How situated is your agent? A cognitive perspective

Daghan L. Acay[1], Liz Sonenberg[1], Alessandro Ricci[2], and Philippe Pasquier[3]

[1] DIS, The University of Melbourne 111 Barry Street Victoria 3010, Australia
lacay@pgrad.unimelb.edu.au, l.sonenberg@unimelb.edu.au
[2] DEIS, U. Bologna in Cesena Via Venezia, 52 Cesena (FC), Italy
a.ricci@unibo.it
[3] SIAT, Simon Fraser University, 102 Ave. Surrey, British Columbia, Canada
pasquier@sfu.ca

Abstract. Software agents are situated in an environment with which they interact reactively or in a goal-directed fashion. Generally, such environments do not assume a structure, hence are deemed to be unpredictable. Recent approaches adopt an environment model where artifacts form the building blocks which represent functional components that an agent can exploit for reaching its goals. It has been argued that software agents can improve/amend their capabilities at run time through the use of (new) artifacts as possible means. We argue that such a run time adaptation by the agents can be realized by creating an appropriate relationship between agent reasoning and the functionality of the artifacts. We have coined the term *extrospection* to refer to the act of an agent *reasoning about the tools*. In this paper, we first identify the features of extrospection, then, we extend the belief, desire, intention (BDI) agent deliberation cycle to encompass extrospection.

1 Introduction

Although, there is a growing body of work in agent literature that highlights the importance of the environment for agent systems [1], the relation between the agent and the environment at the cognitive level has not been well established. For example, an agent designer, in general, is responsible for constructing an internal environment model that may be consulted during deliberation. Such an internal model reflects an agent designer's anticipation about objects and available actions in the environment, in contrast to the actual environment as the agent experiences at run time. Thus, the cognitive awareness of the agent about the environment does not drive from the actual interaction but is limited to the agent designer's intuition at the design time.

As the scale of multi-agent systems (MASs) increases, the above approach to agent design leads to two problems. Firstly, the agent and the environment it acts in need to be developed by different designers. For example, in the context of web services, agents and web services are designed by different designers. Thus, the designer of an agent can not capture all the possible service combinations (i.e. environment). Secondly, even if the agent designer presumes a subset of services and constructs an internal model accordingly, the overall agent environment interaction is still prone to failures. The reason is that, some inconsistency between the internal model and the actual environment may arise over time. For example, some services included in the agent's internal model may become unavailable (off-line) or be inconsistent (due to the service (up/down)grade).

So, agents should learn, understand and adapt to, their environments at run time. In this paper, we argue that the adaptation could be more tractable if agent environments are engineered appropriately.

An analogy can be drawn between such an environment-agent dichotomy and human practices. Humans populate their environment with tools to create a more suitable environment for their practices. They perceive action possibilities in their environment and find alternative means in the context of tool availability [2]. Humans are not designed for an anticipated environment but are capable of adapting their behavior in different environments by exploiting alternate tools. In terms of software agent research terminology, one may argue that humans have *desired ends* in their heads but find *means* to reach those ends by exploiting the tools in the environment.

Considering this insight, we support the idea that one can adopt an explicit tool model [3] for engineering agent environments. We further argue that cognitive awareness of tools can support agent adaptation. Through awareness, software agents can reason about tools and modify their behavior accordingly at run time. By reasoning we mean that agents should flexibly substitute tools as means for their goals.

The first requirement was explored by Ricci et. al. [4] in their agent and artifact framework (A&A), where they have also developed a computational model for the artifact abstraction [5]. In their work they have described dynamic creation and linking of artifacts in distributed environments. They have also discussed the model for sensing the effects of tool use through, what they call, the *agent body*.

We explore the relation between the agent reasoning and the tool[1] availability. Referring to the title, we adopt the perspective that agent's situatedness should not be reduced to sense/act loop but should embrace the cognitive situatedness. By this we mean that agents adapt to the environment by understanding tools in the environment. Thus, agents can realize the full potential of the environment at run time by discovering alternative means.

We have coined the term *extrospection* to refer to the reasoning about the tools (e.g. real time discovery, selection, and use). The term *extrospective agent* refers to the agents endowed with this capability.

It is fair to interpret extrospection as another way to capture adaptation (i.e. learning and planning) and introducing a new term may seem hardly justifiable. On the other hand, the reader should bear in mind that extrospection approaches to the adaptation problem from an environment design perspective instead of an agent design perspective. That is, our use of the term has implications for engineering an agent environment such that the adaptation is tractable. Thus, the fundamental questions for extrospection are: how can we engineer the agent environment? how can agents reason about available action in the environment instead of what they know a priori about it? what are the implications of environment engineering to the reasoning cycle (i.e. query, learn, deliberate, plan, execute)? etc. For that, we believe a new term is helpful.

Engineering the environment for supporting human work is considered also in formative design. Cognitive work analysis (CWA) [7] – a formative design methodology– is employed in human-machine systems and suggests that machine interfaces should be designed such that the interface *explicates* the *functional properties* of the environment. Thus, instead of supporting humans with strict procedures, humans can identify the appropriate behavior from available actions in the environment. Rasmussen [8] calls this "completing the design at run time."

[1] In this paper, the term *artifact* is used for the computational model and the term *tool* is used for emphasizing the reasoning

We believe that tool and agent models are the core components for applying formative design techniques to the MAS development.

Expected benefits of employing formative design ideas for extrospection for MAS development are: (i) agents can complete the design by discovering and opportunistically using tools at run time, (ii) the reuse of components will be enhanced, and (iii) domain independent meta-level reasoning can be built into the agents.

Contributions of this paper are the introduction of our initial ideas and concrete steps taken towards answering questions such as, (i) how does the availability of a tool extend capabilities of agents at run time? and (ii) what are the modifications required to the deliberation cycle of the agent?

