

The Engineering of Emergence in Complex Adaptive Systems

ANET POTGIETER

University of Pretoria, Pretoria, South Africa
and

JUDITH BISHOP

University of Pretoria, Pretoria, South Africa

Agent-oriented software engineering is a new software engineering paradigm that is ideally suited to the analysis and design of complex systems. The main focus of these methodologies is to engineer a complex system in such a way that the correct emergent behavior results. In a complex adaptive system, however, emergent behavior cannot be predicted during analysis and design as it evolves only after implementation. By restricting emergent behavior as is done in most agent-oriented software engineering approaches, a complex system cannot be fully adaptive as well. We propose the BaBe methodology that will enable a complex system to be adaptive as the system can learn from its environment during run-time and modify its behavior in order to adapt to changes in its environment. This methodology adds a run-time emergence model consisting of distributed Bayesian Behavior Networks to the agent-oriented software engineering lifecycle. These networks are initialized by the human software engineer and deployed by Bayesian Agencies (also complex adaptive systems). The distributed Bayesian Behavior Networks, being specialized Bayesian Networks, will enable the Bayesian Agencies to collectively mine relationships between emergent behaviors and the interactions that caused them to emerge. This technology will further enable the Bayesian Agencies to collectively learn from the emerged behaviors and to modify the behavior of the system in order to adapt to changes in the environment. We describe a prototype implementation of the Bayesian Agencies using Sun's Enterprise JavaBeans™ component architecture.

Categories and Subject Descriptors: Adaptive Agent Architectures, Agent-Oriented Software Engineering, Complex Adaptive Systems, Distributed Bayesian Networks

General Terms: Emergence, Belief Propagation

Additional Key Words and Phrases: BaBe Methodology, Bayesian Behavior Networks, Bayesian Agencies

1. INTRODUCTION

Open distributed environments, such as the Internet, place a growing demand on complex systems to learn from and adapt to their uncertain environments. The uncertainty in these environments is mostly due to the behavior of other complex adaptive systems such as users browsing web pages, the behavior of buyers and sellers on the Internet and the behavior of autonomous agents bidding on behalf of persons in Internet auctions. The emergence of global behavior from local interactions in complex adaptive systems cannot be engineered during the analysis and design phases as is done in most agent-oriented software engineering approaches. An adaptive agent architecture, that is an agent architecture that can function as a complex adaptive system, evolves only after implementation in order to adapt to changes in its environment.

Authors' addresses: Anet Potgieter, 10 Bloemendal Road, Mowbray, Cape Town, 7700, anet@bayesbean.com;
Judith Bishop, Department of Computer Science, University of Pretoria, Pretoria, 0002, jbishop@cs.up.ac.za

Adaptive agent architectures have been developed in robotics research. In most of these architectures, the engineering of emergence is a manual iterative process (Brooks [1985], Maes [1994]) in which a human observer places the autonomous agent in its environment and observes its behavior using a laborious process of experimentation, trial and error (Jennings, Sycara and Wooldridge [1998]). In a complex adaptive system, this becomes an impossible task for the human observer to perform.

In this paper, we propose an adaptive agent architecture that uses distributed Bayesian Networks implemented by hierarchies of Bayesian Agencies (also complex adaptive systems) in order to assist in bridging the gap between the implementation and the understanding of emergent behavior in complex adaptive systems.

2. COMPLEX ADAPTIVE SYSTEMS

A complex adaptive system is characterized by complex behaviors that emerge as a result of interactions among system components and the environment. Such a system interacts with its environment by learning from and modifying its behavior in order to adapt to changes in its environment.

Examples of complex adaptive systems include:

- Users “foraging” for information, navigating from web page to web page along Web links;
- The behavior of consumers in a retail environment;
- The interactions between companies, consumers and financial markets in the modern capitalist economy;
- Intelligent autonomous agents bidding on peoples’ behalf in Internet marketplaces;
- Bayesian Networks, Neural Networks, Genetic Algorithms and Artificial Life Systems.

Agent-oriented approaches are well suited for developing complex systems. This involves the analysis, design and implementation of a complex system as a multi-agent system (Jennings [2001]). In order to implement a complex adaptive system, the agents in the multi-agent system must collectively learn from and adapt to their changing environment – this is emergence. In the next section, we describe agent architectures, agents and agencies. In the sections thereafter, we describe emergence and the engineering thereof.

3. AGENT ARCHITECTURES

3.1 Overview

An agent architecture is a software architecture implementing either a single autonomous agent or a multi-agent system. An adaptive agent architecture is an agent architecture that can function as a complex adaptive system.

Although agent architectures are widely used, there is still no consensus on what an agent is. Minsky [1988] first established the concept of simple unintelligent agents combined into intelligent agencies. He describes the mind as a “society” of tiny components that are themselves mindless. He refers to each of these components as agents. His simple agents combine into (sub)societies, called agencies. The agencies are intelligent through the interaction amongst the (unintelligent) agents.

Simple agents differ from autonomous agents used in most agent architectures in that a simple agent forms a *part* of a software system, whereas an autonomous agent implements an *entire* software system. To avoid confusion we will refer to our agents as *simple agents*. We use the following definitions of agents and agencies:

SIMPLE AGENT

Any *part or process* of a software system that by itself is simple enough to understand even though the interactions among groups of such agents are much harder to understand (Adapted from Minsky [1988]).

AUTONOMOUS AGENT

A computer system, *situated* in some environment, that is capable of *flexible autonomous* action in order to meet its design objectives (Wooldridge [1997], Jennings [2001]).

ADAPTIVE AGENT

An agent that can *improve over time*, i.e. the agent becomes better at achieving its goals with experience ... i.e. being able to change and improve behavior over time (Maes [1994])

AGENCY

Any collection of agents considered in terms of what it can accomplish as a unit, without regard to what each of its constituent agents does by itself (Minsky [1988]).

The agents in agencies can be organized into hierarchies or heterarchies, defined as follows:

HIERARCHY

A form of organization resembling a pyramid. Each level is subordinate to the one above it (Heylighen, Joslyn and Turchin [2001]).

HETERARCHY

A form of organization resembling a network or fishnet. Authority is determined by knowledge and function (Heylighen et al.).

Heterarchies are more powerful than hierarchies (Minsky [1988]). We exploit the implicit power of heterarchies in our research in that our agencies are organized into heterarchies. Collective intelligence of a heterarchy emerges through the interaction of the agents within the agencies of the heterarchy. We view a hierarchy as a simplified heterarchy.

Potgieter and Bishop [2001] defined the relationship between agents, agencies and heterarchies as follows:

An **agency** consists of a society of **agents** that inhabit some complex dynamic environment, where the agents collectively sense and act in this environment so that the agency accomplishes what its composite agents set out to accomplish by interacting with each other. If agents in a society belong to more than one agency, the set of “overlapping” agencies forms a **heterarchy**.

A multi-agent system is a system that has more than one interacting (simple or autonomous) agents, which can be grouped into one or more agencies. The characteristics of a multi-agent system are as follows:

- the intelligence lies in the behavior of agencies and not in the individual agents. Agencies are specialized for particular tasks, and contains specialized knowledge;
- global behavior of the multi-agent system emerges from interactions between agents and the environment;

- there is no global system control;
- data is decentralized; and
- computation is asynchronous

Different agent architectures adhere to different methodologies of analysis, design and implementation. The next sections give a brief description of representative methodologies followed in different agent architectures.

3.3 Subsumption Architecture

The Subsumption Architecture (Brooks [1985]) is an adaptive single-agent architecture that implements an autonomous agent consisting of task accomplishing behaviors as simple agents. Each behavior is a finite state machine mapping perceptual input to action output. Behavior interactions include for example the suppression of one behavior by another.

