Formal Specification of Evolutionary Software Agents

Hong Zhu

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

Abstract. How to specify agent's intelligent behaviour is a challenging open problem in the development of agent-based systems. This paper presents a case study of developing the formal specification of the evolutionary multi-agent ecosystem Amalthaea developed at MIT Media Lab. A diagrammatic notation is used for the development of agent models and to derive a formal specification of the system in SLABS, which is a formal specification language for agent-based systems.

1 Introduction

Agent technology is widely perceived to be a viable solution for large-scale industrial and commercial applications in the Internet environment [1~4]. However, it has been recognised that the lack of rigour is one of the major factors hampering the wide-scale adoption of agent technology [5]. How to specify, test and verify the intelligent behaviours of agent-based systems remains an open problem.

Much work has been done on formal modelling of agents' rational behaviour by logic systems and game theories, c.f. [6~11]. On the other hand, research work has also been reported in the literature about the development processes and methods for engineering agent-based systems by utilising diagrammatic notations, e.g. [12~16]. Unfortunately, there is a big gap between these two approaches. 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.

The paper is organised as follows. Section 2 gives the background of the paper by a brief review of the formal specification language SLABS [17~19] and a methodology and a diagrammatic notation [20] for agent-oriented software system analysis, design and modelling. Section 3 presents the case study of an evolutionary multi-agent ecosystem called Amalthaea, which is developed in MIT's Media Lab [21]. Section 4 concludes the paper with discussions of related works and further work.

2 Review of the Language and Methodology

SLABS is a model-based formal specification language designed for engineering multi-agent systems [17, 18]. This section briefly reviews the main features of the language and a methodology of developing formal specifications in SLABS.

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.

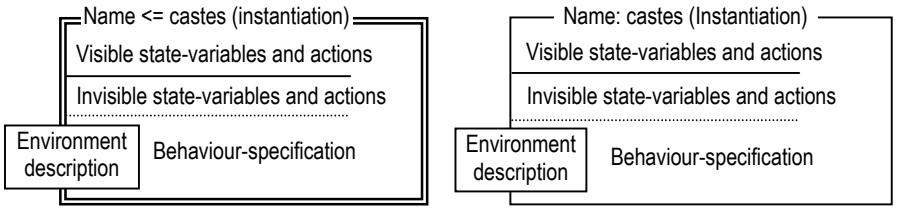
1. *Name*, which is the agent's identity.
2. *Environment description*, which indicates what the agent interacts with.
3. *State,* which consists of a set of variables and is divided into two parts: the visible state and internal state.
4. *Actions*, which are the atomic actions that the agent can take. 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 [22]. 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 [18], our definition implies that objects are special cases of agents in a degenerate form, while agents may be not objects. Finally, various agent models can be naturally defined in our model. Using the SLABS language, we have formally specified examples of personal assistants [17], ants, learning agents [18], communication protocols [19], 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 [18].

The notion of caste plays an important role in our model. It 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 [19] 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 an agent/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 [18].



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 indicates an 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 it is instantiated, it represents an agent in the environment.

Agents behave in real-time concurrently and autonomously. An agent's behaviour is an events sequence indexed by the time. 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, which may have some parameters. The global state of a multi-agent system at any time consists of the states and actions of all agents in the system. However, each agent 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 actions. Scenarios describe global situations of the whole system. Table 1 and 2 below give the formats and the meanings of patterns and scenarios, respectively.

An agent's behaviour is defined by a set of rules that describe its responses in various scenarios. A rule has the following structure.

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

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

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
$\mu X \in C$: 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

