Abstract

This paper proposes a language called CLAIM and its operational semantics. The agents implemented in CLAIM are endowed with cognitive capabilities, are able to communicate with other agents and are mobile. The primitives of mobility are inspired from the ambient calculus.

1. Motivations

We argue in this paper that, for an effective use of MAS paradigm, specific high-level programming languages are required. Our work is motivated by three main objectives:

1. Propose an agent oriented programming (AOP) language that helps the designer to reduce the gap between the design and the implementation phases (i.e. the designer should think and implement in the same paradigm, namely through agents), that allows the representation of cognitive skills such as knowledge, beliefs and more complex mechanisms such as planning, decision making and reasoning and that meets the requirements of mobile computation.

2. Make verification of systems possible. Indeed, at a short term we would also like an AOP language to be amenable to reasoning and verification of the built systems. A first and necessary step towards developing methods for verifying formally agent-oriented programs is the design of a suitable operational semantics. It opens the way to the application of standard techniques like type systems or model-checking to the setting of agent-oriented programming.

3. Provide a distributed platform that supports the proposed language and the deployment of mobile MAS.

To reach our objectives, we propose a declarative language called CLAIM (Computational Language for Autonomous, Intelligent and Mobile agents) where are combined the main advantages of the intelligent agent paradigm (e.g. autonomy and cognitive skills) with those of the concurrent languages such as the ambient calculus [1] which has been recently proposed as a theoretical framework for distributed and mobile objects/agents.

2. CLAIM

In this section we present a brief overview of the specifications of CLAIM, both the syntax of the programming language [2] and its operational semantics. We think the design of the semantics is important to define the operations of this complex language in a non-ambiguous way and to abstract away from irrelevant implementation aspects. The semantics is inspired from the ambient calculus [1] with extensions in three directions: integration of an agent structure in order to model a distributed agent system, security awareness and agents’ knowledge management.

An agent in CLAIM can be seen as a bounded place where the computation happens (similar to ambients) and has a list of local processes concurrently executed and a list of sub-agents. It has mental components such as knowledge, rules and goals, that enable a forward (reactive behavior) or a backward reasoning (goal driven behavior). In order to model correctly such a distributed environment of mobile agents, we define a multi-agent system as a multiset of sites (connected computers). Each site $A @ s$ has an address $s$ and a hierarchy of agents $A$. An agent, semantically noted $a[ M, P, A ]_{R, K,}$, has a name $a$, a message management system, a set of running processes $P$ and sub-agents $A$. $M$ is a set of messages, asynchronously managed and filtered by a set of rules $R$. A rule triggers a process according to a pending message if the rule filters the message (name and parameters), and also if some pre-condition is satisfied. Finally an agent has a set of knowledge $K$.

In the programming language of CLAIM, agents and classes of agent are defined using:

```
defineAgent / defineAgentClass name {  authority=null; | agentName ;  parent=null; | agentName ;  knowledge=null; | { (knowledge;)+}  goals=null; | { (goal;)+}  messages=null; | { (queueMessage;)+}  rules=null; | { (rule;)+}  ```
A process can be a (possibly empty) sequence of actions, where an action is a message sending, a creation or a kill of an agent, a mobility operation, an open action or an action dealing with knowledge:

\[
\text{process } P := 0 | P | P | X \cdot P \\
\text{actions } X := \text{send}(a, m) | \text{kill}(a) | \text{add}(k) | \text{remove}(k) | \text{move}(a) | \text{open}(a) | \text{in}(a) | \text{out}(a) \\
\]

The operational semantics (fig. 1) is a set of rewriting rules. At each step, either a message is processed via a rule, or a running process is executed. For readability reasons, unused fields are not mentioned. In the send rule, \(a\) and \(b\) are two agents acting in parallel (noted \(\parallel\)) in the same site. In the eval rule, a message with name \(m\) and arguments \(\vec{x}\) is filtered by a rule with name \(m\), arguments \(\vec{x}\) and process \(Q\) added to the set of running processes if the rule is triggered.

Distributed systems require specific security functionalities. For this goal, mobility actions distinguish two levels of security according to the relative positions of the agents that want to communicate. Mobility inside a site is uncontrolled, as in the original ambient calculus: with actions in, out, an agent has the opportunity of entering an agent (in the same level in the hierarchy), or exiting his parent. Mobility through sites (move) is subject to control. In the same spirit as in the safe ambient calculus [4], we use co-actions to specify the control.

The programming language includes some additional features, especially for invoking methods implemented in other programming languages (Java, in this version).

3. Conclusion and Future Work

This paper presents the main elements of an operational semantics for the CLAIM language. This semantics is a first step towards the verification of MAS built using CLAIM. To support the CLAIM language, a distributed platform was developed, called SyMP A [3], compliant with the specifications of the MASIF [5] standard from OMG. SyMP A has been used for several applications designed with CLAIM such as e-commerce, resource sharing and load balancing. Our current work tackles the interoperability with agents from other platforms (using a Web Services approach) and the security aspects, both at the language and the platform levels. New primitives of security should be integrated to the syntax of CLAIM while mechanisms of security have already been implemented at the platform level.

References


