A Formal Framework for Reasoning about Goal Interactions (Extended Abstract)

Michael Winikoff[*]
University of Otago
New Zealand
michael.winikoff@otago.ac.nz

ABSTRACT

A defining characteristic of intelligent software agents is their ability to flexibly and reliably pursue goals, and many modern agent platforms provide some form of goal construct. However, these platforms are surprisingly naive in their handling of *interactions* between goals. Whilst previous work has provided mechanisms to identify and react appropriately to various sorts of interactions, it has not provided a framework for reasoning about goal interactions that is generic, extensible, formally described, and that covers a range of interaction types.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents, languages and structures*; I.2.5 [**Artificial Intelligence**]: Programming Languages and Software; F.3.3 [**Logics and Meaning of Programs**]: Studies of Program Constructs; D.3.3 [**Programming Languages**]: Language Constructs and Features

General Terms

Theory, Languages

Keywords

Agent Programming, Goals, Formal Semantics

1. INTRODUCTION

One of the defining characteristics of intelligent software agents is their ability to flexibly and reliably pursue goals, and many modern agent platforms provide some form of goal construct. However, these platforms are surprisingly naive in their handling of *interactions* between goals. Platforms such as Jason, JACK, 2APL and many others don't make any attempt to detect interactions between goals. There has been work on providing means for an agent to detect various forms of interaction between its goals, such as resource contention [3], and interactions involving logical conditions (e.g. [2]). However, this strand of work has not integrated the various forms of reasoning into a single framework: each form of interaction is treated separately[1].

This paper reports on a framework for extending BDI platforms with the ability to reason about interactions between goals. The framework improves on previous work by being **generic**, i.e. can be customised to provide the reasoning that is needed for the application at hand; **presented formally**, and hence precisely, avoiding the ambiguity of natural language; and that **integrates** different reasoning types into one framework. Due to length constraints, this presentation will be informal and example-driven. Formal details are available upon request.

Our running example is a (very!) simple Mars rover that performs a range of experiments at different locations. The first plan below for performing an experiment of type X at location L firstly moves to the appropriate location L, then collects a sample ($samp$) using the appropriate measuring apparatus.

trigger		*context condition*		*plan body*
$exp(L, X)$	:	$\neg locn(L)$	$\leftarrow$	$goto(L)\ ;\ samp(X)$
$exp(L, X)$	:	$locn(L)$	$\leftarrow$	$samp(X)$

We assume for simplicity of exposition that $goto(L)$, and $samp(X)$ are primitive actions, but they could also be defined as events that trigger further plans. The action $goto(L)$ has precondition $\neg locn(L)$ and add set $\{locn(L)\}$ and delete set $\{locn(x)\}$ where x is the current location.

The sorts of interactions that we want to be able to reason about include **resource** and **condition** interactions.

Goals may have resource requirements, including both reusable resources such as communication channels, and consumable resources such as fuel or money. Given a number of goals it is possible that their combined resource requirements exceed the available resources. In this case the agent should realise this, and only commit to pursuing some of its goals or, for reusable resources, schedule the goals so as to use the resources appropriately (if possible). Furthermore, should there be a change in either the available resources or the estimated resource requirements of its goals, the agent should be able to respond by reconsidering its commitments. For example, if a Mars rover updates its estimate of the fuel required to visit a site of interest (it may have found a shorter route), then the rover should consider whether any of its suspended goals may be reactivated.

Goals affect the state of the agent and of its environment, and may also at various points require certain properties of the agent and/or its environment. An agent should be aware of interactions between goals such as after moving to a location in order to perform

[*]This work was partly done while the author was employed by RMIT University.

Cite as: A Formal Framework for Reasoning about Goal Interactions (Extended Abstract), Michael Winikoff, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. 1107-1108.

[1]Recent work [1] does integrate a range of interaction reasoning mechanisms, but does so indirectly, by translation to Petri nets, which has issues such as traceability.

some experiment, avoid moving elsewhere until the experiment has been completed; or if two goals involve being at the same location, schedule them so as to avoid travelling to the location twice.

2. REASONING ABOUT INTERACTIONS

We provide reasoning about interactions between goals by:

1. Extending the language to allow goal requirements (resources, conditions to be maintained etc.) to be **specified** (Section 2.1).
2. Providing a mechanism to **aggregate and propagate** these requirements (Section 2.2).
3. Defining new conditions that can be used to **respond** to detected goal interactions (Section 2.3).

2.1 Specifying Requirements

We extend the language with a construct $\tau(\pi, R)$ which indicates that the plan π is tagged ("τ") with requirements R, where R is a pair of two sets, $\langle L, U \rangle$, representing a lower and upper bound (we abbreviate $\langle X, X \rangle$ to X). Each set contains resource requirements such as $in(c)$ where c is a condition that must be true *during* the *whole* of execution (including at the start); or $re(e, t, n)$ where e is either r or c, denoting a reusable or consumable resource, t is a type (e.g. fuel), and n is the required amount of the resource. Since in some cases the requirements of a goal or plan can only be determined in context, we provide a mechanism for dynamic tagging: $\tau(\pi, f, c)$ where f is a function that uses the agent's beliefs to compute the requirements, and c is a re-computation condition.

