I.J. Intelligent Systems and Applications, **2012, 1, 64-76**
Published Online February 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2012.01.08

Self-Organization and Autonomy in Computational Networks: Agents-based Contractual Workflow Paradigm

E.V.Krishnamurthy,
Australian National University, Canberra, ACT 0200, Australia.
Evk.krishnamurthy@anu.edu.au

***Abstract*- We describe an agents-based contractual workflow paradigm for Self-organization and autonomy in computational networks. The agent-based paradigm can be interpreted as the outcome arising out of deterministic, nondeterministic or stochastic interaction among a set of agents that includes the environment. These interactions are like chemical reactions and result in self-organization. Since the reaction rules are inherently parallel, any number of actions can be performed cooperatively or competitively among the subsets of elements, so that the agents carry out the required actions. Also we describe the application of this paradigm in finding short duration paths, chemical- patent mining, and in cloud computing services.**

***Index Terms*- Agents, autonomy, bio-inspired technique, cloud-computing services, contract -based workflow, self-organization, shortest path computation, template matching**

I. INTRODUCTION

Self-organization and related emergence are important topics of study in complex systems and nonlinear dynamics; for details see Murthy and Krishnamurthy [12]. In a recent paper Prehofer and Bettstetter [14], propose a method of self-organization in communication networks. They propose four basic principles (paradigms) and show how they are reflected in current protocols: design local interactions that achieve global properties, exploit implicit coordination, minimize the maintained state, and design protocols that adapt to changes. However, emergent properties are more complex and we will not consider these aspects here; detailed discussion of emergence requires nonlinear dynamics and associated probabilistic networks [12]. In a practical sense, in the terminology of Kephart and Chess [6], self-organization and autonomic computing have common aims, Ramirez and Cheng [15], Dobson et al [2], with the ability to self –monitor, self-configure, self- optimize, self-protect and being self aware.

A. Prehofer- Bettstetter Principles:

***Principle 1**: Design local behavior rules to achieve global properties*:
That is the entities involved have only a local view of its connected neighborhood, and all the rules (regarding communication and computation) are applied only at the local level and their effects propagate at global level to achieve the required properties.

***Principle 2**: Do not aim for perfect coordination – exploit implicit coordination*:
That is messages among the entities are coordinated at the local level checking for consistency and conflict –freedom.

Principle 3: *Minimize long-lived state information organization*:
This is achieved by employing mechanisms that know its neighborhood entities, their capabilities and their reliability. This enables the system to update global information through local communication.

Principle 4: *Design protocols or algorithmic rules that can adapt to changes:*
Adaptation is the capability of nodes to react to changes in the network and its environment. The need for such adaptation typically arises from changed resource constraints, changed user requirements, node properties, node mobility, or node failures. Since there are no centralized entities that could notify the nodes about changes, each node needs to continuously monitor its local environment and react in an appropriate manner.

Our aim in this paper is to describe the agent- based contractual paradigm that uses the above four principles with the currently available software tools. In Section II we describe the agent-based software paradigm. In Section III we describe the contract- based workflow and its role in failure detection and prevention. Section IV describes an example of shortest delay (or path) determination in a network using local rules on a set of connected agents. Section V studies the problem of matching relational structures (e.g., chemical structure) using a set of cooperating agents using local rules, exploring and exploiting neighborhood information, thus minimizing long-lived information. Section VI deals with the application of agents in cloud computing services and their patterns. Section VII describes in brief some of the currently available agent-tools. Section VIII is the conclusion.

II. AGENT-BASED PARADIGM

An agent is a system [12,13,19] that is capable of perceiving events in its environment or representing information about the current state of affairs and of acting in its environment guided by perceptions and stored information, Woolridge [19]. A set of agent system consists of several single agent-systems (Figure 1),

Murthy and Krishnamurthy [12]. Thus if N agents are involve i = 1,2, , , N, each of the agents will be denoted with a label (i). Here, we will restrict ourselves to the definition that the agent is a software module having the workflow programming model to be described in next section.

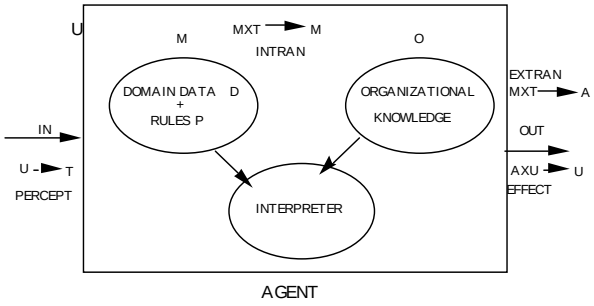


Figure 1. Agent Model

(1) Worldly states or environment U:
All those states that completely describe the universe containing all the agents.
(2) Percept: Depending upon the sensory capabilities (input interface to the universe or environment) an agent can receive from U an input T (a standard set of messages), using a sensory function Perception (PERCEPT): PERCEPT :U $\rightarrow$ T.
PERCEPT can involve various types of perception: see, read, hear,smell. The messages are assumed to be of standard types based on an interaction language that is interpreted identically by all agents. Since U includes both the environment and other agents the input can be either from the agents directly or from the environment that has been modified by other agents. We assume that agents that can communicate directly, as well as, indirectly through the environment (see also EFFECT).
(3) State of Mind M:
The agent has a state of mind M (essentially a problem domain knowledge consisting of an internal database D for the problem domain data and a set of problem domain rules P). Here, D is a set of beliefs about objects in its neighbourhood, their attributes and relationships stored as an internal database. P is a set of rules expressed as preconditions and consequences (conditions and actions). When T is received as the input, if the conditions given in the left-hand side of P match T, the elements that correspond to the right-hand side are taken from D, and suitable actions are carried out locally (internally in M) as well as externally on the environment.

