A Formal Specification Language for MAS Engineering

Hong Zhu
School of Computing and Mathematical Sciences
Oxford Brookes University
Gipsy Lane, Headington, Oxford, OX3 0NW, UK
Email: hzhu@brookes.ac.uk

ABSTRACT

One of the most appealing features of multi-agent technology is its natural way to modularise a complex system in terms of multiple, interacting, autonomous components that have particular objectives to achieve. This paper presents a formal specification language SLAB that supports modular specification of multi-agent systems. The syntax and semantics of the language are defined. The style of formal specification of multi-agent systems is illustrated by an example.

Categories and Subject Descriptors
D.2.1 [**Requirements / Specifications**]: Languages - *formal specification language*; Methodologies - *agent-oriented methodology*. I.2.11 [**Distributed Artificial Intelligence**] Intelligent agents, Languages and structures, Multiagent systems.

General Terms
Languages

Keywords
Formal specification language, Multiagent Systems, Scenario, Software Agent, Caste

1. INTRODUCTION

Agent technology has long been predicted to be the next mainstream computing paradigm, see e.g. [1, 2, 3]. It is perceived to be a viable solution for large-scale industrial and commercial applications. However, researches on agent-based systems have been mainly an AI endeavour so far. The majority of extant agent applications are developed in an ad hoc fashion without proper analysis and specification of system's requirements, and without systematic verification and validation of the properties of the implemented system. For a long time, software engineers and computer scientists alike have learned from many incidents that the behaviours of a system should be understood and documented before the system is put in operation, even before a serious implementation effort is make. One of such incidents that is related to autonomous software agents in particular is the crash of Air France's Airbus 320 at an air show in June 1988 [4, 5]. Airbus 320 was the first fly-by-wire passenger aircraft in the world. In other words, an autonomous agent controls the aircraft. The incident was caused by a conflict between the human pilot's instruction and the autonomous control by the software. While the pilot intended to fly over the airport in the air show, the fly-by-wire control software seems to have instructed the aircraft to land, which was believed to be the cause of the accident. More than a dozen of years has passed and autonomous agents have gained much wider applications (see e.g. [6]), but open questions remain about how to specify autonomous agents' behaviour and how to verify and prove their properties so that such tragedy can be prevented. Being autonomous, proactive and adaptive, an agent-based system can be very complicated, and sometimes may demonstrate emergent behaviours, which are neither designed by the developers nor expected by the users of the system. The new features of agent-based systems demand new methods for the specification of agent behaviours and for the verification and validation of their properties to enable software engineers to develop reliable and trustworthy agent-based systems. It has been recognised that the lack of rigour is one of the major factors hampering the wide-scale adoption of agent technology [7].

The past few years have seen increasing research interests in agent-oriented software development methodology. Existing work can be classified into three main groups. The first is towards the theoretical foundations for specifying and modelling agent-based systems. Much work has been focused on modelling agents' rational behaviour by introducing modalities for belief, desire and intention, e.g. [8, 9, 10, 11]. Game theory has also found its position in the formalisation of agent models, e.g. [12]. A great number of formal models of agents have been proposed and investigated in the literature, see e.g. [13, 14]. Most of them are based on an internal mental state model of agents, yet some are based on a model of the external social behaviours of collaborative agents, e.g. [15]. As pointed out by Michael Fisher [16], a specification method based on a specific model of agents may result in the existence of certain agent theory and systems that do not match the concept in the specification formalism. Moreover, temporal logics, particularly when combined with modalities for belief, desire, etc., can be very complex. The second group of researches is on the development process and development methods for engineering agent-based systems, see e.g. [17, 18, 19, 20, 21]. These works mostly focused on diagrammatic notations that support the analysis and design of multi-agent systems. Some of the notations extend object-oriented methods and notations such as UML. Some introduce new models for agents and corresponding new diagrammatic notations as well. How such diagrammatic notations are related to the logic and formal models of agents remains as an open problem. The third group consists of the researches on the language facilities and features that support the formal specification and verification of agent-based systems in a software engineering context, although there is little such work reported in the literature [7, 22]. The use of existing formal specification languages, such as Z, has also been explored to specify agent architecture [23] and concepts [24]. Despite the large number of publications on agents in the literature, we lack researches on language facilities that support the development of large-scale complicated multi-agent systems. In particular, we lack language facilities to explicitly specify the environment of agents and agent-based systems although it is widely recognised that an important characteristic of agents is that they are entities situated (embedded) in a particular environment [25]. We lack facilities that can clearly state how agents' behaviours are related to the environment. The theme of this paper is to search for such language facilities to support the specification of agent-based systems in the context of software engineering. We present a formal specification language called SLAB [26, 27].

The remainder of the paper is organised as follows. Section 2 defines the syntax and formal semantics of the specification language SLAB. Section 3 illustrates SLAB's style of specification by an example. Section 4 is the conclusion.

2. SLAB's SYNTAX AND SEMANTICS

In this section, we present the syntax and semantics of the SLAB language. The meta-language to define the syntax is EBNF, which is given in Table 1. In a syntax definition, the meta-symbols are in bold font such as **::=** . Terminals are in *italic* font such as *Var*. Non-terminals are in normal font such as agent-description.

Table 1. The meta-symbols in EBNF

Name	Symbol	Means
Definition	**::=**	A **::=** B means that A is defined as B.
Concatenation		AB means that A is followed by B.
Optional	**[ ]**	**[** A **]** means that A is optional.
Choice	**|**	A **|** B means either A or B.
Repetition	**{ }**	**{** A **}** means that A may appear any times including zero times or more times.
Repetition with separator	**{ / }**	**{** A **/** B **}** means a sequence of A separated by B, where the number of A's can be zero or more. For example, A B A B A.
Positive repetition	**{ }**$^+$	**{** A **}** means that A may appear at least once.
Parenthesis	**( )**	They are used to change preference.

