CASA - Structured Design of a Specification Language for Intelligent Agents

Stephan Flake and Christian Geiger

C-LAB VIS (Visual Interactive Systems)
Fuerstenallee 11, 33102 Paderborn, Germany
{flake, chris}@c-lab.de
WWW home page: http://www.c-lab.de/vis

Abstract. The interest in agent based technologies in the sense of distributed computing is continuously increasing in academic and industrial research and development. But the concentration of research in distinct niches has lead to a lack of consensus regarding basic concepts of agent theory and their relation to the development of agent systems and applications. This gap between theory and practice has been recognized and many research groups focus on the relation of formal specification methods for agent properties to the design of practical multi agent systems (MAS). In this paper we describe the design and implementation of CASA, a multi agent specification language that builds on an existing formal agent specification approach [13] and extends it by concepts from concurrent logic programming. With CASA it is possible to design agents with complex behavior patterns (e.g. parallel strategies, speculative computations). Application areas of CASA agents are manufacturing systems, robotics and intelligent user interfaces.

1 Introduction

Agent-based technology in the sense of distributed computing is growing dramatically in many directions. From a design point of view agents provide a natural metaphor for conceptualizing and building a wide range of complex computer systems. These systems contain many passive objects but also many active components which can best be described by the notion of agents. Agent-based systems literature distinguishes micro and macro views. The micro view considers the local behavior of an individual agent whereas its environment and interaction is investigated in the macro view.

From an implementation point of view the existence of special libraries or dedicated programming languages that provide data and control structures for describing and manipulating agent specific properties allows a straight forward implementation of the designed models. Also, there is considerable work on formalizing notions in MAS (e.g. commitments, capabilities, know how). Advanced logics based on temporal and modal logic that capture essential agent properties like concurrency, time dependent behavior or inconsistent states are under development. Well known specification techniques like Z are applied to formally capture all aspects of such systems. Even well-known verification methods like model checking have been successfully applied in the agent oriented domain.

A major criticism of much of the formal work is, that only little advice is given how to apply theoretical results to practical realizations. On the practical side there is a significant number of prototype implementations, some addressing to solve real world problems [1] like flight allocation, manufacturing resource allocation and information retrieval in large databases. These practical approaches mostly follow an ad-hoc approach building systems from scratch. Due to the complexity of the developed code implemented systems often lack an underlying sound theory.

In addition, as reported by Nwana in [11], a couple of new concepts in recent MAS research seems to reinvent the wheel. There is already a large amount of relevant and first class literature published in traditional AI, system design, parallel computing, and software engineering which directly apply to the principles of agent-based systems. Summarizing, the design of a MAS should be based on the following principles: (1) transformation from theoretical phase to practical realization, (2) regard and, if possible, reuse well-known concepts from related fields, (3) extendable and flexible design approach that is efficiently realized.

The work presented in this paper describes CASA, a multi agent specification language, and its efficient implementation. The design of CASA was strongly inspired by the proposed principles and followed a structured design approach consisting of the consecutive design phases of specification, modelling, prototyping and validation. The structure of the remainder mainly follows these phases. After describing some basic concepts of MAS in general and the well-known BDI approach in particular we describe the micro level specification of CASA agents based on the BDI specification language AgentSpeak [13] extended by concepts taken from concurrent logic programming [15]. The next section describes the modelling of CASA agents and concentrates on the transformation of the (semi-)formal concepts to a description language for a MAS. Our current prototype implementation which is using existing Java-libraries is described thereafter. We conclude with some examples that serve as a first validation and the description of future work in this area.

2 Basics of Multi Agent Systems

Agents are characterized as a set of concurrent objects acting locally according to specific incoming messages and perceived events. On a higher description level these objects have complex internal states and act according to strategies. Properties of agents are autonomy, reactive / pro-active behaviour and structured messages for communication. A metaphor for explaining their functionality can assign certain mental capabilities (e.g. beliefs, goals, desires, intentions) to these active objects. Many approaches of agent architectures exist in the AI-literature (see [9]) and a significant part of these is based on Rao's and Georgeff's BDI-approach [12, 13] in which agents have beliefs, desires or goals and build intentions to achieve their goals by means of plans. For the purpose of our work the following naive view of agents is sufficient (see Fig. 1).