The nature of internal production rules P, their mode of application and the action set determines whether an agent is deterministic, nondeterministic, probabilistic or fuzzy. Rule application policy in a production system P can be modified by:
(1) Assigning probabilities/fuzziness for applying the rule
(2) Assigning strength to each rule by using a measure of its past success
(3) Introducing a support for each rule by using a measure of its likely relevance to the current situation.
The above three factors provide for competition and cooperation among the different rules. Such a model is useful for many applications, Murthy and Krishnamurthy [12]. Also, we assume that each agent can carry out other basic computations, such as having memory, arithmetic capability, comparison, simple control rules and the generation of random numbers.
(4) Organizational Knowledge (O): Since each agent needs to communicate with the external world or other agents, we assume that O contains all the information about the relationships among the different agents, e.g., the connectivity relationship for communication, the data dependencies between agents, interference among agents with respect to rules and information about the location of different domain rules.
(5) INFLOW:
On the receipt of input T, the action in the agent M is suitably revised or updated by the function called INFLOW
(6) Revision: Revision means acquisition of new information about the environment that requires a change in the rule system P. This may result in changes in the database D. In a more general sense, revision may be called "mutation' of the agent since the agent exhibits a mutation in behaviour due to change of rules or code.
(7) Update: Update means adding new entries to the database D; the rules P are not changed. In a more general sense, the update may be called "Reconfiguration" since the agent's behaviour is altered to accommodate a change in the data.
Both revision and update can be denoted in set-theoretic notation by:
INFLOW: M X T $\rightarrow$ M(D,P)
Both mutation and reconfiguration play important roles in self-organization. These are achieved in this model by introducing changes in D and P as required. This can be interpreted as updating or revising a set of database instances. Hence, if one or several interaction conditions hold for several non- disjoint subsets of objects in the agent at the same time, the choice made among them can be nondeterministic or probabilistic. This leads to competitive parallelism. The actions on the chosen subset are executed atomically and committed. In other words, the chosen subset undergoes an 'asynchronous atomic update'.
As a result of the actions followed by commitment, we may revise or update and obtain a new database for each agent; this may satisfy new conditions of the text and the actions are repeated by initiating a new set of computations. This set of transformations halt when there are no more actions are executable or the databases does not undergo a change for two consecutive steps indicating a new consistent state of the databases.

However, if the interaction condition holds for several disjoint subsets of elements in the database at the same time, the actions can take place independently and simultaneously. This leads to cooperative parallelism; e.g. vector parallelism, pipeline parallelism.
(8) EXFLOW: External action is defined as an external workflow (EXFLOW) that maps a state of mind and a partition from an external state into an action performed by the agent. That is: EXFLOW: M X T $\rightarrow$ A. That is, the current state of mind and a new input activates an external action from the action set A.

(9) EFFECT: The agent also can affect the universe U by performing an action from a set of actions A (ask, tell, hear, read, write, speak, send, smell, taste, receive, silent), or more complex actions. Such actions are carried out according to a particular agent's role and governed by an etiquette called protocols. The effect of these actions is defined by a function EFFECT that modifies the world states through the actions of an agent:

EFFECT: A X U $\rightarrow$ U;

EFFECT can involve additions, deletions and modifications to U. Thus an agent is defined by a set of nine entities, a 9-tuple:

(U,T,M(P,D),O,A,PERCEPT,INFLOW,EXFLOW,EFFECT).

The interpreter within an agent repeatedly executes the rules in P until no rule can be fired.

An agent-based software system is a collection of agents interacting through messages among themselves and the environment. Each agent maintains its own share of data and has its own program piece to manipulate it. The agents are active and behave like actors in a movie, each following its own script and interacting with other agents. Accordingly agents can model the changing world by perceiving the changes themselves at any specific time. Thus agents can adequately provide distributed-service for a given request in a particular scenario by monitoring, decision making and executing required actions through the functional logic. In addition they have built in local rules, adapt to changes, monitor the environment and be proactive when anticipating failures.

III. AGENT-BASED CONTRACTUAL WORKFLOW

We now describe an agent-based contractual workflow paradigm to support long and short duration transactions in communication, control and commercial environment. In this environment, we use a model called a "workflow model" between the agents (peers) that interact, compete and cooperate, to realise a distributed program [10], [11]. The various types of task patterns that arise in communication, control and commercial environment require a "what if" programming approach consisting of intention and actions for trial-error design, before an actual commitment is made. Such an approach enables us to connect partners anywhere and anytime and take care of the unpredictable nature of connectivity among the devices and the networks. It also provides for seamless integration of differing applications and communications and the trial and error program design required. Thus it helps to design protocols required by principle 4 in selforganization.