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 in 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 behaviour rule of search engines states that if there is an agent A in the environment that calls for search the Web 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)]
```

2.3 The Development Process

In [20], we proposed a process for developing formal specifications of multi-agent systems and devised a simple diagrammatic notation to support the process. As shown in Fig 1, the process is an iteration of the following activities.

- *Architectural analysis.* Its main purpose is to define the overall structure of the system. It is achieved by identifying the agents and castes and their inter-relationships in terms of how agents influence each other.

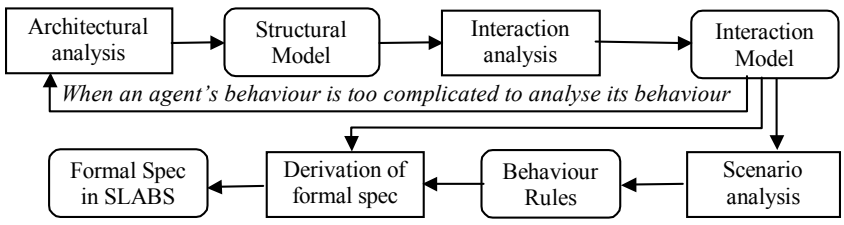


Fig. 1. Process of developing formal specifications in SLABS

- *Interaction analysis.* It aims to further clarify the interactions between the agents by identifying the visible actions and states of each agent or a caste of agents.
- *Scenario analysis.* It 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.
- *Iteration 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 is analysed. An iteration of the process of architectural analysis and interaction analysis starts and continues until scenario analysis can be successfully applied to the agents.
- *Derivation of formal specification.* The formal specification of the multi-agent system is derived based on the results of above analysis.

2.4 Diagrammatic Representation of Agent Models

An architectural model of a multi-agent system can be built and represented in a simple diagrammatic notation given in Fig 2.

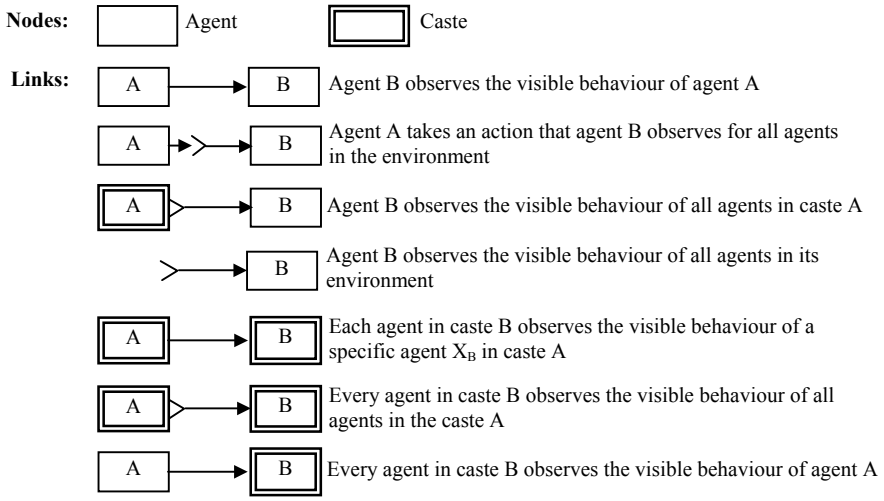


Fig. 2. Notation of agent diagram

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 that all the agents in the caste may influence the agent. If an 'open end arrow' that points to the agent connects to no caste, it means that all agents in the environment influents its behaviour.

An agent or caste in an agent diagram may itself be a complicated multi-agent system. In such a case, 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.

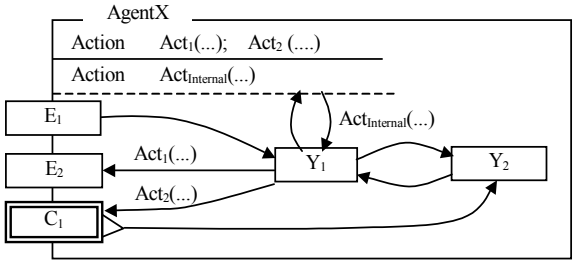


Fig. 3. Lower level agent diagram for a node

3 Case Study: Amalthaea

Amalthaea is an evolutionary multi-agent ecosystem developed at MIT Media Laboratory [21]. Its main purpose was to assist its users in finding interesting information on the web. There are two species of agents in the system: *information filtering agents* (IFA) that model and monitor the interests of the user, and *information discovery agents* (IDA) 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. In this section, we apply the methodology described in the previous section to develop a formal specification of Amalthaea in SLABS.

3.1 System's Architecture and Interactions Between the Agents

Amalthaea [21] is composed of the following components.

