

How situated is your agent? A cognitive perspective

Daghan L. Acay¹, Liz Sonenberg¹, Alessandro Ricci², and Philippe Pasquier³

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. Artifacts 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 the 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 design time.

As the scale of Multi-Agent Systems (MASs) increases, the above approach to agent design leads to two problems. First, when the agent and the environment it acts in are developed by different designers, e.g. in the context of web services, the designer of an agent can not capture all the possible service combinations (i.e. environment). Second, 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 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 in some manner. Ricci et. al. [2], in their agent and artifact framework (A&A), suggested ‘artifact’ as a possible abstraction to model non-agent entities in the environment. Although they suggested an ‘artifact manual’ for autonomous discovery and use of artifacts at run-time, they considered it as future work.

Previously, a possible way of representing artifact manuals using description logic has been explored and a formal language for writing artifact manuals, called OWL-T [3] was introduced. In the context of this paper, we will call the combination of an artifact and the associated manual a ‘tool¹.’ The aim in this paper is to explore the relation between agent reasoning and tool specification so that agents can modify their behavior at run time, based on tool availability. In our case, the behavior modification is realized by agents flexibly substituting tools as means for their goals.

The ‘cognitive situatedness’ mentioned in the title reflects our idea that agents should adapt to the environment by discovering and using tools as alternative means at run-time. Real-time discovery, selection, and use of tools is referred to as *extrospection*, a term that we have coined to emphasize the externally influenced nature of reasoning. The term *extrospective agent* refers to 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 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? 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.

Expected benefits of using tools for MAS development include: (i) agents can complete the design by discovering and opportunistically using tools at run time (in Sec. 5), (ii) the reuse of components will be enhanced (implicitly argued throughout the paper), and (iii) domain independent meta-level reasoning can be built into the agents (in Sec. 6).

¹ One reason is that the term *artifact* has a computational emphasis whereas, *tool* has emphasize on reasoning

2 Background: Philosophical Underpinnings

The A&A framework introduces artifacts as first class entities along with agents for developing a MAS. Similar to the A&A framework, our work on extrospective agents is inspired by the psychological theory called Activity Theory (AT) [4]. 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.

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 [4]. 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 availability of tools influences the agent's behavior through its physical characteristics, e.g. body posture, approaching angle. Moreover, tool availability can modify the choice of action after the knowledge of the tool use is acquired, e.g. in the existence of a table, putting a hot cup on the table instead of dropping the cup.

In that sense, our work is complementary to the A&A framework where a computational model for artifacts has been introduced, as 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, the agent designer does not need anticipate possible actions in the environment. Thus, the agent 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. [5] where a variety of component types with different production demands are produced. 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 such as failure, removal, addition, or upgrade. 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 is considered here. Components that need to be

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 operations that should be executed on the component.

Meneguzzi et. al. [5] 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 moving the components over the tools.

For future reference we give following details. 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.

4 Extrospection Framework

The two important aspects of the extrospection framework are the agents and the tools in the environment. In addition, we introduce a third layer where the functionality of tools is symbolically represented as ‘artifact manuals’ ???. Next we will detail each layer.

4.1 The Artifact Layer

We adopt the A&A meta-model [2], 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 function², 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. Operation steps are guarded by asserting *preconditions* for execution. In order to avoid interferences, the usage interface is disabled during the (atomic) execution

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

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.” 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.

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 require both the `procUnit3` and the `procUnit4`. Thus, the use of tools requires two sorts of scheduling/planning tasks.

The first type of planning is employed for realizing the precondition/guards of the operations of a single tool. 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 second is necessary for orchestrating the use of multiple tools within a single intention. For example, the crane agent is responsible for planning to move components from one machine to another using its *moving* capability.

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 language OWL-T [3] is used for this purpose. T stands for (T)ool and OWL for the variant of description logic, Web Ontology Language (OWL) [6].

Artifact manual written by OWL-T 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 supplied by the agent designer). 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. [3]. 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`, can associate `procesUnit1` as a possible means to process the component.

Secondly, the OWL-T captures the *operations* and the associated *preconditions*. For example, an operation of `procesUnit1`, e.g. `startDrill` with the associated precondition `drillSize(X)` is captured using OWL-T. OWL-T also identifies the link between the precondition of an operation and the beliefs of

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

such as the belief base, the events, the plan library, and the intention we refer the reader to [7]. 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. Query mechanism can be thought as a goal directed perception since, the agent queries the environment based on its goals. If there is a match between agent’s goal and the tool in the environment, *special percepts*⁴ are sent to the agent. Different query strategies are discussed in Sec. 5.1.