We define an agent-based workflow-service thus: Given an input set I= (I1,I2,I3.....) the agent -based service execution W (I1,I2,..In) is an execution of a sequence of valid agent-based states (s0,s1,s2,...sn) that represent a meaningful state transition from s0 to achieve a desired final state sn which satisfies the context specified by the invoker, subject to the condition that all the elements of the input set I are valid objects that exist during the state transformation. A workflow is a distributed task that can be executed partly within that agent as an internal workflow (Inflow) and partly in other agents as external Workflow (Exflow).

We illustrate some of the important applications and the design of suitable protocols. We also classify some workflow patterns that arise in cloud computing services in E-business and the language support needed.

A global workflow (we call it an External workflow or *Exflow*) T(ij) is defined as a workflow between two agents A(i) and A(j) ; this consists of a message sent from A(i) to execute a desired workflow in A(j); this message is received by A(j). A(j) has a behaviour specified by: Pre(T(ij)), G(j), C(j), Post (T(ij))S(j), where Pre() and Post() are respectively the pre and post states of the world that are active before and after the workflow T(ij). G(j) is a guard of A(j) to signal when the required precondition is met, and C(j) is the command function ; S(j) signals when the post condition is achieved. Here the script specifies what message A(j) can accept and what actions it performs when it receives the message while in state Pre(T(ij)) to satisfy the post condition post(T(ij)). The Exflow T(ij) can trigger in A(j) numeric, symbolic or database computations.

Each **Exflow** T(ij) from agent i to agent j triggers a set of serializable computations in A(j) either in a total order or in a partial order depending upon whether parallelism, concurrency and interleaving are possible locally within A(j). If the agent A(j) is "made up" of sub-agents, we may have to execute a workflow consisting of several local workflows (called internal workflow - **Inflow**). After executing Inflow, each agent reaches a new state from an old state using its internal command set C(j); before executing the commands, the required precondition is met, and after completion of the command set , the post condition is ensured in the new state. This is the **design by contract approach**, Kramer [7], Little [8], Meyer [9], and widely used in the language Eiffel. The precondition is specified by "require" and post condition by "ensure".

The principal aim of the contractual paradigm is to ensure that every action is evaluated in terms of the requirements on the state of the world before the execution of the service in a given context (user, time, location, type of data, ordering actions) so that the collaboration, communication and action framework take place infallibly. This paradigm was first suggested by Smith [17].

The concept of a contractual workflow paradigm is quite simple. A contract is a consistent and fault tolerant execution of an arbitrary sequence of predefined actions carried out according to an explicitly specified control flow description called "script". The script has a condition event structure that describes a stereotyped sequence of event in a particular context. Events form a causal chain and during the execution of a workflow, a contractual obligation should take the program from one consistent state of the world to another. That is the precondition and post condition of a contract holds at every elementary contextual step ensuring a consistent and fault tolerant execution of any task.

Remark: The contractual workflow paradigm thus provides a particular design pattern, Jezequel et al. [5], Shalloway, and Trott [16]. See Section VI for examples.

A. Chemical reactivity -like properties

The Exflow and Inflow have general properties called "chemical Reactivity properties", since they resemble chemical reactions: Molecularity, Contractual obligation, Opacity during a molecular action, and retry or rescue through a recovery protocol bringing the system back into the invariant state.

These are defined as below:

(i) Molecularity: If there is a crash during a composite operation all the effects of the sub-operation are lost. If there is no crash the composite or molecular operation is complete. That is a molecule is synthesised fully or not at all.

(ii) Contractual obligation: Invocation of a single composite operation takes the program from one consistent state to another. This means precondition and post condition of a contract holds. Thus conventional consistency is replaced by contractual obligation.

(*iii) Opacity*: The results of the sub-operations of composite operation should not be revealed until the composite operation is complete.

(iv)Durability: If a crash occurs and contract fails and a component cannot meet its obligation and fails in its contract, an exception is raised. Then we have three possibilities:

a. Exception is not justified: it is a false alarm; we may ignore.

b. If we have anticipated the exception when we wrote the routine and provided an alternative way to fulfil the contract, then the system will try that alternative. This is called resumption.

c. If, however, we are still unable to fulfil the contract we go into graceful degradation or surrender with honour. Then bring all the objects to an acceptable state (pre-committed- state) and signal failure. This is called organized panic. This should restore the invariant. At this point we initiate retry. The effect of retry is to execute the body of the routine again.

Remark: In Eiffel Jezequel et al. [5] Meyer [9], the rescue clause does all the above (this is essentially RESTART after recovery).

Also the chemical reactivity properties are generalized further: Since a number of remote objects are invoked, the agents should ensure all the remote actions and the local actions are complete. If any one fails the whole program has to be abandoned, and we need to retry, rescue and bring the system to its invariant state. Contractual obligation is extended to all agents under concurrent invocation and partial failures. No results are revealed from any agents until all the actions are complete and committed.

(v) Retry /Rescue and Reset :If false alarm then retry; else rescue and restart so that all the invariants in all objects are rest to their pre-action state.

Recall that we have two types of transactions - Exflow between peers, and inflow within each peer. We split the Exflow into Intention and Action transactions, where the intention transactions satisfy the ACID properties:

Atomicity: All or none of transaction happens;

Consistency: A transaction preserves the consistency in database before and after its execution;

Isolation: Intermediate results are not externally made visible until commitment;

Durability: The effects are made permanent when a transaction succeeds and recovers under failure.

The Action transactions are long duration transactions supported by a recovery protocol. The intention transactions are again local to each agent and based on the decision in this phase, the action transaction takes place through a protocol (called Intention -Action Protocol -IAP) provided with a time-out strategy and recovery to cope up with failures of disconnection.