- *User Interface*, where the user is presented with the retrieved information and gives feedback on its relevance;
- *The Ecosystem*, which consists of IDA and IFA agents;
- *The WWW search engines* for retrieving documents;
- *WKV Generator*, which extracts keywords from retrieved documents and generates the weighted keyword vectors;
- *A Database* of the retrieved documents.

These components plus the user are the agents of system. The visible states and actions of the agents are determined by how information flows in the system. For example, 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'. 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 4 is the result of the architectural and interaction analysis of Amalthaea.

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.

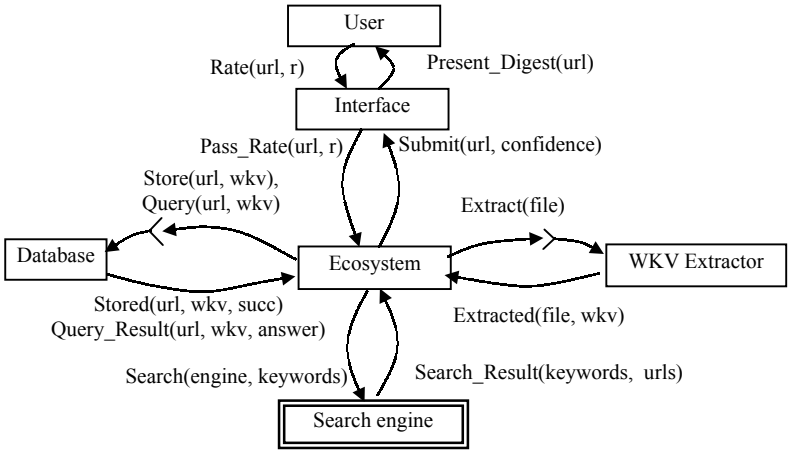


Fig. 4. Agent diagram with observable actions

3.2 Scenario Analysis and Description of Behaviour

Agent and castes of simple behaviour can be easily described via scenario analysis. In Amalthaea, the Search Engines is a caste that consists of a number of search engines, which have a common simple behaviour. 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 by the following rule.

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

Together with the information contained in Fig. 4, we can derive the following specification of the caste of Search Engines.

Search Engines

Action Search_Result (K:Keywords, urls: set of URL)

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

The Database, User, Interface and WKV generator are also simple agents. Their formal specifications can be derived similarly.

User

Action Rate(url: URL, r: {1, 2, 3, 4, 5})

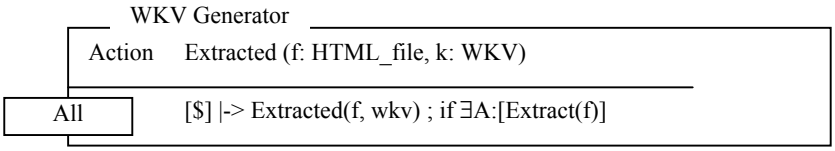
Interface [$] |–> Rate(url, r); if Interface: [$, Present_Digest(url), $^k || N>k≥0]

Interface

Action Present_Digets(url: URL)
Pass_Rate (url: URL, rate: {1, 2, 3, 4, 5})

User [$] |–> Pass_Rate(url, r); if User: [Rate(url, r)]

Ecosystem [$] |–> Present_digest(url); if Ecosystem: [Submit(url)]



3.3 Decomposition and Analysis of Internal Structure

The Ecosystem in Amalthaea has complicated behaviours that cannot be described straightforwardly as above. It is therefore decomposed and analysed for its internal structure. The Ecosystem contains two types of agents: *IFAs* and *IDAs*. IFAs select the information proposed by IDAs and submit it to the interface. IDAs search the Internet by interacting with the search engines, database and WKV generator. The Interface passes user's rating to the Ecosystem, which takes an internal action of assigning credits to the agents who discovered, selected and submitted the information to the Interface. All the agents in the Ecosystem must also pay a fixed amount of rent for each fixed period of time. Hence, we have the refined description of the Ecosystem in Fig 5.

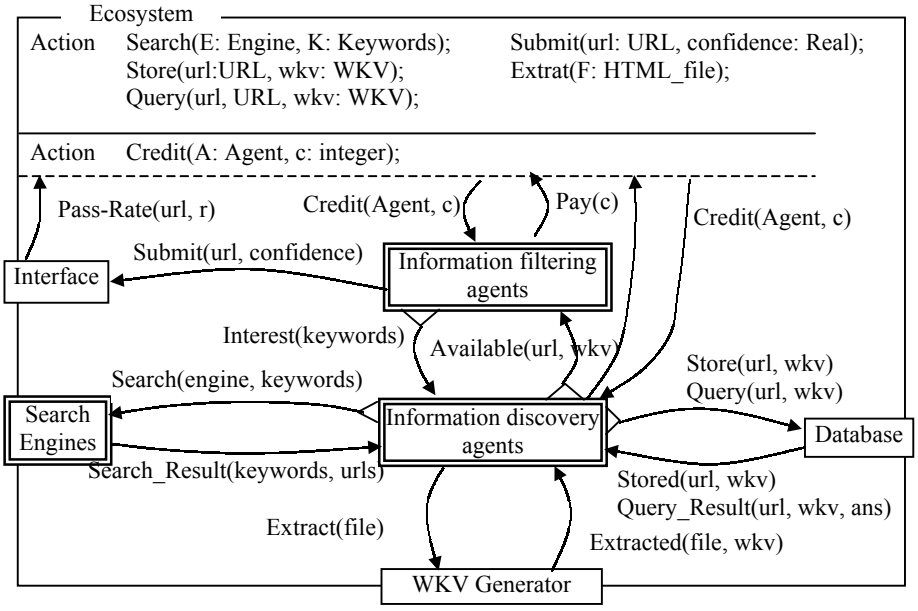


