Knowledge–Based Situated Agents among Us
A Preliminary Report

Renwei Li and Luís Moniz Pereira

Department of Computer Science, Universidade Nova de Lisboa
2825 Monte da Caparica, Portugal
{renwei | lmp } @di.fct.unl.pt

Abstract. A situated agent is a hardware or software-based computer system that is embedded in a real-time world or environment, is ascribed with some mental states, and may enjoy a disconcerting variety of properties such as proactivity, reactivity, etc. In this paper we present the design of a class of knowledge-based situated agents and an agent specification language, called SICLE . After the SICLE interpreter receives a SICLE specification of an agent's initial knowledge base, atomic actions (their conditional effects and executability conditions), initial intentions and some predefined optimal plans, the agent for this specification is produced and become active. The agent will always try to realize all its intentions at its best and may produce new intentions in the real-time. The SICLE interpreter employs a non-monotonic reasoning system to model the real-world and record the past history of the environment. For those situations which may frequently arise, the user can provide optimal plans to achieve timely response. The SICLE language and its associated architecture and interpreter was designed based on the recent non-monotonic reasoning systems and action theory. A simple simulation of human commuter between two cities is used to present our approach.

1 Introduction

A situated agent is a hardware or software-based computer system that is embedded in a real-time world or environment, is ascribed with some mental states, and may enjoy a disconcerting variety of properties such as proactivity, reactivity, rationality, etc. In addition to its computational and sensory hardware, the architecture of a situated agent provides a principled way to organize the agent and determines how it will behave by imposing control strategies and constraints on the way it senses, reasons, and acts in order to realize its intentions (goals).

In the literature there are two basic extremes in the spectrum of architectures of situated agents. At one end, a deliberative planner is used to generate an optimal course of actions to achieve the agent's goals according to its current beliefs on the external world and its capabilities. The deliberative planner can be implemented in many ways including Fikes-Nilsson's STRIPS system and its variants, Sacerdoti's NOAH [28], Tate's NONLIN [38], just to name a few. Although deliberative systems have a very powerful reasoning techniques and generate optimal plans, they often fail to guarantee a timely response in the real-time world. On the other hand, the real world may be changed as a consequence of other agents' activities. What is worse is that it needs a priori model of the real world which is often very difficult, if not impossible, to have. At the other extreme, the behaviour of a situated agent is controlled by a collection of reaction rules of the form condition-action, for example, situated automata architecture [15], the Pengi system [4], the Subsumption system [6], teleo-reactive systems [27], just to name a few. In these purely reactive systems, all reaction rules have to be pre-wired. Whereas reactive systems can provide quick responses to the external world, they also fail to guarantee to respond to all possible situations due to the impossibility of pre-coding reaction rules for all possible situations. On the other hand, even if the designer can pre-code all reaction rules, it is often time-consuming and too complex.

The architecture to be presented in this paper will be between the above two extremes. This kind of compromise architecture, sometimes called hybrid or mixed architecture [3, 30], has a set of pre-compiled reaction rules and a deliberative planner to deal with unpredicted situations in the reaction rules. In contradistinction to existing hybrid architectures, our situated agent will always act according to its situated knowledge and intentions. At any time point, only intentions with higher priorities are considered, due to boundness of resources. The agent can dynamically re-organize and compute the priorities of intentions. The situated knowledge contains what is stored in its knowledge base, plan library, action theory, agenda, sensored data, etc. The knowledge base is maintained and managed by using a non-monotonic reasoning system. With the lapsing of time, the knowledge base evolves non-monotonically both in time and in sensored or learned knowledge. In order to react timely to most frequently arising situations, some optimal plans are pre-coded into the plan library. In order to realize an intention, the plans stored in the plan library will be taken into account first. If there is no such a plan, the agent may generate a plan on the spot. When a plan from the plan library fails, the agent can also generate a substitute plan. For different application domains, it can be made purely reactive, or purely deliberative. For example, a physical device trouble-shooter can be implemented by a proactive agent situated around the physical device; a mobile robot in a completely known environment can be implemented as a purely reactive situated agent; and an electronic taxi driver, on the other hand, can be implemented as a mixture of proactive and reactive situated agents. Our situated agents are based on well-established non-monotonic reasoning techniques and action theories.

The rest of this paper is organized as follows. We start by describing the overall architecture, then proceed to detail its components and the agent specification language SICLE , compare with related work, and finally present an example.

2 An Architecture of Situated Agents

The overall architecture of our situated agents,called SICLE agents[1], in the real world is depicted in Figure 1.

[1] The acronym stands for Situations, Intentions, Choices, Language, Executability. A situation denotes the content of a knowledge/belief base, sensored data, etc. An intention consists of the intended goals. A choice is a decision on actions. Informally, our situated agents will work by repeating the following tasks: From the current situation and intentions the agent will make a choice to determine which action will be taken at the next step. Thus the behaviour and the operational semantics of our agents can be described by a Situation-Intention-Choice.

After a SICLE program (to be presented later) is received by the interpreter, a SICLE agent is produced and will always try to realize its intentions at its best. In order to do so, the agent will dynamically measure the priorities among intentions and their associated procedures, and select the most urgent intention and its associated procedure to invoke its effector to perform atomic actions and monitor its result. Intentions basically correspond to temporal-positional formulas which are intended to be made true by performing some actions. For any intention, the agent will always check if there is a pre-defined plan for it (we assume that the pre-defined plans are better than occasionally generated plans). If there is a plan, the agent takes it, othewise generates one, and then put it in the agenda. The agent will also maintain a model of the world represented by a non-monotonic reasoning system and revise it in real time according to its sensory input.

