UNIT IV

SOFTWARE AGENTS

Architecture for Intelligent Agents – Agent communication – Negotiation and Bargaining – Argumentation among Agents – Trust and Reputation in Multi-agent systems.

4.1 INTRODUCTION TO INTELLIGENT AGENTS

Computers are not very good at knowing what to do: every action a computer performs must be explicitly anticipated, planned for, and coded by a programmer. If a computer program ever encounters a situation that its designer did not anticipate then the result is a system crash at best, loss of everyone's work at worst.

Agents are systems that can *decide for themselves* what they need to do in order to achieve the objectives that we delegate to them. *Intelligent agents,* or sometimes *autonomous agents* are agents that must operate robustly in rapidly changing, unpredictable, or open environments, where there is a significant possibility that actions can *fail*. An expert system is one that is capable of solving problems or giving advice in some knowledge-rich domain.

For example, NASA and the European Space Agency are interested in the possibility of making space probes more autonomous – giving them richer onboard decision-making capabilities and responsibilities. Or an agent can do searches for us.

4.1.1 Agent

An agent *is a computer system that is* situated *in some* environment*, and that is capable of* autonomous action *in this environment in order to achieve its delegated objectives.*

The agent (not intelligent agent) takes sensory input in the form of *percepts* from the environment, and produces as output actions that affect it. The interaction is usually an ongoing, non-terminating one. Actions have preconditions too. *Effectoric capability* is agent's ability to modify its environment. Since the environments are *non-deterministic,*

- agent will not have *complete* control over its environment. It will have at best *partial* control, in that it can *influence.*
- apparently identical circumstances might appear to have entirely different effects.
- it may *fail* to have the desired effect.

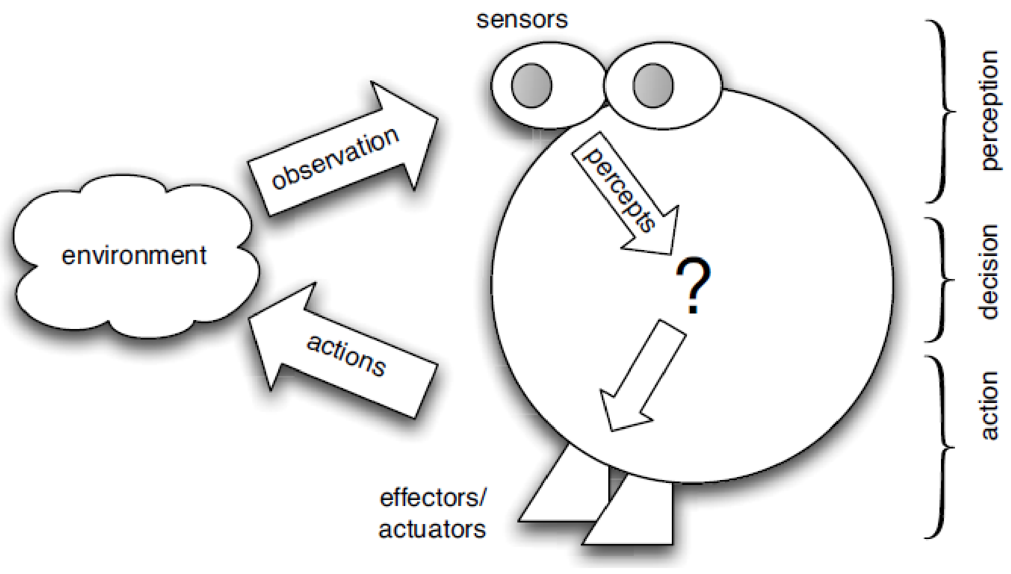

Figure 4.1 Agent and Environment

4.1.2 Environmental Properties

Russell and Norvig suggest the following classification of environment properties.

1. *Accessible* vs. *inaccessible* - *T*he agent can obtain complete, accurate, up-to-date information about the environment's state. Complex environments are inaccessible.
2. *Deterministic vs. non-deterministic* - In deterministic environment any action has a single guaranteed effect – there is no uncertainty. The physical world is non-deterministic.
3. *Episodic* vs. *non-episodic* - In an episodic environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performance of an agent in different scenarios.
4. *Static* vs. *dynamic* - A static environment can be assumed to remain unchanged except for the performance of actions by the agent. A dynamic environment has other processes operating on it, and which hence changes in ways beyond the agent's control.
5. *Discrete vs. continuous* – A Discrete environment has a fixed, finite number of actions and percepts in it. Russell and Norvig give a chess game as an example of a discrete environment, and taxi driving as an example of a continuous one.

4.1.3 Intelligent Agents

An intelligent agent exhibits the following types of behavior in order to meet its delegated objectives.

1. *Pro-activeness*: exhibit goal-directed behavior by *taking the initiative* in order to satisfy their delegated objectives.
2. *Reactivity*: perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their delegated objectives.
3. S*ocial ability*: interacting with other agents (and possibly humans) in order to satisfy their design objectives.

4.1.4 Agents and Objects

Objects are defined as computational entities that *encapsulate* some state, are able to perform actions (*methods)*, on this state, and communicate by message passing. While there are obvious similarities between agents and objects, there are also significant differences.

- Degree to which agents and objects are autonomous - object-oriented programming principle is encapsulation, i.e. objects can have control over their own internal state. Agents performs *requesting* other agents actions to be performed.
- Notions of reactive, proactive, social, and autonomous behavior are where agents act on. OOP's has nothing with those behaviors.
- In the standard object model, there is a single thread of control in the system with recent work in concurrency. A multi-agent system is inherently multithreaded in that each agent is assumed to have its own thread of control, and is continually executing.

4.2 ARCHITECTURE FOR INTELLIGENT AGENTS

Agent architectures are software architectures for decision-making systems that are embedded in an environment. Four important classes of agent architecture are:

1. *Logic-based agents* – Decision about what action to perform is made via logical deduction.
2. *Reactive agents* – Decision making is implemented in some form of direct mapping from situation to action.
3. *Belief-desire-intention agents* – Decision making depends upon the manipulation of data structures representing the beliefs, desires, and intentions of the agent.
4. *Layered architectures* – Decision making is realized via various software layers, each of which is more or less explicitly reasoning about the environment at different levels of abstraction.

4.2.1 Logic-Based Architectures

The "traditional" approach to building artificially intelligent systems suggests that intelligent behavior can be generated in a system by giving that system a *symbolic* representation of its environment and its desired behavior. These symbolic representations are *logical formulae*, and the syntactic manipulation corresponds to *logical deduction*, or *theorem proving*.

To see how an agent might work, we shall develop a simple model of logic-based agents called *deliberate* agents. Such agents are assumed to maintain an internal database of formulae of classical first-order predicate logic, which represents the information they have in symbolic form. For example, an agent's belief database might contain formulae such as the following:

$$Open(valve)$$
$$Temperature(reactor,321)$$
$$Pressure(tank,28)$$

The database is the information that the agent has about its environment. An agent's database is analogous to that of belief in humans. An agent's decision-making process is modeled through a set of *deduction rules*. The *action* takes as input the beliefs of the agent (Δ) and deduction rules (ρ) and returns as output either an action or else null when nothing is found.

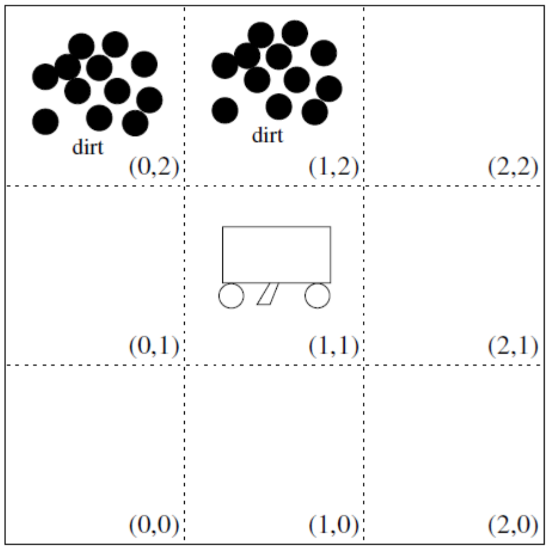

Figure 4.2 Vacuum Cleaning World

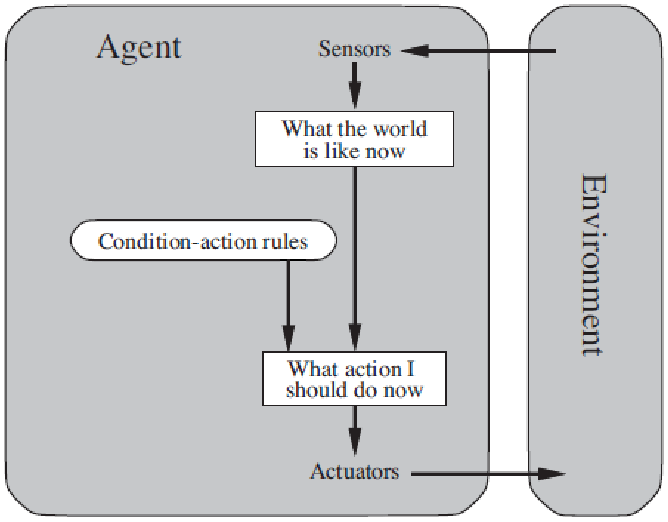

Figure 4.3 Simple Agent Architecture

To illustrate these ideas, let us consider a small example based on the vacuum cleaning world example. The idea is that we have a small robotic agent that will clean up a house. The robot is equipped with a sensor that will tell it whether it is over any dirt, and a vacuum cleaner that can be used to suck up dirt. In addition, the robot always has a definite orientation (North, East, West and South) and turn right 90^{O}. The agent moves around a room, which is divided grid-like into a number of equally sized squares. We will assume that our agent does nothing but clean – it never leaves the room.

To summarize, our agent can receive a percept *dirt*, or *null*. It can perform any one of three possible actions: *forward*, *suck*, or *turn*. The robot will always move from (0,0) to (0,1) to (0,2) and then to (1,2), to (1,1), and so on. The goal is to traverse the room, continually searching for and removing dirt. First, make use of three simple *domain predicates*:

In(*x, y*) agent is at (*x,y*)

Dirt(*x, y*) there is dirt at (*x,y*)

Facing(*d*) the agent is facing direction *d*

The rules that govern our agent's behavior are:

$In(x, y) \wedge Dirt(x, y) \longrightarrow Do(suck)$ ------(4.1)

$In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) \longrightarrow Do(forward)$ ------(4.2)

$In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) \longrightarrow Do(forward)$ ------(4.3)

$In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) \longrightarrow Do(turn)$ ------(4.4)

$In(0,2) \wedge Facing(east) \longrightarrow Do(forward)$ ------(4.5)

Notice that in each rule, we must explicitly check whether the antecedent of rule (4.1) fires. The problems with this vacuum cleaning world are:

1. An agent is said to enjoy the property of ***calculative rationality*** if and only if its decision-making apparatus will suggest an action that was optimal *when the decision-making process began*. Calculative rationality is clearly not acceptable in environments that change faster than the agent can make decisions.
2. Representing and reasoning *temporal information.* Temporal information is how a situation changes over time. Representing it turns out to be extraordinarily difficult.
3. The problems associated with representing and reasoning about complex, dynamic, possibly physical environments are also essentially unsolved.

4.2.2 Reactive Architectures (or) Subsumption Architecture

The *subsumption architecture* is arguably the best-known reactive agent architecture. There are two defining characteristics of the subsumption architecture. The first is a set of *task accomplishing behaviors,* each behavior may be thought of as an individual action selection process, which continually takes perceptual input and maps it to an action to perform. These behaviors are implemented as rules of the form, **situation → action.**

The second defining characteristic of the subsumption architecture is that many behaviors can "fire" simultaneously. There must be a mechanism to choose between the different actions selected by these multiple actions. A *subsumption hierarchy* has behaviors arranged into *layers*. Lower layers in the hierarchy are able to *inhibit* higher layers: the lower a layer is, the higher is its priority, which represent more abstract behaviors. For example, in a mobile robot the behavior "avoid obstacles." It makes sense to give obstacle avoidance a high priority.