Each agent has got a knowledge base in which the agent's beliefs are stored. Beliefs can be seen as facts or data describing the agent's model of the world. Plans provide strategies for possible behaviour in form of rules. To achieve agent specific goals, certain strategies are activated based on available data (beliefs) and incoming messages (from other agents) as well as events generated by the user or the system. Executing these strategies results in certain actions which may generate new messages or change the agent's environment. The control mechanism is an endless cycle in which incoming messages and relevant events are observed. This results in a possible update of the knowledge base (if new information is available) and the selection of suitable strategies in order to fulfill the agent's goals. Agent behaviour is often modeled by operational means [13]. Based on definitions for plans, beliefs, goals, etc. an abstract interpreter

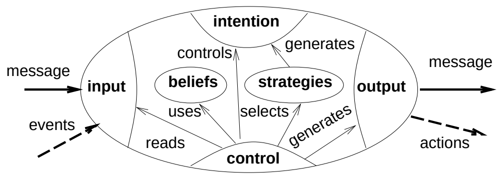

Fig. 1. Naive Agent View

defines the control cycle running in each agent. Strategies are described declaratively in terms of clauses or rules similar to horn clauses or production rules. Additional tests as preconditions ensure the applicability in a given context.

2.1 BDI agents

Within the MAS community, the BDI model [12] has come to be possibly the best known and best studied model of agency. Perhaps the most compelling reason is that this model combines a respectable philosophical model of human practical reasoning [3], a number of well designed implemented systems [1] and several successful applications (e.g. factory process control, business process, fault diagnosis systems). Finally, an elegant abstract logical framework has been developed that serves as theoretical foundation and allows to relate this model to other computational models of concurrency [14]. Beliefs, desires and intentions are an essential part of the state description of a complex system viewed from a high abstraction level. The purpose of the BDI model of agents is to characterize entities using these anthropomorphic notions as a sound formal base. Specification, analysis and verification of rational agents then becomes possible. If these concepts are realized as efficient data and control structures they may form the core components for systems that learn, can handle inconsistent and uncertain information and adapt dynamically to a changing environment.

However, it is generally concerned that there exists a gap between those powerful BDI logics and practical system implementations. Following [10] one reason is that the logical formalisms used to define the models do not have an operational model to support them. Recent work to overcome this problem can be separated in two major directions. One is to define BDI models using a suitable logical formalism that allows to represent mental states and that has operational procedures to use the logic as knowledge representation formalism. Mora et al., for example, defined the notions of belief, desires and intentions using extended logic programming (ELP), an extension of logic programming with a second, explicit negation. This allows the explicit representation of negative information based on a well founded semantics [10].

Another approach is to extend existing BDI logics with appropriate operational models so that agent theories become computational. Schild's representation of a BDI logic follows this approach [14]. Rao also took this approach in [13] where he defines a proof procedure for the propositional version of his BDI logic. He presented AgentSpeak(L), a formal agent description language for an already implemented MAS. Based on this reengineering effort a practical and useful combination of formal description and implementation framework has been achieved.

AgentSpeak(L) allows the specification of an agent in a declarative notation similar to horn clauses used in logic programming languages like PROLOG. An agent's behavior (i.e. plans) and its knowledge are completely and clearly described using this notation. Plans in AgentSpeak(L) refer to horn clauses that are triggered by an event. Rao identified the following differences to conventional horn clauses: there are different types of conditions that allow both, a goal directed and a data driven activation of plans. Plans are event based and context sensitive, i.e. the execution of a plan needs the activation by an event and the successful evaluation of additional context conditions. Finally, the execution of a plan can be suspended and an agent may execute more than one plan at a time. The description of AgentSpeak(L) is similar to the semantics of guarded horn clauses used in concurrent logic programming languages like Concurrent Prolog [15]. We used these clauses as base for the specification of CASA agents. Therefore we will begin the specification section with a brief description of the main concepts of Concurrent Prolog.