The semantics of our agents is defined to be a function $Choice$ from situations and intentions to sets of actions:

$$Choice : SITUATIONS \times 2^{INTENTIONS} \rightarrow 2^{ACTIONS}$$

where $SITUATIONS$ is a type representing all situations, $INTENTIONS$ a type representing all intentions, and $ACTIONS$ a type representing all actions. Given a situation S and a set of intentions I, $Choice(S, I)$ is a set of actions which are meant to be performed in parallel.

A situation is in turn defined to be a disjoint union of the current knowledge base, plans in the plan library, action theory, agenda, and sensored data.

An intention is essentially a temporal-positional formula which can be reified by introducing a meta-predicate $holds(F, T)$ for fluent F and time T.

An action is either a physical atomic action that corresponds to a function of the effector, or an internal action corresponding to query, knowledge expansion, knowledge assimilation, abduction, diagnosis, etc.

Technically, the SICLE agent and its associated language SICLE are based on a non-monotonic reasoning system and an action theory. In the foregoing sections we will discuss its main features and the SICLE language. We will leave out its tedious syntactical details and focus on its major features, in view that the reader can reconstruct the work in this paper.

3 Fluents and Time

In SICLE agents we classify relations among first-order individuals into two categories: time-independent relations and time-varying relations. Time-varying relations will be represented by fluents, and time-independent relations will be represented by predicates. A fluent can be reified by a special predicate $holds(F, T)$ for fluent F and time T. For example, "two plus three makes five" is a time-independent relation, while "the red block is on top of the black block" is a time-varying relation since its truth value depends on time.

In this paper we will use the common-sense calendar-clock style time represented by tuples. For example, "1996/8/12" is a valid calendar-clock style time to represent August 12, 1996. Ladkin [17] showed that the calendar-clock style time with arbitrary

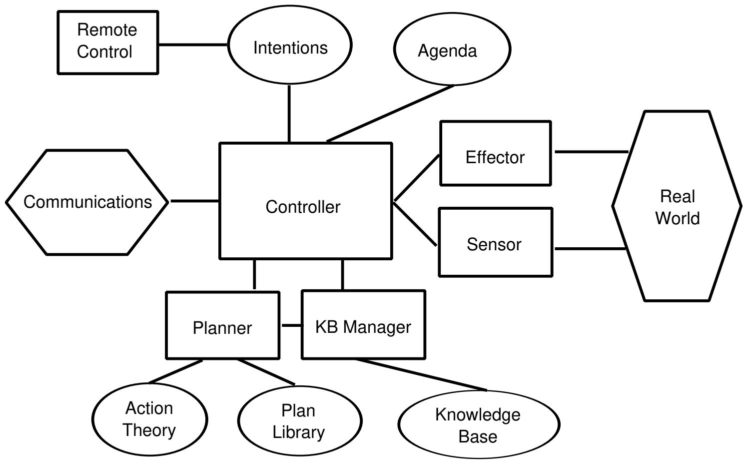

Fig. 1. SICLE Agent Architecture

precision can be isomorphically described by a time structure based on rational numbers. In the SICLE agent we choose to use the common-sense calendar-clock style time with *finite* precision, normally precise to the second for the following reasons: (i) technical reason: the built-in clocks in computers are often calendar-clock style time precise to the second (ii) pragmatical reason: such time representation is enough for many practical domains. Mathematically, the calendar-clock style time can be represented by a tuple $\langle year, month, day, hour, minute, second\rangle$, and an operator to make arbitrary time intervals. In what follows we will use a very relaxed denotation for time.

4 Knowledge Base

The knowledge base in the SICLE agent architecture is a partially-ordered set of flat knowledge bases, and each flat knowledge base is in turn an extended logic program. A flat knowledge base is a collection of the agent's knowledge and beliefs about the external world, its own capabilities, etc.

In general, a flat knowledge base is a collection of extended logic programming rules of the form:

$$L \leftarrow L_1, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n.\ (0 \leq m \leq n)$$

where the symbol *not* denotes the negation-as-failure operator, each of L and L_i is either an atom or an atom preceded by explicit negation $\neg$. Atoms and atoms preceded by explicit negation $\neg$ are called objective literals. The set of all ground objective literals of an extended logic program P is called the extended Herbrand base of P and denoted by $\mathcal{H}(P)$. By $\mathit{not}\ \{a_1, \ldots, a_n, \ldots\}$ we mean $\{\mathit{not}\ a_1, \ldots, \mathit{not}\ a_n, \ldots\}$. An interpretation I of an extended logic program P is denoted by $T \cup \mathit{not}\ F$, where T and F are two subsets of $\mathcal{H}(P)$. Objective literals in T are said to be *true* in I, objective literals in F *false by default* in I, and objective literals in $\mathcal{H}(P) - I$ *undefined* in I.

If the explicit negation $\neg$ does not appear in an extended logic program P, then it collapses to a normal logic program; if the negation-as-operator does not appear, it collapses to a definite logic program; if the body of every rule is empty, it collapses to a relational database. For many interesting domains, relational databases are too weak and extended logic programs are not necessary, but locally stratified logic programs [32] are sufficient[2]. Anyway extended logic programs are sufficient enough to subsume many other knowledge bases.

The semantics of an extended logic program P is defined to be the well-founded semantics of P incorporated with a coherence principle [31]. The coherence principle relates two forms of negation: For any interpretation $I = T \cup \mathit{not}\, F$ and any objective literal L, if $\neg L \in T$ then $L \in F$. We start with the definition of Gelfond–Lifschitz module operator $\frac{P}{I}$.

