ATS4 AppModel Architecture Overview

5/29/2009

Introduction

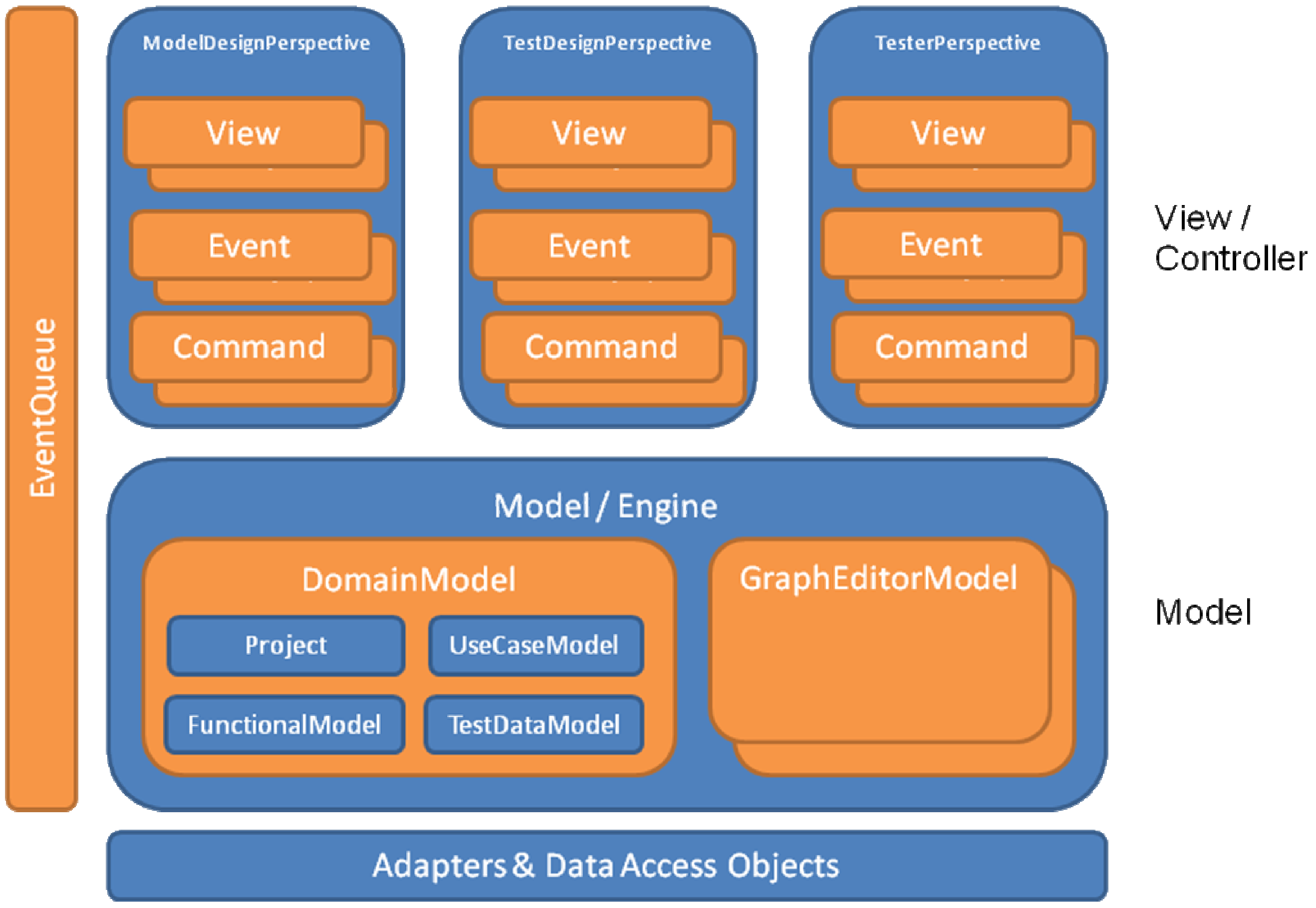
This document describes the overall architecture and design of the ATS4 AppModel tool. This includes the basic design, technologies , the inputs and outputs and files & directories. The detailed design with class diagrams is not documented here as it is laborious to keep up to date with the implementation and would be very complex due to large number of classes. See Javadocs and the source code to get an understanding of the detailed design.

Note: Since the tool is a standalone application which is not deployed on parallel machines and does not have any distributed components, the difference between architecture and design is vague. It could be said as well that this document describes the high-level design.

Basic design

This section documents the overall internal architecture without listing any concrete classes.

The architecture of the ATS4 AppModel application follows well known patterns and practices. The principal pattern is the widely used Model-View-Controller (MVC) design pattern illustrated in the following figure.



