Developing Formal Specifications of MAS in SLABS

-- A Case Study of Evolutionary Multi-Agent Ecosystem

Hong Zhu

Dept of Computing, Oxford Brookes University
Wheatley Campus, Oxford, OX33 1HX, UK
Email: hzhu@brookes.ac.uk

Abstract. The recent years has seen a rapid growth of research interests in agent-oriented software development methodology. A great amount of work has been reported in the literature on formal models and logics of software agents. However, how to use such formalisms in the analysis and specification of agent-based systems remains as an open problem. Alternative approaches using semi-formal diagrammatic notations for the analysis and specification of agent-based software systems have emerged recently. Unfortunately, there is a big gap between the formal and informal approaches. This paper investigates a process and method of agent-oriented software analysis and specification that use a simple diagrammatic notation for the development of agent models and derive a formal specification of multi-agent systems with the help of such diagrammatic notations. The method is illustrated by an example of the evolutionary multi-agent ecosystem Amalthaea developed at MIT Media Lab.

1. Introduction

Agent technology is widely perceived to be a viable solution for large-scale industrial and commercial applications [1,2]. However, it has not been widely adopted by IT industry. It has been recognised that the lack of rigour is one of the major factors hampering the wide-scale adoption of agent technology [3].

Much work has been done on formal modelling of agents' rational behaviour by logic systems and game theories, c.f. [4, 5, 6, 7, 8, 9]. In [10,11], we proposed a formal specification language for agent-based systems called SLABS. The language is powerful enough to specify different types of agent-based systems [10, 11, 12]. However, developing formal specifications of multi-agent systems in SLABS and other formalisms as well is difficult due to the complexity of agent-based systems. On the other hand, in recent years, a large amount of research work has been reported in the literature about the development processes and methods for engineering agent-based systems, e.g. [13, 14, 15, 16, 17]. These works mostly utilise diagrammatic notations to support the analysis and design of multi-agent systems. How such diagrammatic notations are related to the logic and formal models of agents remains as an open problem.

In this paper, we investigate how descriptions of multi-agent systems in a simple diagrammatic notation can be used to derive formal specifications of multi-agent systems in SLABS. A process of analysis and specification of multi-agent systems is

proposed and illustrated by a case study of the evolutionary multi-agent ecosystem Amalthaea, which was developed in MIT's Media Lab [18]. The paper is organised as follows. Section 2 is a brief introduction to SLABS. Section 3 proposes a process model and the diagrammatic notation. Section 4 illustrates the process by an example. Section 5 is the conclusion of the paper.

2 Overview of SLABS language

SLABS is a model-based formal specification language designed for engineering multi-agent systems [10, 11]. This section briefly reviews the main features of the language.

2.1 The underlying model

In our model, agents are defined as encapsulations of data, operations and behaviours that situate in their designated environments. Here, data represents an agent's state. Operations are the actions that an agent can take. Behaviour is a collection of sequences of state changes and operations performed by the agent in the context of its environment. By encapsulation, we mean that an agent's state can only be changed by the agent itself. Moreover, an agent has its own rules that govern its behaviour in its designated environment. Constructively, agents are active computational entities with a structure comprising the following elements.

1. *Name*, which is the agent's identity.
2. *Environment description*, which indicates what the agent interacts with.
3. *State,* which consists of a set of variables and is divided into two parts: the visible state and internal state.
4. *Actions*, which are the atomic actions that the agent can take. Each action has a name and may have parameters.
5. *Behaviour rules*, which determine the behaviour of the agent.

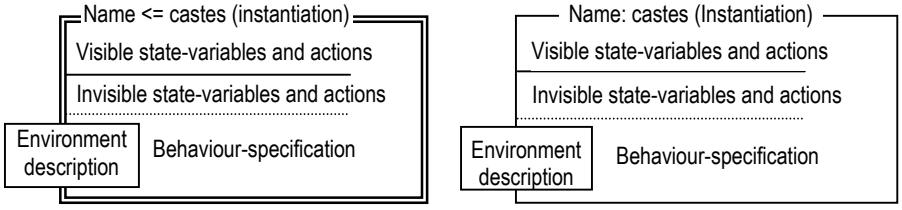
Agents constructively defined above have a number of features. First, they are autonomous in the sense of [17] and [19]. Second, they are communicative and social, yet it is independent of any particular agent communication language or protocol. Third, agents are situated in their designated environments. It requires an explicit and clear specification of the boundary and interface between an agent and its environment as well as the effects of the environment on the agent's behaviour. Fourth, as argued in [11], our definition implies that objects are special cases of agents in a degenerate form, while agents may be not objects. Finally, our model is independent of any particular model or theory of agents. We believe that specific agent models can be naturally defined in our model. In fact, using the SLABS language, we have formally specified examples of personal assistants [10], ants, learning agents [11], communication protocols [12], etc. In this paper, we will also demonstrate how an evolutionary multi-agent ecosystem can be formally specified in SLABS. A formal definition of the model can be found in [11].