2.1 Agents, Castes and Multi-Agent Systems

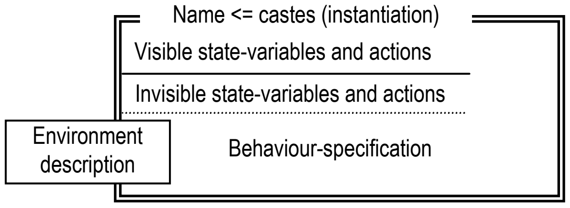
The specification of a multi-agent system in SLAB consists of a set of specifications of agents and castes.

```
    System ::= {Agent-description | caste-description}+
```

There is a most general caste, called AGENT, such that all castes in SLAB are sub-castes of AGENT. The main body of a caste specification in SLAB contains a description of the structure of its states and actions, a description of its behaviour, and a description of its environment. The following gives SLAB's syntax in EBNF of specifications of castes.

```
    caste-description ::=
        Caste name [ <= { caste-name / ,} ]
            [ instantiation ; ]
            [ environment-description ; ]
            [ structure-description ; ]
            [ behavior-description ; ]
        end name
```

It can also be equivalently represented as follows in a graphic form similar to the schema in Z [28].

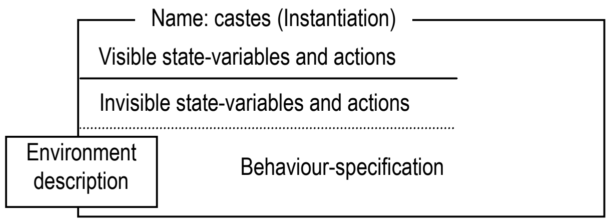

Name <= castes (instantiation)
Visible state-variables and actions
Invisible state-variables and actions
Environment description
Behaviour-specification

The clause 'Caste *New_Caste* <= $Caste_1$, $Caste_2$, ..., $Caste_n$' specifies that the defined caste *New_Caste* is a sub-caste of $Caste_1$, $Caste_2$, ..., $Caste_n$. That is, the defined caste inherits the structure, behaviour and environment descriptions of existing castes $Caste_1$, $Caste_2$, ..., $Caste_n$. When no inherited caste is given in a caste specification, it is by default a sub-caste of the predefined caste AGENT.

In SLAB, every agent must be an instance of a caste. When caste name(s) are given in an agent specification, the agent is an instance of the castes. If no caste name is given in an agent specification, the caste of the agent is by default AGENT. If an agent is an instance of a caste, it must have all the structural, behaviour and environment descriptions given in the caste's specification. Moreover, it may have additional structural, behaviour and environment descriptions to extend its state space, to enhance its ability to take actions and to widen its view of the environment. The following gives the syntax of agent specifications in SLAB.

```
    agent-description ::=
        agent name [ : { caste-name / , } ]
            [ instantiation ; ]
            [ environment-description; ]
            [ structure-description; ]
            [ behavior-description ]
        end name
```

It can also be equivalently represented as follows in a graphic form.


Name: castes (Instantiation)
Visible state-variables and actions
Invisible state-variables and actions
Environment description
Behaviour-specification

If an agent is specified as an instance of a caste, all the parameters in the specification of the caste must be instantiated in the specification of the agent.

Formally, we define that a *multi-agent system* consists of a finite set of *agents* $\{A_1, A_2, ...A_n\}$. These agents belong to a hierarchy of castes C_1, C_2, ...C_m. A binary relation $\prec$, called the inheritance relation, is defined on the castes C_1 and C_2 if C_1 is specified as a sub-caste of C_2. The inheritance relation is required to be a partial ordering on castes. Let $A \in C$ denote that agent A belongs to caste C. We also require that for all agents A and castes C and C',

$$A \in C \wedge C \prec C' \Rightarrow A \in C' . \qquad (1)$$

2.2 Environment

The SLAB language enables software engineers to explicitly specify the environment of an agent as a subset of the agents in the system that may influence its behaviour. The syntax for the description of environments is given below.

```
Environment-description ::=
     ENVIRONMENT { ( agent-name | All: caste-name
                          | variable : caste-name ) / , }+
```

where an agent name indicates a specific agent in the system. 'All' means that all the agents of the caste have influence on its behaviour. As a template of agents, a caste may have parameters. The variables specified in the form of "identifier: class-name" in the environment description are parameters. Such an identifier can be used as an agent name in the behaviour description of the caste. When instantiated, it indicates an agent in the caste. The instantiation clause gives the details about how the parameters are instantiated.

```
Instantiation ::= { variable := agent-name / , } +
```

Formally, we use $Env_A \subseteq \{A_1, A_2, ..., A_n\}$ to denote the environment of an agent A. Let "ENVIRONMENT EC_1, EC_2, ..., EC_K; be agent A's environment description. Then,

$$\text{EnvA} = [\![ENVRIONMENT \ \ EC_1, EC_2, ..., EC_K]\!] = \bigcup_{i=1}^{K} [\![EC_i]\!] ;$$

$$[\![agent]\!] = \{agent\} ;$$

$$[\![All : Caste]\!] = \{A_i \mid A_i \in Caste\} ;$$

$$[\![x : Caste]\!] = \{A_k\} ,$$

where "$x := A_k$" is the instantiation of variable x given in the specification of agent A.

2.3 State Space and Actions

In SLAB, the state space of an agent is described by a set of variables with keyword VAR. The set of actions is described by a set of identifiers with keyword ACTION. An action can have a number of parameters. An asterisk before the identifier indicates invisible variables and actions.

```
structure-description ::=
     [ Var { [ * ] identifier: type; }+ ]
     [ Action { [*] action-declaration / ; }+]
action-declaration ::= identifier
     | identifier ({ [ parameter:] type / , }+)
```

In a caste specification (and agent specification as well), the additional state variables and actions should have no overlap with the state variables, action identifiers and parameter variables defined in the super-castes. Moreover, the castes $Caste_1$, $Caste_2$, ..., $Caste_n$ that it inherits should have no common variables, no common action identifiers, and no common parameters. However, they can overlap with agent names in the environment descriptions.