Fig. 5. Agent diagram of the Ecosystem

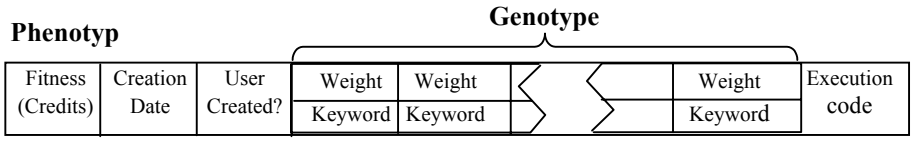


Fig. 6. The phenotype of information filtering agents

Each IFA only selects a specific type of documents that is determined by its 'phenotype'; see Fig 6. Therefore, an IFA has four internal state variables: (1) the fitness, (2) creation date, (3) the tag for if it is user created, and (4) the genotype in the form of a weighted keyword vector.

The following four scenarios trigger the behaviour of IFA.

- *Scenario 1*: when the Ecosystem credits the agent.
- *Scenario 2*: when it is the time to pay the rent.
- *Scenario 3*: when a user's interests in a topic are announced.
- *Scenario 4*: when an IDA retrieved some information.

For each scenario, the behaviour of an IFA can be specified by rules. For example, in Scenario 1, when the Ecosystem credits an agent, it increases 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)]

The specifications of IFAs and IDAs are given below.

Information Filtering Agents

Action Recommend(Title, URL); Request(keywords); Pay Rental (Natural)

Var Fitness: Integer; Creation Date: Date; User Created: Bool;
Genotype: WKV

All: IDA
[!(fitness=x)] |–> !(fitness = x+c); if Ecosystem: [credit(self, c)]
[!(fitness=x)] |–> t: pay(c) !(fitness = x-c); where pay_time(t)

Ecosystem
[$] |–> Interest(keys(Self.genotype));
[Interest(keywords)] |–> Submit(url, confidence);
if ∃A∈ IDA: [Available(urls, wkv)]
where confidence=Distance(genotype, wkv) and confidence > threshold

Interface

Information Discovery Agents

Action Search(engine:Search Engine, keywords: set of keywords, max, min: Natural)
Pay (rent: Natural); Extract(f: HTML_file); Query(url: URL, wkv: WKV)
Store(url: URL, wkv: WKV); Available(url: URL, wkv: WKV)

Var Fitness: Integer; Creation Date: Date; SearchEngine: Search Engines
NumberOfkKeywords, MinimumHits, MaximunHits: Natural

All: IFA
[!(fitness=x)] |–> !(fitness = x+c); if Ecosystem: credit(Self, c)
[!(fitness=x)] |–> t: pay(c) !(fitness = x–c); where pay_time(t)

Ecosystem
[$] |–> search(Self.SearchEngine, keywords, Self.minimum_hit,
Self.maximum_hit);
if ∃A: [Interest(keywords)], where Good_Track_Record(A)=true

E: Search Engines
[search(Engine, keywords, min, max)]
|–> Forall file ∈ {f at u | u ∈ urls}. Extract(file);
if Engine:[Search_Result(keyword, urls)]

Database
[Extract(file)] |–> Query(url, wkv); if WKV Generator : [Extracted(file, wkv)]

WKV Generator
[Query(url, wkv)] |–> Available(url, wkv);
if DataBase:[Query_Result(url, wkv, 'no')]

The credit of an agent serves as the fitness function. The higher the fitness of an agent, the more chances it has 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.
- *Scenario 2*: when it is the time for the agents to pay.

In scenario 1, the system credits the IFA that proposed the item and the IDA that retrieved it. Let *Credit_IFA*:*Rate*×*Confidence*→*N* and *Credit_IDA*: *Rate*×*Confidence* →*N* be functions that calculate the amount of the credit to be given to the IFA and IDA, respectively. Then, we have the following rule.