This paper is structured as follows: the philosophical motivations shared between this work and the A&A framework is set forth in Sec. 2. A running example is introduced in Sec. 3. In Sec. 4 we extend the relation between the A&A framework and our work by identifying the artifact layer, the concept layer and the extrospective agent. Later, in Sec. 5 we elaborate on the extrospective agent's architecture and the associated abstract interpreter as an extension to Jason BDI model [6]. Implications of the existence of tools in the environment for metalevel reasoning are discussed in 6. Sec. 7 is the comparison with other approaches. We briefly present future work and the conclusion in Sec. 8.

2 Background: Philosophical Underpinnings

The A&A framework introduces artifacts as a first class entity along with agents for developing a MAS. Similar to the A&A framework, our work on the extrospective agent is inspired by the psychological theory called Activity Theory (AT) [9]. It is emphasized in AT that humans have the potential to change their environments. In other words, humans no longer live in natural habitats but populate them with tools to make the environment more suitable for their practices. Thus, their speed, power, and intelligence are enhanced beyond their innate nature through the proper use of tools [10].

Another important claim of AT is that tools represent the scrutinization of the experiences of those who have encountered and solved a particular problem in the past [9]. The solutions manifest themselves in (i) the physical properties of the tool (e.g. shape, size, etc.) and (ii) the knowledge of the functionality and the use of tools. The use of tools influences the external behavior through its physical characteristics. Moreover, tools modify the choice of behavior after the knowledge of the tool use is perceived.

In that sense, our work is complementary to the A&A framework where a computational model for the artifact has been introduced and we introduce a computational model for the cognition of tool use. The A&A framework and the extrospective agent together aim to benefit from the claims of the AT in the context of the MAS development and execution.

3 An Example: Production Cell

The problem we are addressing is the run time adaptation of agents to different environments. Firstly, an agent does not need to be coded with the a priori knowledge of possible actions in the environment. Thus, it needs to discover what it can do in the environment. Secondly, actions in the environment may not be fixed due to changes (e.g. some services may go off line, new services can

be introduced, or present services may be updated). Thus, we have chosen an example that reflects these points and rather simple in nature.

The example of production cell is taken from Meneguzzi et. al. [11] where a variety of component types with different production demands are produced. The production units in the production cell may fail, be removed, be added, or be upgraded. An agent *with the knowledge of the production demands of the components* is designed to work in different production cells. That is, the agent should discover and use production units (i.e. adapt) in the production cell it is situated. Moreover, the agent should be responsive to the changes made to the production units over time. The task of the agent is to schedule components to the existing units.

An instance of the production cell with six devices (a Feed Belt, a Deposit Belt, four Processing Units) and a Crane that can freely move the components over the devices in the cell are shown in Fig. 1. Components that need to be

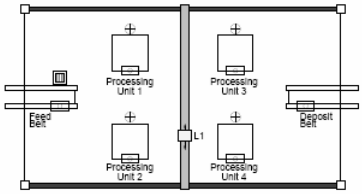

Fig. 1. Production cell taken from [11]

processed enter the production cell through the Feed Belt. After a component is processed, it is removed from the cell through the Deposit Belt. Each Processing Unit can perform a set of operations and can accommodate a single component at a time. The type of a component determines the necessary Processing Unit to be used.

Meneguzzi et. al. [11] have modeled the overall production unit as a single agent with propositional planning capability. Besides, we assume two distinctive entities, e.g. tools, and an agent. The tools such as the Feed Belt, the Deposit Belt, and Processing Units form the environment for the agent. Yet, the Crane is conceived as an agent with the capability of scheduling and moving the components over the tools.

We further say that, the Process Unit 1 `procUnit1` can drill a component whereas the `procUnit2` can both drill and paint. Similarly, the `procUnit3` can cut and the `procUnit4` can polish a component. Moreover, there are three types of components that may come to the production cell. The first type `type1` requires drilling and the `type2` requires painting and drilling. The third type `type3` requires both cutting and polishing. This example is going to be used in the following sections.

4 Extrospection Framework

As mentioned in the introduction, the CWA [7] starts the design with analyzing the environment where work takes place. Later, the CWA considers the cognitive capabilities of employees to finalize the system development. Thus, two important aspects of the analysis are, the components in the environment and

the agents. In our framework, the former corresponds to the tools and the latter corresponds to the extrospective agent. In addition, we introduce a third layer where the functionality of tools is symbolically represented.

The first of the three layers is composed of artifacts that implement some sort of operation. Such operations can be invoked possibly with effects perceived later by the agent. The second layer is called the concept layer and it stores the symbolic information about the functionality and the operating instructions of the artifacts. The two layers belong to the environment design. Finally, the third layer is the extrospective agent which is responsible for reasoning about selecting and using the tools. In the following subsections, a brief summary of the artifact and the concept layers and pointers to related literature are given.

4.1 The Artifact Layer

For this layer we adopt the A&A meta-model [4, 5], where the notion of artifact is used to model non-autonomous state-ful entities, specifically designed by MAS environment engineers to encapsulate some kind of function[2], and to be instantiated and used dynamically by agents to support their activities.

The functionality of an artifact is structured in terms of *operations*, whose execution can be triggered by agents through the artifact's *usage interface* which in turn is composed of *controls*. Agents can trigger and control the operation execution through controls with the necessary input parameters. Besides the controls, the usage interface might also contain a set of *observable properties*; the properties whose dynamic values can be observed by agents without necessarily interacting with (or operating upon) the artifact.

The execution of an operation may result in changing the artifact's inner (i.e., non-observable) state. The operation execution can be conceived as a process, combining the execution of possibly multiple atomic guarded operation steps. In order to avoid interferences, the usage interface is disabled during the (atomic) execution of a single operation step. The operations execute asynchronously to the activity of the agent. The information flow from artifacts to agents is modeled in the form of observable signals that are perceived by agents.