3 Specification of CASA agents

The specification defines a CASA agent by means of goals, plans, beliefs, actions, messages, and intentions. In this article, we focus our investigation on the CASA micro view, i.e. modelling and execution of agent plans. These plans are defined by rules with complex context-sensitive tests. The parallel, event-based evaluation of plans with priorities is defined by extended guarded horn clauses.

3.1 Guarded Horn Clauses

Concurrent Prolog is a concurrent logic programming language [15] and was developed by E. Shapiro during 1985-1989. It provides a process oriented semantics and was implemented as a subset (Flat Concurrent Prolog) on multi processor architectures. The language easily can handle infinite computations typical for the modelling of reactive systems. The language incorporates guarded-command indeterminism, data-flow like synchronization, and a commitment mechanisms. A Concurrent Prolog program is a finite set of *Guarded Horn Clauses* (GHC).

$$\underbrace{H}_{Head} \leftarrow \underbrace{G_1, G_2, ..., G_n}_{Guard} \mid \underbrace{B_1, B_2, ..., B_m}_{Body}$$

The operator '|' separates the guard from the body and is called *commit operator*. The head and body elements define processes with arguments. The components of the guard define test conditions related to constraints that the head's arguments should satisfy. Declaratively, the commit operator is read just like a conjunction: H is true if the G's and B's are true. The abstract computation model of Concurrent Prolog and its derivates is established by process interpretation of the goals forming the resolvent. The goals are regarded as an asynchronous *process network*. The concurrent processes communicate and synchronize via shared logical variables according to an asynchronous communication model. When several clauses are applicable at once, the commit operator '|' acts as a control primitive ensuring that clause selection is carried out in a mutual exclusive manner. The potential for parallelism offered by clause selection might be considered as some form of restricted OR-parallelism. The evaluation of a guard G_i may result in a complex computation if G_i is not unifiable with a fact but with the head of another GHC. Such *deep guards* relate to PROLOG's backtracking mechanism but can hardly be implemented efficiently on parallel architectures.

The concepts of concurrent logic languages received significant attention in the 80's and several parallel programming languages based on these ideas were implemented, e.g. FCP or Strand. Concurrent constraint programming languages and multi-paradigm programming languages like Oz or AKL inherited many ideas from these approaches. Such languages often build the implementation base for multi agent systems.

3.2 Basic Definitions of CASA Agents

The specification of CASA agents is based on AgentSpeak(L), Guarded Horn Clauses and several extensions. The complete specification [6] covers the definition of beliefs, goals, actions, messages, events, plans and agents. In the following we focus on the definition of plans and agents and will describe the other elements only briefly and informally.

Beliefs: Beliefs compare to facts in logic programming. If two facts $a(t)$ and $b(s)$ are processed sequentially this is written as $a(t); b(s)$. A concurrent execution is denoted as $a(t), b(s)$.

Goals: A goal $g(s)$ represents the states an agent wants to achieve in the future. There are two different types of goals. A test $?g(s)$ simply looks for a belief in the agent's knowledge base that unifies with the test. A deep goal $!g(s)$ is a goal in the PROLOG sense and needs the reduction of one or more strategies.

Actions, Messages, Events: An action $a(t)$ refers to a basic behavior block the agent can perform to modify its environment. Messages are special actions with a given structure including sender, receiver, type, and content. If an action from other agents, the system or the user can be perceived by an agent, this perceived action is denoted as an *event*.

Plans: The behavior of an agent is basically defined by specifying strategies (resp. plans). A plan P is formally defined as an *extended guarded horn clause* of form

$$P : H \leftarrow G_1 ... G_n \mid B_1 ... B_m(p).$$

in which H is the head, G_i are guards, B_j are predicates as body goals, and p defines a priority.