Formally, each agent A has its own *state space*, which is a non-empty set S_A. Each state consists of two disjoint parts, the externally visible part and the internal part. The external part is visible for all agents in the system, while the internal part is not visible for any other agents in the system. Therefore, we have that $S_A = S_A^V \times S_A^I$, where S_A^V is the externally visible part of state space and S_A^I is the internal part of state space. An agent is capable of taking an action with various parameters at any particular time when it decides to do so. The set of actions is a finite non-empty set, denoted by Σ_A. An action can also be either externally visible or internal (hence externally invisible). We assume that an agent cannot take two actions at the same time. Thus, we have that $\Sigma_A = \Sigma_A^V \cup \Sigma_A^I$, where $\Sigma_A^V \cap \Sigma_A^I = \varnothing$, Σ_A^V is the subset of externally visible actions and Σ_A^I is the subset of internal actions.

2.4 Behaviour

Agents behave in real-time concurrently and autonomously. To capture the real-time features, an agent's behaviour is modelled by a set of sequences of events indexed by the time when the events happen.

2.4.1 Runs and Time

A *run* r of a multi-agent system is a mapping from time T to the set $\prod_{i=1}^{n} S_{A_i} \times \Sigma_{A_i}$. The behaviour of a multi-agent system is defined to be a set R of possible runs. Instead of defining a fixed set of time moments, the set of time moments are characterised by a collection of properties.

Definition 1.

Let T be a non-empty subset of real numbers. T is said to be a *time index set*, or simply the *time*, if

1) Bounded in the past, i.e. $\exists t_0 \in T . \forall t \in T . (t_0 \leq t)$; (2)

2) Unbounded in the future, i.e. $\forall r \in \mathrm{R} . \exists t \in T . (t > r)$; (3)

3) Uniformity, i.e. $\forall\ t_1, t_2, t_3 \in T.\ (t_2 > t_1 \Rightarrow t_3 + t_2 - t_1 \in T)$. (4)

The following lemma states that a time index set T can be characterised by two real numbers: the *start time* t_0 and the *time resolution* ρ, where $\rho \geq 0$.

Lemma 1.

For all subsets T of real numbers that satisfy properties (2), (3) and (4), we have that either $T=\{t_n \mid t_n = t_0 + n\rho, n=0, 1, 2, ... \}$ for some positive real number ρ, or $T=\{r \mid r \in R$ and $r \geq t_0\}$. In the former case, we say that the time index set T is *discrete*, and in the latter case, we say that T is continuous.

Proof.

Let $\rho(T) = inf\ \{t-s \mid s, t \in T \wedge t>s\}$. By using properties (2)~(4), it is easy to prove that when $\rho > 0$, $T=\{t_n \mid t_n = t_0 + n\rho, n=0, 1, 2, ... \}$. When $\rho=0$, $T=\{r \mid r \in R$ and $r \geq t_0\}$.

The real number $\rho(T)$ defined above in the proof is called the *resolution* of the time index set T.

On the other hand, it is easy to see that any discrete time index set of the form $T=\{t_n \mid t_n = t_0 + n\rho, n=0, 1, 2, ... \}$ satisfies the properties (2) ~ (4). Any subset $T=\{r \mid r \in R$ and $r \geq t_0\}$ of real numbers also satisfies the properties (2) ~ (4). Therefore, the model defined below applies to both discrete time index and continuous time index. Without loss of generality, subsequently, we assume that $t_0 = 0$.

For any given run r of the system, we say that a mapping h from T to $S_A \times \Sigma_A$ is the run of agent A in the context of r, if $\forall t \in T . h(t) = r_A(t)$, where $r_A(t)$ is the part of $r(t)$ in $S_A \times \Sigma_A$. Let r_A denote the run of agent A in the context of r, and $R_A = \{r_A \mid r \in R\}$ denote the behaviour of agent A in the system.

2.4.2 *Assumptions*

We assume a multi-agent system has the following properties.

Instantaneous actions. We assume that actions are instantaneous, i.e. they take no time to complete.

Silent moments. We assume that an agent can take no action at a time moment t. In such a case, we say that the agent is silent at time t. For the sake of convenience, we treat silence as a special action and use the symbol τ to denote silence. Therefore, we assume that for all agents A, $\tau \in \Sigma_A^V$.

Separatebility. We assume that the actions taken by an agent are separable, i.e. for all runs r, and all agents A, there exists a real number $\varepsilon_{r,A}>0$ such that $r_A^C(t) \neq \tau$ implies that for all $x \in T$, $t < x \le t + \varepsilon \Rightarrow r_A^C(x) = \tau$, where $r_A^C(t)$ denotes the action taken by agent A at time moment t in the run r. Consequently, an agent can take at most a countable number of non-silent actions in its lifetime.

Initial time and sleeping state. An agent can join the system at a time, say $t_{init,A}$, later than the system's start time. We say that the agent A is *sleeping* before time moment $t_{init,A}$. We use a special symbol $\perp \notin S_A$ to indicate such a state of an agent. Of course, we require that if an agent is sleeping, it will take no action but silence, i.e. $\forall t \in T . \left(r_A^S(t) = \perp \Rightarrow r_A^C(t) = \tau \right)$, where $r_A^S(t)$ denotes agent A's state at time moment t in the run r. The initial time $t_{init,A}$ of an agent A in a run r can be formally defined as the time moment $t \in T$ that $r_A^S(t) \neq \perp \wedge \forall t' \in T . (t' < t \Rightarrow r_A^S(t) = \perp)$.