Behaviors are organized into a hierarchy of layers in which the level of abstraction increases from the bottom layers up. The complexity of an agent increases with the number of layers it has. Developing an autonomous agent that exhibits coherent behavior is a process of carefully developing and experimenting with new behaviors, usually by placing the agent in its environment and observing the results. Overall behavior emerges from component behaviors when the autonomous agent is placed in its environment.

The methodology followed in this architecture is a manual process in which the software engineer must analyze the environment as well as the tasks that the agent must accomplish. Each behavior must be designed as well as the interactions between the behaviors. Goals are implicit, known only to the designer (Maes [1994]). The engineering of emergent behaviors consists of a laborious manual process of experimentation, trial and error (Jennings, Sycara and Wooldridge [1998]). In a complex autonomous agent with many layers, the emergent behavior becomes too complex to be understood by the human software engineer.

3.4 Behavior Networks

Maes [1989] developed an adaptive agent architecture that uses Behavior Networks, which are graphs used to model the relationships between “competence modules” (also called behaviors) in adaptive autonomous agents. Each competence module can be viewed as a simple agent.

The software engineer designs an autonomous agent in terms of a Behavior Network consisting of a number of nodes, representing behaviors, linked together by causal links. Each node models the selection of a particular behavior as an emergent property of an underlying process. Each behavior node consists of:

- a behavior;
- a list of preconditions that must be true for the competence module to become active and to execute the behavior;
- an add list of predicates which are expected to become true by execution of the behavior. Some of these predicates can be global goals that an autonomous agent must achieve; and
- a delete list of predicates, which are expected to become false by the execution of the behavior.

Behavior nodes are linked together by three types of links that represent causal relations among the behaviors, namely

- predecessor link from a competence module to every competence module that can make a precondition true;
- conflictor link from a competence module to every competence module that would make a precondition false; and
- successor link between all competence modules connected by a predecessor link but going in an opposite direction.

A compiler analyses the Behavior Network and generates a circuit that will implement the desired goal-seeking behavior. Once implemented, the competence modules will activate and inhibit each other along the links specified in the Behavior Network, so that after some time the activation energy accumulates in the competence modules that represent the 'best' actions to take given the current state of the environment and current global goals of the autonomous agent. Once the activation level of a competence module surpasses a certain threshold, and if the module is executable, it becomes active and executes real actions. Testing if the autonomous agent meets its requirements consists of a process of (human) observation, extensive testing and simulation.

3.5 Behavior-Oriented Design (BOD)

Behavior-Oriented Design (BOD) is an object-oriented methodology developed by Bryson [2001], that can be used to analyze, design and implement single agent architectures. This is an iterative design methodology consisting of two phases, an initial decomposition phase and an iterative development phase.

In this manual methodology, emergence is engineered during the iterative development phase by a “hand-cranked” version of the EM algorithm (Bryson [2001]). The first step of an iteration is to elaborate the current model and the second step is to revise the model to find the new optimum representation.

3.6 Agent-Oriented Software Engineering (AOSE)

The analysis and design methodology followed in multi-agent (mostly non-adaptive) agent architectures is referred to as agent-oriented software engineering. Using this approach, a complex system is decomposed into multiple interacting autonomous agents. Griss and Pour [2001] refer to agents as “next-generation” components and to agent-oriented software engineering as an extension to conventional component-based software engineering approaches.

Agent-oriented software engineering is a process of generating increasingly detailed models during the analysis and design phases. Examples of agent-oriented software engineering approaches include the Gaia methodology (Wooldridge, Jennings and Kinny [2000]) and coordination models (Zambonelli, Jennings, Omicini and Wooldridge [2000]).

Figure 1 illustrates the different models in the Gaia analysis and design phases.

During the Gaia analysis phase, the following models are defined:

- roles model - A description of each role in terms of responsibilities, permissions, interaction protocols, and activities; and
- interactions model – Description of each protocol in terms of data exchanged and partners involved.

During the Gaia design phase, the following models are defined:

- agent model - the autonomous agent classes and instances composing the system;
- services model - the services to be provided by each autonomous agent in order to realize its role and

- acquaintance model - a description of the lines of communication between different autonomous agents.

The Gaia methodology is not suitable for complex adaptive systems. According to Zambonelli et al. [2000] it was intended for use in closed systems of distributed problem solvers, in which autonomous agents collaborate to achieve a global goal. Zambonelli et al. adapted the Gaia methodology in order to make it suitable for open environments and defined a coordination-oriented methodology. This methodology tries to anticipate all potential actions that autonomous agents might take. They impose restrictions on the interactions, which they call “social laws”. These “social laws” ensure that agents that adhere to these laws need not worry about undesirable interactions no matter what goals or plans they adopt (Durfee [2001]).

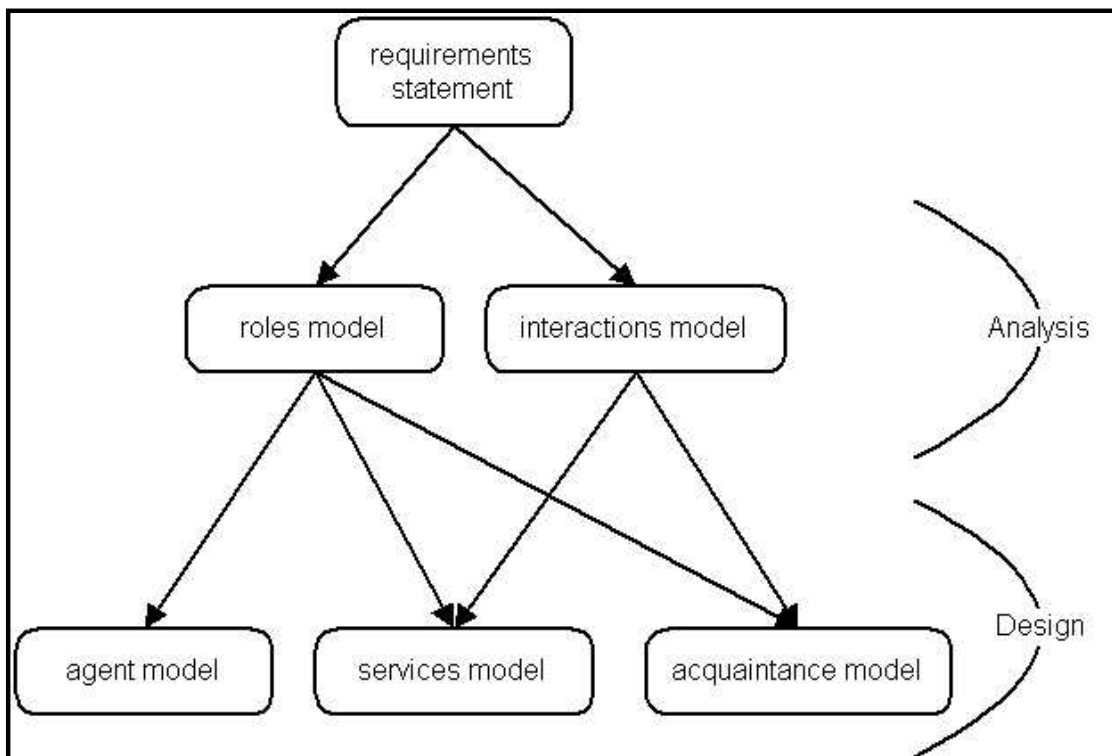


Figure 1: Relationships between the Gaia Models (courtesy of Wooldridge et. al [2000])

The engineering of emergence cannot be done during the analysis and design phases. It must evolve once the system is deployed. In the sections that follow, we will discuss emergence in more detail and propose a methodology that adapts and extends the Gaia methodology, in order to engineer emergence in complex adaptive systems.