In the Mars rover example we have the following requirements. Firstly, $goto(L)$ computes its requirements based on the distance between the destination and current location: $\tau(goto(L), f(L), c)$ where $f(L)$ looks up the current location $locn$ in the belief base, and then computes the distance between it and L. Secondly, $samp(X)$ requires that the rover remains at the desired location, hence its requirement is $\{in(locn(L))\}$. We thus provide requirements by specifying the following plan body (for the first plan):
$\tau(goto(L), f(L), c); \tau(samp(X), \{in(locn(L))\})$

2.2 Propagating Requirements

We define a function Σ that takes a plan body and tags it with requirements by propagating and aggregating the given requirements. Returning to the Mars rover, let $\pi = \tau(goto(L), f, c); \tau(samp(X), \{in(locn(L))\})$ then the following requirements are computed[2] (if we assume that f returns 20 for the fuel requirement of reaching L from the starting location):

$$\begin{aligned}\Sigma(\pi) &= T(\pi_2; \pi_3, \{re(c, fuel, 20), \\ &\qquad in_s(locn(L)), in_s(\neg locn(L))\}) \\ \pi_2 &= T(goto(L), \{re(c, fuel, 20), pr(\neg locn(L))\}, f, c) \\ \pi_3 &= T(samp(X), \{in(locn(L))\})\end{aligned}$$

2.3 Responding to Interactions

The language of conditions is extended with new conditions: rok ("resources are ok"), $interfere$, and $culprit$. The new condition $rok(G)$ means that there are enough resources for all of the goals in G. The new condition $interfere(g)$ is true if g is about to do something that interferes with another goal. Informally, this is the case if one of the actions that g may do next has an effect that is inconsistent with another (active) goal's in-condition. The condition $culprit(g)$ is true iff the goal g is responsible for a lack of sufficient resources, i.e. if removing g from G makes things better.

The language of responses is extended with new responses: $!\pi$ and PICKME. The former simply executes π (we can define synchronous and asynchronous variants of this). The latter specifies that this goal should be given priority when selecting which goal to execute and can be used to prioritise other experiments to be performed at the current location on Mars (details omitted).

We are now in a position to define a new goal type which uses the conditions and responses defined, along with the underlying infrastructure for specifying and propagating requirements, in order to deal with interactions as part of the agent's goal reasoning process. We extend goals into *interaction-aware goals* by simply adding to their set of condition-response pairs the following condition-response pairs[3]:

$$\begin{aligned}\mathcal{I} \;=\; \{&\langle culprit, \textsc{Suspended}\rangle, \langle notculprit, \textsc{Active}\rangle, \\ &\langle interfere, \textsc{Suspended}\rangle, \langle \neg interfere, \textsc{Active}\rangle\}\end{aligned}$$

We now consider how the different forms of reasoning discussed at the outset can be supported by interaction-aware goals.

Scenario 1: A lack of resources causes a goal to be suspended, and, when resources are sufficient, resumed. Since the goals are interaction-aware, suspension and resumption will occur as a result of the conditions-responses in $\mathcal{I}$. Since updates are performed one at a time, this will only suspend as many goals as are needed to resolve the resource issue. If further resources are obtained, then the suspended goals will be re-activated by $\langle notculprit, \textsc{Active}\rangle$.

Scenario 2: Once the Mars rover has moved to a location to perform an experiment, the requirement of the plan (see π_3 in Section 2.2) is $in(locn(L))$, and therefore it avoids moving again until the sampling at L has completed. Should another goal g' get to the point of being about to $goto(L')$, then this next action interferes with the in-condition, and g' will then be suspended, preventing the execution of $goto(L')$. Once the first goal has concluded the experiment, then it no longer has $locn(L)$ as an in-condition, and at this point g' will be re-activated ($\langle \neg interfere, \textsc{Active}\rangle$).

3. REFERENCES

[1] P. H. Shaw and R. H. Bordini. Towards alternative approaches to reasoning about goals. In M. Baldoni, T. C. Son, M. B. van Riemsdijk, and M. Winikoff, editors, *Declarative Agent Languages and Technologies (DALT)*, pages 164–181, 2007.

[2] J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and avoiding interference between goals in intelligent agents. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 721–726, 2003.

[3] J. Thangarajah, M. Winikoff, L. Padgham, and K. Fischer. Avoiding resource conflicts in intelligent agents. In F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 18–22. IOS Press, 2002.

[4] M. B. van Riemsdijk, M. Dastani, and M. Winikoff. Goals in agent systems: A unifying framework. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 713–720, 2008.

[2] $T(\pi, R)$ denotes that R is the *aggregated* requirements of π. We use $in_s(c)$ to indicate that condition c is required at some unspecified period during execution, and $pr(c)$ denotes that c is a precondition, i.e. required to be true at the start of execution.

[3] Our framework specifies the semantics of goals in terms of condition-response pairs [4]. $culprit$ is short for $culprit(g)$ with g being the current goal, and similarly for $interfere$. $notculprit$ differs from $\neg culprit$ in that it includes the current goal g in the computation of resources, whereas $culprit$ treats it as not having any resource requirements, since it is suspended.