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 populated proactively at the run time by special percepts 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 [7] 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. Yet, 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 [5], 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 for pruning the search space for planning using the operating instructions. The related work incorporating planning to the BDI agents is considered 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 interpreter⁵ 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,
- 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

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

⁵ This section relies on the AgentSpeak(L) terminology given in Rao [9].

The extrospective agent program starts with the initialization of the the agent’s goals and beliefs. Beginning of every reasoning cycle starts updating the belief set, $buf(B,\rho)$, the event set, $buf(\rho)$, similar to AgentSpeak agents [7]. In addition, the tool base is the update by processing the set of special percepts⁶ received from the environment, $buf(TB,\rho)$.

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^7 represents the ends that the agent wants to achieve.

Unify event function (UEF) $unifyEvent(S_e(G),PL,TB)$ takes the current goal set, the TB, and the PL and matches the available means with the goals (i.e. ends) of the agent. The UEF returns the selected 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 ($\neg Bel(noTool(G))$). 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.

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 [7]. Here, we are interested in the achieve goals.

```

B ← B0
G ← G0
WHILE true
  WHILE not empty(ρset)%get percepts ρset from the environment
    %get next percept ρ from ρset
    B ← buf(B,ρ)
    E ← buf(ρ)
    T ← buf(TB,ρ)
  END WHILE
  G ← options(G,E)
  Pr ← unifyEvent(SE(G),PL,TB) %unifies plan,goal,tool
  IF Pr = ∅ THEN
    IF not ¬Bel(noTool(G))
      query(G)
    ELSE
      dropGoal(G)
    END IF
  END IF
  CONTINUE
END IF
%At this point we have relevant plan(s)
π ← unifyContext(Pr,B) %unifies plan, belief
IF empty(π)
  dropGoal(G)
  CONTINUE
ELSE
  %At this point we have applicable plan(s)
  πim = SO(π)
END IF
I = pushIntention(G,πim)
πi = SI(I)
WHILE ¬ endOfPlan(πi)
  IF πi ∈ PL
    α = head(πi)
    IF action(α)
      execute(α)
    ELSE IF goal(α)
      updateEvents(α)
    END IF
  ELSE IF πi ∈ TB
    α = head(πi)
    WHILE not empty(preList(α))
      p = next(preList(α))
      IF checkPre(p,B)
        unify(p,B,πi)
      ELSE
        πi = [p|πi] % update intention
      END IF
    END WHILE
    α = head(πi)
    IF action(α) % check the updated intention
      execute(α)
    ELSE IF goal(α)
      updateEvents(α)
    END IF
  END IF
  πi ← tail(πi)
  πi = SI(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 $pushIntention(G, \pi_{im})$ function.

Since the execution of agent and the artifact is asynchronous, the intention selection function $S_I(I)$ is responsible for suspending 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. The plans from the TB can be executed, if the preconditions for each operation unify with the agents beliefs. Otherwise, the agent should plan to satisfy before executing it (see 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 is checked once during *unifyContext*. Thus, agent consumes internal plans blindly whereas executes the operations of tool cautiously [10].

The planning for precondition is achieved by updating the intention stack by pushing the preconditions as goals before the invocation of the operation $\pi_i = [p|\pi_i]$. 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 [7]. 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 plans originating from the PL are assumed to be applicable without further deliberation in the environment since they are anticipated by the agent designer. For the rest, the agent should do the discovery and planing at run time which is more time consuming. The syntax for the rule uses \bowtie to indicate that the plan is supported by the tool, \supset is used as implication, and $appPlan(Goal)$ 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 plans* for goal φ then the agent intends the one which originates from the PL.

The second rule is suggested to capture the preferences of environment designer for the tool use. For example, the environment designer may want agents to use the `procUnit2` for drilling. Such preference is conveyed by the utility function given by agent designer and used by the agents will 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 utility 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 there are two functionally equivalent tools and no selection function. 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 [11] 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. The intention commitment strategies have been focus of attention [10, 12]. However, previous strategies consider only dropping the intention when some conditions hold [10]. In many situations, such a strategy may not be feasible because it wastes the time used for deliberation. In the worst case, some actions cannot be reversed. Thus dropping an intention has adverse side effects. In some cases, suspending an 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 reactivity 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 need to use the same tool, e.g. drill, then the agent suspends one of the goals until it believes that it reaches the first goal.