4.0 EMERGENCE

4.1 Overview

Emergence in a multi-agent system is the collective behavior of the individual (simple or autonomous) agents in such a system. Emergence arises from the agent interactions. This is a very desirable property for a multi-agent system to have, as emergence produces intelligence (Brooks [1991]).

The uncertainty associated with emergence makes the engineering thereof very difficult. Most agent-oriented software engineering approaches attempt to “understand” emergence by engineering it before the fact. Current approaches to the engineering of emergence include:

- using an iterative approach to simulate and test as many behaviors as possible in order to “understand” the emergent behaviors (Bryson [2001]);
- manual engineering of emergence by placing the system in its environment, and observing the emergent behavior (by a human observer) (Brooks [1985], Maes [1989], Bryson [2001]);
- restricting the interactions between the agents by using interaction protocols and imposing social laws (Wooldridge et al. [2000], Zambonelli et al. [2000]).

Emergence cannot be engineered by using static software engineering processes. A multi-agent system in which the agents communicate with each other using pre-defined interaction protocols can be used to implement a complex system, but not necessarily a complex adaptive system. Emergence must be “observed”, modeled, and if the model deviates from the design model, the design model must be adapted. The observation and modeling process must form an integral part of the software engineering cycle.

4.2 The Engineering of Emergence

4.2.1 Overview

According to Minsky [1988] the process of understanding any large and complex “thing” (system) can be broken up into three subprocesses, as follows:

First we must know how each separate part works. Second we must know how each part interacts with those to which it is connected. And third, we have to understand how all these local interactions combine to accomplish what the system does – as seen from the outside.

Minsky's third step above is concerned with the understanding of emergent behavior. This is an iterative and emergent process in itself, and can only be done by observing behaviors and comparing it to a model. An observed behavior is emergent as soon as it deviates from the observer's model of it.

It is as difficult to define emergence as it is to define intelligence. Drawing their inspiration from the Turing Test, Ronald, Sipper and Capcarrère [1999] formulated an emergence test in terms of an observer's incapacity to reconcile observed global behavior from his awareness of the local interactions. This test is described as follows:

Our emergence test centers on an observer's avowed incapacity (amazement) to reconcile his perception of an experiment in terms of a global world view with his awareness of the atomic nature of the elementary interactions.

Ronald et al. described the emergence test in terms of the following three conditions, namely design, observation and surprise of a system designer and a system observer (which could be the same), as follows:

- **Design:** constructed by the system designer in terms of local elementary interactions between components, in a language $L1$.
- **Observation:** The observer is fully aware of the design, but describes global behaviors and properties of the running system over a period of time using a language $L2$.
- **Surprise:** The design language $L1$ differs from the observation language $L2$ and the causal link between the elementary interactions programmed in $L1$ and the behaviors observed in $L2$ are non-obvious to the observer who then experiences surprise.

Engineering emergence can be viewed as a continuous process of bridging the $L1$ - $L2$ gap. In this research, we propose the use of a Bayesian Behavior Network as the observer's model of the elementary interactions programmed in $L1$ and the behaviors observed in $L2$. We implement our observer as a heterarchy of Bayesian Agencies, collectively implementing the Bayesian Behavior Network. As soon as the observer is "surprised" by unexpected behavior, the global view must be reconciled with the awareness of the underlying elementary agent interactions. This will be done by the Bayesian Agencies during Bayesian learning, by incrementally updating the underlying Bayesian Behavior Network. This process forms an integral part of the software engineering lifecycle.

4.2.2 Bayesian Behavior Networks

A Bayesian Behavior Network is a Bayesian Network that represents a probabilistic model of the relationships between individual behaviors, the state of the environment and the global behaviors of the system. A Bayesian Network is a directed acyclic graph (DAG) that consists of a set of nodes, representing random variables, linked together by causal links. The causal dependencies are given in terms of conditional probabilities of states that a node can have given the values of the parent nodes.

Let $B = \{B_1, \dots, B_n\}$ be a set of random variables. Let $P = \{P_i\}$ where P_i is the conditional probability matrix associated with B_i . $P_i = \{P(B_i | pa(B_i))\}$, where $pa(B_i)$ represents the parents of B_i . An assignment $(B_1=b_1, \dots, B_n=b_n)$ can be abbreviated to $b = (b_1, \dots, b_n)$.

The Bayesian Network represents a global joint probability distribution over B having the product form

$$P(b_1, \dots, b_n) = \prod_{i=1}^n P(b_i | pa(b_i)) \quad (1)$$

The joint probability distribution represents the global semantics of a Bayesian Network, and the local semantics of a Bayesian Network is that each variable in the network is independent of its nondescendants in the network given its parents (Pearl and Russel [2000]).

In a Bayesian Behavior Network, the random variables represent local behaviors, global behaviors and environmental states. We use Bayesian Behavior Networks in a similar way as the Behavior Networks defined by (Maes [1989]). Behavior Networks, however, allows for causal modeling using booleans only, whereas Bayesian Behavior Networks allows for powerful probabilistic reasoning in the presence of uncertainty.

Figure 2 illustrates a Bayesian Behavior Network that is a fictitious model of the browsing behavior of users visiting an electronic bookstore website. This network models the relationships between the type of user that browses the site (A), their interests (B), the sequence of hyperlinks that they clicked to access the pages (C), content categories of all the pages on the website (D), the information content of the advertisements on the web pages (E), the pages they view (F), the pages that they will visit next (H) and the buying behavior per page (G).

