Design and Verification of an Agent-Based System

J. SEBESTYÉNOVÁ
Institute of Informatics
Slovak Academy of Sciences
Bratislava, Dúbravská cesta 9
SLOVAKIA

Abstract: - The paper describes design and verification problems of agent-based systems. An agent-based system for support decision making for physicians is proposed. Knowledge base is represented by a propositional logic formulas and we use deductive reasoning. Model of the system is given in Statecharts visual language. Required properties of the system can be given as formulas of the branching-time temporal logic. Statecharts model checking algorithm is used to verify safety and liveness property.

Key-Words: -Agent-based systems, Knowledge-base reasoning, Statecharts, model checking, temporal logic

1 Introduction

An early deliverable in traditional systems design is an architecture of the application, showing which entities interact with which other entities and specifying the interfaces among them. For example, installation of a conventional system for electronic data interchange among trading partners requires that one know the providers and consumers of the various goods and services being traded, so that orders can be sent to the appropriate parties. Sometimes, determining this information in advance is extremely difficult or even impossible. Consider an electronic system to support open trading, where orders are made available to any qualified bidder. Requiring the system designer to specify the sender and recipient of each transaction would quickly lead to "paralysis by analysis"[5]. From a traditional point of view, this application is ill-structured. That is, not all of the necessary structural information is available when the system is designed.

Such an application is a natural one for **agents**. The fundamental distinction in an agent's view of the world is between "self" and "environment." "Self" is known and predictable, while "environment" can change on its own within limits. Other agents are part of this dynamic, changing environment. Depending on the complexity of individual agents, they may or may not model one another explicitly. Instead of specifying the individual entities to be interconnected and their interfaces with one another, an agent-based design need identify only the *classes* of entities in the system and their impact on the environment. Because each agent is designed to interact with the environment rather than with specific other agents, it can interact appropriately with any other agent that modifies the environment within the range of variation with which other agents are prepared to deal. Naturally occurring multi-agent systems often use some form of currency to achieve global selforganization.

An agent is more than an object; it is a pro-active object, a bounded process. It does not need to be invoked externally, but autonomously monitors its own environment and takes action as it deems appropriate. This characteristic of agents makes them particularly suited for applications that can be decomposed into stand-alone processes, each capable of doing useful things without continuous direction by some other process.

An agent is a software program or a particular type of software module that co-operates on behalf of other entities and has some control over their actions and internal state. They perform their actions with some degree of pro-activity and/or reactivity. To compromise group intelligence one should strive for agents with knowledge of other agents so as to co-ordinate themselves with other agents.

We can distinguish many types of agents [6]:

- An Agent as a Single Reactive Process: toward isolated intelligent agents, rather than multi-agent systems.
- Agents as Capitalists: Dissipative mechanisms such as currency flows in markets are a powerful way to achieve coordination in a decentralized system.
- Agents as Travelers: mobile agents
- Agents as Members of a Community: multi-agent system
- Agents as Intelligent Processes:
  The *knowledge management layer* provides general-purpose representation and inference mechanisms that agents can use to model their knowledge and beliefs about the problem domain, the environment (including other agents), and themselves. It supports standard knowledge-representation methods

including nonmonotonic reasoning, deductive reasoning, inconsistency detection, automated concept classification, subsumption-based theorem proving, and truth maintenance.

The *ontology layer* uses the knowledge management layer to construct the specific models that an agent maintains of its domain, its environment, and itself. Ontology is a description (specification) of a domain and of the objects that exist in that domain.

The *cooperation and conflict layer* supports shared knowledge between agents for managing an agent's beliefs when it receives contradictory information from other agents.

The *coordination and communication layer* provides inter-agent communication.

A rational agent acts in its own best interest. Several important categories of possible mental states that may characterize an agent:

- information attitudes: knowledge and belief
- pro-active attitudes: goals, desires, intentions
- normative attitudes: obligations, permissions, authorization.