The logic language we have used to formalize the intention handling is first order multi-modal logic with modalities \diamond (eventually), \square (always), \bigcirc (next), \bigcup (until) defined in Rao and Georgeff [10]. 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 \parallel 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) $INT_T(\varphi) \supset (\bigcirc INT_T^S(\varphi) \cup Bel(\varphi))$
(ii) $(INT_T^S(\varphi) \parallel \diamond INT_T(\psi)) \supset (\bigcirc INT_T(\varphi) \parallel INT_T^S(\psi) \cup Bel(\varphi) \wedge INT_T^S(\varphi))$

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 than the events.

7 Related Work

Planning for BDI agents has recently been considered by several researchers. Walczak et. al. [13] have described a planning approach using utility functions. They have introduced the formal planning problem based on domain object models. Meneguzzi et. al. [5] concentrate on the deliberation cycle for agents and include a planning component, Graphplan, before intention selection. Sardina et. al. [14] concentrate on the similarities between hierarchical task network (HTN) planning [15] and the BDI approach, and also identify a formal operational semantics for their work.

All the planning problem formalizations above – including our formalization of operations – are similar to the STRIPS [16] notation. Further similarities with Walczak et. al. [13] are the use of a utility function and partial plans as heuristics for planning. Their utility function corresponds to our option selection Rule 2. Their partial plan heuristic corresponds to our operating instructions of tools.

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

Second, planners such as HTN planners return a complete plan from the initial state to the goal state. Thus, the agent is *conservative* and does not initiate any action unless the planning problem is fully solved. Such look-a-head planning is useful when the success of the plan should be guaranteed before action commences. On the other hand, 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 does not return complete plans, hence cannot estimate the future success. Thus, the agent is *opportunistic* and acts even though it can not predict possible future failures. If the agent environment is changing faster than the planning time then plan-when-needed will perform better than look-a-head planning [14]. Our planning approach is more conservative than plan-when-needed since it checks the preconditions of an operation before attempting it. Yet, our work is more optimistic than HTN planners, since it does not wait until a full plan is found but only assures that the current step can be executed.

In the planning sense, the work by Hübner et. al. [17] 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 the Jason programming language.

Apart from planning, we also like to remind the reader of the intention handling mechanism in Rao and Georgeff [10] and Cohen and Levesque [12]. As mentioned above, both works consider conditions for dropping an intention. In our case, we have introduced an intention suspension mechanism, 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) designing agents with the capability to reason about the functionality of artifacts as alternate means for achieving their goals.

Drawing from Activity Theory [4], we have identified tools as the basic building blocks of such a design perspective. We have concentrated on the BDI agent framework as a particular agent architecture and elaborated some extensions suited to 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 tool use.

Possible future work includes: a formal theory for tool use; the formalization of the tool concept; and how agents can reason about tools. Selecting a tool is one aspect that we have considered in this paper. A question we have not yet addressed is whether an agent can use one tool as a substitute for another. We believe that consulting situated theories of mind in psychology such as ecological psychology, distributed cognition, and activity theory will be useful in developing our ideas further.

Other future work is to consider practical extensions. It will be useful to combine the A&A platform with an extrospective agent architecture to provide a full MAS development environment. The new framework should then be evaluated in more complex domains, such as web service management.

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. 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
3. Acay, D.L., Pasquier, P., Sonenberg, L.: Extrospection: Agents reasoning about the environment. In: 3rd International Conference “Intelligent Environments”. (2007)
4. Leont’ev, A.N.: *Activity, consciousness and personality*. Prentice Hall, Englewood Cliffs, NJ (1978)
5. Meneguzzi, F.R., Zorzo, A.F., Mòra, M.D.C., Luck, M.: Incorporating planning into BDI agents. *Scalable computing: Practice and experience* **8** (2007)
6. 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)
7. 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
8. Wooldridge, M.: *An Introduction to Multiagent Systems*. John Wiley & Sons (2002)
9. 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)
10. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In: 2nd International Conference “Principles of Knowledge Representation and Reasoning”. (1991)
11. Gibson, J.J.: *The ecological approach to visual perception*. Houghton Mifflin (1979)
12. Cohen, P.R., Levesque, H.J.: Intention is choice with commitment. *Artificial Intelligence* **42**(2–3) (1990) 213–261
13. 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
14. 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
15. 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)
16. 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
17. 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