2.4.3 *Agent's View of the Environment*

The global state of the system at any particular time moment belongs to the set $\prod_{i=1}^{n} S_{A_i} \times \Sigma_{A_i}$. However, each agent A can view the externally visible states and actions of the agents in $Env_A \subseteq \{A_1, A_2, ..., A_n\}$. In other words, an agent A can only view the part $\prod_{X \in Evn_A} S_X^V \times \Sigma_X^V$ of the state of the system. Agent A's view of the system state is defined as a mapping $View_A$ from global state $\prod_{i=1}^{n} S_{A_i} \times \Sigma_{A_i}$ to $\prod_{X \in Evn_A} S_X^V \times \Sigma_X^V$ as follows:

$$\begin{aligned} &View_A\left(\left\langle \langle s_1, s_1', c_1 \rangle, \langle s_2, s_2', c_2 \rangle, ..., \langle s_n, s_n', c_n \rangle \right\rangle\right) \\ &= \left\langle \langle s_{i_1}, c_1' \rangle, \langle s_{i_2}, c_2' \rangle, ..., \langle s_{i_k}, c'_{i_k} \rangle \right\rangle \end{aligned} \quad (5)$$

where $Env_A = \left\{ A_{i_1}, A_{i_2}, ..., A_{i_k} \right\}$, $i_u = v$ implies that $s_{i_u} = s_v$, $c_{i_u}' = c_v$ if $c_v \in \Sigma_{A_v}^V$, and $c_{i_u}' = \tau$ if $c_v \in \Sigma_{A_v}^I$. Because an agent's view is only a part of the system's global state, two different global states become equivalent from its view. The following formally defines the relation.

$$\forall x, y \in \prod_{i=1}^{n} S_{A_i} \times \Sigma_{A_i} . (x \approx_A y \Leftrightarrow View_A(x) = View_A(y)). \quad (6)$$

It is easy to see that the binary relation $\approx_A$ is an equivalence relation.

2.4.4 *Execution History*

Although an agent may not be able to distinguish two global states, the histories of the runs leading to states may be different. An intelligent agent may decide to take different actions according to the history rather than only depending on the visible global state. Let t be any given time moment. The *history* of a run r up to t, written as $r\downarrow t$, is a mapping that is the restriction of r to the subset $\{x \le t | x \in T\}$ of T. The history of a run up to t in the view of an agent A, denoted by $View_A(r\downarrow t)$, is the mapping from the subset $\{x \le t | x \in T\}$ of time moments to its views of the system's states in the run r. It can be defined as follows.

$$View_A\left(r \downarrow t\right)(u) = View_A(r(u)), \text{ for all } u \in T \text{ and } u \le t. \quad (7)$$

Similarly, we define $View_A(r)$ to be an agent A's view of a run r, and $View_A(r_B)$ to be agent A's view of agent B's behaviour in a run r. The equivalence relation defined on the state space can be extended to histories and runs as follows.

$$r_1 \approx_A r_2 \Leftrightarrow View_A(r_1) = View_A(r_2) \quad (8)$$

$$(r_1 \downarrow t) \approx_A (r_2 \downarrow t) \Leftrightarrow View_A(r_1 \downarrow t) = View_A(r_2 \downarrow t) \quad (9)$$

Before we finish this section, we introduce some further notation. Let A be any given agent in a multi-agent system. Let $c_1, ..., c_n, ... \in \Sigma_A - \{\tau\}$ be the sequence of non-silent actions taken by agent A in a run r and $t_1, t_2, ..., t_n, ... \in T$ are the time moments when the actions are taken place, i.e. $r_A^C(t_i) = c_i$ for all $i = 1, 2, ..., n, ...$. At a time moment $t \in T$, we say that c_n is agent A's current action, and c_{n+1} the *next* action, if $t_n \le t < t_{n+1}$. We write $Current(r_A \downarrow t) = <t_n, s_n, c_n>$ to denote that agent A's current action is c_n which was taken at time t_n and its state was s_n. Similarly, we write $Next(r_A \downarrow t) = <t_{n+1}, s_{n+1}, c_{n+1}>$ to denote the next action taken by agent A in a run r at time moment t is c_{n+1} at the time moment t_{n+1} with state s_{n+1}. We also write $Events(r_A \downarrow t) = <<t_1, s_1, c_1>, ..., <t_n, s_n, c_n>>$ to denote the sequence of events taken by agent A in the run r up to time moment t.

2.5 Specification of Behaviour

2.5.1 *Patterns of Behaviours*

A pattern describes the behaviour of an agent in the environment by a sequence of observable state changes and observable actions. A pattern is written in the form of $[p_1, p_2, ..., p_n]$ where $n \ge 0$. Table 2 gives the meanings of the patterns.

```
pattern ::= [ { event || [ constraint ] } ]
event ::= [ time-stamp: ] [ action ] [ ! state-assertion ]
action ::= atomic-pattern [ ^ arithmetic-expression ]
atomic-pattern ::= $ | ~
    | action-variable
    | action-identifier [ ( { arithmetic-expression } ) ]
time-stamp ::= arithmetic-expression
```

where a constraint is a first order predicate.

Table 2. Meanings of the patterns

Pattern	Meaning
\$	The *wild card*, which matches with all actions
~	The *silence* event
Action variable	It matches an action
P^k	A sequence of *k* events that match pattern *P*
! *Predicate*	The state of the agent satisfies the predicate
Act $(a_1, a_2, ...a_k)$	An action *Act* that takes place with parameters match $(a_1, a_2, ...a_k)$
$[p_1,..., p_n]$	The previous sequence of events match the patterns $p_1, ..., p_n$

Formally, Let *p* be a pattern. We write $B : r_A \downarrow t \models p$ to denote that from agent *B*'s viewpoint the behaviour of an agent *A* in a run *r* matches the pattern *p* at time moment *t*. The relationship |= can be defined inductively as follows.

Definition 2.

We write $B : r_A \downarrow t \models p$, if there is an assignment α such that $B : r_A \downarrow t \models_\alpha p$, which is inductively defined as follows.