The head of a CASA strategy describes the event the agent must perceive in order to execute the plan. The guard elements may consist of any number of tests, goals and messages which are processed sequentially (separated by ";") or in parallel (separated by ","). Additionally, a priority (or weight) allows to choose between different applicable strategies. The priority (p) of a plan denotes its importance for the agent.

Based on the type of the guards different strategies are distinguished. If all guards are simple tests, the strategy is considered as *reactive*. If goals are also elements in the strategy's guards the strategy is *deliberative*, because the evaluation of the guard requires a speculative computation which evaluates other strategies in order to reduce the goal (multi level plans). If the context test also requires the communication between agents, the strategy is described as *communicative*. Actions are not allowed in guards. If multiple strategies can be applied reactive strategies take precedence over deliberative strategies. Communicative strategies have lowest priority. During run time, an agent can suspend executing strategies and resume suspended ones by using special operations for suspending / resuming.

3.3 Execution Cycle of a CASA Agent

A CASA agent is described by a tuple

$$(Bel, Plans, Msg, Act, RunTime(Evt, Int, AMsg, AAct, S_e, S_p, S_i))$$

$Bel, Plans, Msg, Act$ are sets with elements representing the agent's beliefs, plans, actions and messages. The structure $RunTime(...)$ describes a component that defines the state of execution in each agent at run time. This structure consists of a set of currently perceived events (Evt), a set of intentions (Int, i.e., strategies that are currently pursued by the agent), a subset of messages / actions ($AMsg$, $AAct$) that are processed in the current execution cycle and a number of selection functions (S_e, S_p, S_i).

The operational semantics of a CASA agent can be best described by means of an abstract interpreter as it is visually given in Figure 2. The interpreter manages the execution of all agent activities in an interpretation loop. The operation of the agent interpreter is controlled by three functions that control event selection, plan selection, and intention selection. By modifying their implementation, the system can be easily tailored the different operational semantics for various other applications.

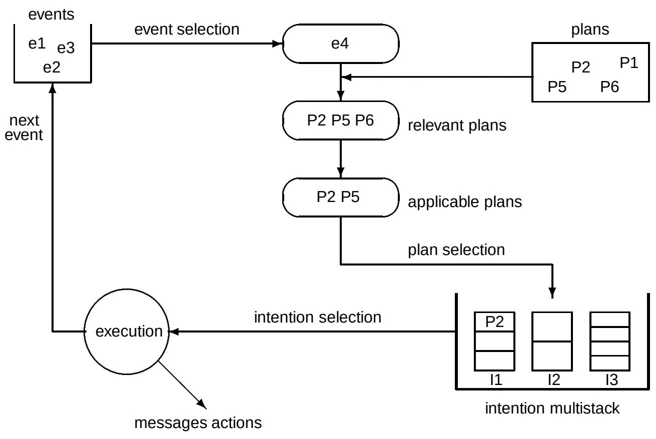

Fig. 2. CASA Execution Cycle

The interpretation starts with the selection of an incoming event. All perceived events are stored and the selection function S_e selects a suitable element. In a second step, a set of relevant plans which are appropriate for processing the selected event are identified. A *relevant plan* is defined by a plan which head matches the selected event. The preconditions of all relevant plans are checked against the facts / plans stored in the agent's beliefs base. This test can include the evaluation of other strategies, e.g. if a precondition is a goal of another strategy. This is realized by generating a corresponding event that is processed in one of the next execution cycles. A relevant plan which preconditions are all satisfiable is called an *applicable plan*. The agent uses the selection function S_p to select one applicable plan from the set of all applicable plans. This plan is processed as the pursued strategy and becomes instantiated as an intention on the multistack. If the event that triggered the plan was generated by the agent itself (as a consequence of a former execution cycle) the plan is pushed onto the intention stack that generated the event. If the event was generated from other agents, the environment or the user, the selected applicable plan is stored as a new intention on the multistack.