B. Multiagent System

A multi-agent system can be defined as a loosely coupled network of agents that interact among them and through the environment to solve a problem. The multiagent system carries out distributed computation by sending, receiving, handshaking and acknowledging messages and performing some local computations and has the following features:

1. An agent has the structure as described in Figure 1 and can carry out elementary computations and can generate random numbers
2. There is a seeding agent who initiates the solution process.
3. Each agent can be active or inactive.
4. Initially all agents are inactive except for a specified seeding agent that initiates the computation.
5. An active agent can do local computation, send and receive messages and can spontaneously become inactive.
6. An inactive agent becomes active if and only if it receives a message.
7. Each agent may retain its current belief or revise its belief as a result of receiving a new message by performing a local computation. If it revises its belief, it communicates its revised state of belief to other concerned agents; else it does not revise its solution and remains silent.
8. Agents are proactive in the sense of being anticipatory and taking charge of situations. Proactive behavior involves acting in advance of a future situation, rather than just reacting. It means taking control and making things happen rather than just adjusting to a situation or waiting for something to happen. Proactive agents do not need to be asked to act, nor do they require detailed instructions.

Hence the basic agent model can realise:

(i)Reactive or proactive agent that make decisions at run time with a limited amount of local information,

(ii) Deliberating agent that has an internal representation of the environment and has a logical inference mechanism for decision making and planning and

(iii) Interacting agent that is capable of coordinating the activities with other agents through communication and negotiation.

C. 1nteraction among Agents

In order to use the multi-agent paradigm to realise cooperative and competitive computational tasks, we need to consider how the agents can interfere with each other.

1. Enabling dependence (ED): Agent A(i) and agent A(j) are called enable dependent

(or dataflow dependent) if the messages from A (i) creates the required precondition in A(j) to carry out a specific action. These messages can be chemical concentration, voltages or communication messages.

2.Inhibit dependence (ID): Agents A (i) and A (j) are called inhibit dependent, if the actions of A (i) creates the required precondition in A(j) to prevent it from executing a specific action. Inhibition may be due to negative weights in connective links or stop signals or other computable entities.
3. INFLOW Conflict (IC) : Agents A (i) and A (j) are opposition dependent (also called data-output dependent) through A(k)), if the order in which A (i) and A (j) enable A(k) and update A(k) produce different results in A(k); that is the objects A(i) and A (j) perform operations on A(k) that are not order reversible. That is, local serializability (or commutability) is not ensured in the INFLOW within A(k), if the actions are carried out within an agent in different partial order. Exception is raised in this case.
4. EXFLOW Conflict (EC): Agents A (i) and A(j) are data antidependent through A(k) if the order in which A(i) enables (inhibits) A(k), and A(j) enables (inhibits) A(k) result in different external actions (EXTRAN) by A(k) on the environment. That is the temporal order in which information arrives from the environment and other agents affects the global serializability (commutability) of the actions of an agent. Exception is raised in this case.

Remark: ED and ID:
The two properties ED and ID are crucial in self-organization.These rules permit an agent to enable itself and also an agent A(i) to enable A(j) and A (j) to enable A(i) cyclically. For example, A(i) can create the required precondition in A(k), so that A(j) can enable A(k). Also, A(i) can inhibit the required precondition in A(k) so that A(j) is prevented from enabling A(k).

D. Concurrency and Conflicts

In distributed computing and transaction processing: we require that the following two conditions be satisfied for global serialization when concurrent operations take place:
1. At each agent the actions in local actions are performed in the non-conflicting order (Local serializability or commutativity).
2. At each agent the serialization order of the tasks dictated by every other agent is not violated. That is, for each pair of conflicting actions among transactions p and q, an action of p precedes an action of q in any local schedule, if and only if, the preconditions required for p do not conflict with those preconditions required for execution of the action q in the required ordering of all tasks in all agents (Global serializability).

The above two conditions require that the preconditions for actions in different agents A(i) and A(j) do not interfere or cause conflicts. These conditions are necessary for the stabilization of the multi-agent systems that the computations are locally and globally consistent.
Termination: For the termination of agent –based program, the interaction among the agents must come to a halt. When the entire set of agents halt we have an equilibrium state (or a fixed point) also called stability while dealing with exact computation in a deterministic system.
Conflicts: Resolution or compromise? In agent -based modelling of behaviour, under concurrency, the conflicts arising in INFLOW and EXFLOW may require resolution or to an agreeable compromise. These rules should be based on the problem domain.

IV. SHORTEST DELAY OR PATH

We now illustrate how to design the multiagent paradigm for the problem of finding a lowest cost path between any two vertices in a directed graph whose edges have a certain assigned positive costs (Figure 2). We use only local rules and communication among neighbours. In the algorithm, we evaluate the goodness of each path chosen by each agent, as well as, those chosen by neighbours. This process is then reiterated until we reach a (stable) fixed point. The likeness of this algorithm with the Swarming intelligence scheme [12] can be inferred. There are very close connections between stabilization, fixed points, chaotic attractors and emergence, see [12]; also see remarks in Section IV B.
The lowest cost path (or shortest delay) problem requires the entity set of vertices, the relationship set of ordered pairs of vertices (x,y) representing edges, and the attribute of cost c for each member of the relationship set, denoted by (x,y,c). Given a graph G the program should give for each pair of vertices (x,y) the smallest sum of costs path from x to y. The vertex from which the lowest cost paths to other vertices are required is called the root vertex r (vertex 1 in this example). Let s denote the sum of costs along the path from the root to y; we assume that c (and hence s) is positive. The ordered 4-tuple describes this information: (x,y,c,s): (vertex label, vertex label, cost, sum of costs from root).