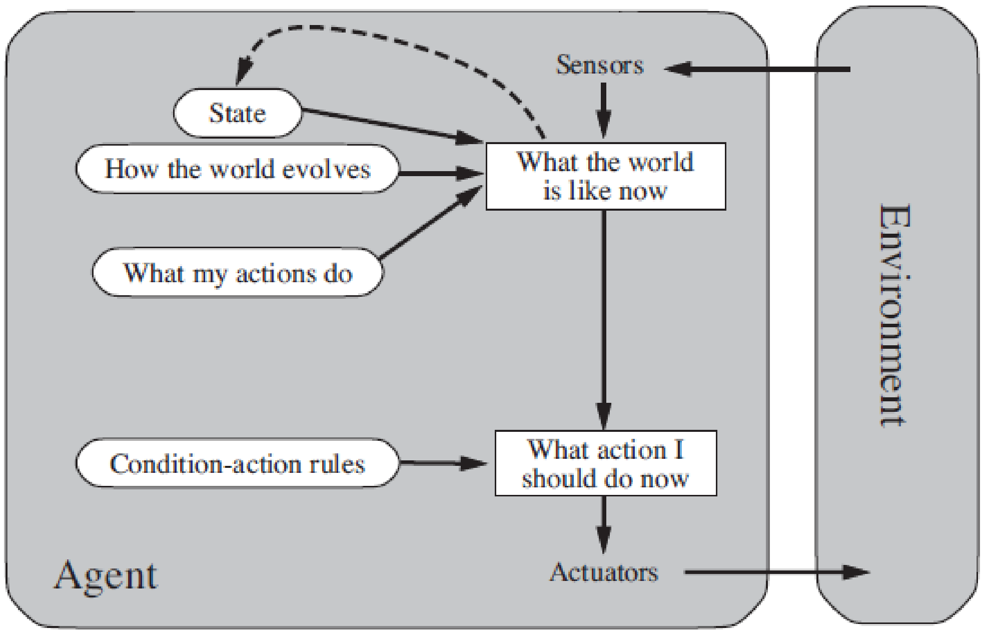

Figure 4.4 Subsumption Architecture

The objective is to explore a distant planet, more concretely, to collect samples of a particular type of precious rock. The location of the rock samples is not known in advance, but they are typically clustered in certain spots. A number of autonomous vehicles are available that can drive around the planet collecting samples and later reenter the mothership spacecraft to go back to earth. There is no detailed map of the planet available, although it is known that the terrain is full of obstacles – hills, valleys, etc. – which prevent the vehicles from exchanging any communication.

The problem we are faced with is that of building agent control architecture for each vehicle, so that they will cooperate among themselves. The solution makes use of two mechanisms introduced by Steels. The first is a *gradient field,* the range of radio signal generated by mothership to find its location. The second is communicate mechanism. agents will carry "radioactive crumbs," which can be dropped, picked up, and detected by passing robots.

The lowest-level behavior is obstacle avoidance, which can be represented in the rule:

if detect an obstacle *then* change direction **------(4.6)**

Other behaviors ensures any samples carried by agents are dropped back at mothership.

if carrying samples *and* at the base *then* drop samples. **------(4.7)**

if carrying samples and *not* at the base *then* travel up gradient. **------(4.8)**

if detect a sample *then* pick sample up **------(4.9)**

if true *then* move randomly. **------(4.10)**

The precondition of 4.10 rule is thus assumed to always fire. These behaviors are arranged into the following hierarchy:

$(4.6) \prec (4.7) \prec (4.8) \prec (4.9) \prec (4.10)$

However, rule 4.8, determining action on carrying sample and not at base is modified as follows.

if carrying samples and *not* at the base *then* drop 2 crumbs *and* travel up gradient. **------(4.11)**

However, an additional behavior is required for dealing with crumbs.

if sense crumbs *then* pick up 1 crumb *and* travel down gradient. **------(4.12)**

These behaviors are then arranged into the following subsumption hierarchy:

$(4.6) \prec (4.8) \prec (4.11) \prec (4.9) \prec (4.12) \prec (4.10)$

Advantages and Disadvantages of Reactive Architecture:

Advantages to reactive approaches with Brooks's subsumption architecture are simplicity, economy, computational tractability, robustness against failure, and elegance all make such architectures appealing. But there are some fundamental, unsolved problems, not just with the subsumption architecture but also with other purely reactive architectures:

- Agents need sufficient information available in their *local* environment.
- It is difficult to see how decision making could take into account *non-local* information.
- Overall behavior *emerges* from the interaction of the component behaviors. But relationship between individual behaviors, environment, and overall behavior is not understandable.
- It is *much* harder to build agents that contain many layers. The dynamics of the interactions between the different behaviors become too complex to understand.

4.2.3 Markov Decision Processes (MDP)

Markov models were originally developed by the Russian mathematician Andrei Markov as models of stochastic process that is, dynamic processes whose behaviors are probabilistic. Markov models are used in operations research for modeling stochastic processes, in which a sequence of decisions must be made over time. The basic components of Markov models are as follows:

An agent in an environment can be any of a set S of **states,** and that the agent can modify its environment by performing **actions** from a set A. A fundamental assumption in Markov models is that the behavior of a system is stochastic, that is, result of performing a particular action in a particular state is not uniquely determined. Here, p(s'| s, a) denote the probability that state s' will result if action a ∈ A is performed in state s.

Next, MDP models make use of the notion of **reward.** We let r(a, s) ∈ R denote the reward obtained from performing action a ∈ A in state s ∈ S. Rewards obtained by performing actions depend solely on the state in which an action is performed, not on what states or actions occur before or afterward. This property is called **Markov assumption.** A policy (sometimes called a decision rule) is essentially a conditional plan that defines an action to perform for every possible state (d : S → A).

Finding an optimal policy using **value iteration** proceeds in two steps. First, we compute the value function, v∗ : S→R, which gives the value v∗(s) of every state s ∈ S. The value v∗ is the expected reward that would be obtained from executing the optimal policy in that state. Now, given the value function v∗, we can then easily "extract" the optimal action.

The algorithm for this is given below; the idea is to iteratively approximate v∗ using two variables v (old) and v∗ (new). We continue to iterate until the difference between the old and new approximation is sufficiently small. "Sufficiently close" means define some convergence threshold ε, and stop when the maximum difference between v and v∗ is ε. When ε = 0, we are requiring exact convergence.

$$d^*(s) = \arg\max_{a \in A} \left(r(s,a) + \lambda \sum_{s' \in S} p(s' \mid s,a) v^*(s') \right)$$

```
function $value\_iteration(S, A, p, r, \lambda)$ return optimal policy $v^*$
    initialize $v^*$ randomly
    repeat
        $v := v^*$
        for each $s \in S$ do
            $v^*(s) := \max_{a \in A} \left( r(a,s) + \lambda \sum_{s' \in S} p(s' \mid s,a) v(s') \right)$
        end-for
    until $v$ and $v^*$ are sufficiently close
    return $v^*$
end function $value\_iteration$
```

Figure 1.4: The value iteration algorithm for Markov decision processes.

4.2.4 Belief-Desire-Intention Architectures

Belief Desire Intention (BDI) architectures have their roots in the philosophical tradition of understanding practical reasoning — the process of deciding, moment by moment, which action to perform in the furtherance of our goals. Practical reasoning involves two important processes:

1. **Deliberation** - deciding what goals we want to achieve, and
2. **Means-ends reasoning** - how we are going to achieve these goals.

The decision process begins by trying to understand what are the options available. After generating this set of alternatives, you must choose between them, called **intentions**, and commit to some, called **future practical reasoning**.

4.2.4.1 Intention

Make a reasonable attempt to achieve the intention. Moreover, if a course of action fails to achieve the intention, then you would expect to try again – you would not expect to simply give up. This intention will constrain future practical reasoning.

From this discussion, we can see that intentions play a number of important roles in practical reasoning:

- Intentions drive means-ends reasoning.
- Intentions constrain future deliberation.
- Intentions persist.
- Intentions influence beliefs upon which future practical reasoning is based.

A key problem in the design of practical reasoning agents is that of achieving a good balance between these different concerns. Specifically, it seems clear that an agent should at times drop some intentions. It follows that, from time to time, it is worth an agent stopping to reconsider its intentions. But reconsideration has a cost, in terms of both time and computational resources. But this presents us with a dilemma. This dilemma is essentially the problem of balancing proactive (goal-directed) and reactive (event-driven) behavior.

- An agent that does not stop to reconsider sufficiently often will continue attempting to achieve its intentions even after it is clear that they cannot be achieved, or that there is no longer any reason for achieving them;
- An agent that constantly reconsiders its intentions may spend insufficient time actually working to achieve them, and hence runs the risk of never actually achieving them.

Let us investigate how bold agents (those that never stop to reconsider) and cautious agents (those that are constantly stopping to reconsider). The rate of world change is γ.

- If γ is low (i.e., the environment does not change quickly) then bold agents do well compared to cautious ones, because cautious ones waste time reconsidering their commitments while bold agents are busy working towards – and achieving – their goals.
- If γ is high (i.e., the environment changes frequently) then cautious agents tend to outperform bold agents, because they are able to recognize when intentions are doomed, and also to take advantage of serendipitous situations and new opportunities.

The lesson is that different types of environments require different types of decision strategies. In static, unchanging environments, purely proactive, goal directed behavior is adequate. But in more dynamic environments, the ability to react to changes by modifying intentions becomes more important.

4.2.4.2 Practical Reasoning

There are seven main components to a BDI agent:

i. A **set of current beliefs,** representing information agent has about its current environment;

ii. A **belief revision function (brf),** which takes a perceptual input and the agent's current beliefs, and on the basis of these, determines a new set of beliefs;

iii. An **option generation function(options),** which determines the options available to the agent (its desires), on the basis of current beliefs and its current intentions;

iv. A **set of current options,** representing possible courses of actions available to the agent;

v. A **filter function (filter),** which represents the agent's deliberation process, and which determines the agent's intentions on the basis of its current beliefs, desires, and intentions;

vi. A **set of current intentions**, representing the agent's current focus – those states of affairs that it has committed to trying to bring about;

vii. An **action selection function (execute),** which determines an action to perform on the basis of current intentions.

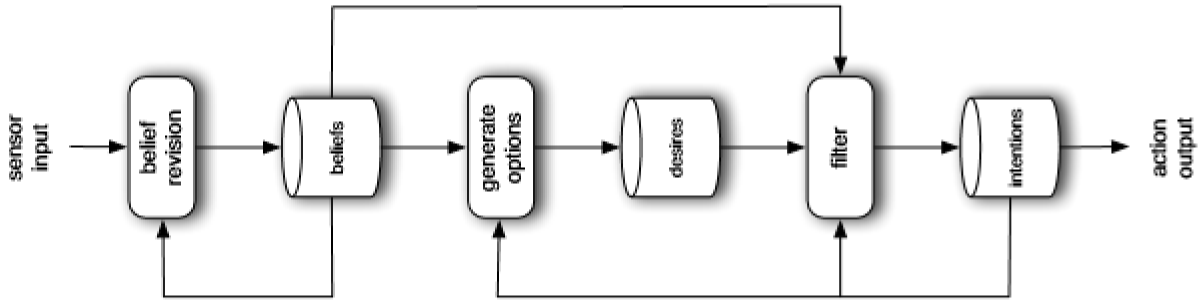

Figure 1.5: Schematic diagram of a generic belief-desire-intention architecture.

The state of BDI agent at any given moment is a triple (B,D,I), where $B \subseteq Bel$, $D \subseteq Des$, and $I \subseteq Int$. If we denote the set of possible percepts that the agent can receive by P, then

$$brf : 2^{Bel} \times P \rightarrow 2^{Bel}$$

$$options : 2^{Bel} \times 2^{Int} \rightarrow 2^{Des}$$

$$filter: 2^{Bel} \times 2^{Des} \times 2^{Int} \rightarrow 2^{Int}$$

Thus filter should satisfy the following constraint:

$$\forall B \in 2^{Bel}, \forall D \in 2^{Des}, \forall I \in 2^{Int}, filter(B, D, I) \subseteq I \cup D.$$

$$execute : 2^{Int} \rightarrow A$$

```
1.  function $action(p)$ returns an action
2.  begin
3.      $B := brf(B,p)$
4.      $D := options(B,I)$
5.      $I := filter(B,D,I)$
6.      return $execute(I)$
7.  end function $action$
```

Figure 4.6 Pseudo-code of function action

4.2.5 Layered Architectures

Layered Architectures involves creating separate subsystems, as a hierarchy of interacting layers, to deal with reactive and proactive behaviors. Two examples of such architectures: INTERRAP and TOURINGMACHINES. There are two types of control flow within layered architectures.

- Horizontal layering - In horizontally layered architectures (Figure 1.6(a)), the software layers are each directly connected to the sensory input and action output. In effect, each layer itself acts like an agent, producing suggestions on action to perform. Advantage – Simplicity. Drawback - overall behavior of the agent will not be coherent.
- Vertical layering - In vertically layered architectures (Figure 1.6(b) and 1.6(c)), sensory input and action output are each dealt with by at most one layer each. A mediator function makes decisions about which layer has "control" of the agent at any given time. Drawback - designer must potentially consider all possible interactions between layers.