The multistack concept allows each agent to investigate several plans in parallel and to instantiate new (sub)intentions. Finally, the interpreter selects an intention by means of the selection function S_i from the multistack and executes it starting from the top element. Execution can result in either a direct action, the generation of an event, or the instantiation of new (sub)intentions. Thereafter, the interpreter advances to process the next event or continues to process the existing intentions on the multistack.

The interpreter and the basic definitions (formally described in [6]) served as base for the modelling phase that will be described in the next section.

3.4 CASA Specification Language

The CASA specification language is syntactically defined similarly to the agent language JAM [8], but the semantics differ with respect to the underlying CASA interpreter model. The language elements are briefly described in the following:

Basic Elements: Expressions in the CASA specification language refer to first-order terms of the formal CASA specification. An expression can be a variable, a constant (a string or a number), a function call (e.g. calculation or comparative operations), or a relation. A relation simply associates a list of data values with an identifier.

Facts: Facts correspond to the beliefs of the formal CASA specification. A fact is defined with the keyword FACT followed by a relation.

Goals: The two different types of goals are handled in the following ways: Syntactically a test is introduced by the keywords EXISTS FACT followed by a relation. The agent's knowledge base is checked if the specified fact with the given argument values exists and returns TRUE if that fact could be found. Deep goals are introduced by the keyword ACHIEVE followed by a relation and a priority value. The relation name specifies the goal to achieve and the relation arguments represent call-by-value-and-result parameters for relevant plans.

Actions: There are a number of pre-defined CASA actions available, e.g. for manipulating the agent's knowledge base or assigning values to variables. For any other action which is not pre-defined in CASA additional functions can easily be declared. Additional functions have to be introduced by the keyword EXECUTE followed by an identifier representing the action and a list of arguments.

Messages: Messages are special actions; they are inter-agent operations and therefore concern both the agent's micro and macro view. CASA has some pre-defined message types to specify messages like REQUEST, INFORM, or REPLY. These keywords are followed by a list of arguments declaring a message receiver, a message label, and the content of the message.

Plans: CASA plans correspond to extended guarded horn clauses described in the formal CASA specification. They are defined along the lines of the following patterns:

```
PLAN:
{ NAME:         <string>;
  DESCRIPTION:  <string>;
  GOAL:         ACHIEVE <relation>;
  TYPE:         <REACTIVE | DELIBERATIVE | COMMUNICATIVE>;
```

```
    PRECONDITION: <list of conditions>;
    BODY:         <list of actions>;
    FAILURE:      <list of actions>;
    PRIORITY:     <numeric value>;
  }
```

In addition to the five plan sections which directly refer to the components of extended guarded horn clauses we have defined a section for informally describing the plan, a section for specifying a plan type, and a failure section. The failure section is executed when errors occur during run time execution of the plan body. Only some actions are allowed in this section, e.g. assignments to variables in order to achieve a consistent state before dropping the plan. Due to a better readability the elements to execute in parallel are explicitly declared in a PARALLEL section (instead of using commas), while sequences are still simply separated by semicolons. Some additional pre-defined structures are available in plan bodies to support easy specification development, e.g. IF-THEN-ELSE or WHILE-loops.

With these language elements it is possible to specify a complete initial state of CASA agents. Each agent specification is composed of four main sections:

```
FUNCTIONS: EVENTSELECT:      <selection function>;
           PLANSELECT:       <selection function>;
           INTENTIONSELECT:  <selection function>;

GOALS:     ACHIEVE <relation> : <numeric value>;
           ...
FACTS:     FACT <relation>;
           ...
PLANS:     PLAN {...}
           ...
```

Three selection functions have to be declared in the first section of an agent specification. The selection functions depend on the agent application, i.e., agent developers provide these functions to their agents in a function library. Initial deep goals are defined in the second section. These goals will be instantiated – together with an applicable plan – in the multistack as separate intentions. Initial facts are simply listed in the third section and added to the agent's knowledge base. Lastly a set of relevant plans has to be defined in order to achieve the goals which are perceived as events during run time execution. The plans are stored in the agent's plan library.