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

In scenario 2, the Ecosystem evolves by purging bad agents and producing new offspring. The overall fitness is measured according to the percentage of positive feedbacks from the user in the past *N* ratings. It decides the numbers of agents to be purged and new agents to be produced. Only the best agents of the whole population are allowed to produce offspring, while the worst are purged. Let *NPurges* (*ratings*) be the number of agents to be purged. We have the following rule.

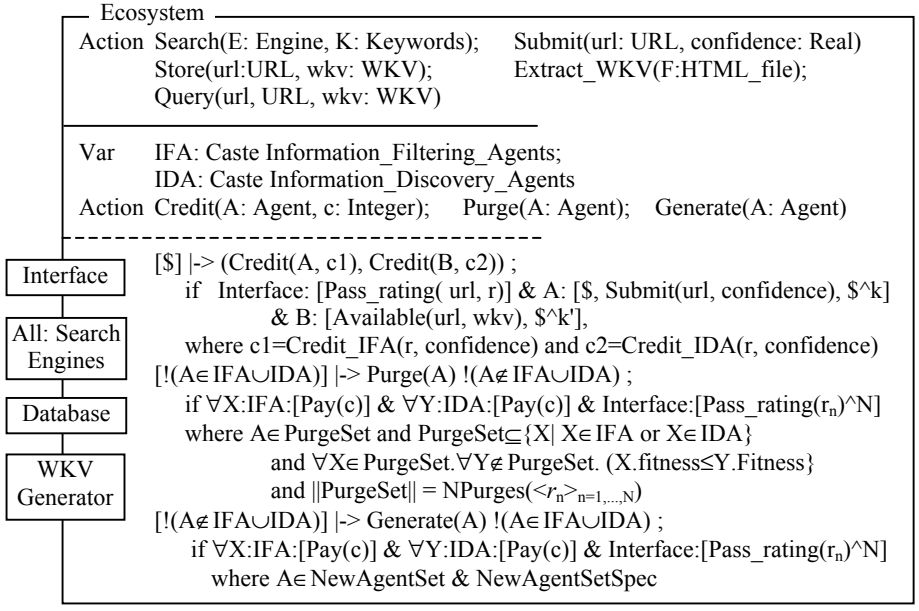
[!(A∈ IFA∪IDA)] |–> Purge(A) !(A∉ IFA∪IDA) ;
if ∀X:IFA:[Pay(c)] and ∀Y:IDA:[Pay(c)] and Interface:[Pass_rating(r_n)^N]
where A∈ PurgeSet & PurgeSet⊆{X| X∈ IFA or X∈ IDA} &
∀X∈ PurgeSet.∀Y∉ PurgeSet. (X.fitness≤Y.Fitness} & ||PurgeSet|| = NPurges(<r_n>$_{n=1,...,N}$)

where *Purge* is an internal action that removes an agent from the system.

Similarly, we have the following rule for producing offspring.

[!(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 that adds an agent to the system. Therefore, we have the following specification of the ecosystem.



This completes the specification of Amalthaea system.

4 Conclusion

In this paper, we presented a case study of developing a formal specification of a non-trivial evolutionary multi-agent system, which has a number of attractive features of its own. The result of the case study was satisfactory. A complete formal specification of the system is obtained with a reasonable effort through a smooth process.

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. Few complete formal specifications of non-trivial multi-agent systems are reported in the literature. It is unclear about how to use formal modal logic systems of BDI such as [8] and game theories [9] to express the behaviours of evolutionary ecosystems, which involves purging existing agents and producing offspring of existing agents.

The case study clearly demonstrated a number of advantages of the language SLABS and the methodology for developing formal specifications in SLABS in comparison with formal specification in general purpose languages such as Z [23~25]. Firstly, the language is expressive and suitable for the formal specification of multi-agent systems. The evolution process as well as other intelligent behaviour of the system can be clearly and naturally described. Second, the model of the system represented in the diagrammatic notation greatly helped to understand the system's behaviour, especially how agents communicate with each other. This model can be naturally transformed into the overall structure of the formal specification, which, from our previous experience, is one of the most difficult tasks in developing formal specifications without such a model. Third, the process 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 specifications in Z [26, 27]. We are also investigating how scenarios analysis can be graphically represented in a diagrammatic notation.