When the user operates on *Views*, they post *Events* to *EventQueue* , which may deliver them to one or more interested *Commands* (aka Controllers) for processing. The Commands may change the state of the *Model*, which triggers change notifications - to which the views react by updating themselves. The Commands may also directly set view properties for which there is no respective model property. The views, events and commands are grouped by the perspective, even though some events trigger Commands in several views.

The Model layer forms a, mostly view technology independent, engine that contains data for the ATS4 AppModel project being worked on, and also view-independent algorithms to operate on the data. The Model layer is divided into two parts: the Domain Model and the GraphEditorModels. The former is independent of the view technology, the latter maintains the graph structure for the JGraph library used for graph visualization. There is one GraphEditorModel instance per each model (system model, application model, sub-model).

The Model / Engine layer utilizes various adapters and Data Access Objects (DAOs) to operate on external data and systems.

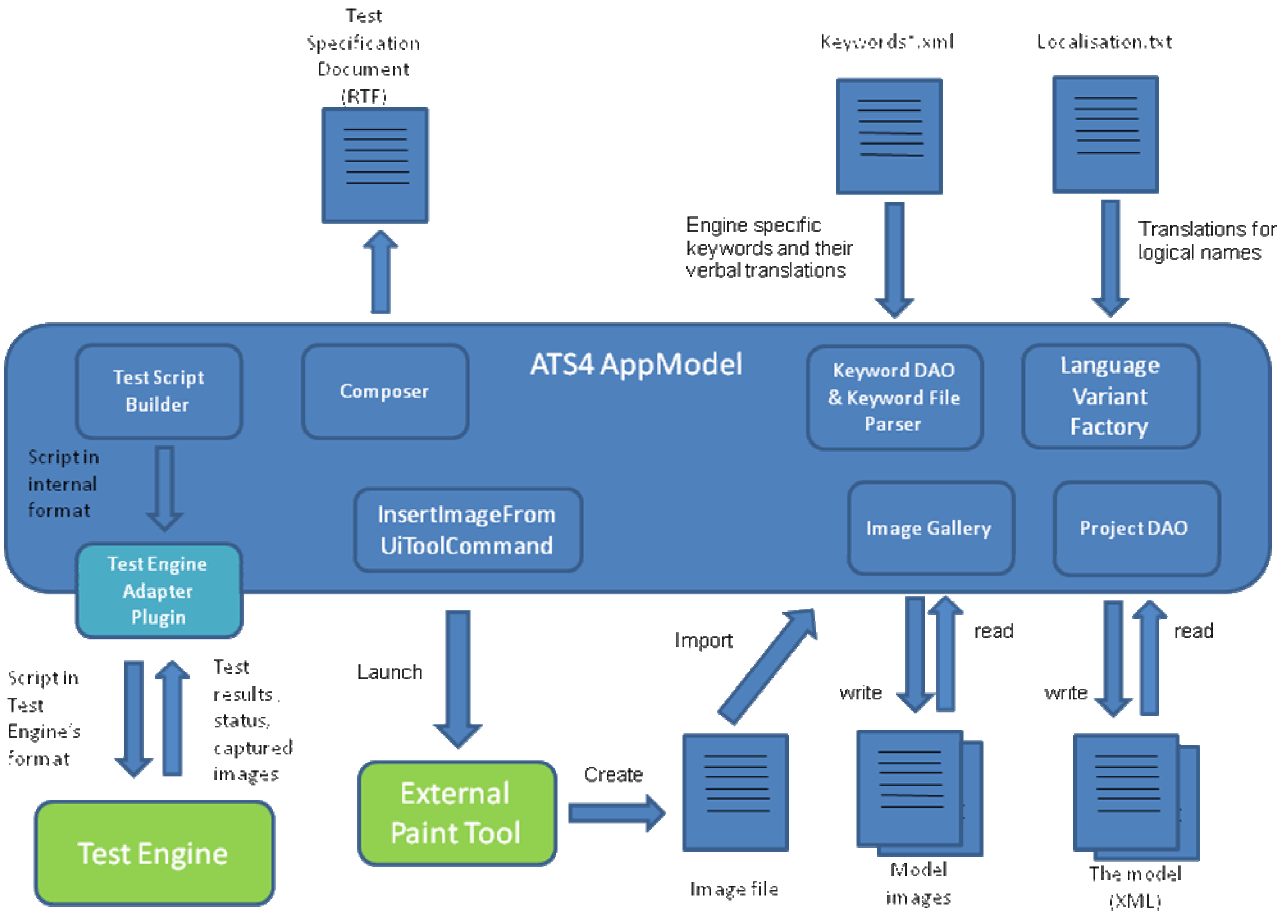
Technologies

ATS4 AppModel is implemented in Java (1.6 or later required), in addition to which the following technologies have been used:

Technology	Usage
Swing	The graphical user interface is implemented using Java's standard Swing technology.
JGraph (www.jgraph.com)	Visualization of the models as graphs. The engine part mostly is independent of JGraph but the view and controller layers depend on it.
iText (http://www.lowagie.com/iText/)	Generation of RTF documents (test specifications)
Log4j	Logging of messages; mostly to help troubleshooting

Inputs and outputs

This section defines the data flows into and out of the application, and the involved components.