There are two important implications of our model of agents and multi-agent systems. First, since agents are not objects, existing language facilities provided by

object-oriented languages cannot solve all the problems that software engineers face in developing agents [20]. Therefore, new language facilities must be introduced to support agent-oriented software development. Second, because agents are generalisations of objects, we believe that agent-orientation should be and can be a natural evolution of object-orientation so that the so-called agent-oriented paradigm can be built on the bases of object-oriented paradigm. In particular, the notion of caste is a natural evolution of the key notion of class in object-oriented paradigm. Here, a caste is a template of agents as class is a template of objects. Similarly, agents are instances of castes just as objects are instances of classes. The agents of a caste, thus, have common structural and behavioural characteristics. Castes also have inheritance relations between them. However, there are a number of significant differences between classes and castes; hence, they deserve a new name. Readers are referred to [12] for more details about the notion of caste and its role in the development of multi-agent systems.

2.2 The SLABS language

The specification of a multi-agent system in SLABS consists of a set of specifications of agents and castes. The main body of a caste specification in SLABS 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 the graphic form of specifications of castes and agents. Their syntax in EBNF can be found in [21].



Every agent must be an instance of a caste. It has 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. 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.

The SLABS 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. Environment description can be in three forms: (a) *an agent-name*, which means the agent is in its environment, (a) *All*: *caste-name*, which means all agents of the caste are in the environment, (3) *variable*: *caste-name*, which is a parameter of the caste. When the parameter is instantiated, it represents an agent in the environment.

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. 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. The global state of a multi-agent system at any particular time consists of the states and actions of all agents in the system. However, each agent *A* can only view the externally visible states and actions of the agents in its environment explicitly specified in its description. Because an agent's view is only a part of the global state, two different global states may become equivalent from its view. Although an agent may not be able to distinguish two global states, the histories of the runs leading to states may be different. The SLABS language provides language facilities to express an agent's view of the current state as well as the history of the run of the system so that intelligent behaviours such as learning and evolution can be easily specified. A pattern describes the behaviour of an agent in the environment by a sequence of observable state changes and observable actions. Table 1 gives the formats and the meanings of patterns.

Table 1. 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

SLABS also provides scenario description facilities to describe global situations of the whole system. The syntax and semantics of scenarios are given below.

Table 2. Semantics of scenario descriptions

Scenario	Meaning
A: P	The situation when agent A's behaviour matches pattern P
$\forall X \in C$: P	The situation when the behaviours of all agents in caste C match pattern P
$\exists_{[m]} X \in C$: 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
$\neg$ S	The situation when scenario S is not true

An agent's behaviour is defined by a set of rules as its responses to environment scenarios.

Behaviour-rule ::= **[**<rule-name>**]** pattern|**[** prob**]**–>event, **[***if* 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 are the behaviours of the agents in the environment. The where-

clause is the pre-condition of the action. 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. SLABS also allows the 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. For example, the following is a behaviour rule of search engine. It states that if there is an agent A in the environment that takes the action of calling the search engine with a set of keywords, it will return a set of urls that matches the keywords.

```
[$]|–> Search_Result(keywords, urls); if ∃A:[Search(Self, keywords)]
```

3 Proposed process and method

In this section, we propose a process for developing formal specifications of multi-agent systems and devise a simple diagrammatic notation to support the process. As shown in Fig 1, the process is an iteration of the following activities.

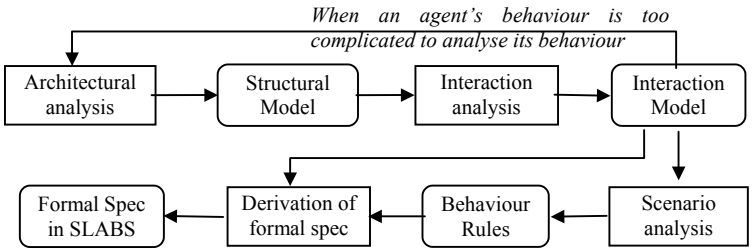


Fig. 1. Process of developing formal specifications in SLABS

Architectural analysis. The process starts with the architecture analysis of the multi-agent system. The main purpose of the analysis is to define the overall structure of the system by specifying the agents of the system and in the environment as well as their interrelationships. The main activity of the step is to identify the agents and castes and the relationships between the agents in terms of how agents influence each other. An architectural model of the system can be built and represented in a simple diagrammatic notation given in Fig 2.

There are two types of nodes in an agent diagram. An *agent node* represents an agent in the system. A *caste node* represents a set of agents in a caste. A link from node A to node B represents that the visible behaviour of agent/caste A is observed by agent/caste B. Therefore, agent/caste A influences agent/caste B. An agent may have an 'open end arrow' from a caste to an agent. It means all the agents in the caste may influence the agent. If an 'open end arrow' pointing to the agent connects to no caste, it means that all agents in the environment influents its behaviour.