3.5 Refined Operational Semantics and Speculative Calculations in CASA

We have refined the abstract interpreter, as it is desirable to distinguish between different types of perceived events (so far all events were processed as new goals). Events get an additional type tag, so that it can now be distinguished between new goals, new facts, new plans, suspend / resume events and replies. New facts and plans are directly inserted into the according components. Suspend / resume events and replies only have effect on single intentions and can therefore be directly passed to the multistack. Only for new goals it is necessary to find an applicable plan and instantiate a new (sub)intention. In the following we focus on the run time execution of new goals in CASA agents and present central aspects of modelling speculative calculations.

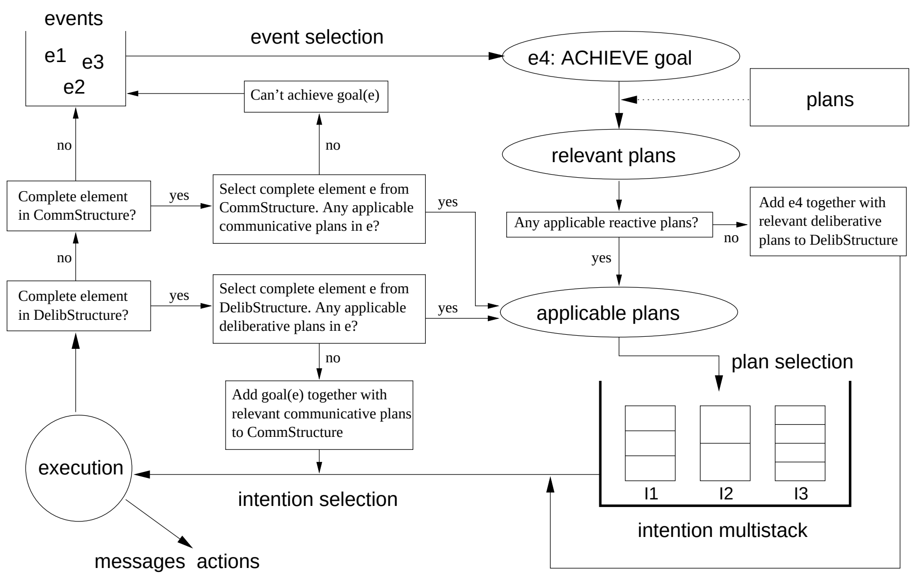
events
e1
e3
e2
event selection
e4: ACHIEVE goal
plans
Can't achieve goal(e)
no
no
relevant plans
Complete element in CommStructure?
yes
Select complete element e from CommStructure. Any applicable communicative plans in e?
yes
Any applicable reactive plans?
no
Add e4 together with relevant deliberative plans to DelibStructure
no
yes
Complete element in DelibStructure?
yes
Select complete element e from DelibStructure. Any applicable deliberative plans in e?
yes
applicable plans
no
plan selection
Add goal(e) together with relevant communicative plans to CommStructure
execution
intention selection
I1
I2
I3
intention multistack
messages actions

Fig. 3. Refined CASA Interpreter Cycle

Speculative calculations appear when a subgoal g in a precondition of a plan has to be tested (deep guards). In order to achieve g, a new goal event has to be generated and eventually some actions have to be performed. The interpreter model is refined with two independent components responsible for speculative calculations: Each element in the *DelibStructure* is composed of a goal event and all relevant deliberative plans to check. Similarly an element in the *CommStructure* is composed of a goal event and all relevant communicative plans. The operational semantics are best described by means of the cycle shown in Figure 3. When no applicable reactive plans are found for a new goal, a new element – composed of the goal and all relevant deliberative plans – is added to the DelibStructure. An element in the DelibStructure is said to be complete, when all its relevant plans are checked, i.e. the according plans' preconditions have been tested. When there is at least one applicable plan in such a complete element, the interpreter cycle is continued with plan selection and intention instantiation. Otherwise all relevant communicative plans have to be checked in the CommStructure. Finally, when even no applicable communicative plan can be found, the new goal can't be achieved and an exception has to be raised. It is important to notice that the DelibStructure, the CommStructure and the interpreter cycle operate independently and elements in the two additional components are usually not completed before several rounds of the interpreter cycle have passed.

