teraim

Rules Component

Functional Specification

Version	Date	Autor	Changes
0.1	2012-12-23	Peter Pavek	Initial version
0.2	2013-02-12	Peter Pavek	Changde property “is visible” to “is available”.

Introduction

Background

Teraim develops software for mobile data collection. The key aspect of this software is its configurability and embedded data value validation.

Scope

This document describes the functionality of a component responsible for data-driven processing. A commonly used paradigm of rules is used and this document describes the basic functionality and the use in this context.

Note that, while rules and their components are described here, their syntax is not specified. Rather, this is left to the implementation specification.

Rules

A “rule” is a way to encode an event trigger combined with a condition and an associated action. The triggering event is when a variable receives a new value. A rule consists of a condition part and an action part[1]. When executed, the condition of a rule is evaluated. If it is found to be true, the action part is executed. A rule can be thought of as having the following form:

if *condition* then *action*

the condition and action parts are described below.

Rule Conditions

A rule condition consist of a logical expression. This may contain variables and constants. Constants are numbers and text strings. Variables are data-entry variables (used to store entered values) and threshold variables. (The use of threshold variables is not certain, as it complicates the mental model and the implementation. Maybe, the threshold variables should be regular variables.) More on variables below.

The following operators are available:

- basic arithmetic (e.g., +, –, ×, /)
- logic (e.g., <, >, ≥, =, ≠, ∃, ∧, ∨, ¬)
- grouping parentheses

Rule Actions

Three (four) types of actions are of interest:

1. Value is valid. This is used in combination with validation of data entry. Note that the opposite, i.e., value is not valid, is not available. Rather, a failed condition indicates that the value is not valid.
2. Variable is available. This is used in conjunction with descriptions of what variables to display (via a suitable data entry field.) If condition is *true*, the variable will be hidden. It must also lose its value, if it had one[2].
3. Page is available. This is used in conjunction with descriptions of when a page should be available. If a link or button exists somewhere to access that page, and it uses this kind of rule, it will visually disappear from the screen or be otherwise disabled when the condition of the rule fails. Per extension, all variables set on that page must lose their values.
4. Change object colour, where object for now is button. Sets the background color of the button to a specified value. The values may be either colour names (html color names) or functional names, where the actual color is defined by the overall system colour theme. Functional names are "completed" and "started".
5. Set threshold variable to an expression. This action is used when setting limits for other variables

[1] In this context, we will only be considering forward chaining, i.e., data-driven processing. Thus, Backward-chaining aspects of rules, i.e., invocation while seeking a value, is not considered. Nor is true forward chaining, i.e., rule-by-rule invocation considered.

[2] If the variable has a spare property field, the value is moved to that field instead of being erased.

in a dynamic way. (This is linked to the concept of threshold variables. Not sure about this.)

Evaluation

Rules are executed whenever a variable in the condition part of the rule receives a new value. From a user's point of view, all rules are executed. From an implementation point of view, only rules mentioning the variable in their condition have to be executed.

If a rule condition contains several variables, and a variable is not defined (i.e., has no value set), the rule is ignored. For different kinds of rules (as distinguished by their actions), this has different implications.

1. When a value validation rule is ignored, the the variable value is considered to be OK.
2. When visibility of a variable rule is ignored, the visibility of the related variable is not changed.
3. When a visibility of a page rule is ignored, the visibility of the page is not changed.

The rule set is executed only once. Thus, when a rule sets a threshold variable, this does not result in further rule activity. This is sometimes called forward chaining and is not considered here.

Rule Attachment

Rules are "attached" to one of three conceptual objects – a variable's correctness, a variable's availability, and a page's availability[3]. These clearly mirror the types of rule actions discussed above.

From the configurator users' perspective, this attachment is vital, as this is the context where the rules are defined. When creating a variable, the configurator may choose to attach one or more validation rules and one or more availability rules. Similarly, when creating a page, the configurator may attach one or more availability rules.

