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

Basic PROforma Concepts

Ayelet Oettinger & Tony Rose

15 July 2005

1. Overview

PROforma is a software technology for specifying clinical guidelines and care pathways and enacting them with the help of a computer to support clinical decision-making and patient care.

Tallis Composer can be used to create and edit process-descriptions, that is, descriptions in the PROforma language of the tasks that are to be carried out by interacting agents (human beings or software components) in order to accomplish some objective.

Process-descriptions can be enacted by the Tallis Engine. The engine keeps track of which tasks need to be performed to advance the process, and provides information to external agents regarding the current state of the process. The engine can also receive messages from agents indicating that they have completed certain tasks or provided data relevant to the running of the process.

2. Tasks: The Building Blocks of the Process-Description

Process-descriptions model processes as sets of interacting tasks.

Tasks represent activities that are performed by external agents (for instance patients or clinicians) who participate in the enactment of the process, as well as activities performed by the Tallis engine itself without any intervention from external agents.

The first stage in creating a process-description is to develop a model of expertise by setting out a collection of tasks that are required in order to achieve a goal.

Once the tasks have been composed into a desired network, the details of each task can be filled in.

The 5 types of Tallis tasks are:

◇ **Enquiry** Represents activities that acquire information.

Example

What is the patient's ethnicity? (Caucasian, Ashkenazi, Other)

● **Decision** Represents activities in which a choice is made between several different options.

Example

Familial genetic risk assessment:

- HIGH (greater than 25% lifetime risk)

- LOW (less than 17% lifetime risk)
- MEDIUM (17 - 25% lifetime risk)

□ **Action** Represents simple activities that effect some change to the external world.

Example

Reassure and discharge patient.

⬭ **Plan** A task container - represents “compound” activities that are defined as a set of tasks.

▽ **Keystone** A placeholder task that can later be modified into any of the above.

Note: By “later”, we typically mean later in the authoring process, i.e. after the overall process structure has been defined, However, in future releases of Tallis we hope to also support dynamic invocation of process-descriptions, so in this context “later” could imply deferring the definition until runtime (enactment).

3. The Four Task States

When a process-description is enacted, tasks generally go through a series of states. All tasks are initially *dormant*; then they are either activated and become *in progress,* or they are ignored and become *discarded*. Finally, tasks that are performed become *completed*.

A task can be in one of four states:

◼ Dormant The Tallis Engine has not yet determined whether the task needs to be performed.

Task colour: Grey

🟨 In progress The task is currently being performed by the Engine.

Task colour: Yellow

⬛ Discarded The Engine has determined that the task does not need to be performed.

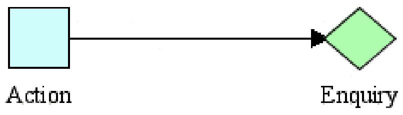
		Task colour: Black
🟦	Completed	The task has been performed. Task colour: Blue

The Tallis Engine determines task states according to the relationships between the tasks, and according to the task properties.

4. Scheduling Constraints

Scheduling constraints are a way of specifying the order in which tasks are enacted.

A scheduling constraint is graphically represented by an arrow connecting two tasks. It indicates that the task at the head of the arrow cannot start until the task at the tail of the arrow (the antecedent task) has completed.



In the figure above, the enquiry's antecedent task is the action. The action has to be completed before the enquiry can be enacted.

A task can have several antecedent tasks. In order for a task to be considered for enactment, *all* its scheduling constraints must be met – that is, *all* antecedent tasks have to be either completed or discarded. Consequently, if a task has one or more antecedent tasks that are still in progress, then its scheduling constraints are not met (yet).

Once the scheduling constraints have been met, the engine then decides whether a task should become in progress or discarded. If *at least one* of the antecedent tasks was completed, then the task can become in progress. However, if *all* the antecedent tasks were discarded, then the task is discarded.

Note: In fact, even if at least one of the antecedent tasks was completed the task might still remain dormant if it has other unmet constraints, such as a state trigger, or precondition, etc.

5. Task Properties

Task properties can be categorised into four groups: general, task specific, scheduling, and execution. This section provides a brief overview of the main task properties.

5.1 General Task Properties

Properties common to all tasks:

Instance Name

An alphanumeric string representing the ID of the task.

Example

PatientDetailsEnquiry

Note: Tallis indexes tasks by Instance Name, i.e. an alphanumeric string chosen by the author. This property is also used to create the task definition, so for this reason, Instance Names must be chosen to be unique (i.e. you cannot create two tasks called “foo” as the definition names will clash). If you want to create a second instance of an existing task, you need to link them.

Caption

A textual label that appears in the tree view and graphical view of the Composer; and as the task’s label when executing via the Tallis engine.