3.6 Implementation

CASA's micro level view is implemented in Java with JDK 1.1.5 integrating modules of the JAM library [8]. A parser written in JavaCC reads CASA specification programs, sets the initial state of CASA agents and starts the execution on the CASA interpreter.

CASA agents are integrated into the MECCA framework [2]. MECCA is an agent management system which implements the FIPA ACL (Agent Communication Language) standard [4]. A CASA agent writes messages through a specific communication adaptor to the internal message transport channel of the MECCA system. This adaptor is also reading incoming messages from the transport channel, converting messages into appropriate CASA events and forwards them to the CASA agent. The adaptor allows CASA agents to communicate with any FIPA compliant agent via the MECCA framework as it is shown in Figure 4.

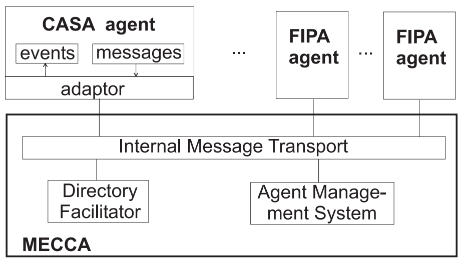

Fig. 4. CASA Agent connected to MECCA

3.7 A Sample CASA Agent

In this section we present a sample CASA agent which was built as part of a MAS that is simulating a color sorting assembly buffer (CSAB) for coloring car bodies. Each station and roboter of the CSAB is represented by an own agent. In this paper we focus on the tasks of a transport roboter. In the corresponding specification file we have listed the robot's strategies to achieve certain goals, e.g. getting an order to deliver, moving to the body depot, or determining the most appropriate destination station. The following CASA code fragment is describing the central part of the agent's specification:

```
FUNCTIONS: EVENTSELECT:     Event_OldestFirst;
           PLANSELECT:      Plan_HighestPriority;
           INTENTIONSELECT: Intention_HighestPriority;
GOALS:     ACHIEVE transport : 1.0;
FACTS:     ...
           FACT orderAgent "orderManager";
PLAN:
{ NAME:    "Master Plan";
  GOAL:    ACHIEVE transport;
  TYPE:    REACTIVE;
  BODY:    GETFACT myId $id;
           WHILE TEST (TRUE) {
             ...
             PARALLEL
             { ACHIEVE getOrder $nr $bodyType $color : 1.0; }
             { ACHIEVE gotoDepot : 1.0; }
             IF TEST (!= $nr "none") {
               PARALLEL
               { ACHIEVE loadRobot $bodyType $bodyNumber :1.0;}
               { ACHIEVE findDestination $color $destLine :1.0;}
               ... // success: move to $destLine and unload
               ... // else:    send order back
             }
           }
  PRIORITY:1.0;
}
```

The specification defines a top-level goal ”transport” and an appropriate master plan to achieve this goal. When the master plan is executed, several subgoals have to be achieved, some of them even in parallel, e.g. the subgoals ”getOrder” and ”gotoDepot”. Each time after this subgoal is achieved a parameter value is checked in order to find out whether an order can be transported or not. The following CASA code is a fragment of the possible plans to achieve the subgoal ”getOrder”:

```
PLAN:
{ NAME:         "taking local order";
  GOAL:         ACHIEVE getOrder $nr $bodyType $color;
  TYPE:         REACTIVE;
  PRECONDITION: GETFACT order $nr $bodyType $color;
  BODY:         EXECUTE printLine "taking order " $nr;
  FAILURE:      ASSIGN $nr "none";
  PRIORITY:     2.0;
}
PLAN:
{ NAME:         "requesting orderAgent";
  GOAL:         ACHIEVE getOrder $nr $bodyType $color;
  TYPE:         COMMUNICATIVE;
  PRECONDITION: GETFACT orderAgent $orderAgent;
                REQUEST $orderAgent "getOrder" : 1.0;
                GETREPLY $nr $bodyType $color;
  BODY:         EXECUTE printLine "received order " $nr;
  FAILURE:      ASSIGN $nr "none";
  PRIORITY:     6.0;
}
```