- $B{:}r_A{\downarrow}t \models_\alpha$ [\$], for all agents *A*, *B*, runs *r* and time moments *t*;
- $B{:}r_A{\downarrow}t \models_\alpha [\tau]$, if $View_B(r_A{\downarrow}t)(t) = \tau$,
- $B{:}r_A{\downarrow}t \models_\alpha [x]$, if $Current(View_B(r_A{\downarrow}t)) = \alpha(x)$;
- $B{:}r_A{\downarrow}t \models_\alpha [\ t_x{:}\ C(e_1,\ e_2,\ ...,e_n)\ !\ pred(s)\ \|\ Constraint\ ]$, if $Current(View_B(r_A{\downarrow}t)) = \langle t_c,\ S,\ C(\alpha(e_1),\ \alpha(e_2),...,\alpha(e_n))\rangle$, *S* satisfies the predicate $\alpha(pred(s))$, $\alpha(t_x)=t_c$, and $\alpha(Constraint)$ is true.
- $B{:}r_A{\downarrow}t \models_\alpha [p\string^k]$, if $Events(View_B(r_A{\downarrow}t))=\langle ...., \langle t_1, s_1, c_1\rangle, \langle t_2, s_2, c_2\rangle, ..., \langle t_v, s_v, c_v\rangle\rangle$, where $v = \alpha(k)$, and for all *i*=1,2,..., *v*, $B{:}r_A{\downarrow}t_i \models_\alpha [p]$;
- $B{:}r_A{\downarrow}t \models_\alpha [p_1, p_2, ..., p_v]$, if $Events(View_B(r_A{\downarrow}t))=\langle ..., \langle t_1, s_1, c_1\rangle, \langle t_2, s_2, c_2\rangle, ..., \langle t_v, s_v, c_v\rangle\rangle$, and for all *i*=1,2,..., *v*, $B{:}r_A{\downarrow}t_i \models_\alpha [p_i]$.

Informally, $B{:}r_A{\downarrow}t \models_\alpha p$ means that agent *A*'s behaviour in a run *r* matches a pattern *p* at time moment *t* from an agent *B*'s point of view under assignment α. An assignment α for a set *X* of variables is a mapping that assigns values to variables in *X*.

2.5.2 Scenarios of Environment

The use of scenarios in agent oriented analysis and design has been proposed by a number of researchers, for example [29, 30, 31]. We define scenario as a set of typical combinations of the behaviours of related agents in the system. In addition to the pattern of individual agents' behaviour, SLAB also provides facilities to describe global situations of the whole system. The syntax of scenarios is given below.

```
Scenario ::=  Agent-Name : pattern
            | arithmetic-relation
            | ∃ [arithmetic-exp] Agent-Var ∈ Caste-Name: Pattern
            | ∀ Agent-Var ∈ Caste-Name: Pattern
            | Scenario & Scenario
            | Scenario ∨ Scenario
            | ~ Scenario
```

where an arithmetic relation can contain an expression in the form of μAgent-var∈ Caste.Pattern. It is an expression whose value is the number of agents in the caste whose behaviour matches the pattern.

```
arithmetic-relation ::= expression relational-op expression
expression ::=  atomic-expression
              | ( expression )
              | expression numerical-op expression
atomic-expression ::=  numerical-constants
              | numerical-variable
              | μ Agent-var∈ Caste.Pattern
relational-op ::= < | > | ≤ | ≥ | = | ≠
numerical-op ::= + | - | / | *
```

The semantics of scenario descriptions are given in Table 3.

Table 3. Semantics of scenario descriptions

Scenario	Meaning
A: P	The situation when agent A's behaviour matches pattern P
∀X∈C: P	The situation when the behaviours of all agents in caste C match pattern P
$\exists_{[m]}$X∈C: P	The situation when there exists at least m agents in caste C whose behaviour matches pattern P where the default value of the optional expression m is 1
μ X∈C: P	The number of agents in caste C whose behaviour matches pattern P
S_1 & S_2	The situation when both scenario S_1 and scenario S_2 are true
$S_1 \vee S_2$	The situation when either scenario S_1 or scenario S_2 or both are true
¬ S	The situation when scenario S is not true

The following are some examples of scenarios.

(1) ∃ p∈ Parties: t_{2000}: [nominate-president(Bush)] \|\| t_{2000}=(March/2000).

It describes the situation that at least one agent in the caste called Parties took the action nominate-president(Bush) at the time of March 2000.

(2) (μ x∈ Voter: [vote(Bush)] > μ x∈ Voter: [vote(Gore)])

It describes the situation that there are more agents in the caste Voter who took the action of vote(Bush) than those in the caste who took the action of vote(Gore).

Let *Sc* be a scenario. We write $A : r \downarrow t \models Sc$ to denote that from agent *A*'s point of view, the scenario *Sc* occurs at time moment *t* in a run *r*.

Definition 3.

From an agent *A*'s point of view, a scenario *Sc* occurs at time moment *t* in a run *r*, iff $A : r \downarrow t \models Sc$, which is inductively defined as follows.

- $A : r \downarrow t \models B : p \Leftrightarrow A : r_B \downarrow t \models p$; (10)
- $A : r \downarrow t \models Sc_1 \wedge Sc_2 \Leftrightarrow A : r \downarrow t \models Sc_1$ and $A : r \downarrow t \models Sc_2$; (11)
- $A : r \downarrow t \models \neg Sc \Leftrightarrow A : r \downarrow t \models Sc$ is not true ; (12)
- $A : r \downarrow t \models \forall x \in G.(x : Sc) \Leftrightarrow A : r_x \downarrow t \models Sc,$ for all $x \in G$; (13)
- $A : r \downarrow t \models \exists x \in G.(x : Sc) \Leftrightarrow A : r_x \downarrow t \models Sc,$ for some $x \in G$ (14)

2.5.3 Rules

In SLAB, an agent's behaviour is defined as its responses to environment scenarios. It is specified by a set of rules.

```
Behaviour-rule ::=
[<rule-name>] pattern|[ prob]–>event, [Scenario] [where pre-cond] ;
```

In a behaviour rule, the pattern on the left-hand-side of the –> symbol describes the pattern of the agent's previous behaviour. The scenario describes the situation in the environment, which specifies the behaviours of the agents in its environment. The where-clause is the pre-condition of the action to be taken by the agent. The event on the right-hand-side of –> symbol is the action to be taken when the scenario happens and if the pre-condition is satisfied. The agent may have a non-deterministic behaviour. The expression prob in a behaviour rule is an expression that defines the probability for the agent to take the specified action on the scenario. SLAB also allows specification of non-deterministic behaviour without giving the probability distribution. In such cases, the probability expression is omitted. It means that the probability is greater than 0 and less than 1.

