ACL

Cancer Research UK

Advanced Computation Laboratory,

Cancer Research UK

44 Lincoln's Inn Fields

London WC2A 3PX

Tel: 020 7269 3627 Fax: 020 7269 3186

Tallis Training

PROforma Expressions

Ayelet Oettinger

18 July 2005

1. Overview

Expressions are text strings that are evaluated by the Tallis Engine in order to obtain information about the current state of a process-description during its enactment. The value of an expression is dependent on the state of the process.

1.1 Conditions

Conditions are expressions that evaluate to truth-values; they occur in the several task attributes:

State trigger	An expression that has to be true before the task can be executed (if false, the task remains dormant and waits for it to become true).
Precondition	An expression that has to be true before the task can be executed (if false, the task is discarded).
Cycle until	An expression defining the conditions under which a task stops cycling.
Abort Condition **(Plans)**	An expression defining the conditions under which a plan's state is set to discarded.
Terminate Condition **(Plans)**	An expression defining the conditions under which a plan's state is set to completed.
Condition **(Decision arguments)**	An expression defining the conditions under which the argument applies.
Recommendation Rule **(Decision candidates)**	An expression defining the conditions under which a candidate is recommended.

Expressions can also evaluate to an integer, a real number, a text string, a set of integers, a set of real numbers, a set of text strings, or to the special value *unknown*.

1.2 Assertions

An assertion is a text string describing values to be assigned to data items during the enactment of a process-description. Assertions occur in postconditions:

Postcondition	An assertion that is executed when the task has been performed; often used to specify information that will be needed by subsequent tasks.

2. PROforma Functions

PRO*forma* functions can be integrated to create complex expressions.

2.1 Logical Functions

AND Expression AND Expression

Returns TRUE if all its arguments are TRUE, returns FALSE if any argument is FALSE. If either operand evaluates to UNKNOWN then the resulting expression evaluates to FALSE.

OR Expression OR Expression

Returns TRUE if any argument is TRUE, returns FALSE if all arguments are FALSE. If either operand evaluates to UNKNOWN then the resulting expression evaluates to FALSE.

NOT not(Expression)

Returns TRUE for a FALSE argument and FASLE for a TRUE or UNKNOWN argument.

IF if(Expression, value_if_true, value_if_false)

Returns one value if the specified Expression evaluates to TRUE and another value if it evaluates to FALSE.

isknown isknown(Expression)

Returns FALSE if Expression is UNKNOWN, returns TRUE otherwise.

2.2 Decision Result Functions

result_of result_of(Decision: Single candidate selection)

Returns the decision's committed candidate if a single candidate was committed; returns UNKNOWN otherwise.

result_set result_set(Decision: Multi candidate selection)

Returns the decision's set of committed candidates; returns UNKNOWN if no candidate were committed.

2.3 Task State Functions

is_completed is_completed(Task)

Return TRUE if a Task is in the State, return FALSE if a Task is not in the State.

is_discarded is_discarded(Task)

Return TRUE if a Task is in the State, return FALSE if a Task is not in the State.

is_dormant is_dormant(Task)

Return TRUE if a Task is in the State, return FALSE if a Task is not in the State.

is_in_progress is_in_progress(Task)

Return TRUE if a Task is in the State, return FALSE if a Task is not in the State.

2.4 Candidate Net Support Functions

netsupport netsupport(Decision, Candidate)

Returns Net Support value for Candidate of Decision.

2.5 Comparison Operators

> Number/TextString > Number/TextString

<	Number/TextString < Number/TextString
> =	Number/TextString >= Number/TextString
= <	Number/TextString =< Number/TextString
=	Number/TextString = Number/TextString
! =	Number/TextString != Number/TextString

A real number may be compared with an integer but a text string may only be compared with another text string. If either of the operands of a comparison evaluates to UNKNOWN then the resulting expression evaluates to FALSE.

2.6 Math & Trig Functions

+ Number + Number

If either of the operands of an arithmetic operator evaluates to UNKNOWN then the resulting expression also evaluates to UNKNOWN.

- Number – Number

If either of the operands of an arithmetic operator evaluates to UNKNOWN then the resulting expression also evaluates to UNKNOWN.

***** Number * Number

If either of the operands of an arithmetic operator evaluates to UNKNOWN then the resulting expression also evaluates to UNKNOWN.