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. Jennings, N. R.: On agent-based software engineering. Artificial Intelligence 117, (2000) 277~296.
4. Lange, D. B.: Mobile Objects and mobile agents: The future of distributed computing? In: Proc. of The European Conference on Object-Oriented Programming, (1998)
5. 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

6. 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.
7. 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
8. Wooldridge, M.: Reasoning About Rational Agents. The MIT Press (2000)
9. Ambroszkiewicz, S., Komar, J.: A Model of BDI-Agent in Game-Theoretic Framework. In: [10] (1999) 8~19
10. Myer, J-J., Schobbens, P-Y. (eds.): Formal Models of Agents - ESPRIT Project ModelAge Final Workshop Selected Papers. LNAI, Vol. 1760. Springer (1999)
11. 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
12. 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)
13. 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
14. 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
15. 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
16. 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
17. Zhu, H.: Formal Specification of Agent Behaviour through Environment Scenarios. In: Proc. of FAABS 2000. LNCS, Vol. 1871. Springer (2001) 263~277
18. Zhu, H.: SLABS: A Formal Specification Language for Agent-Based Systems. Int. J. of Software Engineering and Knowledge Engineering 11(5) (2001) 529~558
19. Zhu, H.: The Role of Caste in Formal Specification of MAS. In: Proc. of PRIMA'2001. LNCS Vol. 2132. Springer (2001) 1~15
20. Zhu, H.: Developing formal specifications of MAS in SLABS, to appear in Proc. of AOIS''2002.
21. Moukas, A.: Amalthaea: Information Discovery and Filtering Using a Multi-Agent Evolving Ecosystem. Journal of Applied Artificial Intelligence 11(5) (1997) 437~457
22. Jennings, N.R.: Agent-Oriented Software Engineering. In: Garijo, F.J., Boman, M. (eds.): Multi-Agent System Engineering, LNAI 1647. Springer, (1999) 1~7
23. Spivey, J.M.: The Z Notation: A Reference Manual. 2nd edn. Prentice Hall (1992)
24. D'Inverno, M., Kinny, D., Luck M. and Wooldridge, M.: A formal specification of dMARS in Singh, M. P. Rao, A. and Wooldridge, M. (eds.): Intelligent Agents IV: Agent Theories, Architectures, and Languages. LNAI Vol. 1365. Springer (1998) 155~176
25. Luck, M. and d'Inverno, M.: A formal framework for agency and autonomy in Proc. of First International Conference on Multi-agent Systems. AAAI Press/MIT Press (1995) 254~260,
26. 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
27. Zhu, H., Jin, L.: Scenario Analysis in an Automated Tool for Requirements Engineering. J. of Requirements Engineering 5(1) (2000) 2~22