As a principle of composition, artifacts can be linked to enable the artifact–artifact interaction. This is realized through the *link interfaces*, e.g. using a remote control with a TV. The artifact topology is handled by the notion of *workspace*. Agents can use and observe only the artifacts belonging to their workspace. Workspaces provide basic default tools (artifacts) that agents can use to dynamically discover the artifacts currently available in the workspace (registry tools), to instantiate dynamically new artifacts (factory tools), to manage organization and security issues (organization tools), and so on." (see accompanying paper [12]) The artifacts of different workspaces – possibly on different network nodes– can be linked through the link interfaces discussed above. Agents can join and work simultaneously on multiple workspaces.

Analogously to the artifacts in the human case, in A&A each artifact is equipped with a "manual" describing: the artifact's function (i.e., its intended purpose), the artifact's usage interface (i.e., the observable "shape" of the artifact), and artifact's *operating instructions* (i.e., the correct use of the artifact). The manual is meant to be inspected and used at run time by agents, for reasoning about how to select and use artifacts. In this paper, the manual is described using the concept layer and discussed in Sec. 4.2.

Considering the A&A framework, we introduce some assumptions that are necessary to limit the reasoning about tools.

[2] The term *function* is used here as in the design theory, a synonym of functionality

Assumption 1 (A1) *All operations supported by a tool are atomic and do not support concurrency.*

A1 simplifies the tool use for metalevel reasoning, as does A2.

Assumption 2 (A2) *Operation generates a* finished event *when execution completes.*

A3 emphasizes the asynchronous execution of tools and adds a temporal constraint.

Assumption 3 (A3) *Operations takes certain amount of time independent of agent activities.*

Finally, we assume that each tool may have more than one functionality. For each functionality there is one and only one operating instruction.

Assumption 4 (A4) *A tool may have multiple functionality. Each functionality is realized through a single operating instruction.*

We may apply these assumptions to the production cell example. The `procUnit2` conforms A4 by having two functionalities. A functionality can be realized by following the respective operating instruction. When the `procUnit2` is drilling a component, it is assumed to be busy (A1, A3). Yet, the agent may concentrate on other activities during drilling (A3). The agent will be informed – regardless of working on another activity– when drilling completes (A2)

In addition to the above assumptions, the production demand of a component may incorporate multiple tools. For example, a component of `type3` may requires both the `procUnit3` and the `procUnit4`.

The use of tools requires two sorts of planning. The first is employed for realizing the precondition of the operations of a single tool. The second is necessary for orchestrating the use of multiple tools within a single intention. For example, the extrospective agent (Crane for this example) needs to plan to acquire the knowledge regarding the preconditions for drilling, e.g. `holeCoord(X,Y)`. Only after this information is available to the agent, the `startDrill` operation can be invoked over the tool. The crane agent is also responsible for planning to move components from one machine to another using its *moving* capability, which corresponds to the planning for the tool orchestration.

4.2 The Concept Layer

The concept layer symbolically describes the functionality and the operating instructions of the artifact (i.e. artifact manual). Through the concept layer, agents can incorporate tools' use knowledge into their deliberation cycle. The ontology called OWL-T [13] is used for this purpose. T stands for (T)ool and OWL for the employed ontology language, Web Ontology Language (OWL) [14].

The OWL-T for tools can be compared – but can not be reduced– to API documentation for software objects. An *API documentation* conveys the functionality of the implemented objects to a human programmer. However, such documents are not useful for the software agents since they are written in natural language. Besides, the OWL-T is a formal language targeting the software agents. Analogous to an API documentation, if the concept layer is not supplied by the environment designer, it does not hamper the artifact operation given that the agent knows the existence of the tool and the corresponding operating instructions (i.e. internal model). The OWL-T is merely useful for run time discovery. That is why we introduce the concept layer as a separate layer.

Here, we will concentrate on the three most relevant aspects that are captured by the OWL-T. The detailed account for the OWL-T has been given by Acay et. al. [13]. Firstly, the OWL-T aims to relate the *goals* of an agent and the *functionality* of a tool. For example, the functionality of the `procesUnit1` can be defined as `drill(C)`[3]. Then, an agent, which has a component of `type1`, queries the concept layer to discover a tool that supports drilling. Upon query, environment returns the information about the `procesUnit1` to the querying agent.

Secondly, the OWL-T captures the *operations* and the associated *preconditions*. If the `procesUnit1` supports the operation `startDrill` which has an associated precondition `drillSize(X)`, the precondition is linked to the beliefs of the agent through the OWL-T. That is, the OWL-T enforces the agent to have a belief of the form `drillSize(X)`. Thus, an agent should plan to acquire drill size as mentioned in Sec. 4.1.

Finally, the OWL-T captures the *operating instructions* of tool for a particular functionality (see A4). We define an operating instruction as a sequence of operations that should be followed to realize the tool functionality. For example, `procesUnit1` may require the agent to enter the coordinates of the hole via the operation `enterCoor(X,Y)` by a number pad. In this respect, `enterCoor(X,Y)` should precede `startDrill`.

The concept layer is updated when there is a change in the artifact layer e.g., new artifact is included, existing artifact is updated, etc. However, concept layer update will not be covered here due to space limitations. The following sections elaborate on the extrospective agent under the assumption that the artifact layer and the concept layer are synchronous.

5 The Extrospective Agent Mind

The extrospective agent architecture depicted in Fig. 2 is based on the Jason BDI agent architecture by Bordini and Hübner [6]. Similar to the BDI agent, the extrospective agent has beliefs about the state of the environment. The beliefs are updated through sensing. The extrospective agent also has a set of goals. Each goal defines a desired state of the environment that the agent wants to reach. Finally, the agent acts in the environment through its effectors. Generally, in BDI agent literature, agents are employed with a library of plans. The plans are partial recipes that guide the agent through means-ends analysis [15].