Definition 4.1 (Modulo Operator $\frac{P}{I}$) Let P be an extended logic program, I an interpretation. Nondeterministically perform the following three operations: (i) Remove from P all rules containing a default literal $L = \mathit{not}\, A$ such that $A \in I$; (ii) Remove from P all rules containing an objective literal L in the body such that $\neg L \in I$; (iii) Remove from all remaining rules of P their default literals $L = \mathit{not}\, A$ such that $\mathit{not}\, A \in I$. $\frac{P}{I}$, P modulo I, is the program obtained from the above result by replacing all the remaining default literals with a new proposition symbol $\mathbf{u}$.

Obviously, the resulting program $\frac{P}{I}$ is non–negative, i.e. no default literals appear in it, thus there is a unique Fitting-least 3-valued model, $least(\frac{P}{I})$, which is a generalization of the van Emden-Kowalski least model operator for definite logic programs. Note that $\frac{P}{I}$ is not unique because of the nondeterministic operations.

Definition 4.2 We define $least(P)$, where P is a non–negative program, as the set of literals $T \cup \mathit{not}\, F$ obtained as follows: (i) Let P' be the non–negative program obtained by replacing in P every negative objective literal $\neg L$ with a new atomic symbol, say $'\neg\_L'$; (ii) Let $T' \cup \mathit{not}\, F'$ be the least 3–valued model of P'; (iii) $T \cup \mathit{not}\, F$ is obtained from $T' \cup \mathit{not}\, F'$ by reversing the replacements above.

Although $least(P)$ uniquely exists for every non–negative program P, it might not satisfy the coherence principle, and hence it isn't always an interpretation.

Definition 4.3 (Coh operator) Let $QI = QT \cup \mathit{not}\, QF$ be a set of literals. We define $Coh(QI)$ to be the interpretation $T \cup \mathit{not}\, F$ such that $T = QT$ and $F = QF \cup \{\neg L \mid L \in T\}$.

Definition 4.4 (Φ operator) Let P be an extended logic program, I an interpretation. We define $\Phi_P(I) = \bigcup Coh(least(P_i)),$ where P_i is some $\frac{P}{I}$.

[2] Note that both stratified logic programs and acyclic logic programs are instances of locally-stratified logic programs. What is important is that the stable models and well-founded models of locally stratified logic programs coincide.

It can be shown that the operator Φ_P for any program P is monotonic, and hence there is a least fixpoint $lfp(\Phi_P)$.

Proposition 4.5 *There is a least fixpoint of Φ_P for any extended logic program.*

Then, the well-founded semantics of P with explicit negation, denoted by WFSX(P), is defined to be the least fixpoint of Φ_P. For lack of space we will not go into deeper discussions on WFSX semantics. Many interesting results are summarized in the monograph [2]. In particular, the following result from [2] acts as the theoretical basis for the knowledge base manager.

Proposition 4.6 Let L be any given objective literal and P any extended logic program. There is a decision procedure taking L and P as inputs such that (i) if $L \in WFSX(P)$ then the decision procedure will always terminate in a finite time and returns the affirmative result, and (ii) if the decision procedure terminates and returns the affirmative result, then $L \in WFSX(P)$.

5 Actions and Changes

The SICLE agent may include an action theory which describes its effectors (and other agents' effectors in multi-agent systems) and enables it to predict the future and explain the past. One of the major difficulties with reasoning about actions is the frame problem. To avoid the frame problem, the STRIPS systems provide an effective way by using three lists (add list, delete list, and precondition list), and are thus accepted by many deliberative planners and agent architectures such as Prodigy [7], but it is interesting to note that the naive use of the STRIPS representation often leads to an unsound action theory [23]. On the other hand, the early attempt at the frame problem using circumscription or normal default logic could lead to intuitively undesirable results [12]. Solutions to the the Yale Shooting Problem stimulated the development of some temporal reasoning techniques such as chronological minimization [16, 24, 37], causal minimization [13, 25], and Baker's circumscriptive theory [5].

Our SICLE agent uses an action theory which is essentially an extension to STRIPS with conditional effects and expected time durations. In order to guarantee the soundness, we adopt some features of the action description language $\mathcal{A}$ [10]. The action theory in SICLE agents is described by action declarations in the SICLE language. In SICLE an action is declared as follows:

action $A = \{$
  when $P_1, P_2, \ldots, P_k$;
  time (min, max);
  causes $Q_{11}, \ldots, Q_{1m}$ **if** $S_{11}, \ldots, S_{1n}$;
  $\ldots$
  causes $Q_{w1}, \ldots, Q_{wm}$ **if** $S_{w1}, \ldots, S_{wn}$
$\}$

where A is an action identifier, min and max are expected minimal and maximal time durations and can be replaced with a single parameter, say d, which abbreviates (d, d).

Each of P_h, Q_{ij} and S_{ik} is an objective fluent expression (atomic fluent names or fluent names with parameters possibly preceded with explicit negation $\neg$.). The **causes** list can be decomposed into add–list and delete–list like in STRIPS by virtue of the explicit negation. All of the **when** list, **time** clause, and **if** list are optional, and it is required that there be at least one **causes** list. The above SICLE declaration means that the action A can be applied at any time when the precondition list (**when** list) is satisfied, and it is expected to take min time to max time (or d time) to finish it, and its effects are specified by the set of conditional **causes** lists.

A set of SICLE action declarations is called a SICLE action theory. The semantics of a SICLE action theory is defined by using states and transitions, as in [10].

A *state* σ is a set of fluent names. Given a fluent name F and a state σ, we say that F holds in σ if $F \in \sigma$, F does not hold in σ otherwise. Given a fluent name F, we also say that $\neg F$ holds in σ if F does not hold in σ.