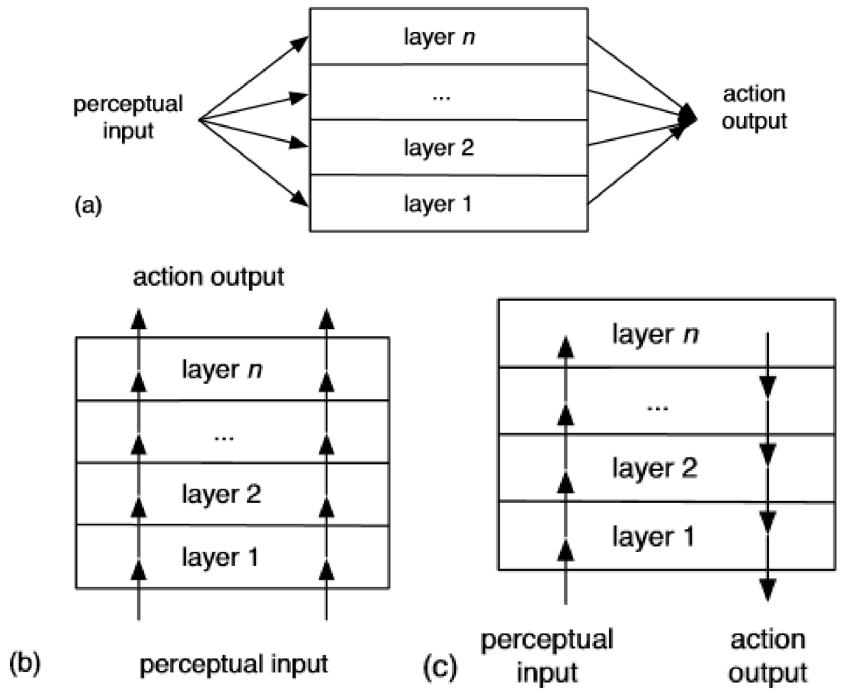

Figure 1.6: Information and control flow in three types of layered agent architectures

Vertically layered architecture is subdivided into one-pass architectures (Figure 1.6(b)) and two-pass architectures (Figure 1.6(c)). The complexity of interactions between layers is reduced in vertical architecture. Since there are n−1 interfaces between n layers, then if each layer is capable of suggesting m actions, there are at most $m^2 (n-1)$ interactions to be considered between layers. This is clearly much simpler than the horizontally layered case. However, this simplicity comes at the cost of some flexibility.

4.2.5.1 Touring Machines

The Touring Machines architecture consists of perception and action subsystems, which interface directly with the agent's environment, and control layers embedded in a control framework, which mediates between the layers. TOURING MACHINES consists of three activity producing layers.

- Reactive layer: immediate response.
- Planning layer: "day-to-day" running under normal circumstances.
- Modelling layer: predicts conflicts and generate goals to be achieved in order to solve these conflicts.
- Control subsystem: decided which of the layers should take control over the agent.

4.2.5.2 INTERRAP

INTERRAP defines an agent architecture that supports situated behavior where the agents are able to recognize unexpected events and react timely and appropriately to them. It show goal-directed behavior in a way that the agent decides which goals to pursue and how. The agents can act under real time constraints and act efficiently with limited resources. The agent can interact with other agents to achieve common goals.

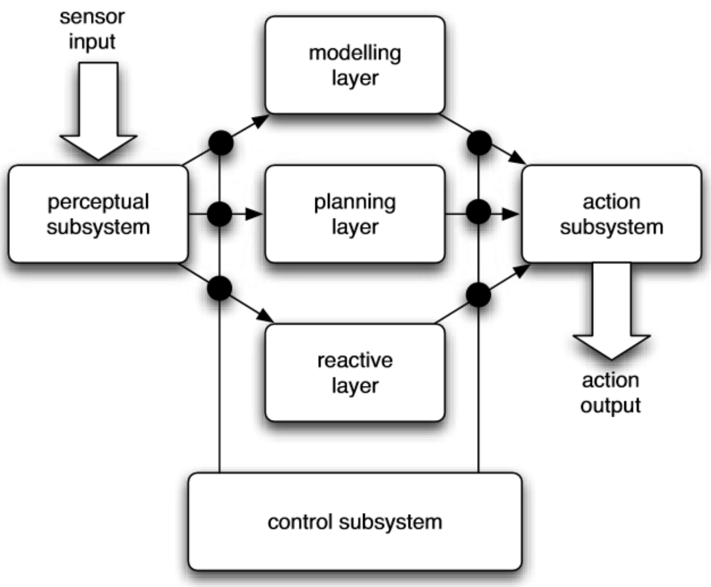

Figure 1.7: TOURINGMACHINES - a horizontally layered agent architecture.

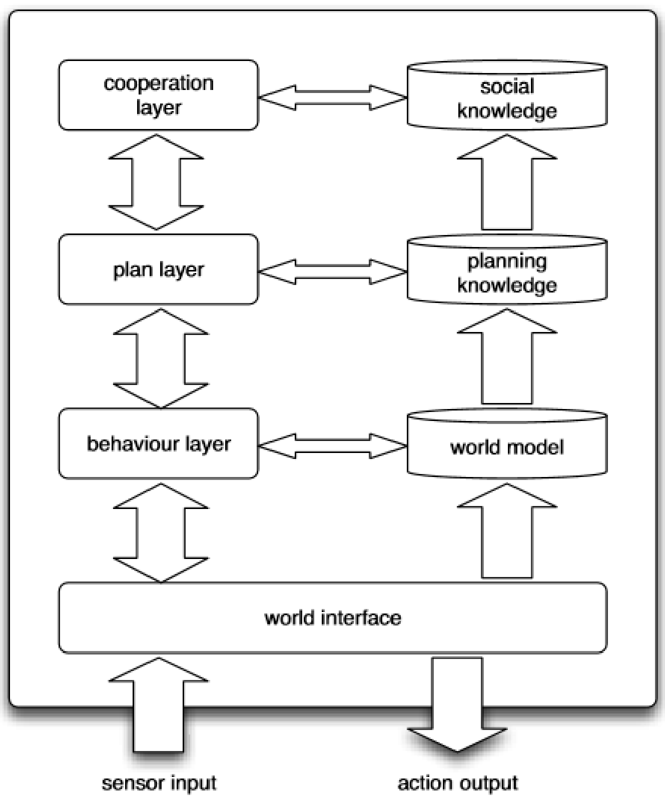

Figure 1.8: INTERRAP - a vertically layered two-pass agent architecture.

Control Component	**Corresponding Knowledge base**	**Function**
Cooperation	Cooperation knowledge	Generate joint plans that satisfy the goals of a number of agents, in response to request from the plan-based component.
Plan-based	Planning knowledge + plan library	Generate single-agent plans to requests from the behavior-based component.
Behaviour-based	World Model	Implement and control the basic reactive capability of the agent. Call on the world interface or a higher-level layer to generate a plan.
World Interface	World Model	Manages the interface between the agent and its environment.

4.3 AGENT COMMUNICATION

Multi-agent System (MAS) are distributed systems. They are collection of collaborative agents. Engineering a multi-agent system means rigorously specifying the communications among the agents by way of interaction protocols. The agents are **autonomous and heterogeneous entities,** and applications are auctions, banking, shipping, and so on. Each application would have its own set of requirements and therefore we would normally find different protocols for each application.

4.3.1 Autonomy and Its Implications

Each agent is an **autonomous entity** in the sense that agent itself is a domain of control, other agents have no direct control over its actions. **Protocols** are modular, potentially reusable specifications of interactions, called messages, between two or more components. To promote reusability, a protocol is specified abstractly with reference to the roles that the interacting components may adopt. A protocol is designed with a certain application in mind. An enactment refers to an execution of the protocol by the components.

As long as components are individually conformant, that is, follow their respective roles in the protocol, they will be able to work together no matter how they are implemented. Interoperation means that the components are loosely coupled with each other; that is, we can potentially replace a component by another conformant one and the modified system would continue to function.

- **The irrelevance of intelligence -** the ability of an agent to perform high-level reasoning or as the degree to which an agent can operate without the supervision of its principal. Two agents may act on two different worlds.
- **Logical versus physical distribution** - Because of their autonomy, agents are the **logical units** of distribution, where two or more agents interacts. Constructs such as processes, are **physical units** of distribution. The choice of whether an application is implemented as a single process or multiple ones is often driven by physical considerations such as geographical distribution, throughput, redundancy, number of available processors and cores, and so on. An agent itself may be implemented via multiple physical units of distribution.

Heterogeneity refers to the diversity of agent implementations. A component can be implemented based on its dependence on other components, without concern for how the others are implemented. Accommodating heterogeneity entails specifying the semantics of the interaction. Making an offer in a Multi-agent system is **social commitment.**

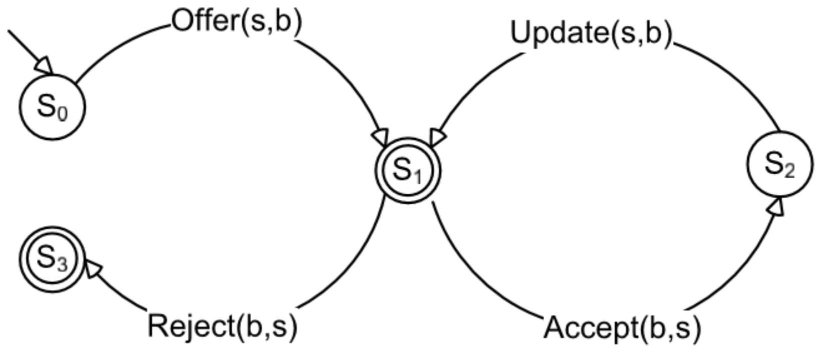

Figure 3.1: Updating an offer.

4.3.2 Criteria for Evaluation

1. **Software engineering –** Protocol specifications should be modifiable, easily understandable, and composable.
2. **Flexibility -** Flexibility is especially important in dynamic settings where agents may come and go, and exceptions and opportunities may arise.
3. **Compliance checking** - Checking an agent's compliance with a protocol means determining if the agent is following the protocol.

4.3.3 Communication in Multi Agent System

The main insight of **Speech act theory,** better known as **communicative act theory,** is that communication is a form of action. For example, the umpire saying, "I declare this player as ejected from the game", is having consequence actions**. Performative verbs** such as, request, promise, etc., makes the sentences stylish.

The agent chooses to inform, request, or promise another agent is entirely within its control, but it may not be the case in real world. It has been customary in agent communication languages to specify a small number of specialized message types as primitives, called **Agent Communication Primitives.** For business applications, today, commitments are the key abstractions employed in the underlying representation. The semantics of the primitives would be expressed in commitments.

4.3.4 Traditional Software Engineering Approaches

Sequence Diagrams - The most natural way to specify a protocol is through a message sequence chart (MSC), formalized as part of UML as sequence diagrams. The roles of a protocol correspond to the lifelines of an MSC; each edge connecting two lifelines indicates a message from a sender to a receiver. Time flows down ward with the orderings of the messages. MSCs support primitives for grouping messages into blocks. FIPA (Foundation of Intelligent Physical Agents) is a standards body, now part of the IEEE, has formulated agent communication standards.

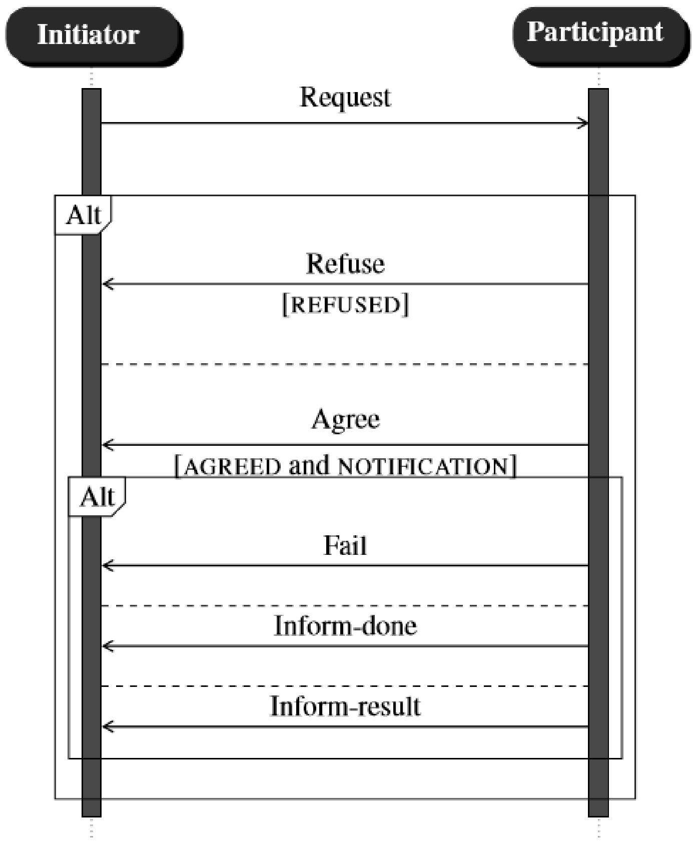

Figure 3.2: FIPA request interaction protocol, expressed as a UML sequence diagram

Choreographies - A choreography is a specification of the message flow among the participants. Typically, a choreography is specified in terms of **roles** rather than the participants themselves. Involving roles promotes reusability by making participants adopt roles. **Orchestration** is workflow, where one party drives all of the other parties.