The architectural additions leading to the extrospective agent aim to support: (i) querying the concept layer to discover tools, (ii) selecting which tool to use, (iii) planning for orchestration and operation enabling, and (iv) focus management during the tool use. The rectangles in Fig. 2 represent the data modules within the agent. The circles are processes that transform or merge the data in the respective data repositories. For the details of data repositories such as the belief base, the events, the plan library, and the intention we refer the reader to [16, 17, 6]. Here, we will concentrate on the extensions.

When an agent enters an environment, it should discover the tools. Run time discovery is done via a query mechanism. The agent queries the environment based on its goals. Upon query, the environment tries to match the agent's goal to a tool functional definition in the concept layer. If matching is successful, *special percepts*[4] are sent to the agent. Different query strategies are discussed in Sec. 5.1.

[3] As convention we use upper case letters for the variables in terms and lower case letters for the constants.

[4] Percepts that are related to tool information are distinguished from other percepts.

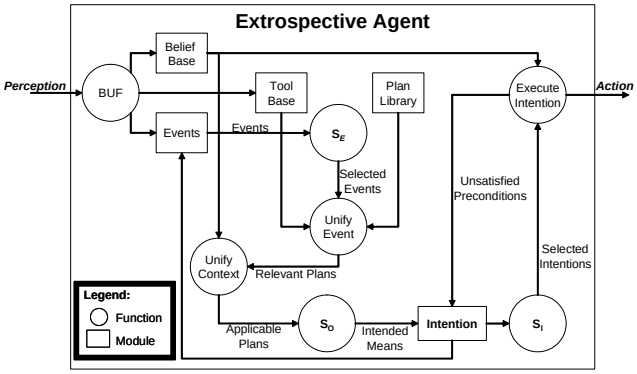

Fig. 2. The extrospective agent architecture

The first addition is the data repository called the *tool base* (TB). Logically, the concept layer resides in the environment and TB resides in the agent. The TB stores the knowledge about a particular tool that the agent has discovered before. The TB is also different form plan library (PL). TB is updated at the run time by special percepts that are retrieved from the concept layer whereas PL is developed by the agent designer at the design time and static. The existence of the TB increases the potential success of the agent in novel and dynamic environments.

The TB is also distinguished from the PL for its support for planning. Planning for realizing the context condition [6] of a plan originating from the PL may seem similar to the planning for realizing the preconditions of each operation of the plan originating from the TB. However, planning for the context condition requires even more anticipation (extended internal model) by the agent designer, e.g. cost of an action, time taken, etc. Planning for the context also has a larger search space [18, 19], hence may not be viable for resource bounded agents. The TB overcomes the first difficulty by mirroring the concept layer that is supplied by the environment designer. Such information is also valuable by guiding the planning process by pruning the search space through the operating instructions. We will consider the related work incorporating planning to the BDI agents in Sec. 7.

The other two extensions to the Jason model are the arrows between (i) the belief update function (BUF) and the TB and (ii) the intention execution function and the intention module. The former arrow represents the process that parses the special percepts into the TB. The latter arrow corresponds to the planning for precondition satisfaction.

5.1 Abstract Interpreter

The overall abstract interpreter[5] of the extrospection agent is given in Table 1. The basic structure for decision making is a loop, in which the agent continuously:

– observes the world and updates its beliefs,

[5] This section relies on the AgentSpeak(L) terminology which can be found in Rao [20].

– deliberates on which ends to achieve,
– uses means-ends reasoning to find the applicable plans from the tool base or internal plan library, queries the concept layer if necessary,
– acts until the entire plan is consumed

The extrospective agent program starts with the agent's goals and the initialization of the beliefs about the environment. Following the initialization, the reasoning cycle starts updating the belief set, the event set, and the tool base by processing the set of percepts ρ_{set} received from the environment. The BUF updates the beliefs buf(B,ρ) and the events $buf(\rho)$ similar to AgentSpeak agents [16]. The only extension to the BUF is the update of the TB buf(TB,ρ). In this case, the BUF detects the special percept[6] regarding the tool information and parses it into the TB. After the BUF is done, the options function *options*(G,E) will generate new goals by taking the unsatisfied goals left from the previous execution cycle and the new events generated either by percepts (external goals) or by the intention stack (internal goals). The resulting goal set G represents the ends that the agent wants to achieve. Later, the goals[7] are sent to the event selection function S_e(G) to select the next goal to be pursued.

Unify event function (UEF) *unifyEvent*(S_e(G),PL,TB) takes the selected goal, the TB, and the PL and matches the available means with the goals (i.e. ends) of the agent. The UEF returns goal-means pairs as *relevant plans* P_r. If P_r is empty then the agent automatically *queries* the environment to discover tools. To avoid infinite loop, the agent should not already *believe* that there is no tool available (¬**Bel**(noTool(G))). In the latter case, the goal and the related intention is dropped. Because the agent is sure that, neither the available tools in the environment nor the plans in the agent's PL can accomplish the goal. Of course, this is true under the condition that the agent does not have any other information source such as another agent or a human user [21].

It is important to note that, querying the concept layer only if the agent could not find any means in its TB or PL corresponds to just one strategy. We employed this strategy in Table 1. A more general approach may include different query strategies. For example, the agent may (i) query before every means-ends reasoning or (ii) query at every percept update. Although, the time required for processing the percepts increases, the former strategy is beneficial for finding the most effective tool as means and the latter is beneficial for synchronizing the TB with the concept layer to decrease the chance of misinformed means-ends reasoning due to out of date TB.

The query strategy becomes important in the context of dynamic environments. As it has been mentioned in Sec. 3, the agent may need to adapt to the changes such as Process Unit upgrades and failures. In such situations, the concept layer is updated accordingly to reflect such changes. The synchronization of the TB with the concept layer is then a question of selecting the appropriate query strategy based on the characteristics of the environment, e.g. the rate of change in the tool composition. At this stage, we will not consider those situations and stay faithful to the query after failure strategy.