We say that the execution of an action A in a state σ *initiates* a fluent expression F if the following conditions are satisfied: Let the SICLE action declaration for A be

action A = {
 when P; **time** (min, max); **causes** Q_1**if** $S_1; \ldots;$ **causes** Q_w**if** S_m
 }

then

- The preconditions P hold in σ;
- there is an **causes–if** list **causes** Q_i **if** S_i such that $F \in Q_i$ and S_i hold in σ.

From this definition we introduce two set-ranged functions as follows:

$$Initiate(A, \sigma) = \{F \ : \ A \text{ initiates } F \text{ in } \sigma\}$$
$$Terminate(A, \sigma) = \{F \ : \ A \text{ initiates } \neg F \text{ in } \sigma\}$$

A transition function is a mapping from the set of pairs (A, σ), where A is an action expression and σ is a state, into the set of states.

Definition 5.7 For a SICLE action declaration for an action A and any world state σ, the state transition function $\tau(A, \sigma)$ is defined as follows:

$$\tau(A, \sigma) = ((\sigma \cup Initiate(A, \sigma)) \setminus Terminate(A, \sigma))$$

When concurrent actions are allowed, the above definition will be accordingly modified as in [21].

An action theory defined above can be reified by a translation from it to abductive logic programs in an analogous way to that of [21], and thus can be used to generate plans, explaining past or predicting the future. More discussions on action theories defined above are omitted.

6 Intentions

An agent's intention is what the agent intends to realize, and thus motivates its actions. In our current implementation, an agent's intention can be described in SICLE by using the intention declaration statement:

intentions = {
intention–declaration$_1$;
$\ldots$;
intention–declaration$_m$
}

Let S be a set of objective literals or fluent expressions. Then, an intention–declaration is any of the following forms:

S **always** ; S **eventually** ;
S **always from** T_1 **to** T_2 ; S **eventually from** T_1 **to** T_2 ;
S **always for** D ; S **eventually for** D ;
S **at** T

where T, T_1 and T_2 are time points, and D is a time duration. It can be seen that the above intention declarations are actually a mixed temporal and positional logic language. The keyword **always** is optional.

The intentions of the form S **always** and S **eventually** have the same meanings as the temporal formulas $\Box S$ and $\Diamond S$. The intention S **at** T has the same meaning as the positional formula $R(T)S$ of [36]. The intention S **always from** T_1 **to** T_2 can be viewed as a conjunction of the intentions S **at** T for all time points T between T_1 and T_2, and roughly corresponds to the positional formula $[\![(T_1, T_2)]\!]S$ of [19]. The intention S **eventually from** T_1 **to** T_2 can be viewed as a disjunction of the intentions S **at** T for all time points T between T_1 and T_2, and roughly corresponds to $\langle\!\langle(T_1, T_2)\rangle\!\rangle S$ of [19]. The intentions S **always for** D and S **eventually for** D are mixed temporal and positional formulas. Suppose *now* it is T. Then, they are transformed into S **always from** T **to** $(T + D)$ and S **eventually from** T **to** $(T + D)$, respectively. The above temporal-positional formulas can be reified in first-order logic by using a predicate $holds(F, T)$.

For example, suppose that an agent intends to work from 10 am to 1 pm and from 2 pm to 6 pm. Then, its intentions can be declared as follows:

intentions = {
$in(work)$ **always from** $10am$ **to** $1pm$;
$in(work)$ **always from** $2pm$ **to** $6pm$
}

If an agent has intentions, the agent will always try to realize them at its best. To each intention a plan can be associated. In our implementation, the agent can re-organize and measure the priority of intentions and plans. Currently we employ the simplest strategy to organize intentions and their plans.

Definition 6.8 *Let Act and $Plan\_Library$ denote the underlying action theory and plan library. Let I and J be two intentions, and P_i and P_j two plans for I and J. We say that (I, P_i) has a higher priority than (J, P_j) iff the following conditions are satisfied: Let the current state be σ_0, then*

- *There is no solution to the planning problem $(Act \cup Plan\_Library,\ \tau(P_j,\ \sigma_0),\ Reify(I))$.*
- *there is a plan for the planning problem $(Act \cup Plan\_Library,\ \tau(P_i,\ \sigma_0),\ Reify(J))$.*

where $Reify(I)$ and $Reify(J)$ denote the reification of intentions in the planner.

Note that there may be many possible plans for an intention. But only one of them is picked up and put in the agenda. Thus, priority among intentions can be measured according to the picked plans.

7 Plans and Agenda

A plan is a specification of sequenced actions to be performed for realizing an intention, and an agenda is a run-time work space (data structure) to store intentions and their related plans, and control pointers which indicate the next action in the plans. A plan is either predefined and stored in the plan library or can be generated by the planner in real time.

The plans stored in the plan library are described in SICLE by the following plan declaration statements:

$$
\begin{aligned}
&\textbf{plan } P = \{\\
&\qquad \textbf{for } P_1, P_2, \ldots, P_k;\\
&\qquad \textbf{when } C_1, C_2, \ldots, C_m;\\
&\qquad \textbf{time } (min, max);\\
&\qquad \textbf{procedure } = Plan\_Body\\
&\}
\end{aligned}
$$

The $Plan\_Body$ consists of primitive actions, conditional actions, looping actions, and seeing-to-it-that actions. A primitive action represents an action that can be performed directly by the agent's effector. A conditional action is of the form

$$\textbf{if } (Cond) \textbf{ then } Plan\_Body \textbf{ else } Plan\_Body$$

where $Cond$ is a set of variable-free literals which may contain the negation-as-failure operator *not* and explicit negation $\neg$. The use of the negation-as-failure operator allows agents to perform default reasoning and perform actions depending on default conditions. A looping action is of the form:

$$\textbf{while } (Cond) \textbf{ do } Plan\_Body$$

And a seeing-to-it-that action is of the form

$$\textbf{stit } \{F_1, \ldots, F_m\}$$

where each of F_i is a fluent expression. The above action means *to see to it that* $F_1, \ldots, F_m$ *are true*, that is, to bring about a world in which $F_1, \ldots, F_m$ are true.

