Sita: A Computational Model for Situated Acting

Thomas Weiser
Chair of Real-Time Systems and Robotics, Department of Computer Science
Technische Universität München
D-80290 München
email: weiser@in.tum.de

Abstract: Sita is a novel language for modeling intelligent behavior in dynamic environments. The central modeling principle is the distinction of recognition and action as the two complementary basic skills essential for situated behavior. The underlying computational model combines an incremental bottom-up logic programming mechanism with a concurrent process calculus. This principle allows for the clear separation and natural description of declarative knowledge about situations and procedural knowledge about acting.

Keywords: reactive logic programming, agent architecture, computational model

I. Introduction

In this paper we propose a new computational model for intelligent agents.

According to Shoham [10] a hardware or software system gets an agent as soon as we use mental concepts (like beliefs, goals, intentions, etc.) to analyze or to model the system. Furthermore agents are self-contained systems interacting with their environment they live in.

These properties make the notion of agent a popular and valuable model building concept in robotics. Examples of applications are: Distributed planning and scheduling in flexible manufacturing plants, sensor data fusion, and recently football playing. See for example [11].

Much work has been done on the theoretical basis of agenthood. Probably the best known is the BDI (Belief, Desire, Intention) approach [9], which builds on a temporal modal logic. The conceptual level of these theories is too abstract to directly derive concrete agents. Moreover, complete proof procedures are too complex to have a feasible implementation.

On the other hand there is much work on agent architectures, like InteRRaP [7] or dMars [2]. They use functional models to describe the agent's behavior, partitioned in several modules. The employed concepts are crude and very abstract again. Therefore these architectures are more helpful for the abstract agent specification, but less for the implementation.

According to these observations there are needs for concrete computational models supporting the agent design process from the implementation language side. Such models should lift the expressiveness of the implementation language to a level and a direction more suitable for agent design, compared to the presently most used languages like C++, Java or Prolog. Those languages suffer from either being very low-level or having no support for reactivity. Narrowing the gap between the architectural level and the language level will substantially enrich the methodology of agent design.

In this paper we present such a novel computational model, which we call Sita (situated action). The aims behind the development of Sita can be summarized as follows:

Support for reactive behavior: An agent lives in a dynamic environment, which is unpredictable in principle. Therefore the agent must be able to trigger adequate, possibly nested reactions.

Reasoning on complex situations: The agent must classify its situations to know how to respond to them. This can be a complex reasoning task, which again needs to be performed in a reactive manner.

Coordination of concurrent activities: Agent's behavior is composed of a varying set of execution threads. As these concurrent threads highly depend on each other, there must be mechanisms to coordinate those processes.
Precise semantics: A firm semantic basis simplifies the understanding. Furthermore it is a prerequisite for formal program verification. (This could be a future development.)
Feasible implementation: The computational model needs to have an effective and efficient implementation.

II. Recognize and Act

The central modeling principle of Sita is the distinction of *recognition* and *action* as the two basic skills essential for situated behavior: Passively waiting for something to happen, and actively making something happen. Accordingly an agent is specified through describing firstly the situations the agent should recognize, and secondly the reactions the agent should respond with.

This distinction is rooted in the tradition of production systems like OPS5 and CLIPS, where programs are given by a set of condition-action-pairs [6]. An example for the use of a production system is the multi agent test-bed Magsy [3]. Each agent is an OPS5 interpreter extended by the capability of asynchronous message passing. Magsy has been used for building a distributed planner for flexible manufacturing plants [4] and for controlling forklifts in an automated loading-dock [8].

However, production systems suffer from two substantial drawbacks, which restrict their usefulness for the mentioned applications:
1. The rule selection process utilizes a very simple pattern matching concept with little expressive power. The condition parts of the rules are composed solely of fact patterns as primitives. There is no concept to abstract condition expressions under a new name. So one cannot compose complex expressions out of other expressions. Furthermore, this makes it impossible to use recursive formulae.
2. The action parts of the rules are simple sequences of actions without any control structures. Complex procedural operations have to be scattered to several rules, whereby the user is forced to manage the execution context by his own.

While preserving the advantages of production systems (reactive, symbolic, event-driven computation), Sita introduces new concepts both for the recognition and the action phase to overcome these drawbacks.

Two different formalisms are used for modeling the two basic skills (see fig. 1). On the one side there is a *deductive knowledge base*, responsible for monitoring the (internal and external) state. This is a process of analyzing the current situation, building an abstract world model, activation of corresponding goals and calling for adequate actions. To serve these complex tasks the Sita knowledge base applies a logic programming system based on Horn clause logic.

On the other side there is a *concurrent procedural language*, responsible for the description and coordination of the action threads the agent is in. It aims at a general but high-level model of coordinated computing. In contrast to production systems there is a powerful set of imperative concepts: sequential and parallel composition, guarded choice based on knowledge base queries, and definition of task abstractions by means of procedures.

These two components are linked through a common interface. Firstly, there are primitive actions to modify the knowledge base, i. e. the insertion and deletion of facts. Secondly, there are queries that trigger procedural actions upon entailment of the query expression.