Agent-based applications provide a new way of viewing problems and designing solutions. Agent-based architectures are robust and dynamic; they can quickly react to unexpected events and adapt to changing conditions. They are inherently distributed and scalable: more agents and more computers can be added as necessary to increase the performance or the capacity of a system. Compared with centralized systems, agent-based architectures are easy to maintain, to modify and to extend as the requirements from the system change and grow with time.

It is possible to distinguish between two main classes of multi-agent systems [9]:

- Distributed problem solving systems in which the component agents are explicitly designed to cooperatively achieve a given goal
- Open systems in which agents are not co-designed to share a common goal; the composition of the system can dynamically vary as agent enter and leave the system.

An agent system development process consists of the following phases: definition of requirements, analysis, design, implementation, test and evaluation. Analysis means that the requirements have to be examined before working toward a conceptual model. The analysis model is made up of entities and collaborations of entities. It represents the structure of a proposed system at a certain level of abstraction. The entities of the analysis will be used during the design phase and within the software architecture. The implementation of the entities means writing and compiling the code together that brings us to the last phase, the evaluation and the deployment of the software development tools. Effectuating of the test and evaluation phases are the last steps for the realisation of the system.

One of the most important aspects for the realisation of a multi-agent system is to specify the capabilities of the interoperable agents and to define the structure of the agent system. The agents are acting in a problem domain, where they try to accomplish the overall goal(s) of the system. In order to share their knowledge and to obtain cooperation the agents communicate with each other. For this purpose, the agents use protocols and one can model a dialogue structure between agents.

The purpose of the interactions is to obtain a joint decision between two or more agents. For these joint decisions also negotiation mechanisms, market mechanisms and voting schemes are often necessarily. Some of these decisions are hard to achieve, because it requires omniscience, a lot of knowledge and for several voting or market mechanisms there doesn't exist an optimal solution. Another complexity is that agents can change their roles.

Two important levels in **design** phase[5] are:

- The Agent Community(Social Level):
  - Protocols (dynamics of communication and co-ordination
  - Organization (roles of services of each agent with respect to the others)
- The individual agent (Knowledge level):
  - Local planning (capabilities and plans)
  - Local behaviour (reactivity, routine tasks)
  - Local knowledge (the agent's beliefs).

Interactions between simple, reactive agents can lead to a global intelligent behaviour of the multi-agent system. The behaviour of the multi-agent system as a whole is said to emerge because it exhibits an intelligence that is not in an obvious way related to the behaviour of the individual agents.

In some systems an agent's code can change during the agent's lifetime. "Code" means a data structure that is executed through time. A simple linear sequence of instructions does not count; there must be some branching or decision-making. The modification may either be imposed on the agent from outside or initiated internally.

Identifying agents, a system developer needs to take into account following problems:

- Thing vs. function: in naturally occurring systems, agents are divided on the basis of distinct entities rather than functional abstractions (functional decomposition)
- Small in size: small specialized agents and using of appropriate aggregation technique
- Decentralized: centralization often appears in artificial systems (central agent - bottleneck), natural systems achieve distribution
- Diversity and generalization (balance)

- Local communication is used instead of broadcast
- Information is shared in space and time – learning
- Decomposition of individual agents into behaviors.

Formal verification using mathematical methods examines the state space of the given design and verifies whether it satisfies the required properties. Computer-aided verification is a general approach with applications to hardware verification, software engineering, multi-agent control systems. It is appropriate for control-intensive applications with interesting interaction among components. Formal analysis has to answer to following questions about system's behavior:

- Are the descriptions logically consistent and complete?
- What kind of behavior emerges from realistic numbers of agents and interchanges?