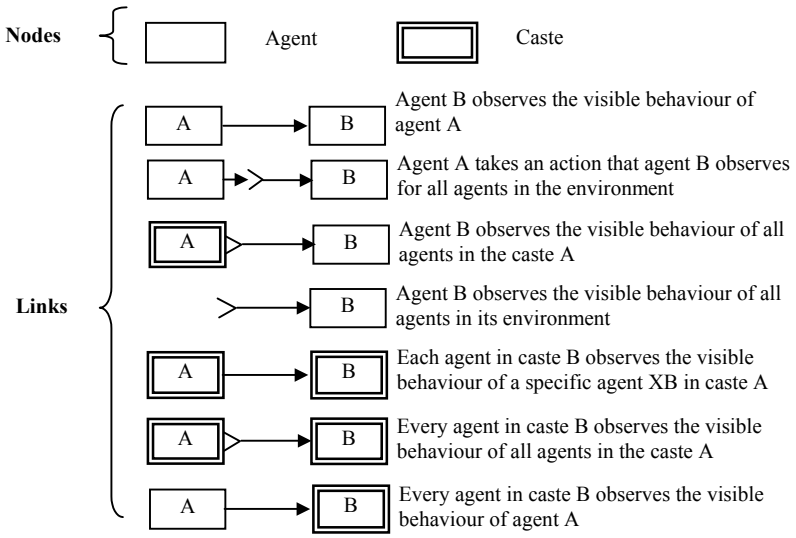


Fig. 2. Notation of agent diagram

Interaction analysis. Based on the result of architectural analysis, interaction analysis further identifies the visible actions and states that are observed by each agent or a caste of agents. The visible actions and states observed by an agent or a caste of agent is then associated with the link between the nodes in the agent diagram. The parameters of the actions and the data type of the state are also identified and annotated on the diagram. The meanings of the state variables and actions and their parameters should be recorded in a separate dictionary.

Scenario analysis. Scenario analysis is applied to each agent or each caste of agents to identify the typical scenarios that the agent will deal with and its designed behaviour in such a scenario. The result of scenario analysis is a set of behaviour rules that characterize the dynamic behaviour of the agent or the caste of agents.

Decomposition and refinement. When an agent's behaviour is too complicated to express in terms of the scenarios in the environment and the events that the agent responds to, the internal structure of the agent must be analysed. The architectural analysis is then applied to the agent so that it can be decomposed into a number of agents or sometimes just a number of state variables. Interaction analysis also follows to identify the interactions between internal components and the agents and/or castes in the environment that interact with the agent. Internal actions can also be introduced. A lower level diagram is drawn for the node that represents the agent. Fig 3 shows an example of lower level diagrams for a node AgentX, where agents E_1, and E_2 and caste C_1 are the agents and castes in the environment that interact with AgentX. Y_1 and Y_2 are the component agents internal to the AgentX.

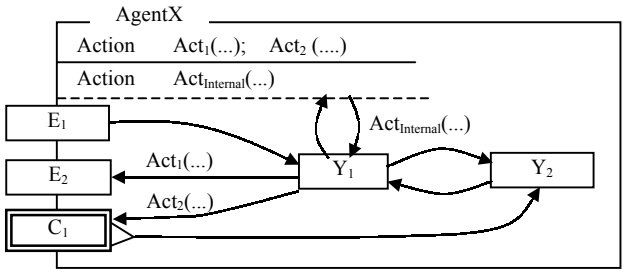


Fig. 3. Lower level agent diagram for a node

Derivation of formal specification. From a set of agent diagrams and a set of behaviour rules, a formal specification in SLABS of the multi-agent system can be derived. The topological structure of the systems determines the environment description of each agent/caste. The actions and states annotated on the links between the nodes determine the visible part of actions and states of the agent/caste. The lower level diagram provides the details of the internal state and actions. The rules are the descriptions of the behaviour.

4 Example: Specification of Amalthaea

Amalthaea is an evolutionary multi-agent ecosystem developed at MIT Media Laboratory [18]. Its main purpose was to assist its users in finding interesting information on the web. There are two species of agents in the system: *filtering agents* that model and monitor the interests of the user and *discovery agents* that model the information sources. These agents evolve, compete and collaborate in a market-like ecosystem. Agents that are useful to the user or other agents reproduce while low-performing agents are destroyed. The evolution of the system enables it to keep track of the changing interests of the user.

4.1 System's architecture

Amalthaea is composed of the *User Interface*, *The Ecosystem*, *the WWW search engines, WKV Generator* and a *Database* of the retrieved documents [18]. These components plus the user are the agents of the MAS system. The structure of the system can be represented in our diagrammatic notation as in Fig 4.

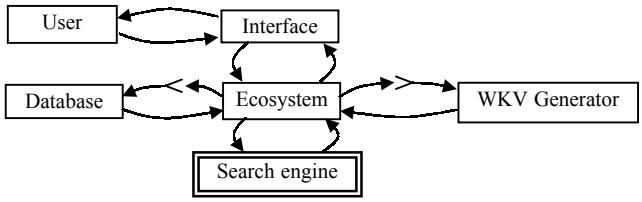


Fig. 4. Agent diagram of the multi-agent system Amalthaea

4.2 Interactions between the agents

The visible states and actions of the agents are determined by how information flows in the system. For example, in Amalthaea, the user browses the information presented on the interface and gives a rating for each item. The only visible action of the user is 'rate on a digest'. Notice that, in some systems that use agent technology for Internet browsing, whether the user browses a webpage and/or how long the user stays at a website are used to indicate whether the user is interested in the information. In such cases, browsing a webpage becomes a visible action.