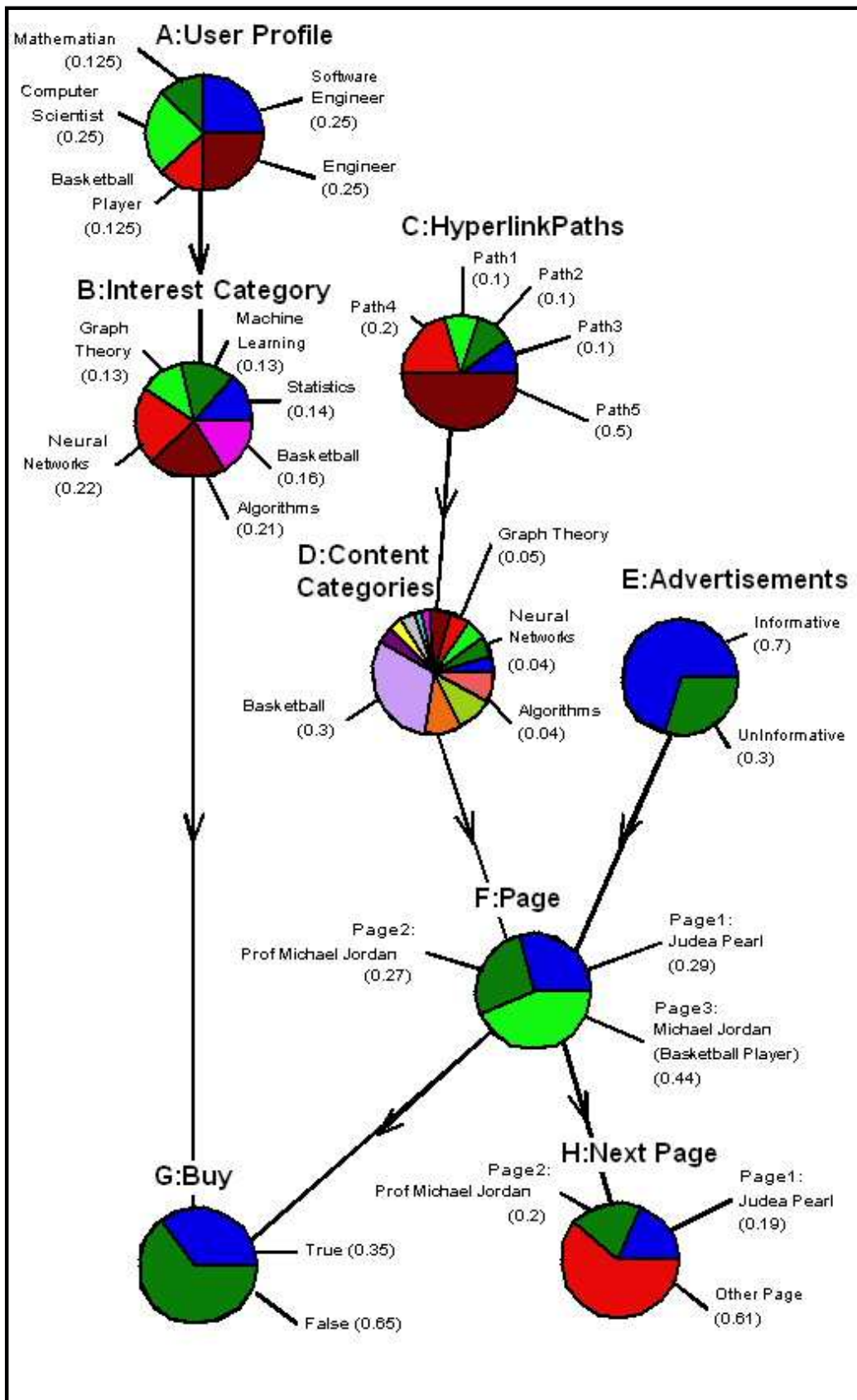


Figure 2. Bayesian Behavior Network

Our example website have hyperlinks to the following pages:

- Page 1: books by Judea Pearl on causality and probabilistic reasoning;
- Page 2: books by professor Michael Jordan on graph theory and probability theory;
- Page 3: books by / related to Michael Jordan, the well-known basketball player;
- Other Page: any other page on the website.

The hyperlink paths to these pages are:

- Path 1: Engineering and Science→Mathematics→Graph Theory→Pages 1 & 2;
- Path 2: Engineering and Science→Mathematics→Probability and Statistics→Pages 1 & 2;
- Path 3: Computers and Internet→Artificial Intelligence→Machine Learning→Neural Networks Page 2;
- Path 4: Computers and Internet→Programming→Software Engineering→Algorithms→Pages 1 & 2;
- Path 5: General Interest→Sports and Adventure→Basketball→Page 3.

In this simple model, buying behavior (G) depends on the current page that is being browsed (F), and the categories of interest of a particular user (B), which in turn depends on the user profile (A). For simplicity, users are profiled on their profession only. The website contents can be categorized into content categories (D), which are distributed between different pages (F). In order not to clutter the diagrams, only a few content categories are indicated next to node D. The choice of a page (F) depends on how well its contents matches the content categories (D) that the user is looking for and how well the content categories were advertised to the user (E). The hyperlinks to the pages (C) are related to the content categories (D) that a user is looking for. The relationship between the current page (F) that is being viewed and the next page (H) that will most probably be browsed next is also modeled in this network. This example network represents the joint probability

$$\begin{aligned}
 &P(a,b,c,d,e,f,h) \\
 &= P(a)P(b|a)P(g|b,f)P(c)P(d|c)P(f|d,e)P(e)P(h|f)
 \end{aligned} \tag{2}$$

From (1) and (2) it can be seen that the global distribution is described in terms of local distributions. Each node's local information consists of the arcs originating from it and terminating in it and the local conditional probability matrix. For example, table 1 lists the conditional probability matrix for Node B: Interest Category.

Bayesian learning can be viewed as the calculation of the parameters of the conditional probability matrices that best model the data (Russel, Binder, Koller and Kanazawa [1995]).

P(B:InterestCategory A:UserProfile)					
B: InterestCategory	A:UserProfile				
	Engineer	Mathematician	Computer Scientist	Software Engineer	Basket ball Player
Graph Theory	0.1	0.25	0.1	0.2	0.01
Statistics	0.2	0.34	0.1	0.1	0.01
Machine Learning	0.1	0.1	0.25	0.1	0.01
Neural Networks	0.3	0.2	0.25	0.2	0.01
Algorithms	0.25	0.1	0.25	0.3	0.01
Basketball	0.05	0.01	0.05	0.1	0.95

Table 1: Conditional Probability Matrix for Node B: InterestCategory

Belief Propagation is the process of finding the belief (also called the most probable explanation) for each node in the presence of evidence. Dechter [1996] defines the most probable explanation (MPE) in the presence of evidence (e) as the maximum assignment b in

$$\max_b P(b) = \max_b \prod_{i=1}^n P(b_i | pa(b_i), e) \quad (3)$$

The belief in the absence of evidence is indicated next to each of the nodes in Fig 2. For example, the beliefs of the user profile node (A) indicate that mathematicians and basketball players browse this site with equal probability of 0.125. The beliefs of the hyperlink paths node (C) indicate that Path 5 will most probably be chosen (0.5) and the beliefs of the content categories (D) indicate that the basketball category is most likely to be searched for (0.3). The beliefs of the page node (F) show that the Michael Jordan (basketball player) page will most probably be viewed (0.44). The beliefs of the advertisements node (E) show that the advertisements that led the user to this page were informative with a probability of 0.7. The beliefs of node (G) show that the probability that a user will buy a book when visiting a page is 0.35. The probability that a user would be interested to view books by Michael Jordan (the professor) next is 0.2.

Figure 3 illustrates the results of belief propagation in the presence of evidence.