Formal agent theories are agent *specifications*, not only in the sense of providing descriptions and constraints on agent behavior, but also in the sense that one understands the term 'specification' from mainstream software engineering, namely that they provide a base from which to design, implement and verify agent systems. Agents are a natural next step for software engineering; they represent a fundamentally new way of considering complex distributed systems, containing societies of cooperating autonomous components. Formalisms and notations can be used to specify the desirable behavior of agents and multi-agent systems. [7] use combinations of modal temporal logics to model checking agent systems. [1] describes symbolic model checking of multi-agent systems.

2 Design of a decision-support system

As an example, we shall use an agent-based support system for physicians to conflict-free prescription of medicines, if the patient suffers from more than one disease.

Stages in designing the multi-agent system consist of conceptual analysis (components) and design of the system architecture:

- What system-level behavior do we want (specification - what a system as a whole will do)?
  a) An apropriate medicine or medicines will be prescribed for the patient to cure his disease or diseases.
  b) Any physician will prescribe no medicine contraindicated in any disease of the patient to him.
- What kind of agents might we need to get it?
  a) An agent supporting the physician's reasoning.
  b) Such an agent will support all of the physicians curing the patient.
- How should they behave? The agents will consult and support physicians decision:
  a) Request information from the patient and receive answers.
  b) Having collected all needed facts, the agent supports the physician by knowledge-base reasoning to prescript a medicine. If the patient already has prescribed any medicine for cure another disease, it is important to check whether the new medicine is not contraindicated to the other disease.
  c) In case, the medicine previously prescribed by another physician-specialist is contraindicated to the new diagnosed disease, the agents supporting these two physicians start to consult. The agent of the first physician informs the agent of the second physician about new facts and requests him to change the prescribed medicine. The agent of the second physician starts a new reasoning process leading to proposal of a new medicine. The agent of the first physician accepts (or rejects) the proposal.
- How do our proposed agents interact with one another in an organization?
  a) Send a message to another physician containing new facts and request to change the prescription.
  b) Receiving a message: search a database for another medicine not contraindicated with the new disease (new fact).
- What low-level behaviors are needed?
  a) Collecting facts
  b) Knowledge-base reasoning.

3 Statecharts model

Internal behavior of an agent and changes of its mental states can be specified using Statecharts. A population of instances of an agent can be modeled as parameterized state in Statecharts.

Statecharts is a graphical formalism [3] to describe hierarchically structured state machines. The formalism extends finite state machines with concepts of hierarchy, concurrency and communication. Statecharts formalism is used for description of the system's behavior.

Semantically, a Statechart may respond to an event entering the system by engaging in an enabled transition. This may generate new events which, by causality, may in turn trigger additional transitions while disabling others. The synchrony hypothesis ensures that one execution step, a so called macro step, is complete as soon as this chain reaction comes to a halt. The Statecharts principle of global consistency prohibits an event to be present and absent in the same macrostep.

Drawing of a statechart begins with a root state (rounded rectangle). If the root state has to be exclusive-OR decomposed, drawing continues with its substates and transitions. If it has to be an AND decomposition,