When no plan is found in the plan library, a plan may be generated by the planner in real time. Roughly speaking, the underlying planner of the current implementation works like this: First it generates an action, whose performance minimizes the difference between the initial state and the goal state, and the intermediate goals are taken as **stit** actions. Meantime, if an intermediate goal is sensorable, a conditional action may be generated, and if an intermediate goal is asserted to be periodical by the built-in predicate $periodical(P, D)$ in the knowledge base (see the commuter example), a looping action may be generated. If necessary, the controller can wait for optimal plans. The planner will generate plans until the controller is satisfied.

8 Knowledge Assimilation

In addition to physical actions the agent can do mental actions which directly modify its knowledge base, action theory, and plan library. In the current implementation, the learning subsystem has not been implemented. In what follows we will discuss the most important internal action, namely knowledge assimilation (also called belief revision): After the agent sensors a new fact N, how will the agent assimilate the new knowledge N into its knowledge base K?

One representative work on belief revision is the so-called AGM theory. As is well-known, some belief revision operators satisfying the AGM postulates give intuitively undesired results. In the literature some other belief revision/update operators have been proposed in order to solve practical problems, and most of them work on finite sets of formulas, called belief base instead of infinite belief set, and can be roughly classified into two types: syntax-based approach, e.g. [11, 26], and semantics-based approach, e.g. [39].

Our SICLE agents adopts a hybrid approach to knowledge assimilation. First all predicates and fluents are classified into following categories: (i) Fluents that can be changed only by the agent itself; (ii) Fluents that can be changed by other agents in multi-agent systems; (iii) Predicates which denote common-sense assumptions on the normality as the $ab(C)$ predicate in model-based diagnosis [35]; (iv) Predicates for which no definitions appear in the knowledge base nor are default assumptions; (v) Predicates which are not of interest. For different categories of predicates, the SICLE agent uses different approaches to assimilate them.

For fluents which can be changed only by the agent itself, when there is a discrepancy between the sensored data and inferred data, then the agent assumes that the sensor is not working well, as these data should be the same as that inferred from its knowledge base. On the other hand, when an effector performs an action, if the action is not successful, the agent assumes that some qualifications of the action are not satisfied[3].

[3] For example, normally if the agent moves a clear block on to the top of another clear block, then the first block will be on the top of the second one. However, this effect description is not always true. If the first block is too heavy, the agent can't move it. If the second block is too small or fragile, the first block can't be put on its top. If there is a fire suddenly in the

For fluents which can be changed by other agents in multi-agent systems, when there is a discrepancy between the sensored data and inferred data, then the agent will take a so-called possible causes approach PCA to knowledge assimilation [22], in contrast to Ginsberg's possible worlds approach PWA [11] and Winslett's possible models approach PMA [39]. A possible cause of new knowledge consists of some abduced occurrences of actions performed by other agents. To assimilate new knowledge, all the effects of the actions which play the role of possible causes for the new knowledge, recursively together with all the effects of actions which play the role of possible causes for the preconditions of the actions whose occurrences are possible causes for the new knowledge, are also assimilated.

For predicates denoting common-sense normal assumptions, the agent will take a consistency-based diagnosis approach [35] to assimilate new data. The general idea is as follows. All default assumptions are collected as a set of predicates, called *abducibles*. When there is a discrepancy between the sensored data and inferred data, then the agent will find a minimal set of abducibles such that disabling them would make the discrepancy vanish. As our knowledge base is essentially non-monotonic and for many interesting domains the description for them is finite, we adopted an ATMS-based computational procedure to maintain the consistency of flat knowledge bases [20]. On the other hand, there is a computational procedure to resume the consistency which is implemented as the system called REVISE [8]. The REVISE system can be used once at a time to resume the consistency. For the moment the REVISE system is directly employed for this kind of knowledge assimilation.

For predicates which have no definitions in the knowledge base nor are defaults, the agent just believes what it sensed at the latest time. This kind of knowledge is easily assimilated.

For predicates which are not of interest, the agent will simply forget them.

9 Related Work

RALPH-MEA

RALPH-MEA (Ogasawara and Russell, 1993)

RALPH-MEA agent = a function $act : P^* \to D$, where
P^* — all perceptions;
D — possible action decisions, either internal or external.

SICLE agent's actions are mainly intention-driven, while RALPH's actions are percepts-driven.

When a SICLE agent has no intentions, it does not deliberatively do any physical actions; it just passively sensors its environment. The RALPH always does some actions according to its percepts.

---

environment, then the two blocks will be burned to ashes. There are potentially many such qualifications for even a simple action. The qualification problem is one of the three major problems with action theories: the frame, qualification, and ramification problems.

Procedural Reasoning System

The design and implementation of the SICLE agents have been influenced by some existing agent architectures such as RALPH-MEA agent [29], the Procedural Reasoning System (PRS) [14], and teleo-reactive (T-R) agents [27]. Although they share a few basic assumptions and characteristics as all agents do, there are many significant differences. In what follows we will briefly compare our work with the Procedural Reasoning System (PRS). The PRS system is an implementation of BDI agents, which is particularly suitable in domains where there are predefined procedures to respond to situations that may arise. The PRS system is composed of (1) a database to hold current beliefs or facts about the world; (2) a set of current goals to be achieved; (3) a set of plans, called Knowledge Area (KA), which describe how certain sequences of actions and tests may be performed to achieve given goals; (4) an intention structure hold the plans which have been chosen from KA for eventual execution. An interpreter controls and manages all these components.