/ Number / Number

If either of the operands of an arithmetic operator evaluates to UNKNOWN then the resulting expression also evaluates to UNKNOWN.

ln ln(Number)

Returns the natural logarithm of a number

exp exp(Number)

Returns e raised to the power of a given number

sin sin(angle: radians)

Returns the sine of an angle.

cos cos(angle: radians)

Returns the cosine of an angle.

tan tan(angle: radians)

Returns the tangent of an angle.

asin asin(number)

Returns the arcsine of a number in radians.

acos acos(number)

Returns the arccosine of a number in radians.

atan atan(number)

Returns the arctangent of a number in radians.

abs abs(Number)

Returns the absolute value of a number.

random random()

Returns a random number greater than or equal to 0 and less than 1.

Note: two occurrences of random() that are evaluated without an intervening change in engine state will return the same value. For instance, if a task has the precondition random() = random() then this precondition will always evaluate to true.

2.7 Set Membership Functions

Include(s) Set includes Member

Returns TRUE if Set includes Member, returns FALSE if Set does not include Member. If either operand evaluates to UNKNOWN then the resulting expression evaluates to FALSE.

Note: `Exp1 oneof Exp2` is equivalent to `Exp2 includes Exp1`

oneof Member oneof Set

nth nth(n,Set)

Returns the nth Member in a Set, returns UNKNOWN if the Set does not have an nth Member.

Examples:

`nth(2,["fee","fi","fo","fum"])` evaluates to `"fi"`

`nth(4,[2,3,5])` evaluates to `unknown`.

2.8 Set Statistic Functions

count count(Set)

Returns the number of Members in a Set.

max max(Set)

Returns the largest value in a set of values.

Note: Text strings are compared lexicographically ignoring case, e.g., `max(["bb","bbb","AAA"])` evaluates to `"bbb"`.

min min(Set)

Returns the smallest value in a set of values.

2.9 Set Miscellaneous Functions

sum sum(Set)

Returns the sum of members of a Set.

Examples:

`SUM[]` evaluates to `0`.

`sum([1,1,2,3,5])` evaluates to `12`.

union union(Set1,Set2)

Returns all Members of both Sets.

Note: Unlike a set theoretic union the PRO*forma* union operator does not eliminate duplicates.

Example:

`union([1,2+2,5],[5,2,1])` evaluates to `[1,4,5,5,2,1]`.

diff diff(Set1,Set2)

Returns Members of Set1 that are not Members of Set2.

Example:

`diff([1,2+2,5,6],[5,2,1])` evaluates to `[4,6]`.

intersect intersect(Set1,Set2)

Returns Members of Set1 that are also Members of Set2.

Example:

`intersect([1,2+2,5],[5,2,1])` evaluates to `[1,5]`.

2.10 Time Functions

Now now()

Returns the current engine time.

completed_time completed_time(Task)

Returns the "engine time" at which a Task last entered a State (or UNKNOWN if the Task has not entered the State).

Example:

If `Task1` has a state trigger `Now() - completed_time(Task2) > 1000` it wouldn't start until 1 second (i.e. 1000 milliseconds) after `Task2` has completed.

discarded_time discarded_time(Task)

Returns the "engine time" at which a Task last entered a State (or UNKNOWN if the Task has not entered the State).

in_progress_time in_progress_time(Task)

Returns the "engine time" at which a Task last entered a State (or UNKNOWN if the Task has not entered the State).

2.11 Miscellaneous Functions

 # (concatenate) TextString # TextString

Joins several text strings into one text string.

Example: the expression `"more " # "beans"` evaluates to the text string `more beans`.

forever() forever()

Always evaluates to FALSE. It is intended to be used as a cycle until condition for tasks that cycle forever.

3. Frequently Used Expressions

In this section you will find expression that are frequently used in preconditions, state triggers and postconditions.

3.1 Preconditions

Preconditions are typically used in combination with scheduling constraints. Different expressions are characteristic of preconditions, depending on the workflow pattern.