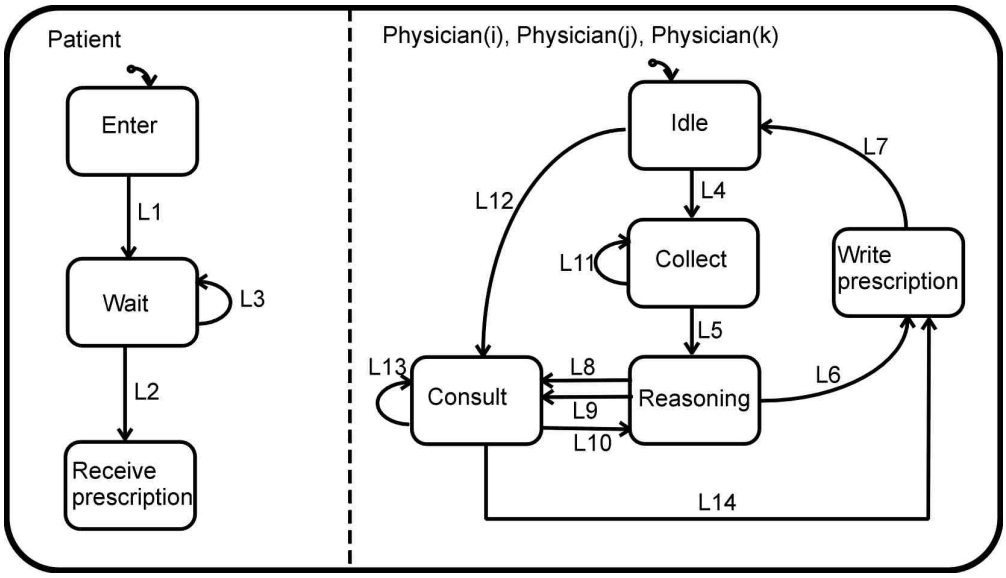

Fig. 1 Statecharts model of the agent-based support system

the state is cut with dashed lines into several parallel parts (orthogonal components).

The behavioral description of complex reactive systems consists of some sequential and parallel processes. The process starts if a proper starting event occurs. If there exists any guarding condition, the process can start provided the condition is not false. If any transition of any process is firable, it is fired. In the substates of an AND state, transitions can be taken simultaneously. Within a XOR state, only one transition can be followed. The complete set of transitions is considered one step.

For all of the transitions, a source state from which the transition is going out, a target state into which the transition is going, and a label of the form *event [condition] / action* are given.

An action takes zero time (it is an event). In behavioral specification of reactive systems, the possibility of describing a non-zero time taking activity is needed, too. An action can be specified along a transition (as a part of a transition's label) and on a state's entrance and exit. An activity will be carried out continuously throughout the system is in the state. To specify the activities, a programming language or another formalism can be used.

Statecharts model of the agent-based support system is given in Fig. 1. The self-explanatory events and states names are mostly used. The transitions labels in the form *event[condition]/action* are:

L1: true / request_medicine
L2: write
L3: query / answer
L4: request_medicine / query ∧ R := 0
L5: answer [¬ need_more_info] / start(reasoning)
L6: end(reasoning) [¬ contraindication ∧ R == 0] / write
L7: timeout
L8: end(reasoning) [contraindication] / req_consultation(send_to_j)
L9: end(reasoning) [¬ contraindication ∧ R==1] / proposal(send_to_k)
L10: to_reasoning / start(reasoning)
L11: answer [need_more_info] / query
L12: req_consultation(from_k) / to_reasoning ∧ R := 1
L13: proposal(from_j) / accept_proposal(send_to_j)
L14: accept_proposal / write

Reasoning activity will be carried out continuously throughout the system is in the state *Reasoning*.

4 Knowledge-base reasoning

Knowledge-base (KB) systems provide an approach to knowledge representation and manipulation. Epistemic logic is a logic of knowledge. It is sufficient to enrich the language of classical propositional logic by unary operators K_i where K_i φ stands for „agent *i* knows φ“. KB system consists of knowledge base (rules and facts) and inference engine. KB systems may be of different types, e.g., rule-based systems (if-then production systems), model based reasoning, case-based reasoning, cost-based reasoning, neural nets, fuzzy logic, decision trees, etc.