If the agent can find a relevant plan – either from the TB or from the PL– it tries to find an *applicable plan* through unify context function (UCF) *unifyContext*(P_r,B). The UCF filters the relevant plans by finding those whose context is satisfied by the beliefs of the agent. Again, if the applicable plan set π is empty then the agent drops the goal as mentioned above.

[6] At the initial state there are no percepts regarding the tools since, they are only available after a query.

[7] Strictly speaking, both the achieve goals and the test goal are events [16]. Here, we are interested in the achieve goals.

```
B ← B_0
G ← G_0
WHILE true
%get percepts ρ_set from the environment
     WHILE not empty(ρ_set)
          %get next percept ρ from ρ_set
          B ← buf(B,ρ)
          E ← buf(ρ)
          T ← buf(TB,ρ)
     END WHILE
     G ← options(G,E)
     P_r ← unifyEvent(S_E(G),PL,TB) %unifies plan,goal,tool
     IF P_r = Ø THEN
          IF not ¬Bel(noTool(G))
               query(G)
          ELSE
               dropGoal(G)
          END IF
          CONTINUE
     END IF
%At this point we have relevant plan(s)
     π ← unifyContext(P_r,B) %unifies plan, belief
     IF empty(π)
          dropGoal(G)
          CONTINUE
     ELSE
%At this point we have applicable plan(s)
          π_im = S_O(π)
     END IF
     I = pushIntention(G,π_im)
     π_i = S_I(I)
     WHILE ¬ endOfPlan(π_i)
          IF π_i ∈ PL
               α = head(π_i)
               IF action(α)
                    execute(α)
               ELSE IF goal(α)
                    updateEvents(α)
               END IF
          ELSE IF π_i ∈ TB
               α = head(π_i)
               WHILE not empty(preList(α))
                    p = next(preList(α))
                    IF checkPre(p,B)
                         unify(p,B,π_i)
                    ELSE
                         π_i = [p|π_i] % update intention
                    END IF
               END WHILE
% check the updated intention
               α = head(π_i)
               IF action(α)
                    execute(α)
               ELSE IF goal(α)
                    updateEvents(α)
               END IF
          END IF
          π_i ←tail(π_i)
          π_i = S_I(I)
     END WHILE
END WHILE
```

Table 1. The abstract interpreter for the extrospective agent

The option selection function $S_O(\pi)$ selects the plan to be executed π_{im}. The selected plan is pushed to the intention set by *pushIntention*(G,π_{im}) function.

At a given time an agent may have concurrent intentions. The intention selection function S_I(I) selects one of the intentions for execution. The S_I(I) may also suspend and resume the currently active intention. The details of the

option selection function and the intention selection functions are given in Sec. 6.

The only branching after the intention selection is based on the origin of the plan. If the plan came from the PL then the standard execution of the Jason agent continues until all the steps of the plan are executed ($\neg endOfPlan(\pi_p)$) by popping the head of the plan repeatedly (α =head(π_i)). If the current step α is an action predicate, α is executed in the environment. On the other hand, if α is a sub-goal then an internal event is generated $updateEvents(\alpha)$.

In the latter case – the plan source is the TB– the plan is the operating instruction as mentioned in Sec. 4.2. The difference between the precondition and the context condition is that the first one applies to each operation in an operating instruction, but the later applies to the whole plan and checked once during *unifyContext*. Thus, even after the tool is intended to be used, the agent must make sure that every precondition of the operation holds in its beliefs. If they do, the agent unifies its beliefs and the precondition of the operation and invokes the operation over the tool. Yet, if the preconditions do not hold, the agent starts planning by updating the intention stack by pushing the preconditions as goals before the invocation of the operation $\pi_i = [\mathrm{p}|\pi_i]$. The goals force the agent to restart the overall deliberation cycle. So, the agent finds new plans and generates new intentions to act until the preconditions of the operation are believed (the preconditions hold in the agent's belief set).

6 Metalevel Reasoning

Metalevel reasoning is concerned with the event focus S_E, the plan preference S_O, and the intention prioritization S_I. Constructing those functions, hence metalevel reasoning, is domain dependent and heavily relies on the knowledge of the domain expert [16]. In this section, we propose a domain independent metalevel reasoning rules based on the assumptions introduced in Sec. 4.1.

6.1 Option Selection

The first rule handles the situations where the agent has relevant plans available both from the PL and the TB. A strategy for S_O is to select the plan originating from the PL. The reason is that the tool use is time consuming since the agent should discover and plan at run time. On the other hand, plans originating from the PL are assumed to be applicable without further deliberation in the environment as expected by the agent designer. The syntax for the rule uses $\bowtie$ to indicate that the plan is supported by the tool, $\supset$ is used as implication, and *appPlan(Goal)* is used as the combination of *unifyEvent* and *unifyContext* functions given in Sec. 5.1. Thus, *appPlan(Goal)* returns applicable plans.

Rule 1 (OS1) $\pi_1 = appPlan(\varphi) \wedge \pi_2 = appPlan(\varphi) \wedge \pi_1 \in PL \wedge \pi_2 \bowtie T \wedge T \in TB \supset INT(\pi_1)$

The Rule 1 states that if there are two *applicable plan*s for goal φ then the agent intends the one which originates from the PL.

The second rule conveys the knowledge of the environment designer to the agent. Through the second rule the *environment designer* has more control over the behavior of the agent. For example, the environment designer may want agents to use the `procUnit2` for drilling. In the presence of the utility function, agents will use it while choosing between functionally equivalent tools.