Example

Patient Details Enquiry

Goal

A truth-valued expression indicating the intention of the task.

Example

```
blood_pressure < 120
```

Note: In the current implementation of the Tallis engine goals are ignored, but it is hoped that future releases of Tallis will allow authors to specify the conditions under which the goal of a task is achieved.

Description

A textual description of the task.

Example

This enquiry collects patient demographics.

5.2 Task Specific Properties

Properties specific to the various task types:

Enquiry: Source	A data item whose value must be supplied. **Example** In the enquiry “Patient Details” a source could be the patient’s age, or height, or weight, etc.
Decision: Candidate	One of a set of options to be selected. **Example** In a decision called “Familial genetic risk assessment”, the candidates could be: • HIGH (greater than 25% lifetime risk) • LOW (less than 17% lifetime risk) • MEDIUM (17 - 25% lifetime risk)
Action: Procedure	An action to be performed. **Example** A procedure can be any number of things from a simple message (“reassure and discharge patient”), to instructions for some external system (e.g. update a database).

5.3 Scheduling Properties

Properties that determine when a task should be executed:

State trigger	An expression that has to be true before the task can be executed. The task remains dormant until it becomes true (even if its scheduling constraints have been met). **Example** The action “Control blood pressure” should only be executed if and when the expression `blood_pressure = HIGH` becomes true
Precondition	An expression that has to be met before the task can be executed.

A task's precondition is examined when, *and only when*, its scheduling constraints are met. If at that moment the precondition is false, the task is discarded. Otherwise, it becomes in progress.

Example

The enquiry "Check pregnancy history" should only be executed if the expression `patient gender = female` is true. If it is false (or unknown), then the enquiry is discarded.

Note: If the task should instead *wait* until the specific condition is met, a state trigger should be used.

Event trigger

A string that enables the end-user to trigger a task during the execution of the process-description.

The event trigger is useful for creating processes that do not have a predefined workflow. During the execution of the process-description, the end-user is able to decide which path to take, by triggering the relevant task.

Example

A clinician in a patient consultation may choose to invoke one of a number of services, by selecting the relevant event trigger for each (e.g. "perform_biopsy", or "perform_imaging", or "discharge_patient", etc.)

Note

1. The task does not execute unless the trigger is invoked, even if scheduling conditions are met.

2. When the trigger is invoked, the task executes, even if scheduling conditions are not met.

3. The task state returns to dormant after completion.

5.4 Execution Properties

Properties that affect the behaviour of a task when it is executed:

Number of cycles

An integer defining the number of times a task will be repeated.

Example

The action “Take blood sample” needs to be executed twice in a Glucose Tolerance test.

Cycle until

An expression defining the conditions under which a task will stop cycling.

Example

The action “control_blood_pressure” should continue to cycle until the expression `blood_pressure = normal` becomes true.

Cycle Interval

An integer defining the time interval between cycles.

Example

The action “administer_drug” should cycle every 4 hours.

Note: The time unit is hours.

Postcondition

An assertion that is executed when the task has been performed (e.g. a value assigned to a data item).

Often used to specify information that will be needed by subsequent tasks.

Example

Once the action “control_blood_pressure” has completed the postcondition `drug administered = true` is applied.

Optional

A boolean specifying whether the task must be completed (or discarded) for its parent plan (i.e. the immediately enclosing plan) to complete.

By default, tasks are *not* optional. This means that their patent plan can complete only if the task is completed or discarded. If a task is optional, its parent-plan can complete even if the task remains dormant.

Automatic

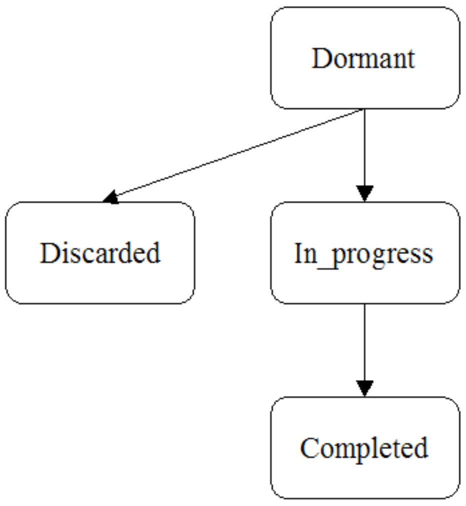
A task that is automatic does *not* require confirmation from the end user. This has different implications on different task types:

- Plan & Enquiry: The automatic property is not applicable for plans and enquiries
- Action: The action is confirmed automatically by the engine (the procedure is not displayed to the end-user).
- Decision: The recommended candidate(s) are committed automatically (the candidates are not displayed to the end-user).

6. Task State Transitions I

The Tallis Engine determines task states according to the scheduling constraints between the tasks, and according to the task properties.