According to this architecture, the agent's state is split into two distinct components: the facts present in the knowledge base which can be queried and updated by processes; and the task, representing the processes to be executed by the agent.

To be linked with the outside world the agent needs capabilities to perceive and to act. Perceived information is stored as messages in the knowledge base. External actions are effected through special primitives in the procedural part.

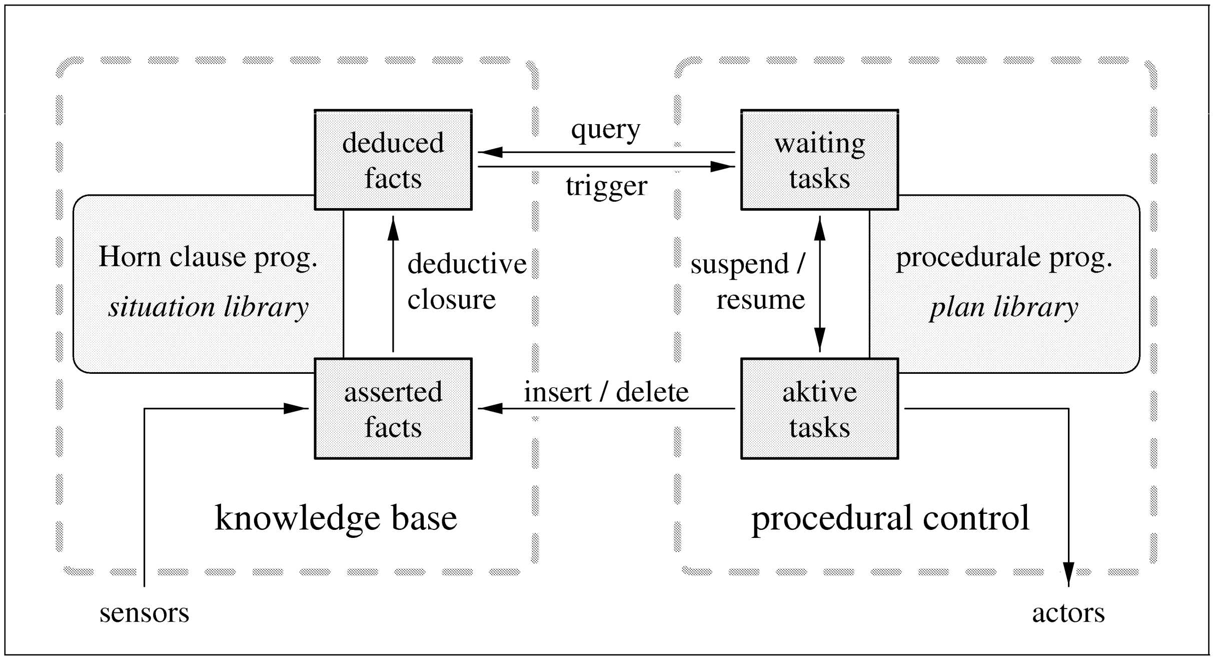

Fig. 1. Sita architecture

III. Knowledge Base

The Sita knowledge base applies Horn clause logic with negation as failure and function symbols in order to handle knowledge representation and abstraction, situation recognition and decision making. It consists of a logic program, a fact base and a forward-chaining inference machine.

The basic expressions of the logic language are predicates, which come in three flavors: Extensional predicates are containers for those facts that may be asserted or retracted through actions or perception. Intensional (or derived) predicates are defined by the clauses of the logic program and are interpreted by the deduced facts. Built-in predicates provide for some basic functions, e.g. arithmetic operations. Accordingly the fact base contains two sets of facts, asserted and deduced ones.

The inference engine continuously maintains the set of deduced facts in dependence of the current set of asserted facts and in correspondence to the logic program. This maintenance is an incremental and active reasoning process. All changes in the extensional part of the fact base will cause corresponding changes in the intensional predicates.

In the following example, it is assumed that `edge` is an extensional predicate. The two clauses define the transitive closure `path` based on the `edge` relation. Whenever the base relation changes, the derived relation will change accordingly. In this example, the `edge` relation may grow by and by through certain perceptions, while the `path` relation may continuously guide the agent's action to find a path between certain nodes.

```
path(X Y) ← edge(X Y).
path(X Y) ← path(X Z) edge(Z Y).
```

The knowledge base is a pure *declarative* formalism, which gets its meaning through the well-founded semantics [12]. The advantages are e.g. that the ordering of the clauses within a program is irrelevant. The same goes for the ordering of the condition elements within the body of a clause. This is a big improvement compared to conventional logic programming systems like Prolog.