Rule 2 (OS2) if $\pi_1 = appPlan(\varphi) \wedge \pi_2 = appPlan(\varphi) \wedge \pi_1 \bowtie T_1 \wedge \pi_2 \bowtie T_2$ and $T_1 \succ T_2 \supset INT(\pi_1)$

where $\succ$ is an ordering relation of the form $T_1 \succ T_2 := f(T_1, \varphi) > f(T_2, \varphi)$ *and* $f : T \times G \rightarrow \mathbb{R}$

Rule 2 states that if there is a function f that orders the tool preference for a goal then the agent will use higher rated tool.

Finally, the third rule is a heuristic form of Rule 2 and handles the situations where no selection function f is defined by the environment designer. In those situations, S_O selects more specific tool. By the more specific tool, we mean a tool with less functionality. For example, if the agent needs to drill a hole, it prefers the `procUnit1` over the `procUnit2`. If one compares Rule 2 to Rule 3, he/she observes that the behavior of the agent differs, though, the behavior due to Rule 3 may be sub optimal.

Rule 3 (OS3) $\pi_1 = appPlan(\varphi) \wedge \pi_2 = appPlan(\varphi) \wedge \pi_3 = appPlan(\psi) \wedge \pi_1 \bowtie T_1 \wedge \pi_2 \bowtie T_2 \wedge \pi_3 \bowtie T_2 \supset INT(\pi_1)$

The Rule 3 states that if tool T_1 is a more specific tool than T_2 then the agent intends to use the tool T_1.

In future, we intend to incorporate a computational notion for affordance [22] by extending the Rule 2. For example, a tool such as a *chair* can afford for *sitting* to *get rest, stepping on* for *elevation, or to stack things* for *organization*. The evaluation function may rank various affordances to guide the agent actions, e.g. suggesting sitting.

6.2 Intention Handling

Intentions are used for balancing deliberation and action in resource bounded agents. If an agent intends to do something, it will no longer consider other options or other events until the intention is reached [23].

Although the intention is a useful mechanism for designing a resource bounded agent, commitment strategies have also been focus of attention [23, 24]. However, previous strategies consider only dropping the intention when some conditions hold [23]. In many situations, such a strategy may not be feasible because it wastes the deliberation time or even some actions cannot be reversed. For example, if a component has already been painted, the intention to drill it can not be dropped just because the `porcUnit1` is occupied. In those cases, suspending the intention is a better strategy. By suspending an intention, the agent also saves time from future deliberation. This section explores conditions for suspending and resuming intentions based on the assumptions introduced in Sec. 4.1.

The three rules with the increasing level of reactiveness to the events are named as (i) blind use, (ii) dedicated use, and (iii) lazy use. An agent that adopts the blind use strategy would not pay attention to new events until it reaches its current goal. That is, if two goals (e.g. the process of component 1 and 2) need to use the same tool, e.g. drill, then the agent suspends one of the intentions until it believes that it reaches the first goal (finish processing the component 1).

The logic language we have used to formalize the intention handling is first order multi-modal logic with modalities $\Diamond$ (eventually), $\Box$ (always), $\bigcirc$ (next), $\bigcup$ (until) defined in Rao and Georgeff [23]. The notation INT_T is used to denote the intention to use a tool with the superscript INT_T^S denotes the suspended intention and the operator $\|$ denotes concurrent intentions. An intention is said to be suspended if it is in the intention set but not pursued and two intentions

are said to be concurrent if both appear in the intention set. So the blind use can be given as follows;

Rule 4 (IS1) $(INT_T(\varphi) \parallel INT_T(\psi)) \supset (INT_T(\varphi) \parallel \bigcirc INT_T^S(\psi) \bigcup Bel(\varphi) \wedge \bigcirc INT_T(\psi))$

The Rule 4 states that if the agent intends to use a tool T for goal φ and for goal ψ then it suspends the second intention (due to A1) in the next step *until* it believes that the goal φ is reached and resumes the intention for goal ψ as soon as φ. For example, an agent in the production cell would ignore all the incoming components until it finishes the current component. The Rule 4 can be used for the situations where the component arrival is scarce and finishing a component is more important than processing more components.

The second rule is called *dedicated use* and allows the agent to suspend its current intention until the tool completes the operation (due to A3). That is, the agent starts the operation and suspends the intention that the operation belongs and handles other events. Meanwhile, the agent is still focused on the state of the tool. When the agent receives the notification from the environment regarding the operation completion (due to A2), the agent resumes its previous intention. The next rule identifies the behavior for dedicated use.

Rule 5 (IS2) $INT_T(\varphi) \supset (\bigcirc INT_T^S(\varphi) \bigcup Bel(\varphi) \wedge Bel(done(o)) \wedge INT_T(\varphi))$

The rule 5 states that the agent suspends the intention to reach φ in the next step until it believes φ is reached and operation o is finished $done(o)$ when it resumes the intention $INT_T(\varphi)$. For example, an agent in the production cell may set one of the Processing Units for drilling a component. The difference between the Rule 4 and the Rule 5 is that the agent is not idle during the operation execution. It will suspend the intention for the component and waits for other components to arrive. The dedicated use rule balances the deliberation and action but may lead to delays in processing times. For example, assume a component of `type2` comes while a component of `type1` was being painted then the agent will dedicate the `procUnit1` to the second component. Thus, drilling the first component will be delayed. The dedicated use rule is suitable for situations where intentions are not time critical and can be suspended indefinitely.

Finally the third rule, called the *lazy use*, favors the event handling to the intention completion. In fact, the agent will not resume an intention until another goal needs one of the tools that the suspended intention is holding. The formalization of the lazy use is composed of two rules for suspending and resuming the intentions.