The analysis of the interactions between the agents can be represented on the diagram by annotating the links with the actions that an agent / caste is interested in. Fig 5 is the result of the analysis of Amalthaea. Details can be found in [22].

The visible actions and states of the agents / castes can be derived directly from such a diagram. For example, the Interface should have two visible actions: *Pass_Rate*(*url, r*) and *Present_Digest*(*url*) according to the diagram. The former is observed by the ecosystem and the later is observed by the User.

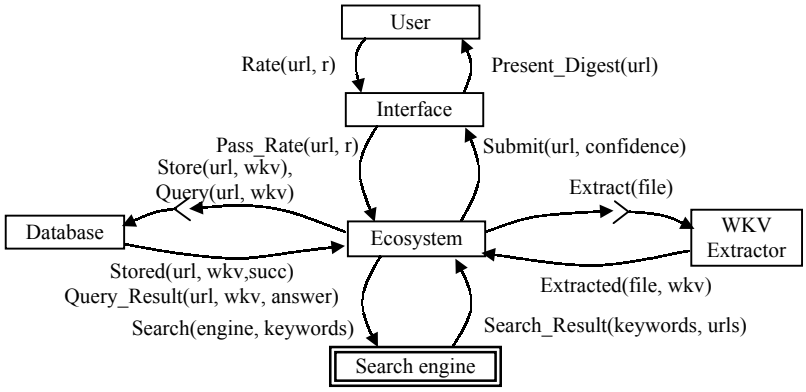


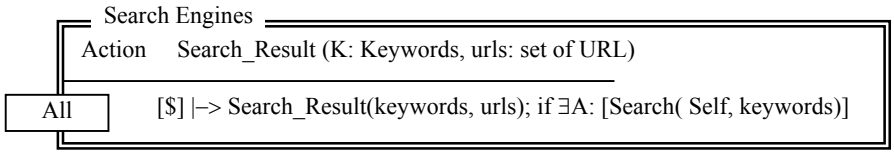
Fig. 5. Agent diagram with observable actions

4.3 Scenario analysis and the description of behaviour

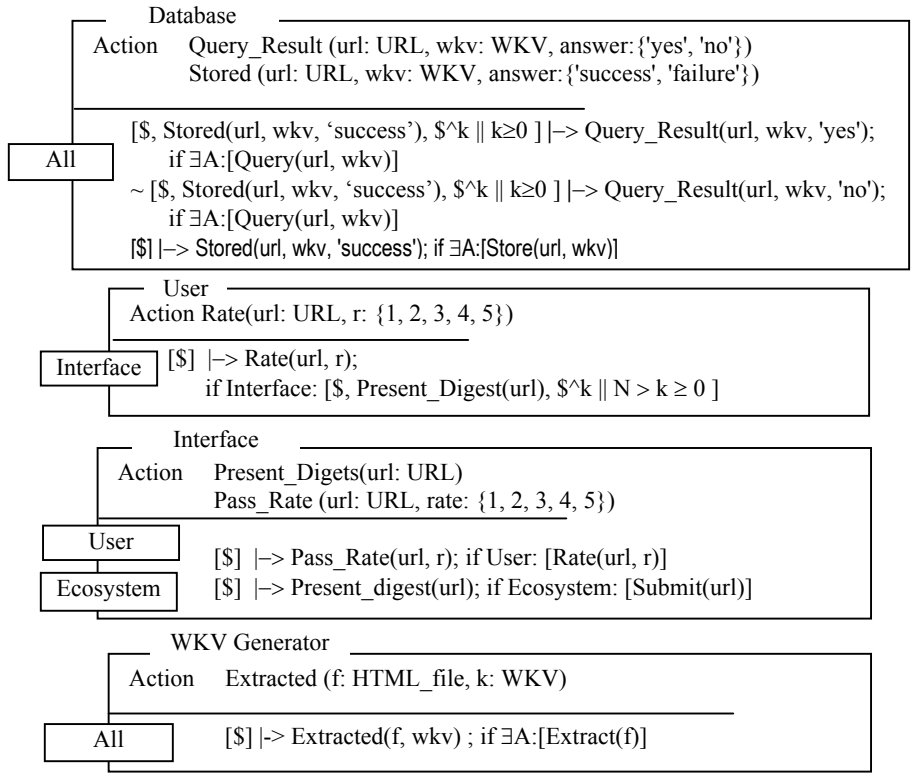
Agent and castes of simple behaviour can be easily described via scenario analysis. For example, the Search Engines in the Amalthaea system is a caste that consists of a number of search engines. All these search engines have a common behaviour. That is, whenever a search engine receives a search request, it performs the Internet search and returns a set of URLs as search results. This can be specified as follows.

```
[$] |–>[Search_Result(keywords, urls)]; if ∃A:[Search(Self, keywords)]
```

We can derive the following formal specification of the caste of Search Engines.