Test Specification Document (RTF)
Keywords*.xml
Localisation.txt
Engine specific keywords and their verbal translations
Translations for logical names
ATS4 AppModel
Test Script Builder
Composer
Keyword DAO & Keyword File Parser
Language Variant Factory
Script in internal format
Test Engine Adapter Plugin
InsertImageFrom UiToolCommand
Image Gallery
Project DAO
Import
read
write
read
write
Launch
Script in Test Engine's format
Test results, status, captured images
Test Engine
External Paint Tool
Create
Image file
Model images
The model (XML)

Component	Description
TestScriptBuilder	Generates a test script from the model in ATS4 AppModel's internal XML format. The script is provided to a test engine specific adapter plugin that converts it into test engine specific format.
TestEngineAdapter plugin	An adapter plugin that isolates ATS4 AppModel from test engine specific details and provides a common interface for executing scripts and monitoring test progress. The default implementation uses an XSL stylesheet to translate the script to appropriate format. The default implementation writes the translated script into a file and does not communicate with any test engine.
InsertImageFromUIToolCommand	Implements a simple and generic mechanism for using any paint tool to draw images for system events. When the user creates and stores an image with the paint tool, this component imports it into the model using ImageGallery (see below).
Composer	Generates a localized test specification document for manual testing. The content of the document is entirely based on the model.
ImageGallery	Manages the images of the model. Can import images from external sources (eg when saved from a painting tool).
Project DAO (**D**ata **A**ccess **O**bject)	Handles loading and saving of the project from/to the disk.
LanguageVariantFactory	Provides dictionaries for all language variants defined in the localisation file. The localisation file defines translations for logical names in one or more languages.

Keyword DAO and keyword file parser	These components let ATS4 AppModel access keyword files. Keyword files define engine specific keywords and their verbal translations (“phrases”).

External interfaces

This section documents the TestEngineAdapter interface in more detail since it must be understood to be able to develop a test engine plugin.

A new test engine plugin may be implemented by extending the class AbstractTestEngineAdapter which provides the basic functionality for event and settings handling. This abstract base class saves a lot of implementation work and unifies the certain plugin functions, and thus is is NOT recommended that plugins are developed against the plain TestEngineAdapter interface. See the API documentation for the TestEngineAdapter interface for more information.

The plugins are installed in the "plugins" folder as a JAR package that has the following attributes set in it's manifest file:

Attribute	Description
Class-Path	List of JAR libraries required by the plugin, given as relative to plugins folder.
ATS4AppModel-Plugin-Name	The plugin name that is displayed in GUI.
ATS4AppModel-Plugin-Class	Fully qualified class name of the adapter class.

Further reading: Java JAR Tutorial

http://java.sun.com/docs/books/tutorial/deployment/jar/

You may use the PluginManager class (requires Log4j in the classpath) to load and test the plugin. When finished, drop the plugin jar in the plugins folder and the libraries required by it in the location as defined in the Class-Path attribute. Activate the plugin by selecting it in the ATS4 AppModel's settings dialog.

Files and directories

This section documents the relevant files and directories below the *ATS4AppModel* directory.

File or directory	Purpose
/readme.txt	Installation and startup instructions
/license.txt	The license statements for ATS4 AppModel itself
/ATS4AppModel.bat	Startup scripts for Windows and Unix shells (see readme.txt for

/ATS4AppModel.sh	startup instructions)
/ATS4AppModel.jar	The Java binary code of the ATS4 AppModel application itself
doc/	Accompanying usage documentation in PDF format (introduction, self-learning etc).
lib/	Any libraries used by the ATS4 AppModel application
licenses/	Licenses of the 3rd party libraries
logs/	Log files. Created at first run
models/	Default storage location for ATS4 AppModel projects/models
resources/	Various resources needed by the application: Configuration files, icons, schema files, localization files, keyword files (see below)
results/	Default location for test result files
scripts/	Default location for generated (exported) test scripts
specifications/	Default location for generated test specifications (rtf)
plugins/	Drop test engine plugins in this directory to make them available for ATS4 AppModel. Plugins are discovered automatically when ATS4 AppModel starts. The plugin to be used can be selected in the settings dialog.
resources/keywords.xml	All files matching the keyword*.xml wildcard pattern in this directory are automatically picked up at startup and used to build a list of known keywords.
resources/settings.properties	Configuration properties. Overwritten by ATS4 AppModel on shutdown. Make any changes only when the application is not running.
resources/localisation.txt	The default localization file containing logical names and their verbal translations for each supported locale.
resources/language.properties	All messages of the ATS4 AppModel application
resources/logging.properties	Configuration file for Log4j logging library