Rule 6 (IS3) *(i)* $INT_T(\varphi) \supset (\bigcirc INT_T^S(\varphi) \bigcup Bel(\varphi))$
(ii)$(INT_T^S(\varphi) \parallel \Diamond INT_T(\psi)) \supset (\bigcirc INT_T(\varphi) \parallel INT_T^S(\psi) \bigcup Bel(\varphi) \wedge INT_T^S(\varphi))$

The Rule 6(i) states that the agent suspends the intention to reach φ as soon as it starts it. The Rule 6(ii) tells how this intention is resumed when eventually another goal ψ appears in the intention set that requires the use of the tool that the suspended intention holds. For example, the agent in the production cell may leave a component over the drilling unit until another component needs to be drilled. Thus, the agent ignores the unfinished component until it needs the tool for another component. Doing so, the agent will have more time for deliberation and event handling.

The lazy use rule is suitable for the situations where the intentions are less important then the events. Although, the rule does not guarantee that the intention will be completed, it handles the higher number of events than both the blind use and the dedicated use have to offer.

7 Related Work

Planning for BDI agents is recently considered by different researchers. Walczak et. al. [25] have described a planning approach using utility functions. They have introduced the formal planning problem based on domain object models. Meneguzzi et. al. [18] concentrate on the deliberation cycle for agents and includes a planning component, Graphplan, before the intention selection. Besides, Sardina et. al. [19] concentrate on the similarities between the hierarchical task network (HTN) [26] and BDI. Sardina et. al. also identify the formal operational semantics for their work.

All the planning problem formalizations above – including our formalization of operations– are similar to the STRIPS [27] notation. Further similarities with Walczak et. al. [25] are the use of the utility function and the partial plans as heuristics for planning. Their utility function corresponds to our option selection Rule 2. The partial plan heuristic corresponds to operating instructions of tools.

The two major differences between our work and the previous planning research are the source of the information necessary for the planning problem and the type of planning. Firstly, in our case, the information regarding the planning problem, e.g. operation, is retrieved from the environment whereas before it was considered within the agent model. In the previous approach, the agent's internal world model should be modified every time the environment changes. In contrast, the concept layer that belongs to the environment allows the extrospective agent to adopt its behavior at run time.

Secondly, planners, such as the HTN, return a complete path from the initial state to the goal state. Thus, the agent is *conservative* and does not initiate any action unless planning problem is fully solved. Such a look-a-head planning is useful when the success of the plan should be guaranteed before action commences. On the other hand, the BDI planners adopt a plan-when-needed approach where planning is only triggered when a sub-goal is encountered in the plan description. Such planning do not return complete plan hence, cannot estimate the future success. Thus, the agent is *opportunistic* and act even though it can not predict possible future failures. If the agent environment is changing faster than the planning time then plan-when-needed perform better then look-a-head planning [19]. Our planning approach for orchestration and the precondition satisfaction is in the plan-when-needed form.

In the planning sense, the work by Hübner et. al. [28] is more similar to our approach. Instead of augmenting the BDI cycle with a look-a-head planner, they use BDI programming patterns to turn the BDI planner into a declarative plan engine similar to the approach explained in Sec. 5.1. They define programming patterns for error recovery and retry condition purely based on Jason programming language and as a result the Jason BDI engine.

Apart from planning, we also like to remind the intention handling mechanism in Rao and Georgeff [23] and Cohen and Levesque [24]. As mentioned above, both works consider the conditions for dropping an intention. In our case, we have introduced the intention suspension, as we rely on the tool assumption that guarantees the operation completion event.

8 Conclusion

In this paper, we have argued that agent adaptation can benefit from (i) engineering the agent environment and (ii) the agents are designed with the capability to reason about the functionality of the artifacts as alternate means for achieving their goals.

Drawing from the activity theory [9], we have identified tools as the basic building block of such a design perspective. We have concentrated on the BDI agent framework as a particular agent architecture and elaborated on the extensions for developing extrospective agents. The extensions concentrate on: (i) querying the environment to discover tools, (ii) selecting which tool to use, (iii) planning for orchestration and operation enabling, and (iv) intention management during the tool use.

Much work needs to be done both in theory and practice. For example, a formal theory for tool use is still missing. Another direction for theoretical research is the formalization of the tool: how can agents reason about tools? An interesting question is "can an agent use one tool as a substitute for another, similar to humans use of a bowl or a cup interchangeably for carrying some water. If so how can an agent relate tools functionality?" Some initial steps towards this direction has been taken into account in our research, however, the details are not well developed yet. We believe that consulting situated theories of mind in psychology such as ecological psychology [22], distributed cognition [29], and activity theory [9] will be useful.

Research questions on the practical aspects are also numerous. Firstly, the implementation details of the extrospective agent should be given. Although, we have extended the Jason [16] interpreter, we did not discuss the details here due to the space limitation. Secondly, combining the A&A platform and the extrospective agent is needed to give a full MAS development environment. A reference architecture from our initial experiences should also be discussed. We believe that combined framework may tackle more complex problems such as use of agents for web service orchestration.