The fourth member of the 4-tuple, namely the sum of costs from a specified root remains initially undefined and we set this to a large number *.We then use the production rules to modify these tuples or to remove them.
To find the lowest cost path to all vertices from a specified root r, we use tuple processing and let the 4-tuples interact; this interaction results in either the generation of modified 4-tuples or the removal of some 4-tuples of the representation.

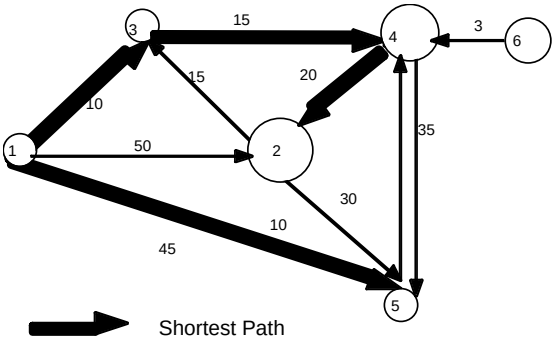


Figure 2

Rule-based protocol to find the shortest path
Let C(i,j) be the cost of path (i,j). A better path is one that can pass through some vertex k such that:

$$C(i,k) + C(k,j) < C(i,j)$$

That is our production rule is: If $C(i,k) + C(k,j) < C(i,j)$ then delete C(i,j) and set
$C(i,j) = C(i,k) + C(k,j)$.

The invariant is: if C(i,j) is the initial cost then all the costs are always less than or equal to C(i,j). We refine this by using the rule : If C(i,k) < C(p,k) , delete C(p,k) and retain C(i,k). Thus the following three production rules result:

Rule 1: If there are tuples of the form (r,r,0,0) and (r,y,c,*), replace (r,y,c,*) by (r,y,c,c) and retain (r,r,0,0).

Rule 1 defines the sum of costs for vertices adjacent to the root, by deleting * and defining the values.

Rule 2: If there are tuples of the form (x,y,c1,s1) and (y,z,c2,s2), where s2 > s1+c2 then replace (y,z,c2,s2) by (y,z,c2,s1+c2); else do nothing.

Rule 2 states that if s2 > s1+c2 we can find a lower cost path to z through y.

Rule 3:If there are tuples of the form (x,y,c1,s1) and (z,y,c2,s2) and if s1< s2 , then remove (z,y,c2,s2) from the tuple set; else do nothing.

Rule 3 states that for a given vertex y which has two paths, one from x and another from z, we can eliminate that 4-tuple that has a higher sum of costs from the root. The above three rules provide for local computation by many agents and we are left with those tuples that describe precisely the lowest cost path from the root.

For simplicity, we assume that there are n agents with names identical to the nodes in the graph and each agent is connected to other agents in an isomorphic manner to the given graph. Such an assumption on the topology of the network simplifies the organizational knowledge O. Thus each agent knows the identity of its neighbours, the direction and cost of connection of the outgoing edges. Thus for the given directed graph the outdegree of each node is the number of sending channels and the indegree is the number of receiving channels.

The revised local rules for multi-agent computation are as follows:

a. Initialization of beliefs: Agent 1 (root) sends to all its neighbours x the tuple (1,x,c,c) describing the name of the root, and the distance of x from the root (c); all the neighbours of the root handshake, receive, and store it. This corresponds to the initialization of beliefs.

b. Initial set of beliefs: Each agent x sends its neighbour y at a distance c1 from it, the tuple (x,y,c1,c+c1) describing its name, its distance to y and the distance of y from the root through x using its distance to the root c. This is the initial set of beliefs of the agents.

c. Update of Beliefs: Each agent y compares an earlier tuple (x,y,c1,s1) got from a neighbour x, or the root, with the new tuple (z,y,c1',s1') from another neighbour z. If s1< s1', then y retains (x,y,c1,s1) and remains silent; else it stores (z,y,c1',s1') and sends out the tuple (y,w,c2,s1'+c2) to its neighbour w at a distance c2, advising w to revise its distance from the root. That is, each agent updates its beliefs and communicates the beliefs to concerned agents.

d. Stability and Halting: An agent does not send messages if it receives a message from another agent that tells a higher value for its distance from the root and ignores the message, i.e., it does not revise its beliefs. Thus it contains only the lowest distance from the root. All the agents halt when no more messages are in circulation and the system stabilizes. An algorithm to detect the termination of negotiation is described in the next section.

Consider the directed graph in Figure 2, in which the edge costs are as shown; we denote this graph by the triplet , a pair of nodes (x,y) followed by the cost c of the edge ,thus: (x,y,c). The graph in Figure 2 is then given by:(1,2,50); (1,3,10); (1,5,45); (2,3,15); (2,5,10); (3,4,15); (4,2,20); (4,5,35); (5,4,30); (6,4,3).