Similarly, we can obtain the following description of the Database, User, Interface and WKV generator.



4.4 Decomposition and analysis of internal structure

An agent may have complicated behaviour that cannot be described straightforwardly as above. The Ecosystem in Amalthaea is such an example. In such cases, it is inevitable to decompose the agent / caste and to analyse its internal structure. Such analysis, therefore, provides the information to specify the internal structure of the agent and enables scenario analysis and description of behaviour according to its internal structure. This refinement step follows the same process as described above and continues until the behaviours of all agents and castes can be clearly specified.

For example, the Ecosystem contains two types of agents: the *information filtering agents* and the *information discovery agents*. Therefore, we add two castes of agents as the internal components of the ecosystem. Hence, we have the following refined description of the Ecosystem in Fig 6.

Although the visible actions and states of the information filtering and discovery agents are clear now, their behaviours are still difficult to specify. The following further analyses the internal structures of these castes.

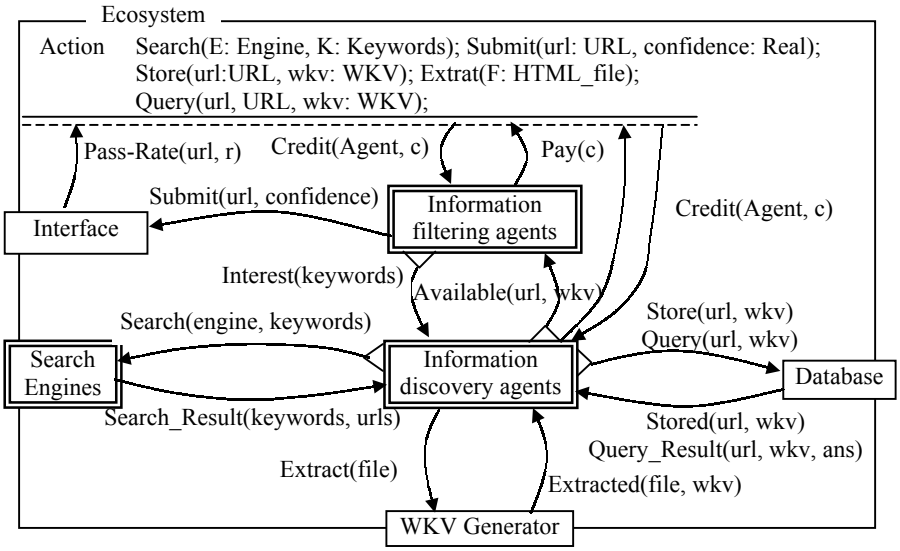


Fig. 6. Agent diagram of the Ecosystem

A. Information Filtering Agents (IFA) and Information Discovery Agents (IDAs)

Each information filtering agent only selects on a specific type of documents that is determined by the 'phenotype' of the agent. As shown in Fig 7, the phenotype consists of the following parts.

- *Fitness*: it is a positive integer value indicating how well the agent is performing. When the fitness is too low, it will be purged by the system.
- *Creation date*: it is the date that the agent was created.
- *User created?*: it is a tag indicating if the agent is created by the user or generated by the system during its evolution process. User generated agents are less easy to be purged by the system.
- *Genotype*: it is a weighted keyword vector. The agent only selects the document that has a weighted keyword vector close enough to the genotype.
- *Execution code*: it is the execution code of the agent.

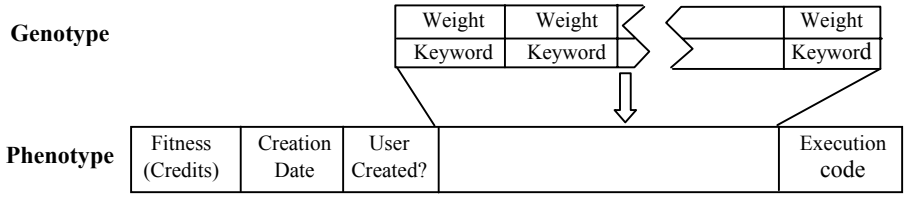


Fig. 7. The phenotype of information filtering agents

The structure of information filtering agents can be specified by the following caste in SLABS. In the specification, an information filtering agent has four internal state

variables: (1) the fitness, (2) creation date, (3) the tag for if it is user created, and (4) the genotype of the agent in the form of a weighted keyword vector.

```
Var  Fitness: Integer; CreationDate: Date; UserCreated: Bool;
     Genotype: WKV;
```

Notice that, the owner of an agent can access the internal state of the agent. Having specified the internal structure of the caste, we can now apply scenario analysis and describe the behaviour. Four scenarios are identified.

- *Scenario 1*: when the Ecosystem credits the agent.
- *Scenario 2*: when it is the time to pay the rent.
- *Scenario 3*: announce interests in a topic.
- *Scenario 4*: select url when the information about a topic is available after an information discovery agent retrieved from the Internet.

For each scenario, a rule is obtained to describe the behaviour of the agent in the scenario. For example, in scenario 1, when the Ecosystem takes an internal action of crediting the agent, the agent will increase the fitness by the amount of the credit assigned. Therefore, the rule is as follows.