The plan constructs in SICLE are very similar to those intended as operational semantics of the BDI agents [34, 40], and the major differences are as follows: (i) PRS does not possess any theory about actions and changes, and thus its planning capability depends on its library of plans (KA). It assumes that all plans have been pre-determined and stored in its KA. It is, however, often impossible to predetermine all plans in many interesting practical domains. The SICLE agent maintains a library of plans and an action theory. Usually, a plan is selected from the plan library. In case of unexpected situations, an ad hoc plan may be generated. (ii) From [14, 18] it can be seen that the database in PRS is actually a relational database, and is thus weak from the viewpoint of knowledge representation. The SICLE agent maintains a knowledge base which is represented by using a non-monotonic reasoning system. (iii) The SICLE agent maintains beliefs about the past world through reification of fluents, but it is not clear whether historical information is also maintained in the database of PRS. (iv) The semantics of SICLE agents is defined to be a function from situations and intentions to sets of actions, and allows for physical and internal actions. The semantics of PRS seems to be, but is not, an approximation to the BDI logic.

Procedural Reasoning System (Georgeff, et al., 1990)

The major differences:

(i) PRS does not possess any theory about actions and changes, and thus its planning capability depends on its library of plans (KA).

(ii) The database in PRS is actually a relational one (Ingrand and Georgeff, 1990), and is thus weak from the viewpoint of knowledge representation.

(iii) A SICLE agent maintains beliefs about the past world, but it is not clear whether historical information is also maintained in the database of PRS. (Ingrand and Georgeff, 1990)

(iv) A SICLE agent is a function from situations and intentions to sets of actions, and allows for physical and internal actions. A PRS seems to be, but is not, an approximation to the BDI logic.

Teleo-Reactive agents

A teleo-reactive (T-R) agent = T-R program.

$$T - R\,program = \left\{ \begin{array}{l} C_1 \rightarrow A_1 \\ C_2 \rightarrow A_2 \\ \ldots \\ C_n \rightarrow A_n \end{array} \right\}$$

A T-R program can be represented as a procedure in a SICLE plan of the form

while (true) **do** $\{$
 if C_1 **then** A_1 **else**
 if C_2 **then** A_2 **else**
 ...
 if C_n **then** A_n
$\}$

Concurrent METATEM

Concurrent METATEM is a multi-agent programming languages [9, 41]. Agents in Concurrent METATEM are concurrently executng entities, able to communicate with each other through asynchronous broadcast message passing, and can be essenticially represented by a triple: $(in, out, rules)$, where in is a set of environment propositions representing messages

10 Commuter: an example

As an example we now consider a commuter who lives in Marques de Pombal of Lisbon and works in Caparica. And thus he commutes between Marques de Pombal and Caparica. The agent commuter can be specified by the SICLE language as follows:

agent commuter = {
 fluent ...;
 predicate ...;
 sensorable ...;
 abstype place = [Pombal, Spain_Square, Sodre, Cacilhas,
  Caparica, Rossio, Campo_Pequeno, ...];
 place Origin, Destination;
% Comments like this one start and end with a percent symbol. The strings in bold form like **fluent** are reserved keywords. The above are declarations for data types and variables. If a string is not declared to have a data type, then it is regarded as a constant. There are a few basic built-in data types such as strings and integers. In SICLE , every statement is called a declaration. A set of SICLE declarations is called a SICLE program. The sequence of SICLE declarations has no effect on its semantics. %
 action take(bus(Origin, Destination)) = {
  when in(Origin), exists(bus(Origin, Destination))
  time (10 minutes, 40 minutes)
  causes $in(Destination), \neg in(Origin)$
 }
 action take(metro(Origin, Destination)) = {

when in(Origin), exists(metro(Origin, Destination))
time (10 minutes, 40 minutes)
causes $in(Destination), \neg in(Origin)$
}
action take(ferry(Origin, Destination)) = {
when in(Origin), exists(ferry(Origin, Destination))
time (10 minutes, 40 minutes)
causes $in(Destination), \neg in(Origin)$
}
knowledge base = {

$holds(in(Pombal), 6\ am).$
$holds(in(work), T) \leftarrow holds(in(office), T), \text{not } ab(in(work), T).$
$holds(in(office), T) \leftarrow holds(in(Caparica), T), \text{not } ab(in(office), T).$
$holds(in(home), T) \leftarrow holds(in(Pombal), T), \text{not } ab(in(home), T).$
$periodical(exists(bus(Origin, Destination)), 10minutes).$
$periodical(exists(ferry(Sodre, Cacilhas)), 10minutes).$
$periodical(exists(ferry(Cacilhas, Sodre)), 10minutes).$

% $holds$ and $periodical$ are built-in predicates for reification of fluents and periodically changed fluents. The predicate $periodical(F, D)$ means that F will be true every D time. %

}
intentions = {
$in(work)$ **always from** $10am$ **to** $1pm$,
$in(work)$ **always from** $2pm$ **to** $6pm$,
$in(home)$ **always from** $10pm$ **to** $6am$
}
plan go_to_Caparica_from_Pombal = {
for in(Caparica)
when in(Pombal)
time (30 minutes, 60 minutes)
procedure = {
while (*not* in(Caparica)) **do** {
if $in(Spain\_Square) \wedge exists(bus(Spain\_Square, Caparica))$
then $take(bus(Spain\_Square, Caparica))$;
if $in(Pombal) \wedge exists(metro(Pombal, Spain\_Square))$
then $take(metro(Pombal, Spain\_Square))$;
}
}
}
plan go_to_Caparica_from_Cacilhas = {
for in(Caparica)
when in(Cacilhas)
time (10 minutes, 30 minutes)
procedure = {
while (*not* in(Caparica)) **do** {

```
        if  in(Cacilhas) ∧ exists(bus(Cacilhas, Caparica))
        then  take(bus(Cacilhas, Caparica));
      }
    }
  }
% The bodies of the following two plans are left out. %
  plan go_to_Pombal_from_Caparica = {
        ...
  }
  plan go_to_Cacilhas_from_Caparica = {
        ...
  }
}
```