State Machines - This state machine supports two executions. One execution represents the scenario where the customer rejects the merchant's offer. The other execution represents the scenario where the customer accepts the offer, following which the merchant and the customer exchange the item and the payment for the item. A state machine does not reflect the internal policies based upon which the customer accepts an offer.

4.3.5 Traditional AI Approaches

The AI approaches presume that the agents are constructed based on cognitive concepts, especially, beliefs, goals, and intentions. Then they specify the communication of such agents in terms of cognitive representations. The AI approaches came from two related starting points:

1. **Human-computer interaction and natural language understanding -** A key functionality of such tools was to infer what task the user needed to perform and to help the user accordingly. Such a tool was obviously cooperative. As the tools became more proactive they began to be thought of as agents.
2. **Building distributed knowledge-based systems** - Each agent would include reasoned, knowledge representation and communication are means to share such knowledge.

4.3.5.1 KQML Agent communication language

KQML was created by the DARPA Knowledge Sharing Effort, and was meant to be an adjunct to the other work on knowledge representation technologies, such as ontologies. KQML assumes that each agent maintains a knowledge base described in terms of knowledge (more accurately, belief) assertions. KQML proposed a small number of important primitives, such as query and tell.

Drawbacks:

1. KQML assumes that the communicating agents are cooperative and designed by the same designers. But autonomous agents may generate spurious messages.
2. KQML did not provide clear basis for agent designers to choose which of the message types to use and how to specify their contents.
3. The above challenges complicated interoperability, so that it is impossible for agents developed by different teams to be able to successfully communicate with one another.

4.3.5.2 FIPA ACL

A goal for the FIPA ACL (Agent Communication Language) was to specify a definitive syntax through which interoperability among agents created by different developers could be facilitated and specified the semantics of the primitives. FIPA provides the semiformal specification of an agent management system and also provides definitions for several interaction protocols, based on beliefs and intentions of agents. One might hope that it would be possible to infer the beliefs and intentions of another party, but it is easy to see with some additional reflection that no unique characterization of the beliefs and intentions of an agent are possible.

Drawbacks:

1. If one developer implements all the interacting agents correctly, the developer can be assured that an agent would send a particular message only in a particular internal state. But it is not the case in multi-agent system.
2. FIPA specifications have ended up with a split personality.

4.3.6 Commitment-Based Multi-agent Approaches

Commitment protocols give primacy to the business meanings of service engagements, which are captured through the participants' commitments to one another. Each participant is modeled as an agent; interacting agents carry out a service engagement by creating and manipulating commitments to one another.

A commitment is an expression of the form

C(debtor, creditor, antecedent, consequent)

where debtor and creditor are agents, and antecedent and consequent are propositions. A commitment C(x,y,r,u) means that x is committed to y that if r holds, then it will bring about u. If r holds, then C(x,y,r,u) is detached, and the commitment C(x,y,T,u) holds (T being the constant for truth). If u holds, then the commitment is discharged and does not hold any longer. All commitments are conditional; an unconditional commitment is merely a special case where the antecedent equals T.

Importantly, commitments can be manipulated, which supports flexibility. The commitment operations are listed below: CREATE, CANCEL, and RELEASE are two-party operations, whereas DELEGATE and ASSIGN are three-party operations. DECLARE is not a commitment operation, but may indirectly affect commitments by causing detaches and discharges.

CREATE(x,y,r,u) is performed by x, and it causes C(x,y,r,u) to hold.

CANCEL(x,y,r,u) is performed by x, and it causes C(x,y,r,u) to not hold.

RELEASE(x,y,r,u) is performed by y, and it causes C(x,y,r,u) to not hold.

DELEGATE(x,y,z,r,u) is performed by x, and it causes C(z,y,r,u) to hold.

ASSIGN(x,y,z,r,u) is performed by y, and it causes C(x,z,r,u) to hold.

DECLARE(x,y,r) is performed by x to inform y that the r holds.

Commitment Protocol Specification

Figure 3.5 (left) shows an execution of the protocol and Figure 3.5 (right) its meaning in terms of commitments. The figures depicting executions use a notation similar to UML interaction diagrams. The vertical lines are agent lifelines; time flows downward along the lifelines; the arrows depict messages between the agents; and any point where an agent sends or receives a message is annotated with the commitments that hold at that point. In the figures, instead of writing CREATE, we write Create. We say that the Create message realizes the CREATE operation. Likewise, for other operations and DECLARE.

Offer(*mer*, *cus*, *price*, *item*)	means CREATE(*mer*, *cus*, *price*, *item*)
Accept(*cus*, *mer*, *price*, *item*)	means CREATE(*cus*, *mer*, *item*, *price*)
Reject(*cus*, *mer*, *price*, *item*)	means RELEASE(*mer*, *cus*, *price*, *item*)
Deliver(*mer*, *cus*, *item*)	means DECLARE(*mer*, *cus*, *item*)
Pay(*cus*, *mer*, *price*)	means DECLARE(*cus*, *mer*, *price*)

Table 3.1: A commitment protocol.

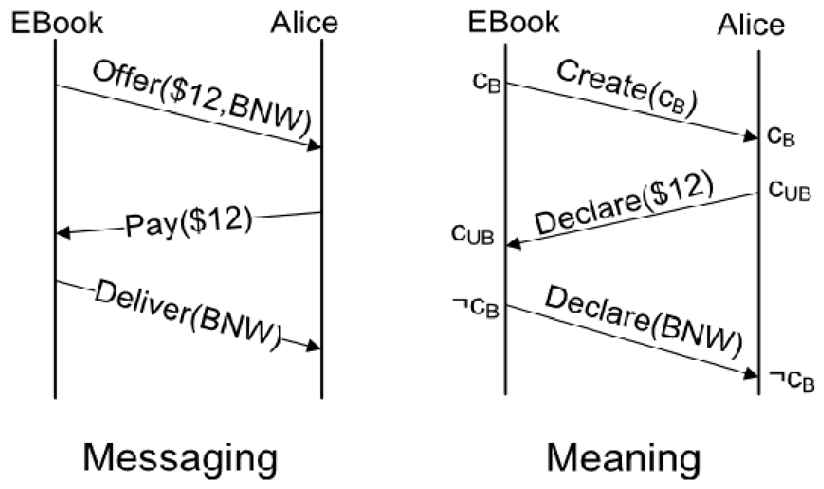

Figure 3.5: Distinguishing message syntax and meaning

	Traditional SE	Traditional AI	Commitment Protocols
Abstraction	control flow	mentalist	business relationship
Compliance	lexical basis	unverifiable	semantic basis
Flexibility	low	low	high
Interoperability	message level	integration	business level

Table 3.2: Comparison of agent communication approaches.

4.3.7 Engineering with Agent Communication

The idea is to use these role specifications as a basis for designing and implementing the agents who would participate in the given protocol. Role specifications are sometimes termed **role skeletons or endpoints,** and the associated problem is called **role generation and endpoint projection.** The agents who correctly implement the roles can interoperate successfully without the benefit of any additional messages than those included in the protocol. The role skeletons are mapped to a simple set of method stubs. An agent implementing a role, by fleshing out its skeleton, provides methods to process each incoming message and attempts to send only those messages allowed by the protocol. Role skeletons do not consider the contents of the messages. As a result, they can be expressed in a **finite-state machine** too.

Programming with Communications:

The Java Agent Development Framework (JADE) is a popular platform for developing and running agent-based applications. It implements the FIPA protocols. JADE provides support for the notion of behaviors. Behavior is an abstract specification of an agent that characterizes important events such as receipt of specified messages and occurrence of timeouts. Behavior involves method definition and callbacks. Skeletons are handlers for any incoming methods.

Modeling Communication

The challenge of specifying the right commitments leads us to guide software engineers in creating appropriate commitment-based specifications. Such guidance is often as state machines and Petri nets that describe interactions in terms of message order and occurrence. For instance, Figure 3.7 shows two common patterns expressed as (partial) state machines. Here, b and s are buyer and seller, respectively. (A) says that the seller may accept or reject an order; (B) says the buyer may confirm an order after the seller accepts it.

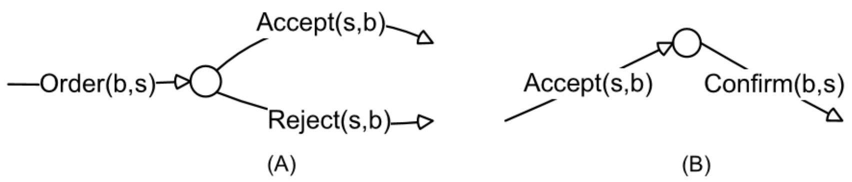

Figure 3.7: Example of operational patterns.

Business patterns characterize requirements that emphasize meanings in terms of commitments. Business patterns support specifying business protocols. These patterns are motivated by the following principles. **Autonomy compatibility** - Autonomy broadly refers to the lack of control: no agent has control over another agent. To get things done, agents set up the appropriate commitments by interacting. Any expectation from an agent beyond what the agent has explicitly committed would cause hidden coupling. **Explicit meanings** - The meaning ought to be made public, not hidden within agent implementations.

Communication Based Methodologies

A number of methodologies for designing and implementing multi-agent systems are based on communications. The common idea behind these methodologies is to identify the communications involved in the system being specified and to state the meanings of such communications. The main protocol concepts are roles, messages, and message meanings. The high-level considerations involved in designing a protocol are:

- Identify stakeholder requirements.
- Identify the roles involved in the interaction.
- If a suitable protocol is available in repository, then choose it and we're done.
- Often the required protocol may be obtained by composing existing protocols.
- Sometimes the protocol or parts of it may need to be written. Identify the communications among the roles and how each message would affect the commitments of its sender and receiver to write a new one.

4.3.8 Advanced Topics and Challenges

- **Primacy of Meaning –** Adopting a meaning-based protocol protects one's models from unplanned dependencies and yields the highest flexibility for the participating agents while maintaining correctness.
- **Verifying Compliance** - When two agents talk to one another, they must agree sufficiently on what they are talking about and verify if the others are complying with expectations of them.
- **Protocol Refinement and Aggregation -** Refinement deals with how a concept refines another in the sense of the is-a hierarchy. Aggregation deals with how concepts are put together into composites in the sense of the part-whole hierarchy.
- **Role Conformance** – Ensure the role as published by a vendor conforms with the role as derived from a protocol.

4.4 NEGOTIATION AND BARGAINING

Negotiation is a form of interaction in which a group of agents with conflicting interests try to come to a mutually acceptable agreement over some outcome. The outcome is typically represented in terms of the allocation of resources (commodities, services, time, money, CPU cycles, etc.). Agents' interests are conflicting in the sense that they cannot be simultaneously satisfied, either partially or fully. Since there are many different possible outcomes, negotiation can be seen as a "distributed search through a space of potential agreements"

Different aspects of Negotiation are (i) the set of possible outcomes; (ii) the agents conducting the negotiation; (iii) the protocol according to which agents search for a specific agreement in this space; and (iv) the individual strategies that determine the agents' behavior.

The set of possible outcomes may be represented as a number representing who gets what amount of resource. It is worth noting that agents may already have a particular allocation of resources before they begin negotiation. Negotiation becomes an attempt to reallocate the resources in order to reach a new allocation that is more preferable to both. In this case, the conflict deal (also known as the no negotiation alternative) refers to the situation in which agents do not reach an agreement in negotiation. The approaches to defining the space of possible outcomes of negotiation are :

1. Task-oriented domains: domains involving the division of tasks to execute.
2. State oriented domains: domains involving a joint decision about what state agents will achieve;
3. Worth-oriented domains: domains involving a joint decision about what goals to achieve.