[!(fitness=x)] |–> !(fitness = x+c); if Ecosystem: [credit(self, c)]

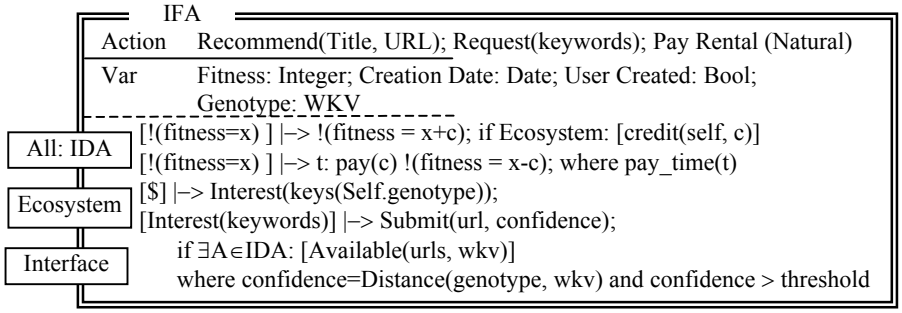
In scenario 2, when it is the time to pay the rent, the agent will take a visible action of pay and decrease the fitness by the amount of payment. Hence, we have the following behaviour rule.

[!(fitness=x)] |–> t: pay(c) !(fitness = x-c); where pay_time(t)

where pay_time is a predicate, which is true if the time t is for the agent to pay. Such concepts can be specified using schemas in Z. For example, the following schema specifies that the agents pay once in every 100 units of time.

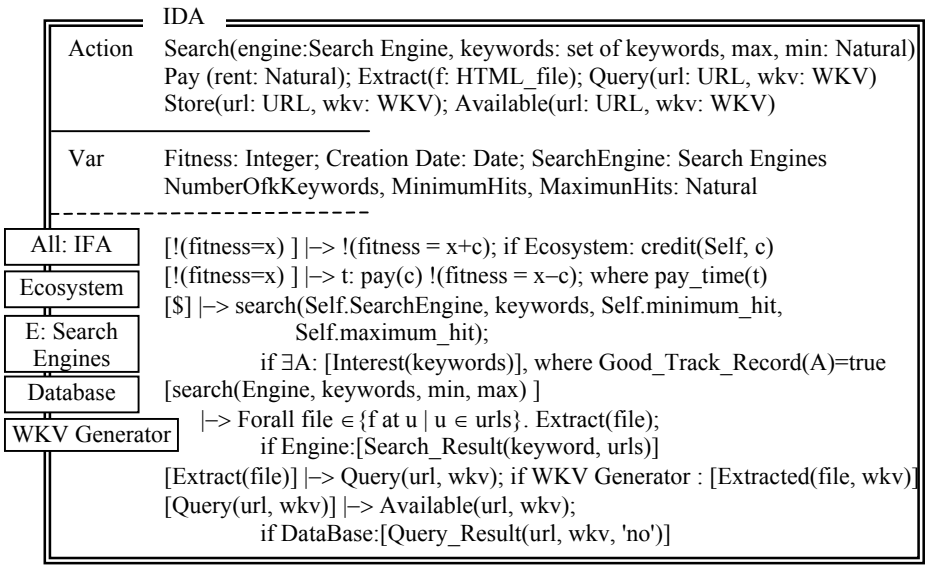
$$\begin{array}{|l} pay\_time: Time \rightarrow Bool \\ \hline pay\_time(t) = true;\ \ t \bmod 100 = 0 \\ pay\_time(t) = false;\ \ t \bmod 100 \neq 0 \end{array}$$

The complete specification of information filtering agents is given below.



The information discovery agents are responsible for posting queries to various Internet search engines, collecting the results and presenting them to the information filtering agents that requested them. Each information discovery agent also has a genotype that contains information on the keywords it should utilise when querying the search engine, along with the canonical URL of the engine (or information source) that it contacts. Depending on the search engine it uses, each IDA uses different query

types. Similar to information filtering agents, the caste IDA can be specified as follows.



B. Behaviour of the Ecosystem.

The interactions between filtering agents and discovery agents determine the global behaviour of the system. Amalthaea is a miniature economy model. The agents that constitute the Ecosystem operate under a penalty / reward scheme that is supported by the notion of credit. Each agent has fitness level expressed in the form of the accumulated amount of credit it has received. Credit, thus, serves as the fitness function of the agents. The higher the fitness of an agent, the more chances it gets to survive and produce offspring. In the analysis of the behaviour of the Ecosystem, the following scenarios are identified.

- *Scenario 1*: when the user's rating on a presented digest is passed to the Ecosystem through the interface.
- *Scenario 2*: when it is the time for the agents to pay.