After the above SICLE program is submitted to the SICLE interpreter, an agent will be born and will work as follows. Suppose now it is 6 am. First the commuter asks the planner to generate plans for each intention and measure their priority. As there are existing plans from the plan library, the commuter will pick up the plan $go\_to\_Caparica\_from\_Pombal$ for the intention of working from 10 am to 1 pm in Caparica. Since the intention of working from 10 am to 1 pm in Caparica is most urgent, the commuter will make a choice: to perform the plan $go\_to\_Caparica\_from\_Pombal$. As he is in Pombal, he must determine if there is a metro to Spain Square. Suppose there is a metro. Then, the commuter will take it to get to $Spain\_Square$. Then, the commuter must make another choice for the next action. At this time, the intention of going to work still has a higher priority over other intentions, then commuter needs to determine if there is a bus to Caparica, as the commuter is now in $Spain\_Square$. Suppose there is no bus. Then it will actually wait there by doing a looping action. Repeating this process, it finally finds a bus, takes it and gets to Caparica. The plan is fulfilled, and thus removed from the agenda.

Normally the behaviour of the commuter looks like the above. But sometimes there are exceptions. Now suppose there is a traffic accident so that the traffic route from $Spain\_Square$ to $Caparica$ is stopped. Then, after the commuter waits in $Spain\_Square$ for a sufficient long time (as the plan says, the expected maximal time is 60 minutes to fulfil this plan), it must discard the existing plan in its agenda and generates another plan whose procedure is as follows:

```
while  (not in(Sodre)) do  {
    if  exists(bus(Spain_Square, Sodre))
    then  take(bus(Spain_Square, Sodre)
};
stit in(Cacilhas);
go_to_Caparica_from_Cacilhas
```

After the agent finishes the looping action, it will be in Sodre. In order to achieve

$in(Cacilhas)$, the commuter simply generates another plan like this:

$$\begin{array}{l} \textbf{while}\ \ (not\, in(Cacilhas))\ \textbf{do}\ \{ \\ \quad \textbf{if}\ \ exists(ferry(Sodre, Cacilhas)) \\ \quad \textbf{then}\ \ take(ferry(Sodre, Cacilhas)) \\ \} \end{array}$$

Then the commuter can realize its intention of working in Caparica.

Certainly there are a very great number of choices the commuter has made in one day. The above example is over-simplified. But it can be seen that the reactive plan stored in the plan library normally works. But when some unpredicted phenomenon arises, the commuter can deliberatively make new plans. On the other hand, the capability of the agent depends on its knowledge bases and action theory, etc.

11 Conclusion

In this preliminary report we have presented the design and architecture of a knowledge-based situated agent system embedded in the real world, called SICLE agents, and an agent specification language called SICLE . Although its major modules such as knowledge assimilation, planning, and knowledge base manager have been implemented by using the latest version of the REVISE system [8], the simulated SICLE agent has not been fully implemented and the implementation of a user interface is in progress. The SICLE architecture and its language SICLE are related to each other, and its general semantics is defined to be a function from situations and intentions to choices of actions. In our (partially) implemented SICLE agents, after the SICLE interpreter receives a SICLE specification of an agent's initial knowledge base, atomic actions (their conditional effects and executability conditions), initial intentions and some predefined optimal plans, the agent for this specification is produced and become active. The agent will always try to realize all its intentions at its best and may produce new intentions or receive new intentions from its user through the remote control in the real-time. The SICLE interpreter employs a non-monotonic reasoning system to model the real-world and record the past history of the environment. For those situations which may frequently arise, the user can provide optimal plans to achieve timely response. A simple simulation of human commuter between two cities has been used to present our approach.

Acknowledgements

This work was supported by JNICT PROLOPPE project under PRAXIS/3/31/TIT/24/94 and a post-doctoral fellowship from JNICT under PRAXIS XXI/BPD/4165/94.

References

1. Alchorrón, C., Gärdenfors, P. and Makinson, D., On the logic of theory change: Partial meet contraction and revision functions, *J. of Symbolic Logic*, 50:2, 1985, 510 – 530