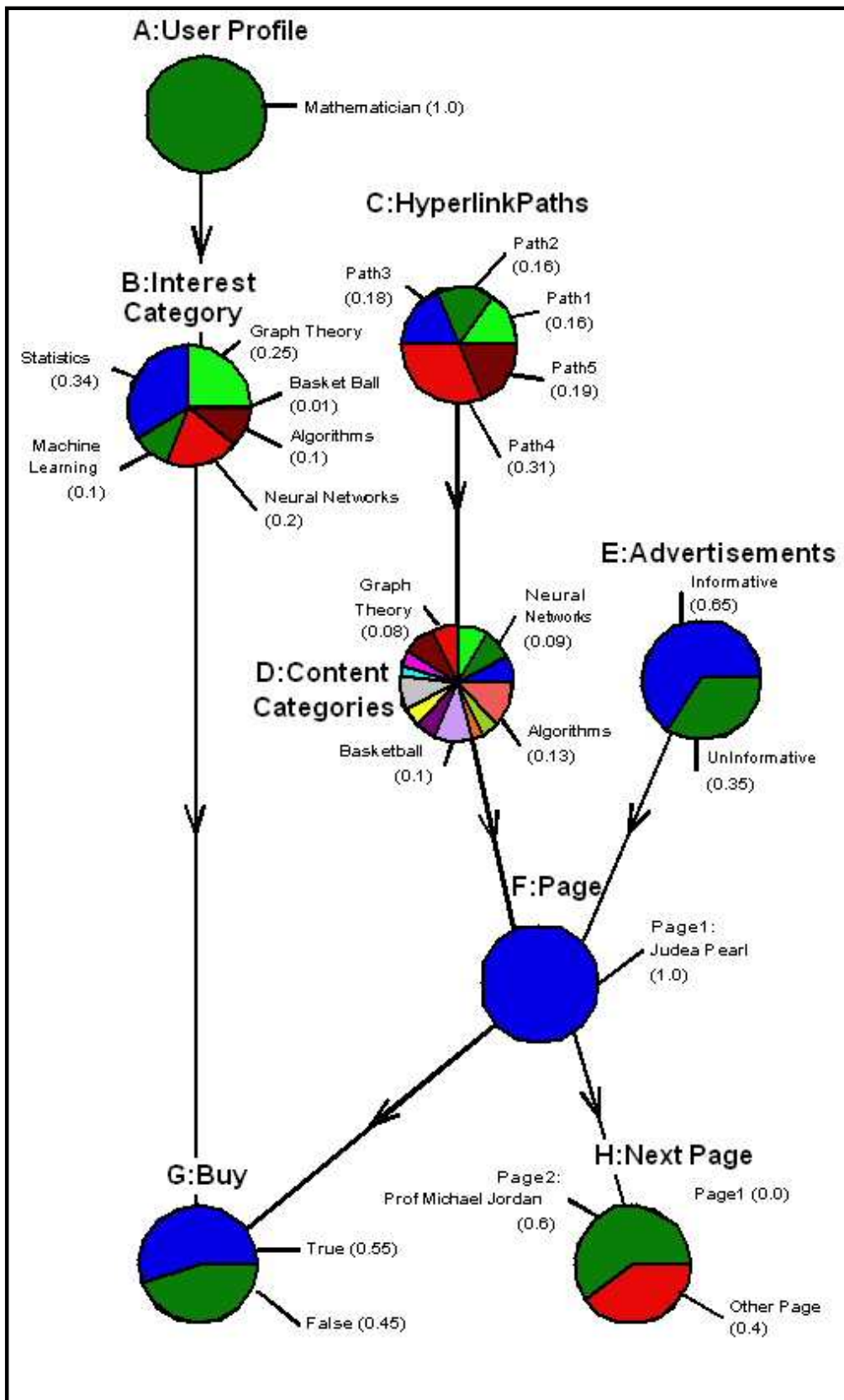


Figure 3. Belief Propagation

A mathematician that browses a website listing books by Judea Pearl is most probably interested in statistics (0.34), graph theory (0.25) and neural networks (0.2). He would have chosen hyperlink path 4 with the highest probability (0.31) in order to search for algorithms related to his field if interest. He will buy a book from this page with a probability of 0.55. The probability that this user will be interested to view books by professor Michael Jordan next has now risen to 0.6 and the probability that the advertisements were informative has now decreased to 0.65.

Using Bayesian Behavior Networks, Bayesian Agencies can “observe” their own behaviors and adapt these behaviors to changes in their environment. Minsky [1988] described the use of an A-Brain and a B-Brain to achieve self-awareness. In this arrangement, the A-Brain’s inputs and outputs are connected to the environment so that it can sense and act upon changes in the environment, while the B-Brain “sees” and influences what happens inside the A-Brain. Bayesian Networks provide the ideal technology to achieve self-awareness or reflection. The Bayesian Agencies uses Bayesian belief propagation to implement the A-Brain, and Bayesian learning to implement the B-brain.

5. BABE: A METHODOLOGY FOR THE ENGINEERING OF EMERGENCE IN COMPLEX ADAPTIVE SYSTEMS

We propose a methodology that uses Bayesian Behavior Networks to model and observe emergent behavior, which we called the BaBe (Bayesian Behavior) methodology. The BaBe models and the relationships between them are illustrated in figure 4.

During the BaBe analysis phase, the following models are defined:

- roles model – this model describes each role in terms of responsibilities, permissions, interactions, and activities.
- interactions model – description of the interactions between agents and the environment.

During the BaBe design phase, the following models are defined:

- agency model - the agents and agencies composing the system, as well as the organization of agencies into heterarchies;
- internal behavior model - the behaviors of and interactions between agents and the environment;
- external behavior model – the behavior(s) associated with each agency.

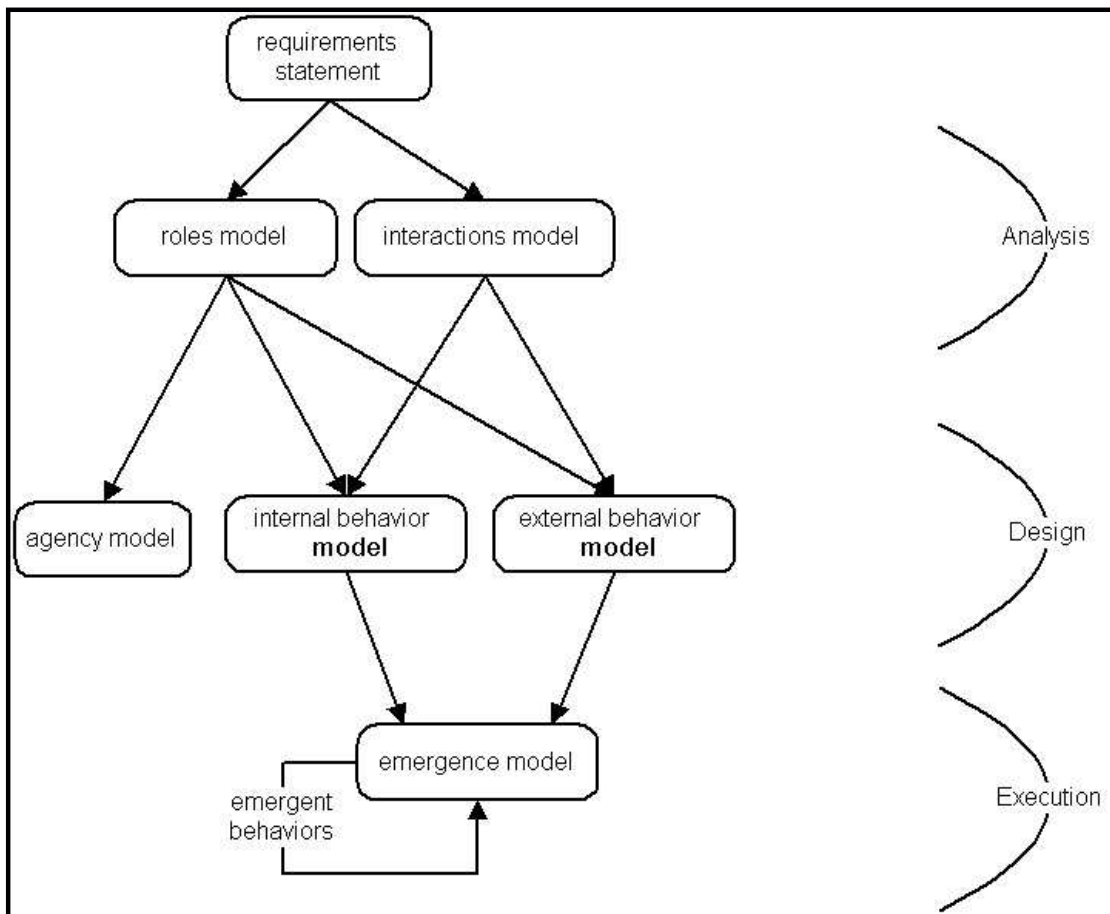


Figure 4 The BaBe Methodology

During the BaBe execution phase, the following model is maintained:

- emergence model – the relationships between local behaviors and emergent global behaviors

The BaBe roles, interactions and agency models are similar to the Gaia models, with the exception that interactions in the BaBe models do not necessarily adhere to interaction protocols. Interaction protocols are not excluded from our models, but are viewed as special cases of interaction handling. The Gaia agent model further applies to autonomous agents only. The BaBe agency model applies to simple or autonomous agents, grouped into agencies. The BaBe internal and external behavior models replace the Gaia services and acquaintance models.