In scenario 1, when credit is assigned by the user indirectly through feedback on the relevance of an item in the digest, the system relates this feedback to the filtering agents that proposed the item and the discovery agent that retrieved it and assigns the credit. We assume the function *Credit_IFA*: *Rate* × *Confidence* → *N* calculates the amount of the credit give to the information filtering agent and *Credit_IDA*: *Rate* × *Confidence* → *N* for information discovery agents. Then, the behaviour of the ecosystem in the scenario that a rating on a digest is received can be described as follows.

```
[$] |–> (Credit(A, c1), Credit(B, c2)) ;
    if Interface: [Pass_rate( url, r)] & A: [$, Submit(url, confidence), $^k]
        & B: [Available(url, wkv), $^k'],
    where c1=Credit_IFA(r, confidence) and c2=Credit_IDA(r)
```

The evolution of the Ecosystem is triggered by scenario 2. It depends on two

factors: the fitness of individual agent and the fitness of the whole system. The overall fitness is measured according to the percentage of positive feedbacks from the user in the past N ratings. Only a variable number of the best performed agents of the whole population are allowed to produce offspring, while a number of worst performed agents are purged. Assume that function *Number_of_Purges*(*N_rating*) calculates the number of agents to be purged from the most recent N ratings. After all agents have paid their rents, decisions are made on who will be purged and who will produce offspring according to the following rule.

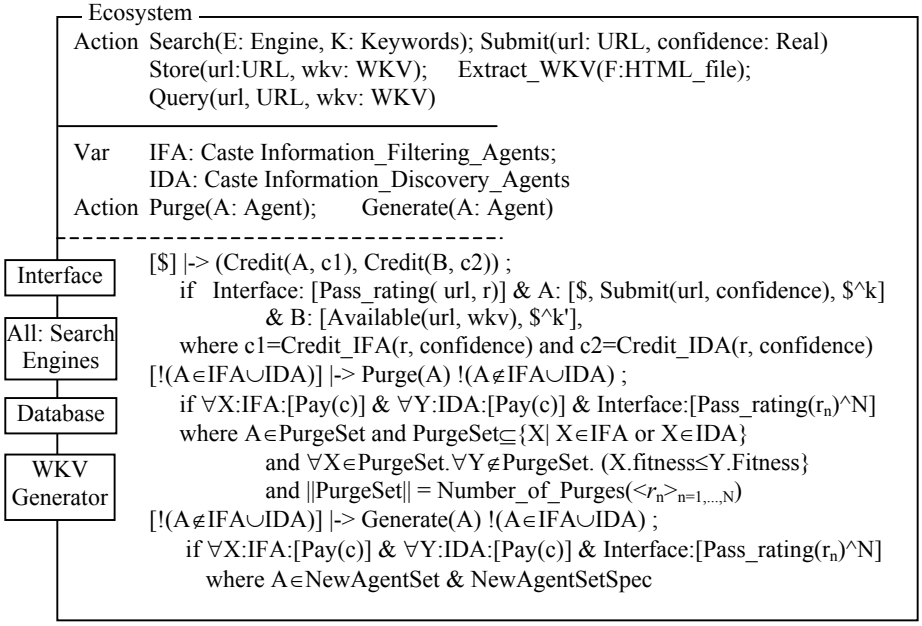
[!(A∈IFA∪IDA)] |–> Purge(A) !(A∉IFA∪IDA) ;
    if ∀X:IFA:[Pay(c)] and ∀Y:IDA:[Pay(c)] and Interface:[Pass_rating(r_n)^N]
    where A∈PurgeSet & PurgeSet⊆{X| X∈IFA or X∈IDA} &
∀X∈PurgeSet.∀Y∉PurgeSet. (X.fitness≤Y.Fitness} & ||PurgeSet|| = Number_of_Purges(<r_n>$_{n=1,...,N}$)

where *Purge* is an internal operation. Its function is to remove the agent from the system.

Assume that function *Number_of_Offspring*(*N_rating*) calculates the number of agents to be generated. Also, assume that *Offspring_Set* is the set of agents that are allowed to produce offspring, and the set *NewAgentSet* are the agents generated from the *OffspringSet* by applying mutation and crossover operations. The *NewAgentSet* can be specified similarly to the *PurgeSet*. The behaviour of producing offspring can be described by the following rule.

[!(A∉IFA∪IDA)] |–> Generate(A) !(A∈IFA∪IDA) ;
    if ∀X:IFA:[Pay(c)] & ∀Y:IDA:[Pay(c)] & Interface:[Pass_rating(r_n)^N], where A∈NewAgentSet

where *Generate* is an internal action of the Ecosystem. Its function is to add an agent to the system. Therefore, we have the following specification of the ecosystem.



This completes our specification of the Amalthaea system.

5 Conclusion

In this paper, we proposed a process model for developing formal specifications of multi-agent systems in SLABS. The process is an iteration of the following activities. Architectural analysis identifies the component agents / castes of a multi-agent system. Interaction analysis identifies the visible actions and states for each agent / caste. Scenario analysis identifies the rules that govern the dynamic behaviour of each agent /caste. Further decomposition and analysis of internal structures of an agent / caste are applied to the agents / castes if necessary. It continues until the behaviours of all agents can be clearly specified. A simple diagrammatic notation is devised based on our general model of multi-agent systems to support the development process. The development of a formal specification of an evolutionary multi-agent ecosystem Amalthaea, which was developed at MIT Media Lab, is presented in the paper to demonstrate the method proposed in this paper and to illustrate the style of formal specification in SLABS.