Let *R* be the set of runs in a formal model of agent-based system. To define the semantics of rules, we first define a probabilistic space as follows.

For each scenario *Sc*, agent *A*, and constraint $Cn(r, t) \rightarrow \{tt, ff\}$, we define $R\lrcorner(Sc, A, Cn)$ as a subset of histories $H=\{r{\downarrow}t \mid r\in R, t\in T\}$ such that

$$R\lrcorner(Sc, A, Cn) = \{\, r{\downarrow}t \mid r\in R,\ t\in T,\ A : r \downarrow t \models Sc\ ,\ Cn(r, t) \,\} \qquad (15)$$

Let H^* be the set that contains *H* and all the subsets in the form of $R\lrcorner(Sc, A, Cn)$ and closed under set complementary, finite intersections and countable unions. Therefore, H^* constitutes a σ–field. A probabilistic space can then be constructed over the σ–field H^* by associating a probabilistic distribution over H^*. Let $Pr(R\lrcorner(Sc, A, Cn)$ be the probability that the scenario *Sc* with constraint *Cn* occurs from agent *A*'s point of view. Notice that agent *A*'s behaviour matches a pattern *p* can be expressed equivalently as a scenario $(A{:}p)_A$. The order pair <*R*, *Pr*> is called the probabilistic model of the agent-based system.

Definition 4.

Let R_A= 'p |(exp)→e if *Sc* where *Cn'* be a rule for agent *A*, where *Sc* is a scenario and *Cn* is a constraint. We say that in a probabilistic agent-based system <*R, Pr*> the agent *A*'s behaviours satisfy the rule R_A and write <*R*, *Pr*>: $A \models R_A$, if

$$Pr(R\lrcorner(A{:}p\#e, A, True) \mid R\lrcorner(Sc \wedge (A{:}p), A, Cn)\ ) = exp, \qquad (16)$$

where $p\#e = [p_1, p_2, ..., p_n, e]$, if $p = [p_1, p_2, ..., p_n]$.

3. EXAMPLE: ANTS

In this section, we give an example to illustrate SLAB's style of formal specification of multi-agent systems. For the sake of space, some details are omitted in this paper. More examples can be found in [26, 27].

The multi-agent system of ants is a typical example of reactive agent systems that may demonstrate emergent behaviours. In this system, food is scattered in the field, and ants try to find food and to move the food back to their home. When an ant walks across the field, its hormone spreads on the path in the field. The density of ant's hormone decreases as time ticks away. The food in the field decreases when taken by the ants.

```
Field
 VAR Hormone: N×N→N;  Food: N×N→N;
 ---------------------------------
All: Ants
 [$] |→ Hormone'(n,m) = Hormone(n,m)–1 ,
      if ¬∃x∈ Ants:[!position=(n,m)]  ;
 [$] |→ Hormone'(n,m) = Hormone(n,m)+100,
      if ∃x∈ Ants:[!position=(n,m)]
 [$] |→ Food'(n,m)=Food(n,m)–k,
      if ∃[k]x∈ Ants:[Take_Food ! position=(n,m)]
```

An ant can move around in the field searching for food. In such a search mode, the ant's movement is rather random. Once it has found some food, it takes a bite and carries it back home by tracing the hormone it left behind its path. Once back home with food, the ant comes back with other ants to the location to get more food. In the specification of ants, each ant can be in one of three mental states, which are *search*, *way_back* and *way_out*. Here, way_back is the state when an ant has found some food in the field and is carrying some food back home. An ant is in the state of way_out when it goes to the field to get food after an ant comes home with food. The following gives a specification of the *Ants*.

```
Ants
VAR      Position: N×N;
ACTION Move(Direction); Take_Food; Save_Food;
---------------------------------------------
VAR      State: {search, way_back, way_out};
- - - - - - - - - - - - - - - - - - - - - - -
Field; all: Ants
[Move(u) ! State=search]
     |(0.5)→  Move(u)!(position' = New(position, u)),
             if field: [!Food(position)<ε];
[Move(u) ! State=search]
     |(0.5)→  Move(v)!(position' = New(position, v)),
             if field:[!Food(position)<ε]; where u≠v
[$] |→   Take_Food !State = Way_back,
          if field: [!Food(position)≥ε];
[!State = Way_back & position ≠ home ] |→  Move(u),
     where u = arg Min'_{u∈Direction} field:
          Hormone(New(position, u));
[!State = Way_back & position = home ]
     |→ Save_Food ! State = Way_out;
[$] |(0.8)→ ! State = Way_out, if ∃x∈ Ants:[Save_Food]
[!State = Way_out] |→  Move(u),
     where u=arg Max_{u∈Direction} field:
             Hormone(New(position, u));
```

In the above specification, we defined *Home*, *Direction* and the function *New*(*position*, *u*) as follows.

```
| Home ≅ (0, 0)
| Direction ≅ {east, west, north, south}
| New: N×N× Direction → N×N
|----------------------------------------------
| New((n, m), east )    =   (n+1, m);
| New((n, m), west )    =   (n–1, m), if n > 0;
| New((n, m), north )   =   (n, m+1);
| New((n, m), south )   =   (n, m–1), if m > 0.
```

4. CONCLUSION

4.1 Summary

In this paper, we presented the formal specification language SLAB for multi-agent systems. The SLAB language integrates a number of novel language facilities that support the development of agent-based systems, especially the specification of such systems. Among these facilities, the notion of caste plays a crucial role. A caste represents a set of agents in a multi-agent system that have same capability of performing certain tasks and have same behaviour characteristics. Such common capability and behaviour can be the capability of speaking the same language, using the same ontology, following the same communication and/or collaboration protocols, and so on. Therefore, caste is a notion that generalises the notion of types in data type and the notion of classes in object-oriented paradigm. This notion is orthogonal to a number of notions proposed in agent-oriented methodology research, such as the notions of role, team and organisation, but it can be naturally used to specify or implement these notions. A caste can be the set of agents playing the same role in the system. However, agents of the same caste can also play different roles especially when agents form teams dynamically and determines its role at run time. Using the caste facility, a number of other facilities can be defined. For example, the environment of an agent can be described as the agents of certain castes and/or some particular agents. A global scenario of a multi-agent system can be described as the patterns of the behaviours of the agents of a certain caste. The example systems and features of agent-based systems specified in SLAB show that these facilities are powerful and useful for the formal specification of agents in various models and theories.