3.1.1 Preconditions Following Decisions

3.1.1.1 Exclusive Choice

A point where, based on a decision, one of several tasks is chosen. Preconditions control task activation and typically have the following structure:

Result_of(Decision) = Candidate

3.1.1.2 Multi Choice

A point where, based on a decision, a number of tasks are chosen. Preconditions control task activation and typically have the following structure:

Result_set(Decision) includes Candidate

If more than one candidate can trigger task activation, the expression can be structured as follows:

result_set(Decision) = [Candidate1, Candidate2, Candidate3]

3.1.2 Preconditions Following Enquiries

A point where, based on workflow data, one or more of several tasks are chosen. Preconditions control task activation and typically have the following structure:

Data Item Comparison Operator Value

3.2 State Triggers

State triggers are commonly used in dataflow without scheduling constraints. They monitor state and the data changes in the process-description, and typically have the following structure:

Data Item Comparison Operator Value

or one of the following:

- is_completed(Task)
- is_discarded(Task)
- is_dormant(Task)
- is_in_progress(Task)

3.3 Postconditions

Assertions that typically take the form:

Data Item = Value

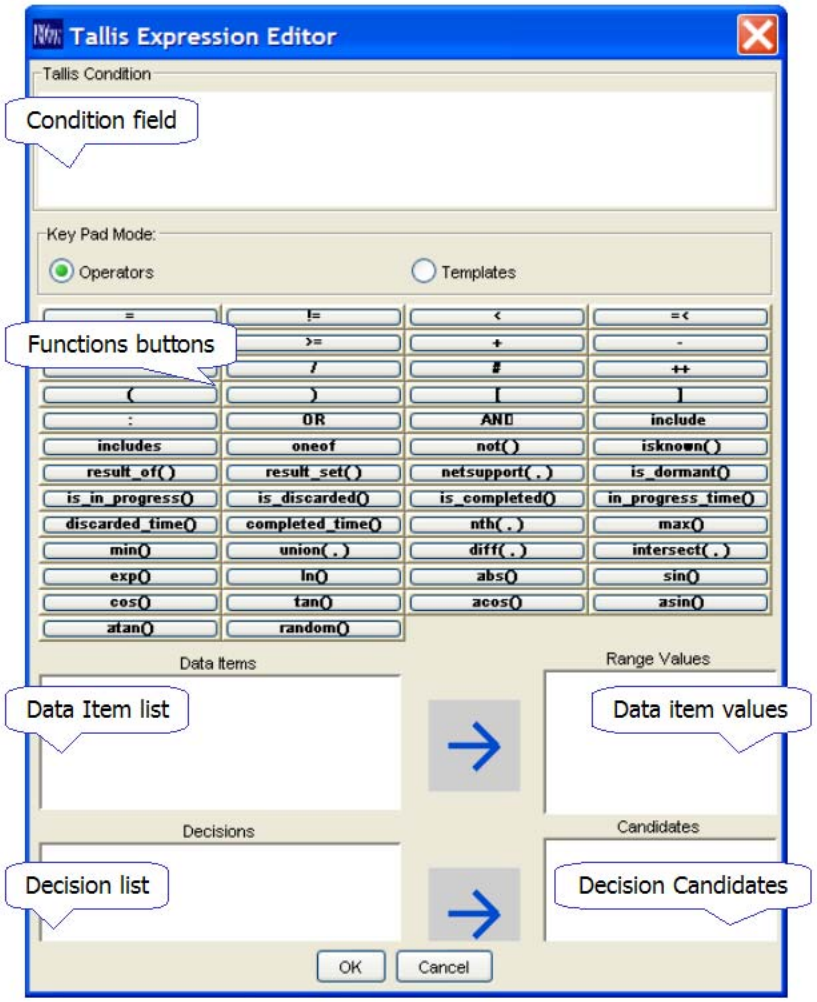
or

Data Item = if(Expression, value_if_true, value_if_false)

4. Expression Editor

The Expression Editor is accessible from every field in the task properties window that contains an expression (e.g., State Trigger, Precondition, Cycle Until). To display it, click on the ellipsis (**...**) button to the right of the field.

The figure below is a screen capture of the Expression Editor:



Expressions are typed in the Condition field.

You can add functions to the Condition field by double-clicking the function buttons. Function buttons have two modes:

1. Operators – double-clicking a button inserts a function into the Condition field

2. Templates – double-clicking a button inserts a template of the function into the Condition field. The template includes both function and placeholders for parameters and values.

   The placeholders and what they stand for:

   X Expression

   T Task

   D Decision

   C Candidate

   S Set

You can add data items or data item values to the Condition field by double-clicking a selected item. The Range Values list displays the values of the selected data item.

You can add decisions or candidates to the Condition field by double-clicking a selected item. The Candidates list displays the candidates of the selected decision.