Existing work on agent-oriented software development methodology has been focused on process models for analysis and design of agent-based systems using diagrammatic notations. Little work has been reported that enable software engineers to use formal logic and other formalisms that have been investigated in the literature for agent technology. The method proposed in this paper naturally bridges the gap between informal and formal notations. We are further investigating how tools can be developed to automate the transformation from the diagrammatic notation to formal specification in SLABS in the way that structured requirements definitions are translated into formal specification in Z [23, 24].

References

1. Jennings, N.R., Wooldridge, M.J. (eds.): Agent Technology: Foundations, Applications, And Markets. Springer, Berlin Heidelberg New York (1998)
2. Huhns, M., Singh, M.P. (eds.): Readings in Agents. Morgan Kaufmann, San Francisco (1997)
3. Brazier, F.M.T., Dunin-Keplicz, B.M., Jennings, N.R., Treur, J.: DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework. Int. J. of Cooperative Information Systems 1(6) (1997) 67–94
4. 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) 473–484.
5. Singh, M.P.: Semantic Considerations on Some Primitives for Agent Specification. In: Wooldridge, M., Muller, J., Tambe, M. (eds): Intelligent Agents. LNAI, Vol. 1037. Springer (1996) 49–64
6. Wooldridge, M.: Reasoning About Rational Agents. The MIT Press (2000)
7. Ambroszkiewicz, S., Komar, J.: A Model of BDI-Agent in Game-Theoretic Framework. In: [8] (1999) 8–19
8. Myer, J-J., Schobbens, P-Y. (eds.): Formal Models of Agents - ESPRIT Project ModelAge Final Workshop Selected Papers. LNAI, Vol. 1760. Springer (1999)

9. Wooldridge, M.J. and Jennings, N.R.: Agent Theories, Architectures, and Languages: A Survey. In: Intelligent Agents. LNAI, Vol. 890. Springer-Verlag (1995) 1–32
10. Zhu, H.: Formal Specification of Agent Behaviour through Environment Scenarios. In: Proc. of FAABS 2000. LNCS, Vol. 1871. Springer 263–277
11. Zhu, H.: SLABS: A Formal Specification Language for Agent-Based Systems. Int. J. of Software Engineering and Knowledge Engineering 11(5) (2001) 529–558
12. Zhu, H.: The Role of Caste in Formal Specification of MAS. In: Proc. of PRIMA'2001. LNCS, 2132. Springer (2001) 1–15
13. Kinny, D., Georgeff, M., 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)
14. Moulin, B., Brassard, M.: A Scenario-Based Design Method and An Environment for the Development of Multiagent Systems. In: Lukose, D. and Zhang C. (eds.): First Australian Workshop on Distributed Artificial Intelligence. LNAI, Vol. 1087. Springer-Verlag (1996) 216–231
15. Wooldridge, M., Jennings, N., Kinny, D.: A Methodology for Agent-Oriented Analysis and Design. In: Proc. of ACM Third International Conference on Autonomous Agents, Seattle, WA, USA (1999) 69–76
16. Iglesias, C.A., Garijo, M., Gonzalez, J.C.: A Survey of Agent-Oriented Methodologies. In: Muller, J. P., Singh, M. P., Rao, A., (eds.): Intelligent Agents V. LNAI, Vol. 1555. Springer, Berlin (1999) 317–330
17. Bauer, B., Muller, J.P., and Odell, J.: Agent UML: a Formalism for Specifying Multiagent Software Systems. In: Ciancarini, P. and Wooldridge, M. (Eds.): Agent-Oriented Software Engineering. LNCS, Vol. 1957. Springer (2001) 91–103
18. Moukas, A.: Amalthaea: Information Discovery and Filtering Using a Multi-Agent Evolving Ecosystem. Journal of Applied Artificial Intelligence 11(5) (1997) 437–457
19. Jennings, N.R.: Agent-Oriented Software Engineering. In: Garijo, F.J., Boman, M. (eds.): Multi-Agent System Engineering, Proc. of 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Valencia, Spain (June/July 1999). LNAI, Vol. 1647. Springer, Berlin (1999) 1–7
20. Lange, D.B., Oshima, M.: Mobile Agents with Java: The Aglet API. World Wide Web Journal (1998)
21. Spivey, J.M.: The Z Notation: A Reference Manual. 2nd edn. Prentice Hall (1992)
22. Zhu, H.: Developing Formal Specifications of Multi-Agent Systems in SLABS, Technical Report CMS-02-01, School of Computing and Mathematical Sciences, Oxford Brookes University, UK (April 2002)
23. Jin, L., Zhu, H.: Automatic Generation of Formal Specification from Requirements Definition. In: Proc. of IEEE 1st Int. Conf. on Formal Engineering Methods, Hiroshima, Japan (1997) 243–251
24. Zhu, H., Jin, L.: Scenario Analysis in an Automated Tool for Requirements Engineering. J. of Requirements Engineering 5(1) (2000) 2~22