4.4.1 Game-Theoretic Approaches for Single-Issue Negotiation

Let us begin by considering the following scenario. There is a single resource and there are two agents competing for the resource. Each agent wants to get as large a share of the resource as possible, so there is a conflict between the agents with regard to how the resource must be divided between them. To resolve this conflict, the agents must negotiate or bargain and decide upon a division that will be acceptable to both parties. So each party can only obtain what the other is prepared to allow it. Given this, the negotiation will either end successfully, whereby the parties reach an agreement on a mutually acceptable split, or else it will end in a failure to reach an agreement. In the event of the latter happening, both agents get nothing. Hence, each agent will prefer to get a non-zero share than allow the negotiation to break down.

There are two ways to model such bilateral negotiations: using cooperative game theory and using non-cooperative game theory. In **cooperative games,** agreements are binding or enforceable, possibly by law. When agreements are binding, it is possible for the players to negotiate outcomes that are mutually beneficial. In **non-cooperative games,** agreements are not binding. Here, the players are self-interested and their focus is on individually beneficial outcomes. So a player may have an incentive to deviate from an agreement in order to improve its utility.

	B: Quiet/cooperate	B: Defect/testify against partner
A: Quiet/Cooperate	Both serve 1 month	A: serves 1 year, B: goes free
A: Defect/testify against partner	A: goes free, B: serves 1 year	Both serve three months

Table 4.1: Prisoner's Dilemma game

Consider Prisoner's Dilemma game given in Table 4.1. Assume that this game is non - cooperative. Then the dominant strategy for both players will be to confess. The outcome is not Pareto optimal. In contrast, if the same game was played as a cooperative game, and the players agreed not to confess, then both players would benefit. The agreement (deny, deny) would be binding and the resulting outcome would be Pareto optimal.

Cooperative Models of Single-Issue Negotiation

Consider a two-person bargaining situation with two individuals who have the opportunity to collaborate for mutual benefit. Thus, situations such as trading between a buyer and a seller, or an employer and a labor may be regarded as **bargaining problems.** There will be more than one way of collaborating, and how much an individual benefits depends on the actions taken by both agents. Nash analyzed the bargaining problem and defined a solution/outcome, by determining how much each individual should expect to benefit from the situation, using an **axiomatic approach.**

There are two players (say a and b) who want to come to an agreement over the alternatives in an arbitrary set A. Failure to reach an agreement, i.e., disagreement, is represented by a designated outcome denoted{D}. Agent i $\in$ {a, b}. The set of all utility pairs that result from an agreement is called the bargaining set (S).

Definition: A **bargaining problem** is defined as a pair (S, d). A bargaining solution is a function f that maps every bargaining problem (S, d) to an outcome in S, i.e., f : (S, d)$\rightarrow$Ş

The payoff allocations that two players ultimately get should depend only on the following two factors:

1. The set of payoff allocations that are jointly feasible for the two players in the process of negotiation or arbitration, and

2. The payoffs they would expect if negotiation or arbitration were to fail to reach a settlement.

Axiom 1 (Individual Rationality) - The bargaining solution should give neither player less than what it would get from disagreement, i.e., f(Ş,d)≥d.

Axiom 2 (Symmetry) - When the players' utility functions and their disagreement utilities are the same, they receive equal shares.

Axiom 3 (Strong Efficiency) - The bargaining solution should be feasible and Pareto optimal.

Axiom 4 (Invariance) - The solution should not change as a result of linear changes to the utility of either player.

Axiom 5 (Independence of Irrelevant Alternatives) - Eliminating feasible alternatives that would not have been chosen should not affect the solution.

Non-Cooperative Models of Single-Issue Negotiation

A key difference between the cooperative and non-cooperative models is that the former does not specify a procedure, whereas the latter has a procedure. There are two players and a unit of good, an apple, to be split between them. If player a gets a share of $x_a \in [0,1]$, then player b gets $x_b = 1 - x_a$. Neither player receives anything unless the two players come to an agreement. Here, the apple, can be split between the players. So the issue is said to be divisible.

Player b can accept or reject the offer. If player b accepts, the game ends successfully with the pie being split as per player a's proposal. Otherwise, the game continues to the next time period in which player b proposes a counteroffer to player a. This process of making offers and counteroffers continues until one of the players accepts the other's offer. Since the players take turns in making offers, this is known as **alternating offers protocol.**

4.4.2 Game-Theoretic Approaches for Multi-Issue Negotiation

The four key procedures for bargaining over multiple issues.

1. **Global bargaining:** Here, the bargaining agents directly tackle the global problem in which all the issues are addressed at once.

2. **Independent/separate bargaining:** Here negotiations over the individual issues are totally separate and independent, with each having no effect on the other.
3. **Sequential bargaining with independent implementation**: Here the two parties consider one issue at a time. There are several forms of the sequential procedure. These are defined in terms of the *agenda* and the *implementation rule*. For sequential bargaining, the agenda specifies the order in which the issues will be bargained. The implementation rule specifies when an agreement on an individual issue goes into effect. There are two implementation rules that have been studied in the literature: the rule of *independent implementation* and the rule of *simultaneous implementation*. For the former, an agreement on an issue goes into effect immediately
4. **Sequential bargaining with simultaneous implementation:** An issue does not take effect until an agreement is reached on all the subsequent issues.

Cooperative Models of Multi-Issue Negotiation

In addition to the efficiency, invariance, and symmetry axioms, these include the following:

1. **Simultaneous implementation agenda independence:** Global bargaining and sequential bargaining with simultaneous implementation yield the same agreement.
2. **Independent implementation agenda independence**: Global bargaining and sequential bargaining with independent implementation yield the same agreement.
3. **Separate/global equivalence:** Global bargaining and separate bargaining yield the same agreement.

Non-Cooperative Models of Multi-Issue Negotiation

The package deal procedure is similar to the alternating offers protocol in that the parties take turns in making offers. However, here, an offer must include a share for each issue under negotiation. An agent must now make trade-offs across the issues in order to maximize its cumulative utility.

Heuristic Approaches for Multi-Issue Negotiation

Since the problem of finding optimal agendas may be computationally hard, a heuristic approach will be useful for solving this problem. In order to generate a counteroffer for an issue under negotiation, there are three types of strategies: *time dependent*, *resource dependent*, and *behavior dependent*, or *imitative*, strategies.

1. A time dependent strategy takes "time" as an input and returns an offer such that concessions are small at the beginning of negotiation but increase as the deadline approaches.
2. Resource dependent strategies are also similar to the time dependent functions except that the domain of the function is the quantity of resources available instead of the remaining time.
3. Behavior dependent strategy simply imitates its opponent's strategy in order to protect itself from being exploited by the opponent.

Theorem 4.1 *For the package deal procedure, the following strategies form a subgame perfect equilibrium. The equilibrium strategy for $t = n$ is:*

$$\text{STRATA}(n) = \begin{cases} \textit{OFFER } (\delta^{n-1}, \mathbf{0}) & \textit{If a's turn to offer} \\ \textit{ACCEPT} & \textit{If b's turn to offer} \end{cases}$$

$$\text{STRATB}(n) = \begin{cases} \textit{OFFER } (\mathbf{0}, \delta^{n-1}) & \textit{If b's turn to offer} \\ \textit{ACCEPT} & \textit{If a's turn to offer} \end{cases}$$

where $\mathbf{0}$ is a vector of m zeroes. For all preceding time periods $t < n$, the strategies are defined as follows:

$$\text{STRATA}(t) = \begin{cases} \textit{OFFER } \text{SA}(\text{TA}(t)) & \textit{If a's turn to offer} \\ \textit{If } (U^a(x^a, t) \geq \text{UA}(t+1)) & \textit{If a receives an offer } (x^a, x^b) \\ \quad \textit{ACCEPT} & \\ \textit{else REJECT} & \end{cases}$$

$$\text{STRATB}(t) = \begin{cases} \textit{OFFER } \text{SB}(\text{TB}(t)) & \textit{If b's turn to offer} \\ \textit{If } (U^b(x^b, t) \geq \text{UB}(t+1)) & \textit{If b receives an offer } (x^a, x^b) \\ \quad \textit{ACCEPT} & \\ \textit{else REJECT} & \end{cases}$$

where $\text{UA}(t)$ ($\text{UB}(t)$) denotes a's (b's) equilibrium utility for time t. An agreement takes place at $t = 1$.

Heuristics can also be used to predict the opponent's preferences for the issues. This prediction is relevant to situations where the negotiators have limited information about their opponents. Here, any information gained from the opponent's offers in the past rounds is used to heuristically form a prediction of the future. An agent's preferences will be revealed when it makes offers to the mediator, because an agent will only propose those offers that are optimal from its individual perspective.

Negotiating with Humans

When agents negotiate with humans, a completely new challenge arises. This is because, humans make systematic deviations from the optimal behavior prescribed by normative theory. For example, people often change their valuations based on how the choices are framed, are opposed to inequity, and are willing to engage in irrational behavior such as costly punishment

One of the earliest agents capable of negotiating with humans was designed by Kraus and Lehmann to play the game *Diplomacy* using a variety of heuristics. Surprisingly, humans were unable to discern whether they were playing with a human or an agent. More recently introduced agents use reinforcement learning to participate in single-shot auctions or games, and have been shown to achieve higher payoffs than humans

Argumentation-Based Negotiation

Automated negotiation are characterized by the exchange of offers between parties with conflicting positions and are commonly referred to as *proposal-based* approaches. That is, agents exchange proposed agreements in the form of bids or offers and when proposed deals are not accepted, the possible response is either a counterproposal or withdrawal.

Argumentation-based negotiation (ABN) approaches, on the other hand, enable agents to exchange additional *meta*-information (i.e., arguments) during negotiation. Consider the case in which an agent may not be aware of some alternative plans of achieving some goal. Exchanging this information may enable agents to reach agreements not previously possible. This was shown through the well known painting/mirror hanging example. The example concerns two home-improvement agents – agent *i* trying to hang a painting, and agent *j* trying to hang a mirror.

There is only one way to hang a painting, using a nail and a hammer. But there are two ways of hanging a mirror, using a nail and a hammer or using a screw and a driver, but *j* is only aware of the former. Agent *i* possesses a screw, a screw driver, and a hammer, but needs a nail in addition

to the hammer to hang the painting. On the other hand, *j* possesses a nail, and believes that to hang the mirror, it needs a hammer in addition to the nail. Now, consider the dialogue depicted in Figure 4.3 between the two agents.

Drawback is that agents may withhold or misreport arguments in order to influence the negotiation outcome to their own advantage.

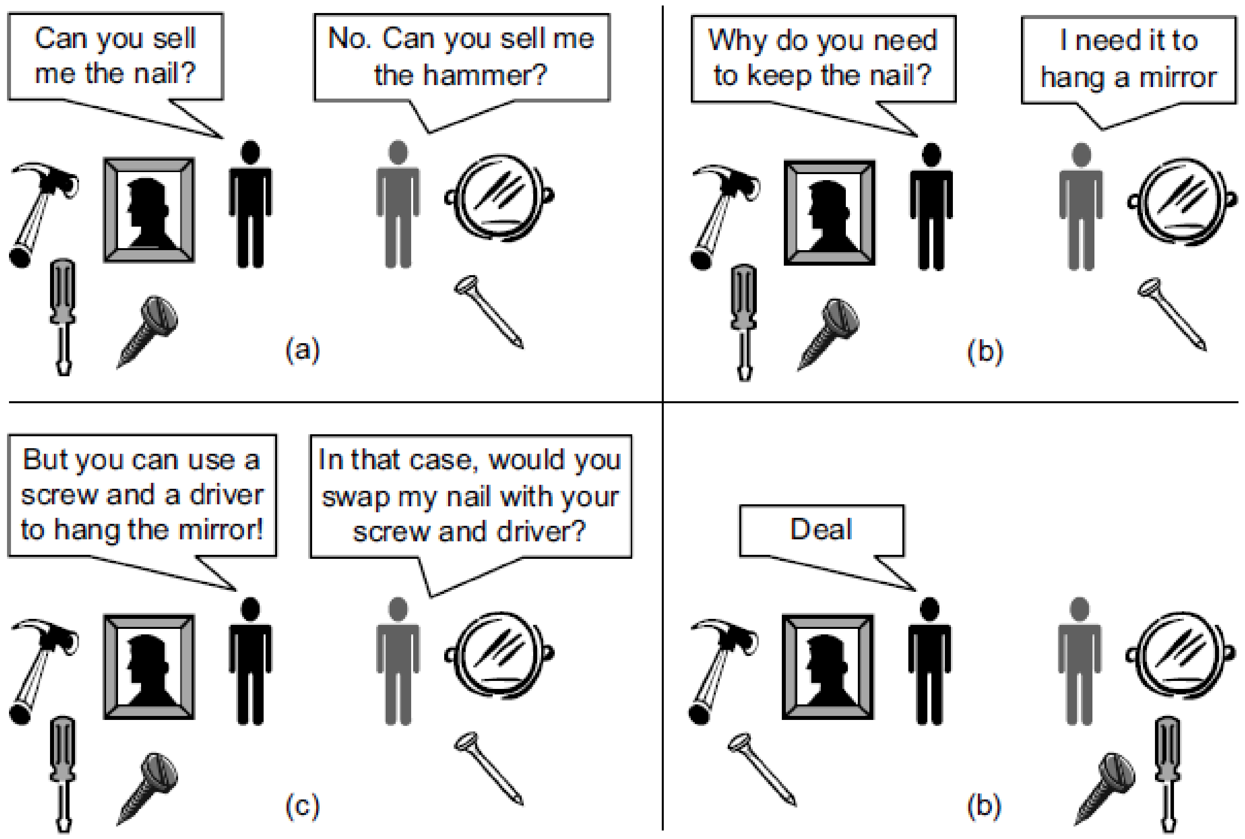
Can you sell me the nail?
No. Can you sell me the hammer?
(a)
Why do you need to keep the nail?
I need it to hang a mirror
(b)
But you can use a screw and a driver to hang the mirror!
In that case, would you swap my nail with your screw and driver?
(c)
Deal
(b)

Figure 4.3: Dialogue between agents i (black) and j (gray).