References

1. Weyns, D., Omicini, A., Odell, J.: Environment as a first-class abstraction in multi-agent systems. Autonomous Agents and Multi-Agent Systems **14**(1) (2007) 5–30
2. Rome, E., Hertzberg, J., Dorffner, G., Doherty, P.: 06231 executive summary – towards affordance-based robot control. In Rome, E., Doherty, P., Dorffner, G., Hertzberg, J., eds.: Towards Affordance-Based Robot Control, Dagstuhl, Germany (2006)
3. Omicini, A., Ricci, A., Viroli, M.: *Agens Faber*: Toward a theory of artefacts for MAS. Electronic Notes in Theoretical Computer Sciences **150**(3) (2006) 21–36
4. Ricci, A., Viroli, M., Omicini, A.: "Give agents their artifacts": The A&A approach for engineering working environments in MAS. In Durfee, E., Yokoo, M., Huhns, M., Shehory, O., eds.: 6th International Joint Conference "Autonomous Agents & Multi-Agent Systems", Honolulu, Hawai'i, USA (2007)
5. Ricci, A., Viroli, M., Omicini, A.: CArtAgO: An infrastructure for engineering computational environments in MAS. In Weyns, D., Parunak, H.V.D., Michel, F., eds.: 3rd International Workshop "Environments for Multi-Agent Systems", Hakodate, Japan (2006) 102–119
6. Bordini, R.H., Hübner, J.F.: Jason: A Java-based interpreter for an extended version of AgentSpeak. February 2007
7. Vicente, K.J.: Cognitive Work Analysis : Toward safe, productive, and healthy computer-based work. Lawrence Erlbaum (1999)
8. Rasmussen, J., Vicente, K.J.: Ecological interfaces: A technological imperative in high-tech systems? International Journal of Human-Computer Interaction **2**(2) (1990) 93–110
9. Leont'ev, A.N.: Activity, consciousness and personality. Prentice Hall, Englewood Cliffs, NJ (1978)
10. Norman, D.A.: Cognitive artifacts. In Carroll, J.M., ed.: Designing interaction: Psychology at the human-computer interface. Cambridge University Press (1991)

11. Meneguzzi, F.R., Zorzo, A.F., da Costa Móra, M.: Propositional planning in BDI agents. In: 19th Annual symposium "Applied Computing". (2004)
12. Ricci, A., Piunti, M., Acay, L.D., Bordini, R., Hübner, J., Dastani, M.: Integrating artifact-based environments with heterogeneous agent-programming platforms. In: To be published. (2008)
13. Acay, D.L., Pasquier, P., Sonenberg, L.: Extrospection: Agents reasoning about the environment. In: 3rd International Conference "Intelligent Environments". (2007)
14. Knublauch, H., Horridge, M., Musen, M., Rector, A., Stevens, R., Drummond, N., Lord, P., Noy, N.F., Seidenberg, J., Wang, H.: The Protégé OWL experience. In: 4th International Conference "Semantic Web", Galway, Ireland (2005)
15. Wooldridge, M.: An Introduction to Multiagent Systems. John Wiley & Sons (2002)
16. Bordini, R.H., Hübner, J.F.: BDI agent programming in AgentSpeak using Jason. In Toni, F., Torroni, P., eds.: 6th International Workshop "Computational Logic in Multi-Agent Systems". (2006) 143–164
17. Bordini, R.H., Bazzan, A.L.C., Jannone, R.O., Basso, D.M., Vicari, R.M., Lesser, V.R.: AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In Castelfranchi, C., Johnson, W.L., eds.: 1st International Joint Conference " Autonomous Agents and Multi-Agent Systems". (2002)
18. Meneguzzi, F.R., Zorzo, A.F., Mòra, M.D.C., Luck, M.: Incorporating planning into BDI agents. Scalable computing: Practice andexperience **8** (2007)
19. Sardina, S., de Silva, L., Padgham, L.: Hierarchical planning in BDI agent programming languages: A formal approach. In: 5th International Joint Conference "Autonomous Agents and Multiagent Systems", New York, NY, USA (2006) 1001–1008
20. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In van Hoe, R., ed.: 7th European Workshop "Modelling Autonomous Agents in a Multi-Agent World", Eindhoven, The Netherlands (1996)
21. Ancona, D., Mascardi, V., Hübner, J.F., Bordini, R.H.: Coo-AgentSpeak: Cooperation in AgentSpeak through plan exchange. In: 3rd International Joint Conference "Autonomous Agents and Multiagent Systems". (2004) 696–705
22. Gibson, J.J.: The ecological approach to visual perception. Houghton Mifflin (1979)
23. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In: 2nd International Conference "Principles of Knowledge Representation and Reasoning". (1991)
24. Cohen, P.R., Levesque, H.J.: Intention is choice with commitment. Artificial Intelligence **42**(2–3) (1990) 213–261
25. Walczak, A., Braubach, L., Pokahr, A., Lamersdorf, W.: Augmenting BDI agents with deliberative planning techniques. In: Programming Multi-Agent Systems. Springer Berlin / Heidelberg (2007) 113–127
26. Erol, K., Hendler, J., Nau, D.S.: Semantics for hierarchical task-network planning. Technical report, Univ. of Maryland Institute for Advanced Computer Studies Report No. UMIACS-TR-94-31, College Park, MD, USA (1994)
27. Nilsson, N.J., Fikes, R.E.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2**(3-4) (1971) 189–208
28. Hübner, J.F., Bordini, R.H., Wooldridge, M.: Programming declarative goals using plan patterns. In: Declarative Agent Languages and Technologies IV, Springer (2006) 123–140
29. Hutchins, E.: How a cockpit remembers its speeds. Cognitive Science **19** (1995) 265–288

University Library

A gateway to Melbourne's research publications

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Acay, Daghan; Sonenberg, Liz; Ricci, Alessandro; Pasquier, Philippe

Title:
How situated is your agent? A cognitive perspective

Date:
2008

Citation:
Acay, Daghan and Sonenberg, Liz and Ricci, Alessandro and Pasquier, Philippe (2008) How situated is your agent? A cognitive perspective, in Proceedings, Programming multi-agent systems (PROMAS) Workshop at AAMAS 2008, Lisbon.

Publication Status:
Inpress

Persistent Link:
http://hdl.handle.net/11343/34897

File Description:
How situated is your agent? A cognitive perspective

University Library

A gateway to Melbourne's research publications

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
ACAY, LD; SONENBERG, E; Ricci; Pasquier

Title:
How situated is your agent? a cognitive perspective

Date:
2009

Citation:
ACAY, L. D., SONENBERG, E., Ricci & Pasquier (2009). How situated is your agent? a cognitive perspective. International Workshop on Programming Multi-Agent Systems, pp.136-151. Springer Verlag.

Persistent Link:
http://hdl.handle.net/11343/31891

File Description:
Accepted version