4.2 Related Work

The model of software agents used in this paper is closely related to the work by Lesperance *et al* [32], which also focused on the actions of agents. However, there are two significant differences. Firstly, they consider objects and agents are different types of entities, while we consider them as the same type of encapsulated computational entities. As a consequence of regarding objects as deferent entities from agents, they allow an agent to change the state of objects in the environment while we only allow an agent to effect its own state. Secondly, the most important difference is, of course, there is no notion of caste or any similar facility in their system.

The notion of groups of agents have been used in a number of researches on the multimodality logic of rationale agents, such as in Wooldridge's work [33], etc. However, such notion of groups of agents is significantly different from the notion of caste, because there is neither inheritance relationships between the groups, nor instance relationship between an agent and a group. Their only relationship is the membership relationship.

Many agent development systems are based on object-oriented programming. Hence, there is a natural form of castes as classes in OO paradigm. However, although agents can be regarded as evolved from object and caste as evolved from class, there are significant differences between agents and objects, and thus between caste and class. Therefore, the notion of caste deserves a new name.

4.3 Further Work

There are a number of open problems that need further investigation in the design of formal specification languages for multi-agent systems. For example, in SLAB an agent's membership to a caste is statically determined by agent description. Static membership has a number of advantages, especially its simplicity and easiness to prove the properties of agents. A question is whether we need a dynamic membership facility in order to specify and implement dynamic team formation. Examples have shown that introducing some state variables to represent the role that an agent is playing can specify dynamic team formation. An alternative approach to the problem of team formation is to define aggregate structures of agents and castes. An advantage of this approach is that the organisational structure of a team can be explicitly specified.

Another design decision that we faced in the design of SLAB was whether we should allow redefinition of behaviour rules in the specification of sub-castes. An advantage of disabling redefinition is that provable properties of a supper caste are inherited by all sub-caste.

Although the language facilities in SLAB, such as caste and scenario, were first introduced as a specification facility, we believe that they can be easily adopted in an agent-oriented programming language for the implementation of multi-agent systems. How to implement these facilities is an important issue in the design and implementation of agent-oriented programming languages. It also deserves further research. Moreover, although the design of SLAB is aimed to support as many agent-oriented methodologies as possible, how to link from such methodologies to formal specifications in SLAB deserve further investigation.

ACKNOWLEDGEMENT

The author is most grateful to his colleagues at Oxford Brookes University, especially Dr. Nick Wilson, Mr. Ken Brownsey, Ms. Sue Greenwood, Prof. David Duce, Mr. John Nealon, Mr. Qingning Huo, Mr. Yanglon Zhang, and Dr. Lu Zhang, Mr. Jiangrong Chen, et al., for their comments on earlier drafts of the paper and discussions on many related subjects. The author would also like to thank Dr. Huaglory Tianfield for many invaluable discussions.

REFERENCES

[1] Janca, P. C., *Pragmatic application of information agents*, BIS Strategic Report. 1995.

[2] Sargent, P., Back to school for a brand new ABC, The Guardian, 12 March, 1992, p28.

[3] Ovum Report, Intelligent Agents: The New Revolution in Software, 1994.

[4] Webster, P., Smith, M., Murtagh, P., Four Killed as Airbus Crashes, The Guardian, 27 June 1988, p1.

[5] ACM, The Risks Digest: Forum on Risks to the Public in Computers and Related Systems, Vol. 7, Issues 10~12, 27~30 June, 1988. (Available online at http://catless.ncl.ac.uk/Risks/ (7.10.html | 7.11.html | 7.12.html).

[6] Jennings, N. R., Wooldridge, M. J. (eds.), Agent Technology: Foundations, Applications, And Markets. Springer, Berlin, Heidelberg, New York, 1998.

[7] Brazier, F. M. T., Dunin-Keplicz, B. M., Jennings, N. R., Treur, J., DESIRE: Modelling Multi-Agent Systems in a

Compositional Formal Framework, in Int. Journal of Cooperative Information Systems, Vol. 1, No. 6, 1997, pp67~94.

[8] Rao, A. S., Georgreff, M. P., Modeling Rational Agents within A BDI-Architecture. in Proc. of the International Conference on Principles of Knowledge Representation and Reasoning, 1991, pp473~484.

[9] Singh, M. P., Semantical considerations on some primitives for agent specification, in Intelligent Agents: Agent Theories, Architectures, and Languages, Wooldridge, M., Muller, J. & Tambe, M. (eds), LNAI, Vol. 1037, Springer, 1996, pp49~64.

[10] Chainbi, W., Jmaiel, M., Abdelmajid, B. H., Conception, Behavioural Semantics and Formal Specification of Multi-Agent Systems, in Multi-Agent Systems: Theories, Languages, And Applications, 4th Australian Workshop on Distributed Artificial Intelligence Selected Papers, Bristane, QLD, Australia, July 1998. Zhang, C., Lukose, D. (eds), LNAI Vol. 1544, Springer, Berlin, Heidelberg, New York, 1998, pp16~28.

[11] Wooldrighe, M., Reasoning About Rational Agents, The MIT Press, 2000.