4.5 ARGUMENTATION AMONG AGENTS

Argumentation can be defined as the interaction of different arguments for and against some conclusion. In multi-agent systems, argumentation has been used both for automating individual agent reasoning, as well as "rational interaction" (i.e., interaction which involves the giving and receiving of reasons). This is because argumentation provides tools for designing, implementing and analyzing sophisticated forms of interaction among rational agents.

Argumentation can be seen as a reasoning process consisting of the following four steps:

1. Constructing arguments (in favor of/against a “statement”) from available information.
2. Determining the different conflicts among the arguments.
3. Evaluating the acceptability of the different arguments.
4. Concluding, or defining, the justified conclusions.

Arguments as Chained Inference Rules

A number of recent attempts have been made to provide a general, unifying definition. Prakken's recent unifying framework is quite general and highlights most important concepts. Prakken defines an argumentation system as a tuple (L,−,Rs,Rd,≤), consisting of a logical language L, two disjoint sets of strict rules Rs and defeasible rules Rd,and a partial order ≤ over Rd. The contrariness function: L → 2L captures conflict between formulas, with classical negation ¬ being captured by ¬ϕ ∈ ϕ and ϕ ∈ ¬ϕ. A particular knowledge base is a pair (K, ≤'), with K ⊆ L divided into the following disjoint sets: Kn are the necessary axioms (cannot be attacked); Kp are the ordinary premises; Ka are the assumptions; Ki are the issues. Finally, ≤' is a partial order on K\Kn.

From a knowledge base, arguments are built by applying inference rules to subsets of K. The left-hand side of the rule ϕ1,...,ϕn is called the premises (or antecedents),while the right-hand side ϕ is called the conclusion (or consequent). An argument is intuitively any of the following structures:

Φ ∈ K, where Prem(A)={ϕ}, Conc(A) = ϕ, and Sub(A) = {ϕ}.

A1,...,An → ψ, where A1,...,An are arguments, and there exists in Rs a strict rule Conc(A1),...,Conc(An) → ψ.

A1,...,An ⇒ψ, where A1,...,An are arguments, and there exists in Rd a defeasible rule Conc(A1),...,Conc(An)→ψ.

Argument as an Instance of a Scheme

Argumentation schemes are forms (or categories) of argument, representing stereotypical ways of drawing inferences from particular patterns of premises to conclusions in a particular domain. Walton's "sufficient condition scheme for practical reasoning" may be described as follows [1]: In the current circumstances R, We should perform action A, Which will result in new circumstances S, Which will realize goal G, Which will promote some value V.

Abstract Arguments

An argument can be seen as a node in an argument graph, in which directed arcs capture defeat between arguments. Despite its simplicity, this model is surprisingly powerful. Argument α1 has two defeaters (i.e., counterarguments) α2 and α4, which are themselves defeated by arguments α3 and α5, respectively.

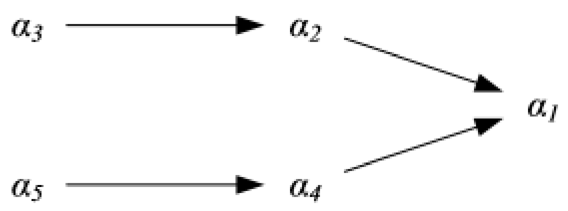

Figure 5.1: A simple argument graph.

4.5.1 Evaluating an Argument

To evaluate whether an argument is acceptable, take into account how it interacts with other arguments. This turns out to be a non-trivial problem, and the source of much research and controversy

Definition 5.2(Conflict-free, defense)

Let A be an argumentation framework and let $S \subseteq A$ and let $\alpha \in A$.

- S is conflict-free if $S \cap S+ = \theta$.
- S defends argument α if $\alpha' \subseteq S^+$. Argument α is acceptable with respect to S.

Definition 5.6(Argument labeling)

Let AF = A,labeling is a total function $L : A \rightarrow \{in, out, undec\}$such that:

$\forall\ \alpha \in A : (L(\alpha) = out \equiv \exists\beta \in A$ such that $(\beta \quad \alpha$ and $L(\beta) = in))$

$\forall\ \alpha \in A : (L(\alpha) = in \equiv \forall\beta \in A : ($ if $\beta \quad \alpha$ then $L(\beta) = out))$

Otherwise, $L(\alpha)$ = undec (since L is a total function).

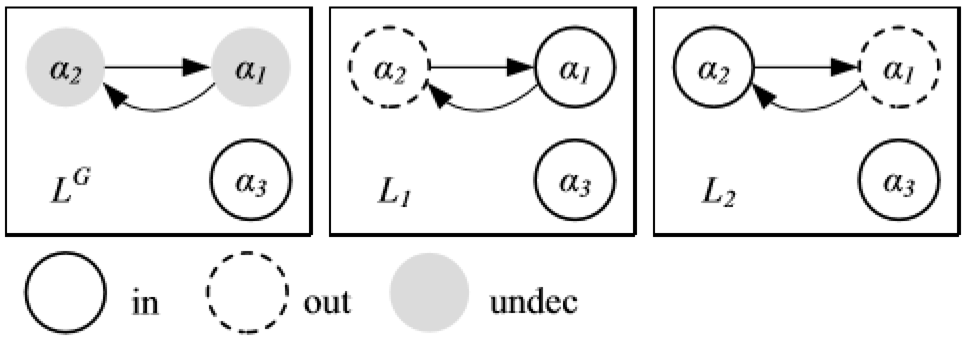

Figure 5.2: Graph with three labelings / complete extensions.

4.5.2 Argumentation Protocols

An argumentation protocol is, therefore, a set of rules that govern the argumentation process. For example consider Abstract Argument Games based on argumentation protocols. Rules for governing argumentation can be specified without regard to the specific internal structure of the arguments. These protocols typically assume two agents, one PRO (the proponent) and the other OPP (the opponent). Dialogue begins with PRO asserting an argument x. Then PRO and OPP take turns, in a sequence of moves called a dispute, where each player makes an argument that attacks its counterpart's last move. A player wins a dispute if its counterpart cannot make a counterattack. But the counterpart may try a different line of attack, creating a new dispute. This results in a dispute tree structure that represents the dialogue.

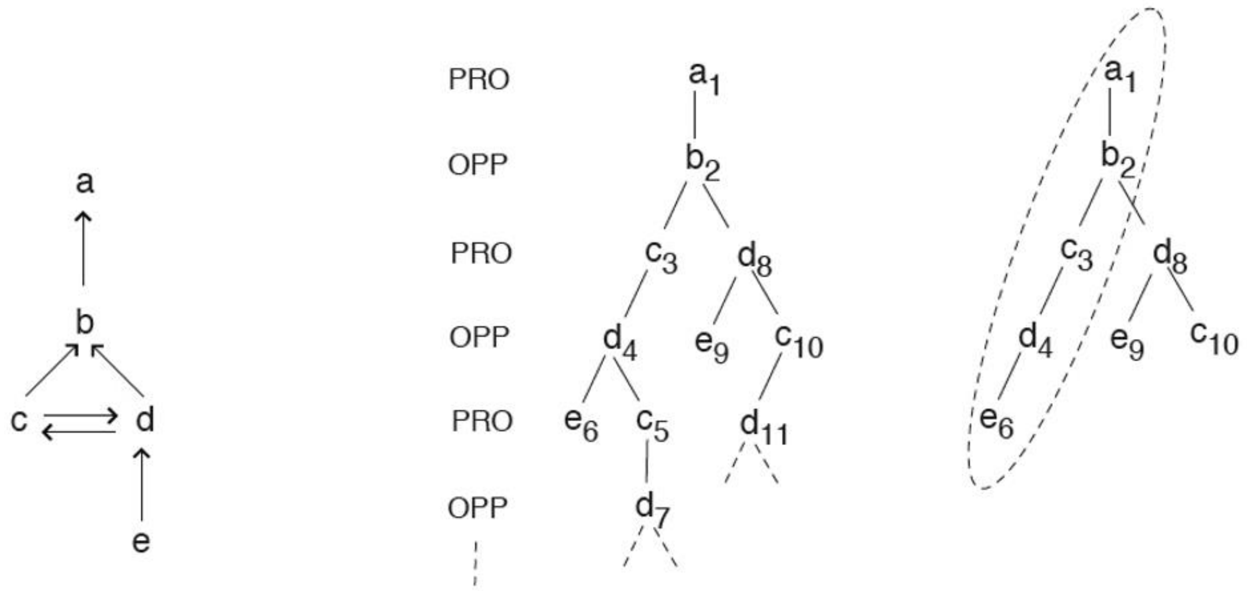

Figure 5.3: (i) Argumentation framework, (ii) Dispute tree induced in a, and (iii) Dispute tree induced by under protocol G, with the winning strategy encircled.

Definition 5.10 (Protocol G) Given argument graph A, and a dispute D whose tail is argument x. Let PRO(D) be the arguments uttered by PRO.

- If dispute length is odd (next move is by OPP), then possible next moves are{y|y→x}.
- If the dispute length is even (next move is by PRO), then the possible next moves are {y|y→x and y ! ∈ PRO(D)}.

Theorem 5.1 Let A,→ be an argument graph. There exists a winning strategy T for x under protocol G such that the set of arguments uttered by PRO in T is conflict-free, if and only if x is in the grounded extension of A,→.

Dialog System relies on having the explicit contents of the arguments presented.

P1[−]: claim r	O2[P1]: why r
P3[O2]: r since q,q⇒r	O4[P3]: why q
P5[O4]: q since p,p⇒q	O 6[P5]: concede p⇒q
	O 7[P5]: why p

Note that at this point, player P has many possible moves. It can retract its claim or premises of its argument, or give an argument in support of p. It was also possible for player O to make queries against P's original claim.

Acts	Intuitive Meaning	Attacks	Surrenders
claim φ	Assert φ is true	*why* φ	*concede* φ
φ *since S*	Support φ by argument *S*	*why* ψ(ψ ∈ *S*) φ′ *since S*′ (defeats φ *since S*)	*concede* ψ (ψ ∈ *S*) *concede* φ
why φ	Challenge φ	φ *since S*	*retract* φ
concede φ	Concede φ claimed by other		
retract φ	Take back own claim φ		

Table 5.2: An example in Prakken's framework

4.5.3 Strategic Argumentation and Game Theory

Researchers started exploring strategies for agents to choose their next moves. For example, in their dialogue system, Parsons et al. defined a number of attitudes of an agent, which specify the criteria according to which it evaluates and asserts arguments. These attitudes assume the availability of a preference relation over arguments, which can be used to compare their relative strength.

In terms of asserting arguments, **a confident agent** can assert any proposition for which it can construct an argument, while **a careful agent** can do so only if it can construct such an argument and cannot construct a stronger counterargument. **A thoughtful agent,** on the other hand, can assert a proposition only if it can construct an acceptable argument for the proposition.