The BaBe models can be defined using the following graphical diagrams, where applicable:

- Use case diagrams
- Class diagrams

- Behavior Diagrams
 - State chart diagrams
 - Activity diagrams
 - Interaction Diagrams
 - Sequence Diagrams
 - Collaboration Diagrams
 - Bayesian Behavior Networks
- Implementation Diagrams
 - Component Diagrams
 - Deployment Diagrams

All the above graphical diagrams, except Bayesian Behavior Networks, are defined in the Unified Modeling Language (UML) defined by the Object Management Group (OMG). A number of researchers view agents as next generation objects or components (Odell, Van Dyke Parunak and Bauer, 2001) and is extending the UML to Agent UML (AUML), which will make it more suitable to the specification of agent interactions and interaction protocols.

We added the Bayesian Behavior Networks to the other behavior diagrams in order to model interactions, behaviors and the relations between them. We use these networks in the BaBe interactions model, the internal and external behavior models as well as in the emergence model.

The emergence model consists of Bayesian Behavior Networks, initialized by the software engineer and maintained by the Bayesian Agencies. In the example Bayesian Behavior Network in figures 2 and 3, possible emergent behaviors include the addition of new hyperlink paths to be traversed by visitors, the discovery of new user profiles and the discovery of new content categories. The Bayesian Agencies will incrementally learn the emergent behaviors by updating the conditional probability tables with the behaviors of each visitor to the website. Adaptive behavior can include the personalized recommendation of other web pages to the visitor.

The conditional probability tables of the Bayesian Behavior Network nodes, as well as the network configuration can be viewed as the language L1 described in the emergence test. Bridging the L1-L2 gap involves the learning of the conditional probability tables from the observed external and internal behaviors, as well as the discovery of new relationships between behaviors. This process is indicated by the

feedback arrows to the emergence model in figure 4. In the next section, the Bayesian Agencies are described in more detail.

6. BAYESIAN AGENCIES – COMPLEX ADAPTIVE SYSTEMS

In Behavior Networks (Maes [1989]), each node determines the activation of a particular behavior as an emergent property of an underlying process. We implement a Bayesian Behavior Network using a heterarchy of Bayesian Agencies. Each Bayesian Agency activates one or more behaviors as emergent properties of the belief propagation in a particular subtree of a Bayesian Behavior Network. For example, in figure 2, a possible set of Bayesian Agencies and their associated subtrees can be:

- Personalization agency implementing the subtree with nodes A, B, G and F;
- Hyperlink agency implementing the subtree with nodes C, D and F;
- Advertisement agency implementing the subtree with nodes E, F and G; and
- Next page agency implementing the subtree with nodes F and H.

The personalization agency can be responsible for the personalization of the web pages depending on the user profile (A), the interest of the user (B) and the buying behavior (G). The hyperlink agency can maintain the reachability of web pages. The advertisement agency can inform the marketing department how informative advertisements were (E), and how it influenced the buying behavior (G). The next page agency can display links to web pages that might interest the user next.

The emergence model will be maintained by heterarchies of Bayesian Agencies that distribute Bayesian Behavior Networks in distributed environments. We developed prototype Bayesian Agencies consisting of simple agents that handle uncertainty by collectively propagating belief in the presence of evidence from their environment. As our agents are simple, and the communication between them minimal, we implemented them as reusable components using Sun's Enterprise JavaBeans™ component architecture (Potgieter and Bishop [2001]).

Figures 5-7 are screen dumps of the prototype components.

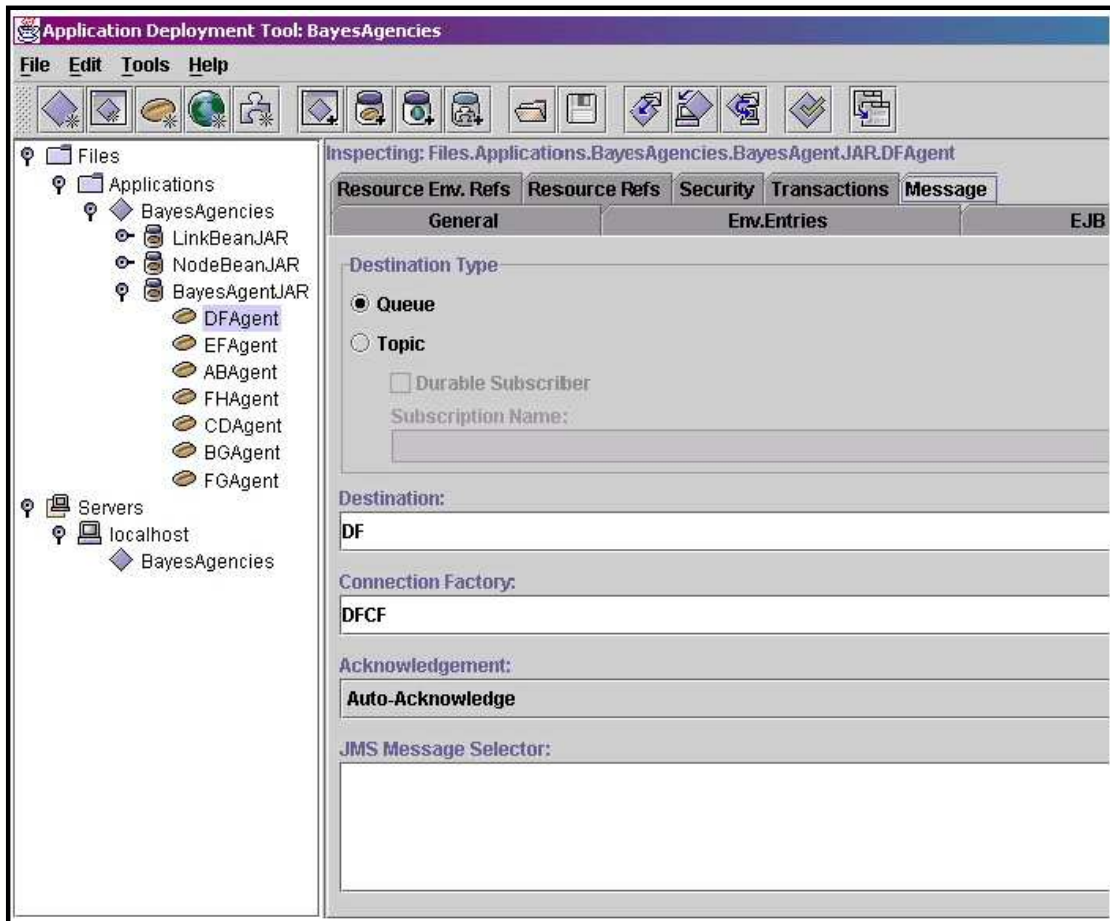


Figure 5: Belief Propagation Agents

The prototype components include

- Belief Propagation Agent – an EJB messagebean per link, its parent node and its child node. Each link is implemented by a JMS queue, and has a corresponding EJB messagebean listening on the queue (see figure 5);
- NodeBean - an EJB entity bean per Bayesian Behavior Network node to administer the conditional probability tables and the beliefs of the nodes, maintained in a database (see figure 6). Each belief propagation agent creates an instance for its parent node and one for its child node;
- LinkBean - an EJB entity bean per Bayesian Behavior Network link to keep track of messages received on links during belief propagation (see figure 6). Each belief propagation agent creates an instance for the link it administers.

The belief propagation agents are organized into Bayesian Agencies. The configuration of the Bayesian Behavior Network as well as the grouping of agents into agencies is maintained in a database. Each Bayesian Agency is responsible for one or

more behaviors depending on the belief propagation in a subtree of the underlying Bayesian Behavior Network.