The figure below shows the basic state transitions:



Initial state

All tasks are initially dormant.

A dormant task cannot change state until its scheduling constraints are met (scheduling constraints are met if all the antecedent tasks are either completed or discarded).

Scheduling constraints are met

The task becomes either in progress or discarded, depending on the value of its precondition and on the state(s) of its antecedent task(s).

For it to become in progress the following two conditions must be true:

- If it has a precondition then that precondition must be met, and
- If it has any antecedent tasks, at least one is completed.

If either of these conditions is not met, the task becomes discarded.

Note: In fact, the task might still remain dormant if it has other unmet constraints, such as a state trigger.

Task Completion

In progress enquiries become completed when values have been supplied for their mandatory (i.e. non-optional) sources.

In progress actions become completed when they have been confirmed.

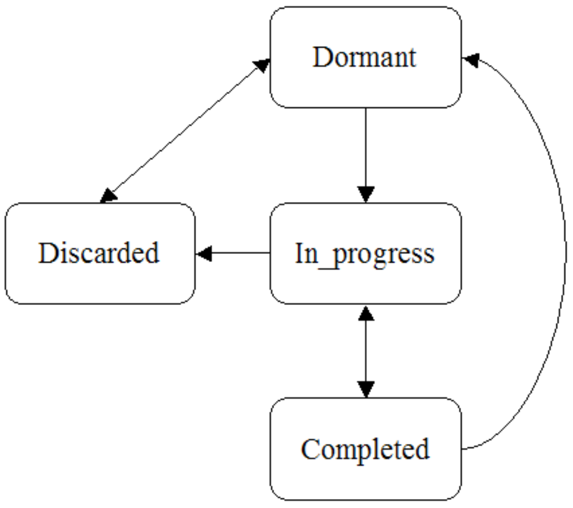
In progress decisions become completed when they have been confirmed.

In progress plans become completed when all their mandatory (non-optional) sub-tasks have completed.

State transitions are further complicated by other task properties. You can learn more about it in the next section.

7. Task State Transitions II

The diagram below shows all possible state transitions:



However, the precise combination of circumstances under which these transitions actually do occur is quite complex. The best way to understand them is to start with the basic rules given in the previous section and be aware of the following complications:

- Tasks that are cyclical return to the dormant state on completion (and then back to in progress, if their cycle Interval is zero or unspecified)

- Tasks can only become in progress if their parent plan is in progress. This applies also to tasks with event triggers.

- Tasks that have been event triggered return to dormant on completion, and their parent plan remains in progress. One consequence of this is that any downstream tasks can never be reached, as their antecedents have not all been performed (recall that scheduling constraints are met if only if *all* the antecedent tasks are either completed or discarded). Other characteristics of event triggers include:

    1. The triggerable task does not execute unless the trigger is invoked, even if scheduling conditions are met.

    2. When the trigger is invoked the task executes, even if scheduling conditions are not met.

    3. The task state returns to dormant after completion.

- Tasks with a state trigger remain dormant until the state condition becomes true, at which point they follow the usual transition path, i.e. they become in progress, and then after being confirmed they become completed. Unlike event triggers they do not return to dormant, and hence are not “re-triggerable” (for this reason state triggers are perhaps better thought of as “wait conditions”).

- Plans can have termination conditions and abort conditions. If a parent plan terminates, it becomes completed, and its contents become discarded. If a parent plan aborts, it becomes discarded, and its contents become discarded. Note also that abort conditions are only checked when scheduling constraints have been met, whereas termination conditions are checked in advance.

An enquiry completes only when values have been supplied for all its mandatory sources. Any outstanding optional sources will remain requested. The same applies to a decision with sources.

8. Enquiries, Sources and Data Definitions

Enquiries are tasks that collect information from agents (end-users or software components), to be used later in the process. An enquiry can hold several requests for data, each known as a source.

Each source is based on a data definition, which defines the structure of the data that the enquiry is requesting. A data definition will typically define the following attributes: name, caption, description, data type, range, default value, and selection mode.

Note:

- Data definitions that are not used by sources are not saved with the process-description.
- Data definitions can be saved separately in data definition libraries (.dd files) to facilitate reuse.
- Data definitions can be imported into a process-description either from a library, or from another process-description.