Conversely, when it comes to evaluating arguments presented by the opponent, an agent can have one of the following attitudes. **A credulous agent** accepts a proposition if it can construct an argument for it, while **a cautious agent** does so only if it is also unable to construct a stronger counterargument. **A skeptical agent,** however, accepts an argument only if it can construct an acceptable argument for the proposition.

Generally, game theory can be used to achieve two goals:

1. Undertake precise analysis of interaction in particular strategic settings, with a view to predicting the outcome;
2. Design rules of the game in such a way that self-interested agents behave in some desirable way (e.g., tell the truth); this is called mechanism design.

Glazer and Rubinstein's Model - Glazer and Rubinstein's Model explore the mechanism design problem of constructing rules of debate that maximize the probability that a listener reaches the right conclusion given arguments presented by two debaters.

Game Theory Background – The field of game theory studies strategic interactions of self-interested agents. There is a set of self-interested agents, whose preferences are outcomes over set of all possible outcomes. When agents interact, we say that they are playing strategies. A strategy for an agent is a plan that describes what actions the agent will take for every decision the opponent agent might make. Strategy profile denote the outcome that results when each agent is playing strategy, which is used as the utility for future moves.

A solution called, **Nash equilibrium** is a strategy profile in which each agent is following a strategy that maximizes its own utility, given its type and the strategies of the other agents.

- **Mechanism Design -** Mechanism design ensures that a desirable system-wide outcome or decision is made when there is a group of self-interested agents who have preferences over the outcomes.
- **The Revelation Principle -** The revelation principle states that we can limit our search to a special class of mechanisms. A social choice function is a rule $f : \Theta 1 \times ... \times \Theta I \rightarrow O$, that selects some outcome $f(\theta) \in O$, given the type of the agent. **Theorem5.2 (Revelation principle)** If there exists some mechanism that implements social choice function f in dominant strategies, then there exists a direct mechanism that implements f in dominant strategies and is truthful.

4.5.4 The Argument Interchange Format

Argumentation Interchange Format (AIF) provides a common language for argument representation, to facilitate argumentation among agents in an open system. The core AIF has two types of nodes: information nodes (or I-nodes) and scheme nodes (or S-nodes).

Information nodes are used to represent passive information contained in an argument, such as a claim, premise, data, etc. **Scheme nodes** capture the application of schemes (i.e., patterns of reasoning). The present ontology has three different types of scheme nodes: rule of inference application nodes (RA-nodes), preference application nodes (PA nodes) and conflict application nodes (CA-nodes).

4.6 TRUST AND REPUTATION IN MULTI-AGENT SYSTEMS

Trust is the outcome of observation leading to the belied that the actions of another may be relied upon, without explicit guarantee to attain the goal. The field of **trust-based computing** has been interested in developing secure systems aimed at preventing a set of well-defined attacks. Proposed solutions often rely on cryptography algorithms.

Even if these techniques can be used to ensure specific properties such as authentication or message integrity, they do not secure a multi-agent system regarding aspects like the truth of messages or the subjective fulfillment of service. The term **soft security**refer to control techniques that provide a degree of protection without being too restrictive for the system development.

4.6.1 Computational Representation of Trust and Reputation Value

There is no unique way to represent trust and reputation values. Existing models use different formalisms, its choice being justified by the type of reasoning the agents have to do. The two main characteristics of this choice are the **simplicity** and **the expressiveness of values.**

- **Boolean Representation** - True means the trustee is trustworthy, while False means the other way around. This representation is very limited and rarely used in current models.
- **Numerical Values** - The trust in an agent X is 0.4 or that the reputation of agent Y is −1. This is by far the most used representation. We can then say that two agents A and B are trusted (distrusted) but that we trust (distrust) A more than B if its trust value is higher (lower) than the one attached to B. Mostly used range is [-1, 1]
- **Qualitative Labels** – Is a trust of 0.6 really different from a trust of 0.7 in terms of making trust decisions? Probably not, so in order to express this lack of precision inherent to both concepts, some models use finite sets of labels in an ordered set to give value to trust and reputation. Usually the set {bad, neutral, good} or {very_bad, bad, neutral, good, very_good} are used.
- **Probability Distribution and Fuzzy Sets** - A discrete probability distribution over a sorted discrete set is a richer representation of reputation value. Figure 9.1-b states that the behavior of the agent has a probability of 0.5 of being very bad and a probability of 0.5 of being very good, that is, an agent that is capable of the best and the worse but will never behave in between. Fuzzy set adds degree of precision to the reputation value. (For example "Ram is tall with probability 0.5").

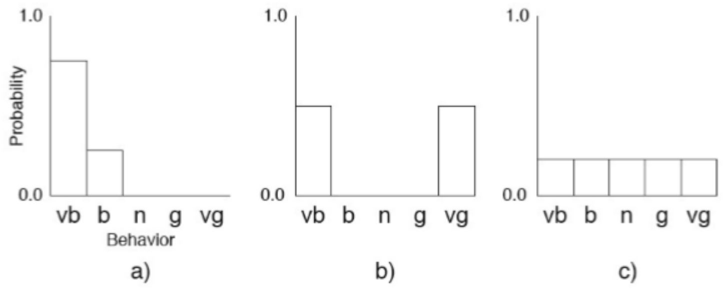

Figure 9.1: Representing reputation as a probability distribution.

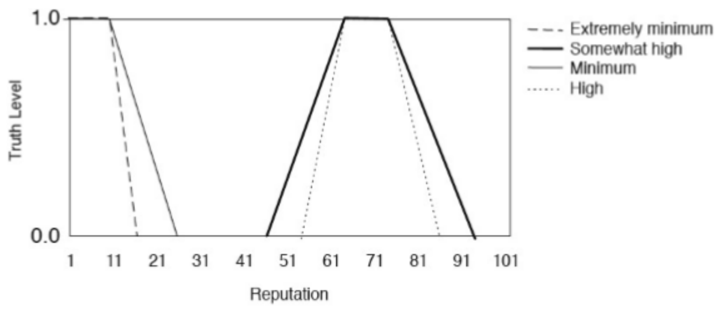

Figure 9.2: Representing reputation using fuzzy sets.

4.6.2 Trust and Reputation as Beliefs

In BDI architecture, the trust and reputation values should be represented in terms of beliefs. Using beliefs to represent trust or reputation raises two main issues. The first one is to define the content and the semantics of this specific belief. The second issue consists of linking the belief to the aggregated data grounding it.

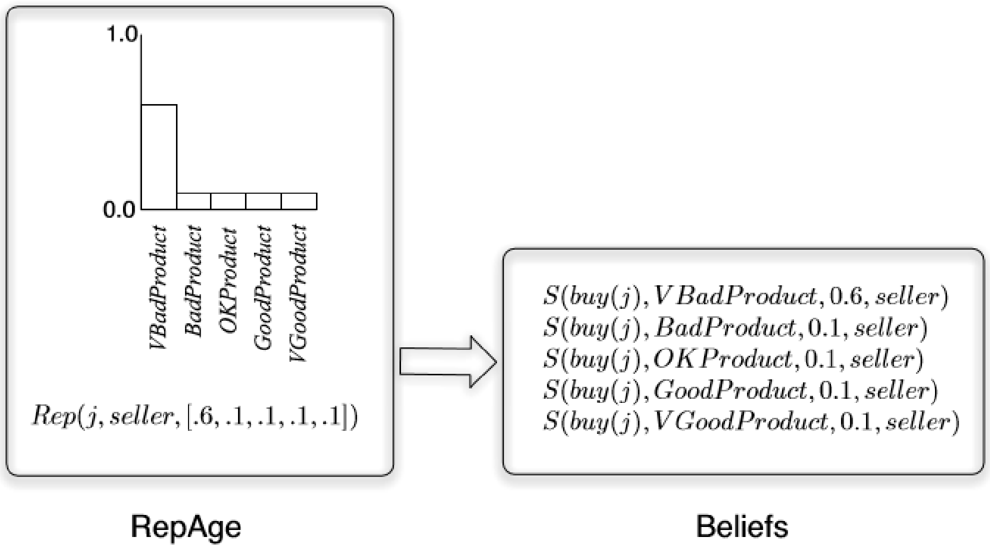
1.0
0.0
VBadProduct
BadProduct
OKProduct
GoodProduct
VGoodProduct
$Rep(j, seller, [.6, .1, .1, .1, .1])$
$S(buy(j), VBadProduct, 0.6, seller)$
$S(buy(j), BadProduct, 0.1, seller)$
$S(buy(j), OKProduct, 0.1, seller)$
$S(buy(j), GoodProduct, 0.1, seller)$
$S(buy(j), VGoodProduct, 0.1, seller)$
RepAge
Beliefs

Figure 9.3: Reputation as beliefs in the BDI+RepAge model.

The Reliability of a Value

The Reliability of a Value can be assured by associating a number to the trust or reputation value that reflects how reliable it is. The calculation of this number is based on several aspects regarding how the trust or reputation value has been calculated. Aspects that can be taken into account are the number of opinions grounding the value, variance of those opinions, redundancy of the opinions, credibility of the informers, and so on.

4.6.3 Trust Processes in Multi-agent Systems

"Trust is the subjective probability by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends." The figure shows how an agent, called the **trustor,** can use the different sources of information to decide if other agents, called the trustees, are trustworthy or not in order to take some actions.

Trust evaluations and **trust decisions** are the central parts of the model and correspond to the two steps that a trust process has to follow: evaluation and decision making. Trust evaluations can be seen as a summary of all the beliefs, mental states, and values known by the trustor to assess trustees. Two elements feed trust evaluations: images and reputation. There are three sources that agents can use to create images: Direct experiences, Communicated experiences, and Social information.

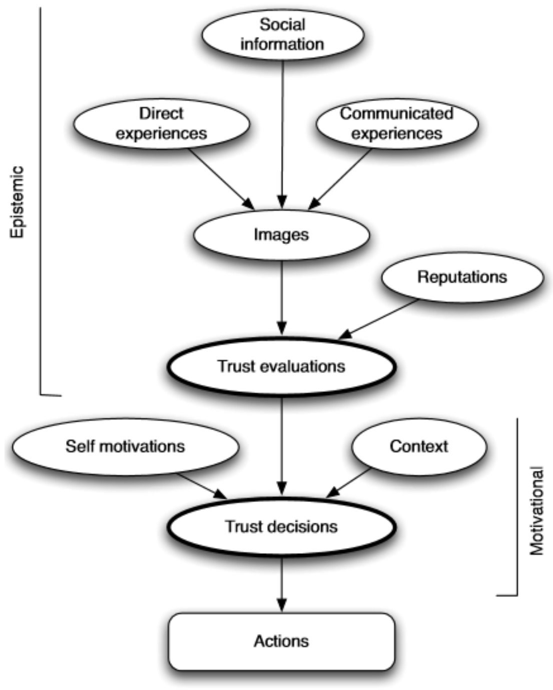

Figure 9.4: The dual nature of trust.

The Trust Evaluation process involves:

1. **Filtering the Inputs – 1) Context** -Take into account only a subset of the available experiences about the trustee. 2) Communicated experiences are not always reliable and their reliability depends on the **source of the communication.**
2. **Statistical Aggregation -** The direct way to define an image is to represent it as a single value.
3. **Logical Beliefs Generation -** Given the nature of trust, a cognitive view is an interesting perspective that makes it possible to build agents that deal with trust in a more "human like" way.

The goal of **Trust Decision process** is to determine if a trustee should be trusted for a given task. The trustor is now in a situation where it is in its own interest that the trustee behaves in an expected way, and the trustor has to decide if it will intend to rely on the trustee. The decision process corresponds to the concept of trust as an act. If the decision is positive (negative), the trustor will intend to act in a way such that it trusts (distrusts) the trustee.

- **Trust evaluations represented as a single value,** either Boolean, qualitative, or numerical, require a simple decision process. This is usually done by comparing the value to a given threshold. If the trust value is above the threshold, trust the trustee.
- The **trust belief** resulting from a trust decision corresponds to the concept of occurrent trust. **Occurrent trust** is a trust attitude that holds here and now. It means that, given the current contextual conditions and a goal that the trustor wants to achieve now, the trustor trusts a trustee for performing now a given action that will achieve the goal. Occurrent trust is obviously linked to the concept of dispositional trust.

4.6.4 Reputation in Multiagent Societies

A Vietnamese proverb says: "After death, a tiger leaves behind his skin, a man his reputation." Reputation has been present in human societies since the beginning of time as a social mechanism that helps these societies to regulate the behavior of their individuals.