The belief propagation agents cooperate to obtain the beliefs for each node of the Bayesian Behavior Network in the presence of evidence received from the environment by communicating λ and Π messages amongst themselves as in Judea Pearl's message passing algorithm (Carnegie Mellon University [1991]).

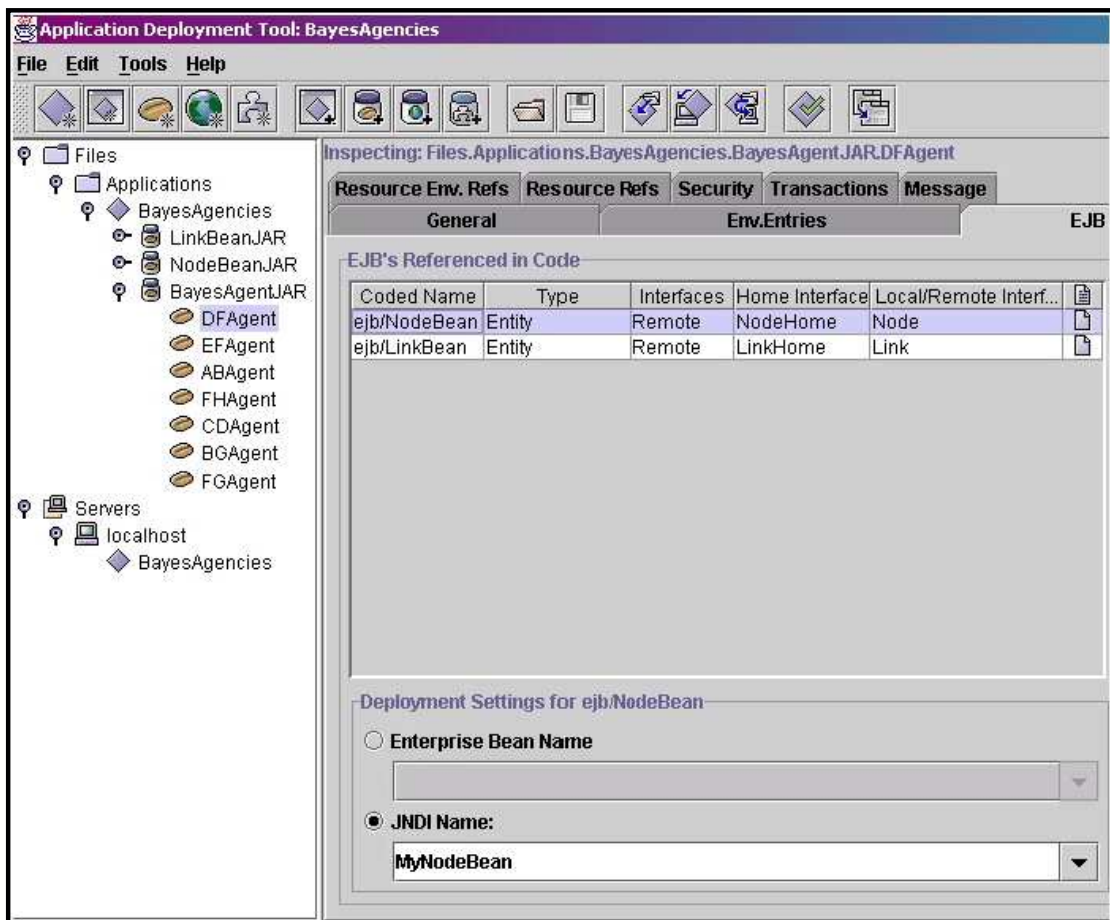


Figure 6: NodeBean and LinkBean

Each belief propagation agent places Π messages on all the outgoing links of the child node and λ messages on all the incoming links of its parent node. For example, figure 7 shows the JMS queues that belief propagation agent DFAgent use (implementing the belief propagation on the link between nodes D and F in figures 2 and 3). This belief propagation agent will communicate Π messages to JMS queues FG and FH and will communicate λ messages to JMS queue CD.

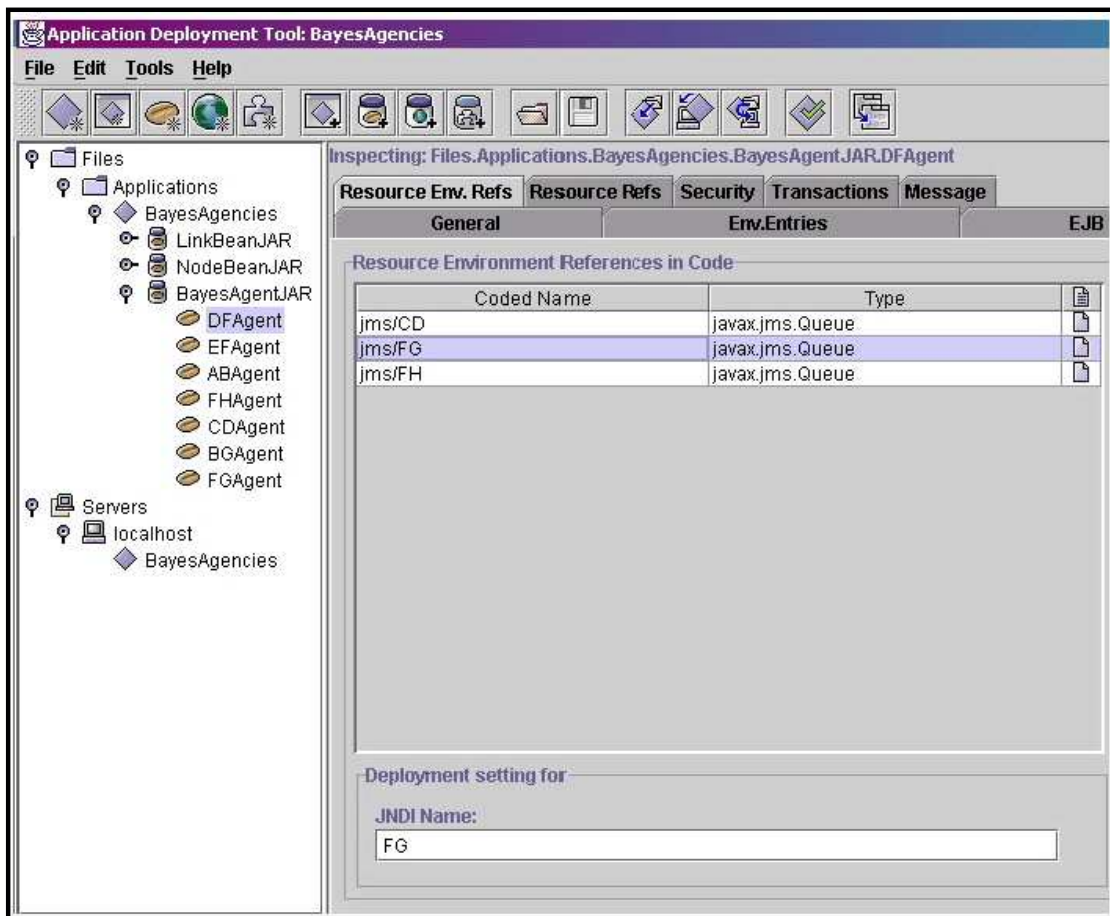


Figure 7: Belief Propagation Agents Neighboring Links

7. FUTURE RESEARCH

With our prototype implementation we demonstrated that it is possible to use a commercially available component architecture to implement an adaptive agent architecture, which can then form the basis of an agent-oriented software engineering methodology. Our Bayesian agents perform collective Bayesian belief propagation in Bayesian Behavior Networks and adapt the behavior of the system in response to evidence from the environment, therefore collectively functioning as a complex adaptive system.