We choose the vertex 1 as the root; we use the following format for representing the distances in the graph:(vertex label, vertex label, cost, sum of costs from root). Thus the graph is encoded in the form:

(1,1,0,0);(1,2,50,*);(1,3,10,*);(1,5,45,*);(2,3,15,*);
(2,5,10,*);(3,4,15,*);(4,2,20,*);(4,5,35,*);(5,4,30,*);
(6,4,3,*).

We then apply the three rules systematically. This results in the following tuples that describe the lowest cost path subgraph: (1,1,0,0); (1,3,10,10); (1,5,45,45); (3,4,15,25); (4,2,20,45). Note that the 4-tuple (6,4,3,*) gets eliminated as vertex 6 cannot be reached from the root vertex 1. Figure 3 shows the computation and communication tree, and its termination.

Self-Evaluation of Stabilization

An important property of self organization is its ability to test the fitness through *self-awareness,* i.e. the individual who does the work evaluates itself, ensuring that the global fitness is guaranteed. This is widely prevalent in Nature for activities such as: nest building (stigmergy), food searching (foraging). We imitate this process here, so that the agents *self-evaluate* themselves to determine the stabilization.

We now describe an algorithm, called "Commission-Savings-Tally Algorithm ", for the global stabilization detection of a Multi-agent computation. This is a general algorithm. For simplicity of illustration we use this algorithm to find the shortest path in a graph of Figure 2.The agent negotiation algorithm terminates or stabilizes with appropriate distances as shown in Figure 3. Let us assume that the N agents are connected through a communication network represented by a directed graph G with N nodes and M directed arcs, as in Figure 3. Let us also denote the outdegree of each node i by Oud(i) and indegree by Ind(i). Also we assume that an initiator or a seeding agent exists to initiate the transactions. The seeding agent (SA) holds an initial amount of money C.

When the SA sends a data message to other agents, it pays a commission:

C/(Oud (SA) + 1) to each of its agents and retains the same amount for itself.

When an agent receives a credit ,it uses the following rules:

Rule a. Let agent j receive a credit C(M(i)) due to some data message M(i) sent from agent i . If j passes on data messages to other agents j retains C((M(i)) / (Oud(j)+1) for its credit and distributes the remaining amount to other Oud(j) agents. If there is no data message from agent j to others, then j credits C(M(i)) for that message in its own savings account; but this savings will not be passed on to any other agent, even if some other message is received eventually from another agent.

Rule b. When no messages are received and no messages are sent out by every agent, it waits for a time-out and sends or broadcasts or writes on a transactional blackboard its savings account balance to the initiator.

Rule c. The initiator on receiving the message broadcast adds up all the agents' savings account and its own and verifies whether the total tallies to C.

Rule d. In order to store savings and transmit commission we use an ordered pair of integers to denote a rational number and assume that each agent has a provision to handle exact rational arithmetic. Note that, if we choose C=1, it is sufficient to store the denominator of the rational number remembering that the actual value is a reciprocal for summing over the credits.

We prove the following theorems to describe the validity of the above algorithm:

Theorem 1: If there are cycles present among the agents (including the initiator itself) then the initiator cannot tally its sum to C within a finite time period. Hence negotiation fails and the algorithm has to abort after a properly chosen time-out period.

Proof: Assume that there are two agents i and j engaged in a rule dependent cycle. This means i and j are revising their beliefs forever without coming to an agreement. Let the initial credit of i be x. If i passes a message to j, then i holds x/2 and j gets x/2. If eventually j passes a message to i, then its credit is x/4 and i has a credit x.3/4; if there is continuous exchange of messages for ever then their total credit remains $(x - x/2^k)$ with $x/2^k$ being carried away by the message at the k-th exchange. Hence the total sum will never tally in a finite time period.

Theorem 2: The stabilization terminates within a finite time-out period, if and only if, the initiator tallies the sum of all the agents savings to C. In other words, there is no situation in which the sum has tallied to C but stabilization is incomplete, or sum has not tallied to C but the stabilization is reached.

Proof:

If part: If the initiator tallies the sum to C within a finite time out period, it implies that all the agents have sent their savings and no message is in transit carrying some credit and there is no chattering among agents.

Only if part: The credit assigned can be only distributed in the following manner:

a. An agent has received a message and credit in a buffer; if it has sent a message then a part of the credit is lost; else it holds the credit in savings.

b. Each message carries a credit; so, if a message is lost in transit or communication fails then total credit cannot be recovered.

Thus stabilization is reached within a properly chosen time-out period when the total sum tallies to C; then all the agents have reached an agreement on their beliefs.

A. Example

The collective agent communication protocol and computational tree of Figure 3 is obtained from Figure 2, using the rules 1, 2, 3. At initiation, the node labelled 1 is the root and the seeding agent. It contains the pair (0,0) indicating that it is the root and its distance to the root through itself is zero. It transmits the information to each neighbour its distance from the neighbour and the distance of its neighbour to the root through itself. Also it starts with a credit 1and retains a credit of (1 / Oud (SA)+ 1) to itself, and transmits the same amount to its neighbours 2, 3, 5 which in this case is 1/4. Along each edge from each node x to node y the credits transmitted are indicated. The retained credit for each transmission is indicated near the node. Then the algorithm proceeds as indicated generating the communication tree of Figure 3. Note that in this process, agent node 2 revises its earlier belief from the new message received from 4 ; but the other nodes 3, 4, 5 do not revise their initial beliefs and remain silent , since the later message received by them contained a longer distance path than what they received earlier from node 1. Finally as indicated in the rules a,b,c,d in this section we sum over all the retained credits after each transmission. These are respectively: Node 1: 18/72; Node 2: 7/72; Node 3: 16/72; Node 4: 12/72; Node 5: 19/72.Note that the sum tallies to one. Also the shortest distance to the root from each node is: Node 1: 0; Node 2: 45; Node 3: 10; Node 4: 25; Node 5: 45.

