Model for iTasks agents

Wessel van Staal

April 12, 2013

1 Idea

In this document, we present an idea for an agent model for iTasks. We give an informal explanation of the features. Each agent is associated with an unique name and is defined in terms of beliefs and capabilities for tasks. Beliefs are assertions about the environment perceived to be true by the agent. An example of a belief is '$direction := "North"$' where an agent believes its current direction is to the north. An agent can use these beliefs to make decisions when performing tasks.

Capabilities are descriptions of actions the agent can perform to complete tasks, given that the tasks are available and preconditions are met. Each capability is a combination of a *pattern* on the tasks at hand and a possible guard condition. Pattern $t_n \leftarrow task(someTask)$ matches when the task with identifier 'someTask' is available for the agent. The matched task is associated with the identifier (t_n). A capability can be used once the pattern match succeeds and the guard condition holds. Each capability concludes with a sequence of actions to be performed on tasks and beliefs that are updated.

Important is that agents only produce the input necessary to complete tasks, but do not define tasks themselves. The task specification is defined in a separate iTasks model. They do reason about how to complete a task.

Agents obtain information about the environment by inspecting their beliefs and querying the value of tasks. The idea is that agents obtain their information from iTasks in the same way humans do: through observing the information that tasks present.

2 Grammar

⟨*agent*⟩ ::= '`Agent`' ⟨*id*⟩ ⟨*beliefs*⟩ ⟨*tasks*⟩

⟨*beliefs*⟩ ::= ‘`Capabilities`’ ⟨*belief*⟩*

⟨*belief*⟩ ::= ⟨*id*⟩ ‘`:=`’ ⟨*expr*⟩

⟨*tasks*⟩ ::= ‘`Tasks`’ ⟨*task*⟩+

⟨*task*⟩ ::= ⟨*patterns*⟩ { ‘`|`’ ⟨*expr*⟩ } ‘`=>`’ ⟨*actions*⟩

⟨*patterns*⟩ ::= ⟨*pattern*⟩ { ‘`,`’ ⟨*pattern*⟩ }*

⟨*pattern*⟩ ::= ⟨*id*⟩ ‘`<-`’ ‘`task`’ ‘`(`’ ⟨*id*⟩ ‘`)`’ | ⟨*id*⟩ ‘`<-`’ ‘`action`’ ‘`(`’ ⟨*id*⟩ ‘`)`’

⟨*actions*⟩ ::= ⟨*action*⟩ { ‘`,`’ ⟨*action*⟩ }*

⟨*action*⟩ ::= ‘`act`’ ‘`(`’ ⟨*id*⟩ ‘`)`’
| ‘`edit`’ ‘`(`’ ⟨*id*⟩ ‘`,`’ ⟨*expr*⟩ ‘`)`’
| ‘`updateBelief`’ ‘`(`’ ⟨*var*⟩ ‘`,`’ ⟨*expr*⟩ ‘`)`’

⟨*id*⟩ ::= (‘`a`’..‘`z`’)+

⟨*expr*⟩ ::= (‘`0`’..‘`9`’)+
| ‘`"`’ (‘`a`’..‘`z`’)* ‘`"`’
| ⟨*expr*⟩ = ⟨*expr*⟩ | ⟨*expr*⟩ + ⟨*expr*⟩ | ⟨*expr*⟩ - ⟨*expr*⟩ | ⟨*expr*⟩ * ⟨*expr*⟩
| ⟨*expr*⟩ . ⟨*expr*⟩ | ⟨*expr*⟩ ‘`[`’ ⟨*expr*⟩ ‘`]`’ | ‘`[`’ { ⟨*expr*⟩ { ‘`,`’ ⟨*expr*⟩ }* } ‘`]`’
| ⟨*id*⟩
| ‘¬’ ⟨*expr*⟩ | ‘∃’ ⟨*id*⟩ ‘∈’ ⟨*expr*⟩. ⟨*expr*⟩ | ‘∀’ ⟨*id*⟩ ‘∈’ ⟨*expr*⟩. ⟨*expr*⟩
| ‘`value`’ ‘`(`’ ⟨*id*⟩ ‘`)`’
| ‘`hasValue`’ ‘`(`’ ⟨*id*⟩ ‘`)`’

3 Examples

This example shows agents suitable for the 'planning dates' example in Bas Lijnse's thesis. The goal of participants is to select a date for a meeting. The coordinator proposes a few initial dates and other participants select one date. The coordinator makes the final decision.

```
Agent Coordinator
Beliefs
    favoriteDate := "10-04-2013"
Capabilities
    enterDateTimes <- task(enterDateTimes), done <- action(enterDateTimesDone)
    => edit(enterDateTimes, ["10-04-2013", "11-04-2013"]), act(done)

    monitorDateTimes <- task(monitorDateTimes), decide <- action(makeDecision)
    | ∃ v ∈ value(monitorDateTimes). v = favoriteDate
    => act(decide)
```

```
Agent DateChooser1
Capabilities
    chooseDate <- task(chooseDate), done <- chooseDateDone
    => edit(chooseDate, value(chooseDate)[0]), act(done)

Agent DateChooser2
Capabilities
    chooseDate <- task(chooseDate), done <- chooseDateDone
    => edit(chooseDate, value(chooseDate)[1]), act(done)
```

The agent *Coordinator* is responsible for entering a few initial dates. When some other participant selects the coordinators favourite date, the agent makes the decision to pick that date. Agents *DateChooser1* and *DateChooser2* pick the first and second proposed date respectively.