According to the CASA interpreter model the reactive plan is checked first. When there is no order in the agent’s knowledge base, the GETFACT precondition fails and the plan is not applicable. The communicative plan is only considered when the precondition of the reactive plan is false. When the communicative plan’s precondition is checked, the agent will get a new order from the ”order agent” on request within a reply message. After execution of one of the plans the variables specified in the ACHIEVE-action of the invoking intention are automatically updated, as they are interpreted as call-by-value-and-result parameters.

4 Summary and Outlook

We have described the design of the agent specification language CASA, its efficient realization and the integration in the agent management framework MECCA. CASA combines the BDI agent approach with concepts from concurrent logic programming and allows the design of complex agent strategies. Based on the elements of the CASA specification a description language was designed that allows to textually specify a multi agent system on a high abstraction level. This description is efficiently executed using the CASA programming environment. First examples include simple applications taken from holonic manufacturing [5]. Future work will concentrate on the development of visual tools for the design of CASA agents and the application in the area of flexible manufacturing systems and intelligent user interfaces.

References

1. Agent Web Pages: http://www.agentlink.org, http://www.cs.umbc.edu/agents
2. B. Bauer, D. Steiner. MECCA - System Reference Manual. (*Internal Documentation*) Siemens, Munich, 1998.
3. M. E. Bratman. Intentions., Plans, and Practical Reason. *Harvard University Press*, Cambridge, USA, 1987.
4. FIPA 97 Specification - Part 2: Agent Communication Language. *FIPA - Foundation for Intelligent Physical Agents*, Geneva, Switzerland, 1997.
5. S. Flake, Ch. Geiger, G. Lehrenfeld, W. Mueller, V. Paelke. Agent-Based Modeling for Holonic Manufacturing Systems with Fuzzy Control, NAFIPS'99, *18th International Conference of the North American Fuzzy Information* Processing Society, New York, USA, June 10-12, 1999.
6. C. Geiger. Rapid Prototyping of interactive 3D animations. *PhD Thesis*, Paderborn University, September 1998 (in German).
7. J. Kiniry, D. Zimmerman.A Hands-On Look at Java Mobile Agents. *IEEE Internet Computing*, Vol. 1, No. 4, July 1997.
8. J.M. Huber. JAM - A BDI-theoretic Mobile Agent Architecture. *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, Washington, USA, May 1-5, 1999.
9. J. Mueller et. al. Chapter Belief Desire Intention. *Intelligent Agents V. Agent Theories, Architectures, and Languages*. 5th International Workshop, ATAL 98, Paris, France, July 1998. Springer LNAI 1555.
10. M. Mora et. al. BDI Models and Systems: Bridging the GAP. *Intelligent Agents V. Agent Theories, Architectures, and Languages*. 5th International Workshop, ATAL 98, Paris, France, July 1998. Springer LNAI 1555.
11. H. Nwana, D. Ndumu. A Perspective on Software Agents Research. (To appear). *Knowledge Engineering Review*, Cambridge University Press, Cambridge, USA, 1999.
12. A.S. Rao, M.P. Georgeff. Modeling Rational Agents within BDI-Architecture. *Tech. Report 64*, Australian Artificial Intelligence Institute, Melbourne, Australia, February 1996.
13. A.S. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. *7th European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.
14. K. Schild. On the Relationship Between BDI Logics and Standard Logics of Concurrency. *Intelligent Agents V. Agent Theories, Architectures, and Languages*. 5th International Workshop, ATAL 98, Paris, France, July 1998. Springer LNAI 1555.
15. E. Shapiro. The Family of Concurrent Logic Programming Languages. *ACM Surveys*, 21(3) 1989.