2. Alferes, J. J. and Pereira, L. M., Reasoning with Logic Programming, LNAI, Springer-Verlag, 1996 (forthcoming)
3. Arkin, R. C., Towards the unification of navigational planning and reactive control, AAAI Spring Symposium on Robot Navigational Working Notes, 1989, 1–5
4. Agre, P., and Chapman D., Pengi: An implementation of a theory of activity, AAAI'87, 268–272
5. Baker, A.B., Nonmonotonic reasoning in the framework of situation calculus, *Artificial Intelligence*, 49, 1991, 5–23
6. Brooks, R. A., A robust layered control system for a mobile robot, IEEE Journal on Robotics and Automation, RA-2(1), 1982
7. Carbonell, J., Etzioni, O., Gil, Y., Joseph, R., Knoblock, C., Minton, S., Veloso, M., PRODIGY: An Integrated Architecture for Planning and Learning. *SIGART Bulletin*, 2, 51-55, 1991
8. Damásio, C. V., Pereira, L.M., and Nejdl, W. , REVISE: An Extended Logic Programming System for Revising Knowledge Bases, *Proc. of KR'94*, 1994
9. Fisher, M., A survery of Concurrent METATEM — the language and its applications, in: D. M. Gabbay and H. J. Ohlbach (eds.), *Temporal Logic — Proc. of 1st International Conf., LNAI 827*, Springer-Verlag, 1994, pp. 480–505
10. Gelfond, M. and Lifschitz, V., Representing action and change by logic programs, *Journal of Logic Programming*, Vol.17, 1993, 301–322
11. Ginsberg, M., Counterfactuals, *Artificial Intelligence*, 30:1, 1986, 35 – 79
12. Hanks, S, and McDermott, D., Nonmonotonic logics and temporal projection, *Artificial Intelligence*, 35, 1988, 165–195
13. Haugh, B.A., Simple causal minimizations for temporal persistence and projection, *Proc. of the AAAI87*, 1987, 218–223
14. Ingrand, F. F., and Georgeff, M. P., Managing deliberation and reasoning in real-time AI systems, *Proc. of the 1990 DARPA Workshop on Innovative Approaches to Planning Scheduling and Control*, 1990, 284–291
15. Kaelbling, L. and Rosebschein, S., Action and planning in embedded agents, Robotics and Autonomous Systems, 6, 1990, 35–48
16. Kautz, H.A., The logic of persistence, *Proc. of the AAAI86*, 1986, 401–405
17. P. Ladkin, The completeness of a natural system for reasoning with time intervals, *Proc. of IJCAI' 87*, Milan, 1987, 462-467
18. Lee, J., Huber, M. J., Durfee, E. H., Kenny, P. G., UM-PRS: An implementation of the procedural reasoning system for multirobot applications, *Proc. of CIRFFSS '94, Conference on Intelligent Robotics in Field, Factory, Service, and Space*, 1994, 842–849
19. Li, R. and Carmo, J., On completeness of a positional interval logic with equality, overlap and subinterval relations, *Journal of the IGPL*, 3:5, 1995, 765-790
20. Li, R. and Pereira, L. M., Maintenance of Consistency in Well-Founded Models of Extended Logic Programs with Explicit Negation, Technical Report, DI-FCT-UNL, 1995, available through http://www-ia.di.fct.unl.pt/ renwei/for-consistency.ps.gz (submitted for publication)
21. Li, R. and Pereira, L. M., Temporal reasoning with abductive logic programming, *Proc. of ECAI'96*, Budapest, 1996
22. Li, R. and Pereira, L. M., What is believed is what is explained (sometimes), *Proc. of AAAI'96*, Portland, Oregon, 1996
23. Lifschitz, V., On the semantics of STRIPS, *Reasoning about Actions and Plans: Proc. of the 1986 Workshop*, Morgan Kaufmann Publishers, 1987
24. Lifschitz, V., Pointwise circumscription, *Readings in Nonmonotonic Reasoning*, M.L.Ginsberg (ed.), Morgan Kaufmann, 1987, 410–423

25. Lifschitz, V., Formal theories of action, *The Frame Problem in Artificial Intelligence*, Morgan Kaufmann, 1987, 35–57
26. Nebel, B., Belief revision and default reasoning: Syntax-based approaches, *Proc. of KR'91*, 1991
27. Nilsson, N. J., Teleo-reactive programs for agent control, JAIR, 1, 1994, 139–158
28. Sacerdoti, E. D., The nonlinear nature of plans, *IJCAI 75*, 1975, 206 – 214
29. Ogasawara, G., RALPH-MEA: A Real-Time Decision-Theoretic Agent Architecture, Technical Report CSD-93-777, Uni. of California, Berkeley,1993
30. Payton, D., Internalized plans: A representation for action resources, in: P. Maes (ed.), Designing Autonomous Agents, MIT Press, 1991, 89–103
31. Pereira, L. M. and Alferes, J. J., Well-founded semantics for logic programs with explicit negation, in *European Conf. on Artificial Intelligence*, B. Neumann (ed.), John Wiley & Sons, 1992, 102-106
32. Przymusinski, T., On the declarative semantics of stratified deductive databases and logic programs, In J. Minker, editor, , Foundations of Deductive Databases and logic Programming, Morgan Kaufmann, 1987, 193-216
33. Rao, A. S., and M. P. Georgeff, Modelling rational agents within a BDI arcitecture, *KR'91*, Morgan-Kaufmann Publishers, San Mateo, CA, 473–484, 1991
34. Rao, A. S., Operational semantics of BDI agents, in: J. L. Fiadeiro and P.-Y. Schobbens (eds.), *Proc. of the 2nd ModelAge Workshop*, Lisbon, 1995, 211–222
35. Reiter, R., A theory of diagnosis from first principles, *Artificial Intelligence*, 32:1,1987, 57-96
36. N. Rescher and A. Urquhart, *Temporal Logic*, Springer-Verlag, 1971
37. Shoham, Y., *Reasoning about Change*, The MIT Press, 1987
38. Tate, A., Generating project networks, *IJCAI 77*, 888 – 893
39. Winslett, M., Reasoning about action using a possible models approach, *Proc. of AAAI'88*, 89 – 93
40. Wooldridge, M., A logic of BDI agents with procedural knowledge, in: J. L. Fiadeiro and P.-Y. Schobbens (eds.), *Proc. of the 2nd ModelAge Workshop*, Lisbon, 1995, 301 –315
41. Wooldridge, M., A knowledge-theoretical semantics for concurrent Metatem, in: J. P. Muller, M. J. Wooldridge, and N. R. Jennings (eds.), *Intelligent Agents III - Proc. of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96), Lecture Notes in Artificial Intelligence*, this volumn, Springer-Verlag, 1996

This article was processed using the LaTeX macro package with LLNCS style