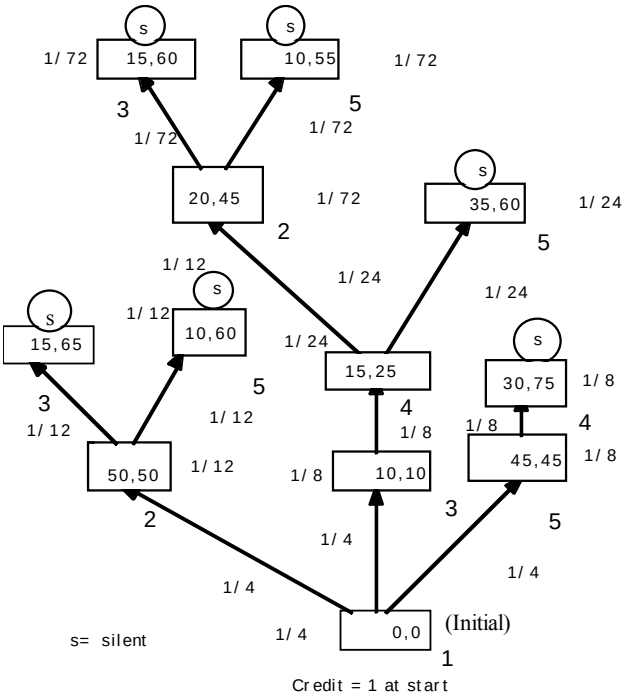


Figure 3

B. Relation to Swarming

Swarming tactics are widely used in nature by ants,flock of birds and in warfare. A set of agents that uses inferences, beliefs and computation can evolve into self-organizing swarms [12]. We can use two different forms of communication to enable (connect) or inhibit (disconnect) agents to form interactive networks.

1.Tacit (Indirect) communication: Agents with simple intelligence (e.g., ants): *Use of markings similar to a chemical gradient or diffusion mechanism or a communication field.* This provides a common spatial resource, where each agent can leave a mark that can be perceived by other agents. This requires minimal amount of memory and communication overhead.

2. Explicit (Direct) communication: Agents with more complex intelligence: *Use of voice, signals, radio resulting in a positive feed-back or nonlinear response to*

the information available from knowledge other agents may possess (by connecting or disconnecting with other agents). This involves a greater amount of memory and communication overhead. Also this would require that *each agent knows what other agents know, and how much they know measured in a taxonomic scale* so that each agent can have a score about its neighbours to link, de-link and revise its belief. The system dynamics is formulated using the rules:

(1) Stepping (or local coupling) rule:

The state of each individual agent is updated or revised in many dimensions, in parallel, so that the new state reflects each agent's previous best success.

(2) Landscaping (or global coupling) rule:

Each agent assumes a new best value of its state that depends on its past best value and a suitable function of the best values of its interacting neighbours, with a suitably defined neighbourhood topology and geometry.

All agents in the universe or selected chunks are updated using rules (1) and (2).

The above two rules permit us to model self-avoiding, self-repelling, communicating, and active random-walker models. This can result in various kinds of attractors having fractal dimensions presenting swarm-like, bacterial colony-like appearance.

V MATCHING TEMPLATES

One of the essential problems encountered in dealing with the information in chemical patents is the matching of chemical structures (templates). In this section we describe an efficient screening and matching procedure suitable for implementation in a distributed multiagent system so as to gain speed in the template matching process. This scheme is bio-inspired in the sense the agents mark their presence in a territory (as animals do) to mark their territorial presence in an environment. These bio- markings are then checked for consistency and conflict freedom so that each place is marked at most once. In this scheme, we connect each piece of new information with the available known information in memory to increase our prior knowledge and integrate this new knowledge into a knowledge network. This happens iteratively.

The multi agent scheme is a concurrent self-organizing iterative method for matching any type of relational structures. This procedure can be used for matching structures drawn from several domains. Also, it can be used for exact matching or similarity matching of structures, according to the requirement and the availability of the data on exactness or similarity. The exact (or identity) matching problem can be described as follows:

Suppose we have a structure (called W, the "world") described in terms of its parts, their properties, and the relations between them. For example, W could be a chemical structural formula the parts being atoms, their properties being their elemental identities (and possibly other features such as oxidation state), and the relations would be the various sorts of chemical bonds between the atoms. These bonds could be single, double, or triple bonds between pairs of atoms or higher order bonding among groups of atoms as occurs, for example, in a benzene ring. In W we are searching for instances of a given substructure (called M, the "model" or a "template") which is described in the same terms as W.