For our example, we shall use knowledge base represented by a propositional logic formulas and we shall use deductive reasoning. Knowledge can be divided to common knowledge what all agents (everybody in a community) know, and local knowledge of an agent. Table 1 contains rules of common

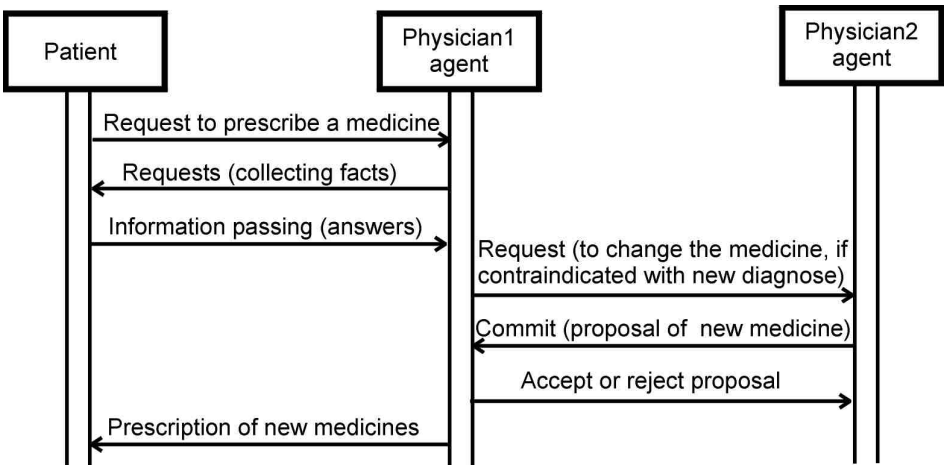

Fig. 2 Communication and co-operation between the agents

knowledge specifying which physician is specialist for a disease, and contraindications for medicines.

Table 1 Rules of common knowledge

Disease1 → Phisician1	Medicine1 → ¬ Med3
Disease2 → Phisician2	Medicine2 → ¬ Disease3
Disease3 → Phisician2	Medicine1 == drops
	Medicine2 == pills

Special knowledge about which medicine can be used to cure a given disease is local to a physician-specialist (and only the specialist is enabled to prescribe it). Constraints on usage of the medicines are given here, too. The local knowledge is given in table 2.

Table 2 Local knowledge of two specialists

Physician1:	Physician2:
Disease1 → Medicine1	Disease2 → Medicine3
Disease1 → Medicine2	Patient.age<5 → ¬ Med3

For reasoning, we also need to know some facts. Some of them are stored in knowledge base and others will be collected by the physician from the patient during an examination. For the example, we shall work with facts given in table 3.

Table 3 The facts used in the example

Patient.age == 12
Patient.prefers == drops
Patient.Diseases == Disease1 ∧ Disease2

The inference sequence of the example is given in table 4.

Table 4 The inference sequence

Physician.expertise == Physician1
Patient.Diseases == Disease1
Patient.age == 12
Patient.prefers == drops
Patient.Medicines == (Med1, Disease1, Physician1)
Physician.expertise == Physician2
Patient.Diseases == Disease1 ∧ Disease2
Patient.Medicines == (Med3, Disease2, Physician2)
Patient.Medicines == (Med2, Disease1, Physician1)

Co-operation between agents, information flow and communication are given in Fig. 2.

Reasoning can be described as an inferential process moving from a problem to an appropriate response. Full statement of the problem, whether theoretical or practical, will involve all the relevant information and this provides the premises from which a conclusion can be inferred representing an answer to the problem. On this view, an agent's drawing that conclusion is an appropriate response to its asking a question.

5 Verification

The formal approach uses the modeling language to system description, the specification language to description of the required correct system behavior, and it provides an analysis technique. The model has to describe not only the designed system but also the environment in which it will work. The system can be modeled at different abstraction levels. For the purposes of modeling, specification, verification, and synthesis of discrete event systems, Petri nets, temporal logics, different algebras of concurrent processes, etc., have been developed.

Formal logic provides a basis for showing that a program will behave as the user intends. A formalism for a multi-agent system must also deal with the multiplicity of agents; group properties of agent systems, such as common knowledge and joint intention; interaction among agents, such as communication and cooperation.