[12] Ambroszkiewicz, S. and Komar, J., A model of BDI-agent in game-theoretic framework, in [13], 1999, pp8~19.

[13] Myer, J-J., Schobbens, P-Y. (eds.), Formal Models of Agents - ESPRIT Project ModelAge Final Workshop Selected Papers, LNAI Vol. 1760, Springer, Berlin, Heidelberg, 1999.

[14] Wooldridge, M. J. and Jennings, N. R., Agent theories, architectures, and languages: A survey, in Intelligent Agents: Theories, Architectures, and Languages, LNAI Vol. 890, Springer-Verlag, 1995, pp1~32.

[15] Ossowski, S., and Garcia-Serrano, A., Social structure in artificial agent societies: implications for autonomous problem-solving agents, in Intelligent Agents V: Agent Theories, Architectures, and Languages, Muller, J. P., Singh, M. P. and Rao, A. S. (eds.), LNCS Vol. 1555, Springer, 1999, pp133~148.

[16] Fisher, M., If Z is the answer, what could the question possibly be? On developing formal methods for agent-based systems, in Intelligent Agents III: Agent Theories, Architectures, and Languages, Muller, J., Wooldridge, M., Jennings, N. (eds.), LNAI, Vol. 1193, Springer, 1997, pp65~66.

[17] Kinny, D., Georgeff, M., and Rao, A., A methodology and modelling technology for systems of BDI agents, in Agents Breaking Away: Proc. of MAAMAW'96, LNAI Vol. 1038, Spriger-Verlag, 1996.

[18] Moulin, B. and Cloutier, L., Collaborative work based on multiagent architectures: a methodological perspective, in Soft Computing; Fuzzy Logic, Neural Networks and Distributed Artificial Intelligence, Aminzadeh, F. and Jamshidi, M. (eds.), Prentice-Hall, 1994, pp261~296.

[19] Moulin, B., and Brassard, M., A scenario-based design method and an environment for the development of multiagent systems, in First Australian Workshop on Distributed Artificial Intelligence, Lukose, D. and Zhang C. (eds.), LNAI Vol. 1087, Springer-Verlag, 1996, pp216~231.

[20] Wooldridge, M., Jennings, N., and Kinny, D., A methodology for agent-oriented analysis and design, Proc. of ACM Third International Conference on Autonomous Agents, Seattle, WA, USA, 1999, pp69~76.

[21] Iglesias, C. A., Garijo, M. Gonzalez, J. C., A Survey of Agent-Oriented Methodologies. in Intelligent Agents V: Agent Theories, Architectures, and Languages, Muller, J. P., Singh, M. P., Rao, A., (eds.), LNAI Vol. 1555. Springer, Berlin, Heidelberg, New York, 1999, pp317~330.

[22] Conrad, S., Saake, G., Turker, C., Towards an Agent-Oriented Framework for Specification of Information Systems, in [13], 1999, pp57~73.

[23] D'Inverno, M., Kinny, D., Luck, M., and Wooldridge, M., A formal specification of dMARS, in Intelligent Agents IV: Agent Theories, Architectures, and Languages, Singh, M. P., Rao, A. Wooldridge, M. (eds.), LNAI Vol. 1365, Springer, 1998, pp155~176.

[24] Luck, M. and d'Inverno, M., A formal framework for agency and auotnomy, in Proc. of First International Conference on Multi-agent Systems, AAAI Press / MIT Press, 1995, pp254~260.

[25] Jennings, N. R., Agent-Oriented Software Engineering, in Multi-Agent System Engineering, Proceedings of 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Valencia, Spain, June/July 1999, Garijo, F. J., Boman, M. (eds.), LNAI Vol. 1647, Springer, Berlin, Heidelberg, New York, 1999, pp1~7.

[26] Zhu, H. Formal Specification of Agent Behaviour through Environment Scenarios, Proc. of NASA First Workshop on Formal Aspects of Agent-Based Systems, (to be published by Springer in LNCS, Also available as Technical Report, School of Computing and Mathematical Sciences, Oxford Brookes University, 2000).

[27] Zhu, H., SLAB: A Formal Specification Language for Agent-Based Systems, Technical Report, School of Computing and Mathematical Sciences, Oxford Brookes University, Feb. 2001. (Submitted to the International Journal of Software Engineering and Knowledge Engineering, Special issue on Multi-Agents and Mobile Agents)

[28] Spivey, J. M., The Z Notation: A Reference Manual, (2nd edition), Prentice Hall, 1992.

[29] Iglesias, C. A., Garijo, M. Gonzalez, J. C., A Survey of Agent-Oriented Methodologies. in Intelligent Agents V: Agent Theories, Architectures, and Languages, Muller, J. P., Singh, M. P., Rao, A., (eds.), LNAI Vol. 1555. Springer, Berlin, Heidelberg, New York, 1999, pp317~330.

[30] Iglesias, C. A., Garijo, M., Gonzalez, J. C., Velasco, J. R., Analysis And Design of Multiagent Systems Using MAS-Common KADS, in Intelligent Agents IV: Agent Theories, Architectures, and Languages, Singh, M. P., Rao, A., Wooldridge, M. J. (eds.), LNAI Vol. 1356, Springer, Berlin, Heidelberg, New York, 1998, pp313~327.

[31] Moulin, B. Brassard, M., A Scenario-Based Design Method And Environment for Developing Multi-Agent Systems, in Proc. of First Australian Workshop on DAI, Lukose, D., Zhang, C. (eds.), LNAI Vol. 1087, Springer Verlag, Berlin, Heidelberg, New York, 1996, pp216~231.

[32] Lesperance, Y., levesque, H. J., Lin, F., Marcu, D., Reiter, R. and Scherl, R., Foundations of logical approach to agent programming, in Intelligent Agents II, Eds. Wooldridge, M., Muller, J., and Tambe, M., LNAI, Vol. 1037, Springer-Verlag, 1996, pp331~346.

[33] Wooldrighe, M., Reasoning About Rational Agents, The MIT Press, 2000.