In our prototype, we demonstrated the concepts of collective adaptive behavior only with respect to Bayesian belief propagation in singly connected Bayesian Behavior Networks. In order for a fully functional adaptive agent architecture, we need to implement collective distributed Bayesian learning, and allow the use of Bayesian Behavior Networks that are multiply-connected.

8. CONCLUSION

Agent-oriented software engineering is a new software engineering paradigm used to decompose complex systems into multiple interacting autonomous agents. The agents in these agent architectures are generally complex in order to be autonomous, trying to anticipate all possible interactions amongst themselves and the environment using complex interaction protocols. These agents can usually not cope with emergence, which characterizes complex adaptive systems, as the learning and predictive capabilities that they will need to evolve will make them even more complex.

In this research effort we extended agent-oriented software engineering in order to decompose complex adaptive systems into multiple interacting simple agents, organized into autonomous, adaptive agencies. Collectively these agencies constitute an adaptive agent architecture in which the agencies can cope with emergence through the collective behavior of the simple agents.

In this paper we described the BaBe methodology to analyze, design and implement a complex adaptive system using an adaptive agent architecture. This architecture is adaptive through the integration of specialized distributed Bayesian Networks, which we called Bayesian Behavior Networks, into the agent architecture. As Bayesian Networks are ideally suited for probabilistic reasoning in uncertain environments, this technology is ideally suited to be used by an agent architecture to evolve and adapt its behavior in response to environmental changes.

In the BaBe agent architecture, simple agents called Bayesian Agents, implement the nodes of the Bayesian Behavior Networks. These agents communicate with each other amongst links defined by the topology of the underlying Bayesian Behavior Network. The Bayesian Agents are organized into Bayesian Agencies, where each Bayesian Agency implements a subtree of the underlying network. The agencies can overlap, causing a heterarchy of agencies to be formed. Associated with each agency is a set of actions that must be performed, depending on the beliefs of the agents in the agency, updated during belief propagation in the subtree in response to evidence from the environment.

The BaBe models include various analysis and design models as well a run-time emergence model consisting of Bayesian Behavior Networks, initialized by the software engineer during the analysis and design phases, and maintained by the Bayesian Agencies during the execution phase. We described a prototype implementation of the Bayesian Agencies using Sun's Enterprise JavaBeans™ component architecture.

REFERENCES

- BROOKS, R. A. 1985. A Robust Layered Control System for a Mobile Robot. *MIT AI Memo 864*. Retrieved 18 July 2000, <http://www.ai.mit.edu/people/brooks/papers.html>
- BROOKS, R. A. 1991. Intelligence without Reason. *MIT AI Memo 1293*. Retrieved July 18, 2000, <http://www.ai.mit.edu/people/brooks/papers.html>
- BRYSON, J. J. 2001. Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents. *PhD Dissertation*, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology. Retrieved September 5, 2001, <http://ftp.ai.mit.edu/pub/users/joanna/pdh.pdf>
- CARIANI, P. 1991. *A Review of Emergence and Artificial Life*. Retrieved September 17, 2001, <http://www.arch.usyd.edu.au/~rob/study/EmergenceAndArtificialLife.html>
- CARNEGIE MELLON UNIVERSITY. 1991. *BAYES: Tree-structured Bayesian belief network*. Retrieved May 5, 2001, <http://www.cs.cmu.edu/~mkant/Public/util/areas/reasonng/probabl/bayes/bayes.aug>
- DECHTER, R. 1996. Bucket Elimination: A Unifying Framework for Probabilistic Inference. *Uncertainty in Artificial Intelligence, UAI96*, 211-219. Retrieved October 8, 2000, <http://www.ics.uci.edu/~dechter/publications/>
- DURFEE, E. H. 2001. Scaling up Agent Coordination Strategies. *COMPUTER, IEEE Computer Society*, 34(7), 39-46.
- GRISS, M. L., AND POUR, G. 2001. Accelerating Development with Agent Components. *COMPUTER, IEEE Computer Society*, 34(5), 37-43.
- HEYLIGHEN, F., JOSLYN, C. AND TURCHIN, V. 2001. *Principia Cybernetica Web*. Retrieved March 18, 2002, <http://pcp.lanl.gov>
- JENNINGS, N. R., SYCARA, S. AND WOOLDRIDGE, M. 1998. A Roadmap of Agent Research and Development. *International Journal on Autonomous Agents and Multi-Agent Systems*, 1(1), 7-38. Retrieved July 11, 2000, <http://www.ecs.soton.ac.uk/~nrj/pubs.html#1998>
- JENNINGS, N. R. 2001. An Agent-based Approach for Building Complex Software Systems. *Communications of the ACM*, 44(4), 35-41.
- JENSEN, F., JENSEN, F. V., AND DITTMER, S. L. 1994. From Influence Diagrams to Junction Trees, In *Proceedings of the Tenth Conference of Uncertainty in Artificial Intelligence*. Retrieved February 13, 2001, <http://www.cs.auc.dk/research/DSS/abstracts/jensen:jensen:dittmer:94.html>
- MAES, P. 1989. *How to do the Right Thing*. Retrieved July 18, 2000, <http://agents.www.media.mit.edu/groups/agents/publications/>

- MAES, P. 1994. *Modeling Adaptive Autonomous Agents*. Retrieved June 21, 2000, <http://agents.www.media.mit.edu/groups/agents/publications/>
- MINSKY, M. 1988. *The Society of Mind* (First Touchstone Edition ed.). Simon & Schuster, New York.
- ODELL, J., VAN DYKE PARUNAK, H. AND BAUER, B. 2001. Extending UML for Agents, In *AOIS Workshop at AAAI2000*. Retrieved September 4, 2001, <http://auml.org.auml/working/main.html>
- PEARL, J. AND RUSSEL, S. 2000. Bayesian Networks, *Technical Report R-277, UCLA Cognitive Systems Laboratory*. Retrieved May 5, 2001, http://bayes.cs.ucla.edu/csl_papers.html
- POTGIETER, A. AND BISHOP, J. 2001. Bayesian Agencies on the Internet. In *Proceedings of the 2001 International Conference on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC '2001)*.
- RONALD, E. M., SIPPER, M. AND CAPCARRÈRE, M. S. 1999. An excerpt from Design, observation, surprise ! A test of emergence, *Artificial Life 5(3)*. Retrieved July 3, 2001, <http://www.cs.bgu.ac.il/~sipper/emertest.html>
- RUSSEL, S. J., BINDER, J., KOLLER, D. AND KANAZAWA, K. 1995. Local Learning in probabilistic networks with hidden variables In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Retrieved September 19, 2000, <http://robotics.stanford.edu/~koller/papers/apnijcai.html>
- WOOLDRIDGE, M. 1997. Agent-based Software Engineering. *IEE Proceedings of Software Engineering*, 144(1), 26-37. Retrieved January 26, 2001, <http://www.csc.liv.ac.uk/~mjw/pubs/>
- WOOLDRIDGE, M., JENNINGS, N. R., AND KINNY, D. 2000. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3), 15. Retrieved December 17, 2000, <http://www.ecs.soton.ac.uk/~nrj/pubs.html#1998>
- ZAMBONELLI, F., JENNINGS, N. R., OMICINI, A. AND WOOLDRIDGE, M. 2000. Agent-Oriented Software Engineering for Internet Applications, *Coordination of Internet Agents*. Retrieved January 23, 2001, <http://www.csc.liv.ac.uk/~mjw/pubs/>