The evaluation is done by an incremental bottom-up algorithm, which is an essential property for obtaining reactive, event-driven situation recognition. To accomplish this, we extended the well-known Rete-algorithm [5] in several ways. Firstly, in the presence of recursive defined relations care must be taken about facts, that either support themselves or negate themselves. Hereto we have developed appropriate reason maintenance al-

elementary tasks:	**insert** *fact*
	delete *fact*
	send *address msg*
sequence:	$t_1\ t_2$
parallel:	$t_1 \parallel t_2$
guarded choice:	$\quad q_1 \Rightarrow t_1$
	$[]\ q_2 \Rightarrow t_2$
priority assignm..:	**prio** n
procedure def.:	**proc** $name$:
	$\quad t$
	end

Fig. 2. task composition

gorithms. Secondly, we apply the magic set transformation [1], known from deductive databases. With this technique, instead of generating the whole set of consequences, only those parts of the logic program are evaluated that contribute to answering the current queries. Thirdly, we found that the efficiency of the algorithm is highly dependent on the order in which changes a propagated through the clauses. As this goes beyond the scope of this paper, we omit the details here.

IV. Procedural Language

The agent's knowledge base is complemented by a concurrent procedural language.

The current state w.r.t. procedure execution is represented by the agent's present task. A task is either a elementary action, a compound task, or the name of a procedure (see fig. 2).

The elementary actions are the insertion and deletion of facts. In addition certain builtins provide for special actions, like sending a message to an other agent, commanding an effector, or accessing the operating system.

Tasks can be combined by sequential and parallel composition. In addition a guarded choice is available: given a set of query expressions, the task execution suspends until at least one of the expressions is entailed by the knowledge base. In that case the execution is continued with the body of one of the entailed branches. The query expressions used as guards have the same syntax as clause bodies.

A procedure definition takes a task expression and makes it available under a given name. Procedures may have formal arguments. Procedures are not only used to structure the program, they are essential to express recursive task structures.

Through the use of the parallel composition multiple threads of execution are introduced, which allow for the required concurrency. A special statement is available to assign relative priorities to threads. This is useful to distinguish between, for example, an urgent reactive behavior and a background planning task.

V. Discussion

The Sita architecture combines a deductive knowledge base with a concurrent procedural control component. This structure reflects a basic model for intelligent agents. On the one side an agent has to maintain its current beliefs about the world and itself. This knowledge has to be represented on different abstraction levels. Higher levels model the agent's view of its situation and current goals. The Sita knowledge base is a tool to describe such abstraction processes by means of Horn clause logic in a purely declarative manner. This enables the agent to make reasoned choices. On the other side an agent has to change the world as well as its own beliefs and intentions. Procedures are a natural way to describe these active aspects. The presented combination of declarative and procedural concepts results in a novel programming model for reactive, intelligent agents.

We have implemented most parts of the presented architecture. In particular, we have a fast and complete implementation of the knowledge base, together with a subset of the procedural language. This has given us the ability to gather some first experiences regarding our approach of declarative bottom-up logic programming. First example applications are the n-queens problem, heuristic reactive search algorithms, and a blocks world planning system.

We have found that most aspects of these problems can be modeled declaratively (and nevertheless efficient), i.e. solely through the knowledge base. This confirms our assumption that the expressiveness of the knowledge base formalism is sufficient to perform complex situation recognition tasks.

Future work will focus on applications in more complex scenarios, where the need for intelligent reactivity is more evident. As a first step towards this direction we are currently working on the integration of the Sita programming system into our robotics laboratory.

References

[1] C. Beeri and R. Ramakrishnan. On the power of magic. In ACM, editor, *PODS '87. Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 23–25, 1987, San Diego, California*, pages 269–284, New York, NY 10036, USA, Mar. 1987. ACM Press.

[2] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In M. P. Singh, A. Rao, and M. J. Wooldridge, editors, *Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, volume 1365 of *LNAI*, pages 155–176, Berlin, July 24–26 1998. Springer.

[3] K. Fischer. The rule-based multi-agent system MAGSY. In *Proceedings of the CKBS'92 Workshop*. DAKE Centre, University of Keele, UK, 1993.

[4] K. Fischer. *Verteiltes und kooperatives Planen in einer flexiblen Fertigungsumgebung*, volume 26 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. Infix, St. Augustin, Germany, 1993.

[5] C. L. Forgy. *On the Efficient Implementation of Production Systems*. PhD thesis, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1979.

[6] T. Ishida. Parallel, distributed and multi–agent production systems – A research foundation for distributed artificial intelligence. In V. Lesser, editor, *Proceedings of the First International Conference on Multi–Agent Systems*, pages 416–422, San Francisco, CA, 1995. MIT Press.

[7] J. P. Müller. *The design of intelligent agents: a layered approach*, volume 1177 of *LNAI*. Springer, New York, 1996.

[8] J. P. Müller and M. Pischel. The agent architecture inteRRaP: Concept and application. Research Report RR-93-26, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, Germany, 1993.

[9] A. S. Rao and M. P. Georgeff. Modeling Agents Within a BDI-Architecture. In R. Fikes and E. Sandewall, editors, *Proc. of the 2rd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484, Cambridge, Mass., Apr. 1991. Morgan Kaufmann.

[10] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, Mar. 1993.

[11] H.-J. Siegert and S. Bocionek. *Robotik: Programmierung intelligenter Roboter*. Springer-Verlag Berlin Heidelberg New York, 1996.

[12] A. van Gelder, K. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.