Having specified a solution for a problem, and implemented a system that should do the job, one needs to show that the implemented specification is correct, or if the implemented system satisfies certain properties. Verification often involves temporal properties [4]. Examples of general properties of programs are safety and liveness. We can divide approaches to the verification of systems into two classes:

- Axiomatic verification reduces to a proof problem.
- Semantic verification: given a formula φ of language *L*, and a model *M* for *L*, model checking [2] problem is to determine whether or not φ is valid in *M*, i.e., whether or not $M \models_L \varphi$.

One of the problems with using model checking to verify properties involving knowledge is that existing model checkers are designed to verify temporal, rather than epistemic properties. Combined modal and temporal logics form the basis for agent-based formal methods. The logics have:

- an informational component to represent an agent's beliefs or knowledge
- a dynamic component allowing the representation of dynamic activity (temporal logics)
- a motivational component representing the agent's desires, intentions or goals.

For the Statecharts model-checking algorithm described in [8], the required properties of the system can be specified as propositional, branching-time temporal logic formulas consisting of: propositions, that can be of type *in(state)* condition, event, assertion about variable's value, e.g., statement of a program: y == 1; Boolean connectives: $\neg(p)$, $(p \vee q)$, $(p \wedge q)$, $(p \rightarrow q)$, where *p* and *q* are subformulas; temporal operators (use Clarke-Emerson [2] notation): **G** globally, **F** finitely, **X** next time, **U** until, **A** for all paths, **E** for some path.

The first property that will be verified, for example, is safety: **AG** ¬ (medicine1 ∧ medicine3). If the inference rules system is sound and complete, this requirement is fulfilled.

The second verified property will be a liveness, i.e., a system's response to a stimulus:

A (req_consultation) → **F** (proposal).

For this type of required property (implication), the first step of the algorithm is to find states where the source part of the implication holds. These states will be considered initial states. The second step of the algorithm is to find all traces from the initial state to the state in which the second part of the implication holds.

6 Conclusion

An agent-based system for support decision making for physicians is designed. Model of the system is given in Statecharts visual language. Knowledge base is represented by a propositional logic formulas and deductive reasoning is used. Problems of verification of agent-based systems are discussed. Safety and liveness properties of the system given as formulas of the branching-time temporal logic are verified using Statecharts model checking algorithm.

The author is grateful to the Slovak Grant Agency for Science (grant No. 2/4148/04) for partial support of this work.

References:

[1] Benerecetti M., Cimatti A., Symbolic Model Checking for Multi-Agent Systems, Istituto Trentino di Cultura, 2001.

[2] Clarke E. M., E. A. Emerson, A. P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, In: *ACM Trans Program Lang Syst*, Vol 8, No 2, 1986, pp. 244-263.

[3] Harel D., Statecharts: a visual formalism for complex systems, In: *Science of Computer Programming*, Vol 8, No 3, 1987, pp. 231-274.

[4] Pnueli A., The temporal logic of programs, In: *Proceedings of 19th IEEE Symposium on Foundations of Computer Science*, 1977, pp. 46-57.

[5] Parunak H. V. D., A. D. Baker, and S. J. Clark, The AARIA Agent Architecture: An Example of Requirements-Driven Agent-Based System Design. In: *Proceedings of First International Conference on Autonomous Agents (ICAA-97)*, 1997.

[6] Parunak H. V. D., Practical and Industrial Applications of Agent-Based Systems, Environmental Research Institute of Michigan (ERIM), 1998

[7] Rao A. S., Georgeff M. P., A model-theoretic approach to the verification of situated reasoning systems, In: *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI- 93)*, Chambery, France, 1993, pp. 318–324.

[8] Sebestyénová J., Hierarchical verification of reactive systems with timing constraints, In: *WSEAS Transactions on Computers*, Issue 4, Vol 2, Oct 2003, pp. 1174-117.

[9] Zambonelli F., Jennings N. R., Wooldridge M., Developing Multiagent Systems: The Gaia Methodology. In: *ACM Transactions on Software Engineering and Methodology*, Vol. 12, No 3, July 2003, pp. 317-370.