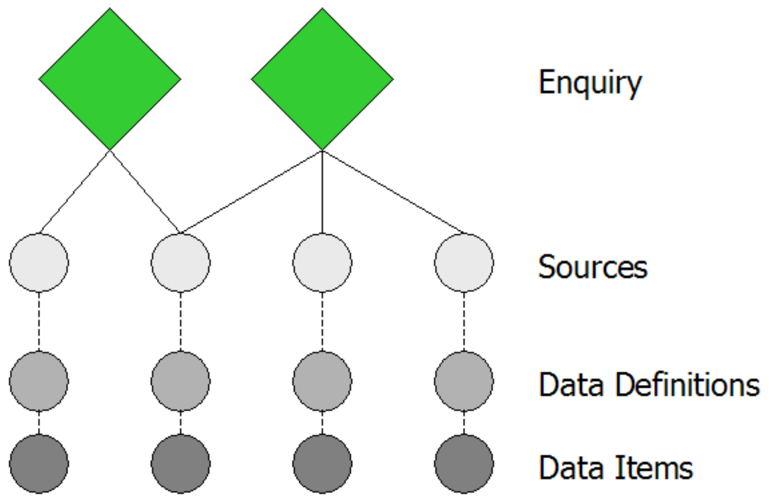
The key difference between a source and a data definition is that a source only exists within the context of an enquiry (or a decision) and therefore also has a status attribute.

The status of a source determines whether data entry is required for the enquiry to complete. By default, source status is mandatory: this means that the enquiry only completes after a data value has been supplied (either by the user, or from some other external agent or system, e.g. a database). If a source's status is optional, the enquiry can complete without it.

When a piece of data is collected during runtime enactment, it is held in an instance of a data definition, and referred to as a data item.

Note: The current version of Tallis allows for only one instance of a data Item per data definition.

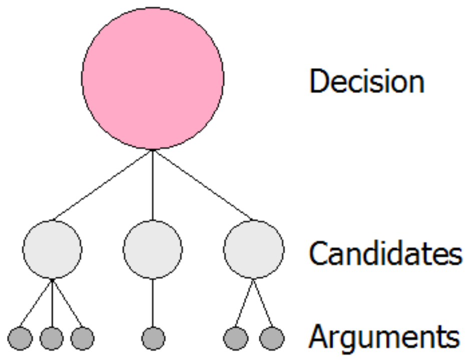
The figure below describes the relationships between enquiries, sources, data definitions and data items: different enquiries can have a common source; each source is based on a singular data definition; and one instance of data item per data definition is formed during runtime enactment.



9. Decisions, Candidates and Arguments

Decisions are tasks in which a choice is made (either by the Tallis Engine or by the end-user) between several different options, known as candidates. PRO*forma* supports decision-making through a mechanism for generating arguments that may be either *for* or *against* a given candidate.

The figure below describes the structure of a decision: candidates are a property of a decision, and arguments are assigned to each candidate.



An argument is defined by two components:

Condition	A truth-valued expression that represents the circumstances under which the argument applies.

	Example `blood_pressure < 120`
Support	The support that the argument offers the candidate if the condition is true. The support format can be: Symbolic – four support categories: the argument can be either *for* or *against* the candidate, or it can *confirm* or *exclude* the candidate Numeric – the support is defined using a real value (i.e. the argument has a weight assigned to it)

The aggregated support of all of a candidate's arguments is known as netsupport.

A candidate's decision rule is used to decide whether a candidate is to be recommended by the Tallis engine. The recommendation rule defines an expression that must be true for the candidate to be recommended (in practice this usually entails setting a minimum netsupport threshold).

Example

```
netsupport(some_decision, some_candidate) >= 1
```

By default, decisions are non-automatic. This means that the candidates are displayed to the end user, who must then select one of them.

Note

- End-users can select non-recommended candidates if they wish
- One or more candidates may be selected if the candidate selection mode is multiple selection

If several candidates have the same netsupport, candidate priority is used to determine the order in which they are displayed to the end user. When a decision is automatic, candidates are not displayed to the end-user, and the recommended candidate(s) are committed automatically.

Note: Although priority is primarily used to order candidates for display, it *can* sometimes affect which candidate gets committed. For example, in an automatic, single selection decision, if more than one candidate is recommended, the one with the highest priority is committed.

Another property of decisions that should be noted is sources. These are requests for data values, and when defined within a decision, this data has to be collected before the decision can complete (see the previous section for more information about sources).

10. Plans

Plans are task containers; they form the hierarchical structure of a process-description: each plan defines a new level in the hierarchy.

Plans have two unique properties:

Abort Condition	If this condition is met, the plan's state is set to *discarded*. All tasks within the plan that have not been completed are discarded, and enactment of the process is halted (i.e., 'downstream' tasks are consequently discarded as well).
Terminate Condition	If this condition is met, the plan's state is set to *completed*. All tasks within the plan that have not been completed are discarded, but enactment continues with the task(s) scheduled next (i.e., 'downstream' tasks).

Note: Abort conditions are only checked when scheduling constraints have been met, whereas termination conditions are checked in advance.

Note: Plans can also affect the scoping of other task properties. For example, a task's event trigger is only active when its parent plan is in progress.