Examples – Validity Rules

This set of examples illustrates the use of rules for validation of entered values.

Example 1

A rule is attached to the variable X and determines if the variable is valid.

(1) if $X > 5$ then conclude that X is valid

When X receives a value this rule is triggered. Its condition is evaluated. Assuming that the entered value is 7, the condition is found to be true and the action – concluding that the X value is valid – is executed.

When X receives the value 4, the condition is false. This implicitly implies that the entered value is not valid

[3] When a page becomes available, all buttons that lead to that page become active. Implicitly, the availability of the page leads to the availability of the buttons.

for *X*.

Example 2

Consider now two validity rules for *X*:

> (1) if $X > 5$ then conclude that *X* is valid
> (2) if $X < 10$ then conclude that *X* is valid

Evaluation is done one rule at a time. As long as the rules conclude that the variable is valid, evaluation proceeds to the next rule. Thus, the value 6 would be OK for both rules.

When *X* receives the value 11, rule (1) is executed. Its condition is evaluated to true. So far, the value is OK. When rule (2) is executed, the condition is evaluated to false.

When *X* receives the value 4, rule (1) condition is false. Execution of rules stops for this variable and rule (2) is not executed. Thus, if a rule condition fails, the variable value is considered invalid, independent of the order of rule execution.

Example 3

Consider now these validity rules for *X*:

> (1) if $X > 5$ and $Y < 0$ then conclude that *X* is valid
> (2) if $X < 10$ then conclude that *X* is valid

When *X* receives the value 6, *Y* has no value. Rule (1) is ignored and rule (2) is executed. So far, 6 is considered to be a valid entry for *X*. When, at a later time, *Y* receives the value 2, rule (2) is executed again. Now both *X* and *Y* have a value and the condition can be evaluated. The rule (2) condition is false and consequently, the value 6 is not valid for *X*.

Examples – Visibility Rules

These examples illustrate the use of rules for controlling of the visibility of variables and pages. This implicitly controls if a variable may be edited or not, because if a variable is “visible” it is also editable.

Example 4

Consider the following visibility rule for variable *X*.

> (1) if $Y > 5$ then *X* is hidden

When *Y* receives a value, this rule is executed. If the condition is *true*, the variable *X* is hidden. Further, it loses its value, if it had one. If the condition is *false*, the variable *X* is left visible on all pages where it is present.

Example 5

Consider the following visibility rules for variable *X*.

> (1) if Y > 5 then *X* is hidden
> (2) if Y < 0 then *X* is hidden
> (3) if Z > 7 then *X* is hidden

When *Y* receives a value, rules (1) and (2) are executed. If at least one of the conditions is *true*, *X* is hidden. Further, it loses its value, if it had one. If both conditions are *false*, the variable remains as it is. Rule (3) is ignored and has no influence on the visibility if *X*.

When *Z* receives a value, rule (3) is executed. If the condition is *true*, the variable *X* is hidden. Further, it loses its value, if it had one. If the condition is *false*, the variable *X* is left visible on all pages where it is present.

Example 6

Consider the following visibility rule for variable *X*.

> (1) if *Y* > 5 and *Z* > 6 then *X* is hidden

When *Y* receives a value, *Z* has no value. This rule is ignored. When also *Z* receives a value, this rule is executed. If the condition is *true*, the variable *X* is hidden. Further, it loses its value, if it had one. If the condition is *false*, the variable *X* is left visible on all pages where it is present.
Example 7

Consider the following visibility rule for page *P*.

(1) if X < 5 then P is hidden

When

Variables

In this context, variables are structures of data. Variables have parts, called attributes.

Current value

Historical value

Value type

List of values (if applicable)

Write about “existence” and corresponding tests, ∃.

Definitions

Page	Used in this document to mean the contents of the screen – both the visible parts and the parts that may be hidden, but can be scrolled to.
Variable	Data structure for storing a number of data values.