The Reputation Building Process analyze how the calculation of reputation has been approached from the point of view of multi-agent systems and to present the main advantages and drawbacks of each approach. Basically there are three sources that can be used to evaluate reputation: communicated image/third-party image, communicated reputation, and social information.

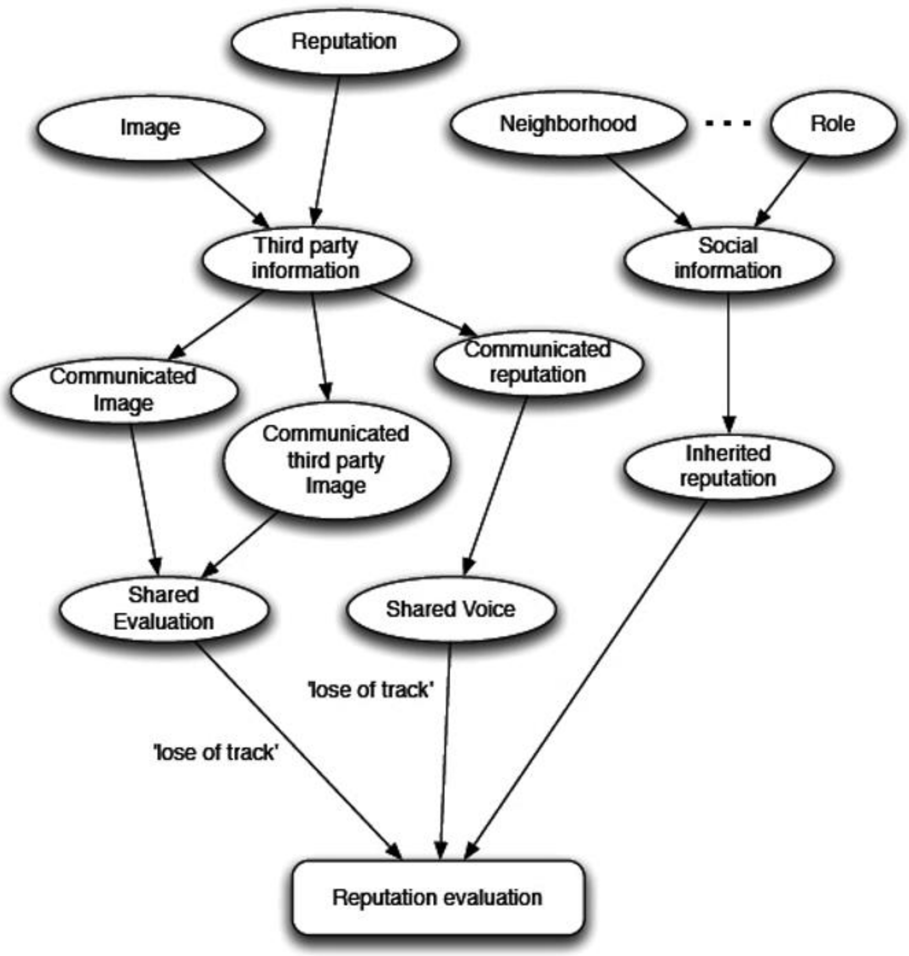

Figure 9.5: Reputation evaluation.

4.6.5 Centralized vs. Decentralized Models

Centralized Approaches

A central service is responsible for collecting the raw information from individuals in a social entity and for calculating a reputation value using an aggregation mechanism. This value is then available to the individuals as a measure of reputation. The advantages are:

- All of the community is contributing to improving the reputation value calculation and therefore that value should be more accurate.
- Given that usually there is a lot of information to make the calculation, possible wrong or biased information provided by a few individuals has less of a bad impact on the final value.
- The fact that reputation values are public allows new comers to benefit from this information even without having any previous knowledge about the society.

But it also brings some drawbacks:

- Individuals have to trust central service regarding the impartiality of the calculation.
- Personal preferences and biases are not taken into account.
- The central repository is a bottleneck for the system, in case of failure of the central service. It may also cause a system overload or slowness in case agents require very frequent access.
- Providing information to a central authority can create a security problem.
- Aspects like social relations, for example, are difficult to collect in a central repository given their sensitive nature.
- Related to the above, it has been observed that when the source for calculating reputation is based on images, people avoid giving bad evaluations in central repositories because there is fear of retaliation, making the calculated reputation too optimistic.

Decentralized Approaches

The decentralized approach relies on the individual information that each agent can obtain about the society. This information can be images and reputation values coming from third-party agents but also can be information about social relations, roles, etc. Decentralized approaches have the following advantages:

- No trust of an external central entity is necessary.
- They are suited to build scalable systems as they do not introduce any bottleneck.
- Each agent can decide the method that it wants to follow to calculate reputation.

But they imply the following drawbacks:

- It can take some time for the agent to obtain enough information from the rest of the society to calculate a reliable reputation value.
- It is not so easy for newcomers to start using reputation in a society that does not have a centralized reputation service.
- It demands more complex and “intelligent” agents as they need to encapsulate processes for reasoning on reputation messages received, calculating reputation, and deciding when to communicate reputation to others.

4.6.6 Using Reputation

Reputation is considered as a source for building trust and an intermediate solution to the problem of social order. It has some pitfalls, since reputation is prone to receive attacks from malicious agents. Defense methods against attacks are usually prone to give false positives if they are not used cautiously. It is easy to regard those that are actually correct ones to be unfair.

- **Unfair Ratings attack** occurs when an agent sends deliberately wrong feedback about interactions with another agent.
- **Ballot-stuffing** is an attack in which an agent sends more feedback than interactions it has been involved in as a partner.
- **Dynamic Personality** is value imbalance exploitation, where the agent provides a number of low value, high-quality services (to get a good reputation) and a small number of deceptive, high-value services.
- **White washing** occurs when an agent changes its identifier in order to escape previous bad feedback.
- **Collusion** occurs when a group of agents cooperate with one another in order to take advantage of the system and other agents.
- **Sybil attacks** are sort of security threat that create enough fake identities so that a single agent can subvert the normal functioning of the system.
- **Reputation Lag Exploitation** attack consists of using the lag (reputation values that oscillate from one value to another in reacting to the minimum change) that the reputation mechanism needs to reflect the new reality and exploiting it to obtain benefit.

4.6.7 Trust, Reputation, and Other Agreement Technologies

- **Argumentation** can be used to explain a trust/reputation value.
- **Negotiation** with other agents is a skill usually demanded of an autonomous agent.
- Even if the expectations are often hidden in the trust calculation function, they can also be made explicit by using **norms.**
- When agents are **organized** in groups, reputation can be attached to an explicit group of agents, and an agent may have different reputations in different groups.
- Some kind of **ontology mechanism** is necessary to guarantee that the exchanged information has meaning for both.

PART – A (2 MARK QUESTIONS)

1.Define an agent?

Agents are systems that can decide for themselves what they need to do in order to achieve the objectives that we delegate to them. Intelligent agents, or sometimes autonomous agents are agents that must operate robustly in rapidly changing, unpredictable, or open environments, where there is a significant possibility that actions can fail.

2. Give the classification of environment properties.

- Accessible vs. inaccessible
- Deterministic vs. non-deterministic
- Episodic vs. non-episodic
- Static vs. dynamic
- Discrete vs. continuous

3.List down the four important classes of agent architecture?

- Logic-based agents
- Reactive agents
- Belief-desire-intention agents
- Layered architectures

4.What are the rules that govern vaccum cleaning agent's behavior?

$In(x, y) \wedge Dirt(x, y) \longrightarrow Do(suck)$ ------(4.1)

$In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) \longrightarrow Do(forward)$ ------(4.2)

$In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) \longrightarrow Do(forward)$ ------(4.3)

$In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) \longrightarrow Do(turn)$ ------(4.4)

$In(0,2) \wedge Facing(east) \longrightarrow Do(forward)$ ------(4.5)

5.Give the disadvantages of Reactive Architecture

- Agents need sufficient information available in their local environment.
- It is difficult to see how decision making could take into account non-local information.

- Overall behavior emerges from the interaction of the component behaviors.
- It is much harder to build agents that contain many layers. The dynamics of the interactions between the different behaviors become too complex to understand.

6.List the seven main components to a BDI agent:

i. A set of current beliefs, representing information agent has about its current environment;

ii. A belief revision function (brf), which takes a perceptual input and the agent's current beliefs, and on the basis of these, determines a new set of beliefs;

iii. An option generation function(options), which determines the options available to the agent (its desires), on the basis of current beliefs and its current intentions;

iv. A set of current options, representing possible courses of actions available to the agent;

v. A filter function (filter), which represents the agent's deliberation process, and which determines the agent's intentions on the basis of its current beliefs, desires, and intentions;

vi. A set of current intentions, representing the agent's current focus – those states of affairs that it has committed to trying to bring about;

vii. An action selection function (execute),

7.Define the two types of control flow within layered architectures.

- Horizontal layering - In horizontally layered architectures, the software layers are each directly connected to the sensory input and action output.
- Vertical layering - In vertically layered architectures, sensory input and action output are each dealt with by at most one layer each

8.What is autonomy in agent?

Each agent is an autonomous entity in the sense that agent itself is a domain of control, other agents have no direct control over its actions.

9.Define protocols?

Protocols are modular, potentially reusable specifications of interactions, called messages, between two or more components. To promote reusability, a protocol is specified abstractly with reference to the roles that the interacting components may adopt. A protocol is designed with a certain application in mind. An enactment refers to an execution of the protocol by the components.

10.Why agents have Heterogeneity?

Heterogeneity refers to the diversity of agent implementations. A component can be implemented based on its dependence on other components, without concern for how the others are implemented. Accommodating heterogeneity entails specifying the semantics of the interaction. Making an offer in a Multi-agent system is social commitment.

11.Give Traditional Software Engineering Approaches

Sequence Diagrams, Choreographies, and State Machines

12.What is FIPA ACL? What is it's use?

A goal for the FIPA ACL (Agent Communication Language) was to specify a definitive syntax through which interoperability among agents created by different developers could be facilitated and specified the semantics of the primitives. FIPA provides the semiformal specification of an agent management system and also provides definitions for several interaction protocols, based on beliefs and intentions of agents.

13.What is JADE?

The Java Agent Development Framework (JADE) is a popular platform for developing and running agent-based applications. It implements the FIPA protocols. JADE provides support for the notion of behaviors.

14.Define Negotiation?

Negotiation is a form of interaction in which a group of agents with conflicting interests try to come to a mutually acceptable agreement over some outcome. The outcome is typically represented in terms of the allocation of resources

15.What are the factors in payoff allocations?

- The payoff allocations that two players ultimately get should depend only on following factors:

- The set of payoff allocations that are jointly feasible for the two players in the process of negotiation or arbitration, and
- The payoffs they would expect if negotiation or arbitration were to fail to reach a settlement.

16.What is ABN?

Argumentation-based negotiation (ABN) approaches, on the other hand, enable agents to exchange additional meta-information (i.e., arguments) during negotiation.

17.State the steps in argumentation?

Argumentation can be seen as a reasoning process consisting of the following four steps:

1. Constructing arguments (in favor of/against a “statement”) from available information.
2. Determining the different conflicts among the arguments.
3. Evaluating the acceptability of the different arguments.
4. Concluding, or defining, the justified conclusions.

18.Define Argument labeling?

Let AF = A, labeling is a total function $L : A \rightarrow \{in, out, undec\}$such that:

$\forall\ \alpha \in A : (L(\alpha) = out \equiv \exists\beta \in A$ such that $(\beta\ \alpha$ and $L(\beta) = in))$

$\forall\ \alpha \in A : (L(\alpha) = in \equiv \forall\beta \in A : ($ if $\beta\ \alpha$ then $L(\beta) = out))$

Otherwise, $L(\alpha) = undec$ (since L is a total function).

19.What is the idea of Glazer and Rubinstein’s Model

Glazer and Rubinstein’s Model explore the mechanism design problem of constructing rules of debate that maximize the probability that a listener reaches the right conclusion given arguments presented by two debaters.

20.Define Trust and give trust evaluation process?

Trust is the outcome of observation leading to the belied that the actions of another may be relied upon, without explicit guarantee to attain the goal. The Trust Evaluation process involves:

1. Filtering the Inputs
2. Statistical Aggregation
3. Logical Beliefs Generation

PART B (13 MARK QUESTIONS)

1. Illustrate the Architecture for Intelligent Agents and explain?
2. Write a short note on Agent communication?
3. Describe in detail about Negotiation and Bargaining
4. Give a complete process of Argumentation among Agents
5. Why Trust and Reputation in Multi-agent systems is so important. Give a detailed explanation?