In the agent –based scheme, we build an actual or simulated network of agents based on the structure W. Here, we assign one agent to each node (atom) in the structure and make inter-agent connections, which correspond to the relations (chemical bonds). Each agent contains a complete description of the sought substructure M, although it can communicate only with those neighboring agents to which it is directly connected. The complete description, that is local with respect to each atom in M consists of substituent variation, possible atom lists, bond lists, link nodes (atoms that can repeat between two of their designated bonds called outer bonds, denoted by brackets), position variation bonds, homology groups), Since each agent has adequate memory, and reasoning power based on rule-based systems and they can communicate with neighbors, the multiagent scheme turns out to be efficient in patent datamining. .

In each iteration, the agents match the concepts in their long-term memory with the perception. When a concept matches, the agent adds an instance of the concept to its short-term memory making it available to support other inferences. The system operates in a bottom-up manner, starting from primitive concepts which match against percepts, and working up to higher level concepts, which matches with lower level concepts. The iterations continue until the agents have deduced all inferences that are implied from the conceptual knowledge base and immediate perception. Using the above principle, each agent maintains a list of labels, the labels referring to the atoms of M with which this atom of W might possibly be identified. This list is initially set to contain those matches that are possible considering only intrinsic properties of each atom. Then, in each iteration, each agent eliminates from its current list, those labels for which there is no possible consistent labeling of neighboring nodes. That is each agent checks for consistency of its local neighborhood as to the global structure W. These iterations continue until no further eliminations can be made only checking for local consistency.

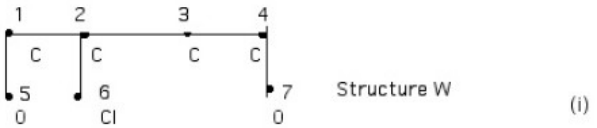
To illustrate the process, we consider a very simple example.

Example 1.

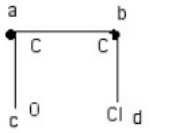
W: structure : $CH_2OH\text{-}CHCl\text{-}CH_2\text{-}CH_2OH$

M : substructure : > COH-C C1 <

Ignoring hydrogen atoms, and unspecified bonds, and numbering the other atoms for reference, we have

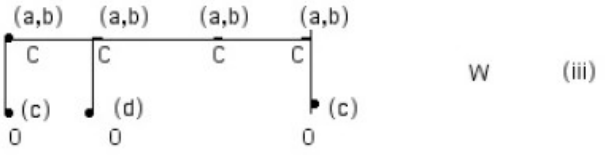


Structure W (i)

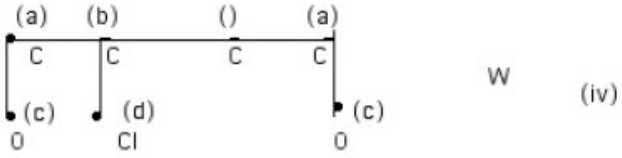


Structure M (ii)

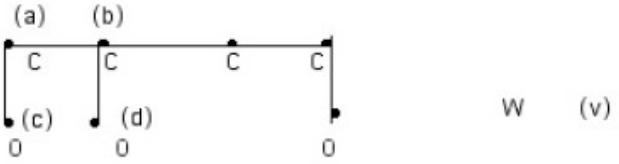
The initial labeling of W with possible matches gives (iii) below, since only carbon can be matched with C, O with O, and Cl with Cl.



To bear label **a**, an atom must be bonded to an atom with label **b** and to an atom with label c. In the first iteration, we can eliminate the label **a** from carbons 2 and 3. However, carbon 4 retains this label. Similarly, label **b** can be eliminated from carbons 1, 3, and 4.No other eliminations are possible now and after the first iteration we have (iv):



On the second iteration, we can eliminate the label **a** from carbon 4, since it has lost support from carbon 3. On the third and last iteration, oxygen 7 loses the label **c**, because carbon 4 has now lost label **a**. No further eliminations are possible, leaving the structure W correctly labeled with the names of the matching atoms of M; see (v) below:



Notice that while the computations at each node are purely local, during successive iterations of the process, information is able to propagate through the structure as in ant swarming. Because the computations at each node can proceed concurrently, as in an ant swarm, this process is suited to implementation on a properly configured multiagent system permitting many-fold reductions in computation time. However, note that the substructure matching is an *NP-complete problem* and so the speed of computation grows exponentially on the size of the problem with only marginal reduction.

In the above example, the loss of label **b** on carbon 3 in the first iteration causes the loss of label **c** on Oxygen 7 two iterations later. One should be aware, however, that the scheme provides only a screening process. There are certain circumstances under which the results of the process indicate that a match may be possible when in fact no match exists.

Selecting a "False match": If the molecules look identical at a local level (e.g. as in a lattice structure with no node distinction) the scheme cannot make any eliminations. However, the scheme is a safe screening procedure in that its failures will all be of the above type, of suggesting a match that cannot exist. It is impossible for it to fail in the other way by rejecting a match that does exist. That is it can select false matches but cannot reject true matches. In these situations we need more than the local information based on computing higher order resemblance matrix.

Stabilization: The labels generated by the agents in the iterative procedure can stabilize only in the following three types:

1. Firstly, we have the situation in which all atoms in W lose all labels. In this case we can safely conclude that no match exists.
2. Secondly, if every node in W has at most one label and every label from M is used just once, then that labeling describes the unique embedding of M into W.
3. Thirdly, if there are results with some remaining ambiguity, then some atom in W may bear more than one label or some labels from M may occur more than once. This third situation can arise either when there are multiple embeddings of M into W (from actual multiple instances or from internal symmetries of M) or when no match exists, but its refutation requires the use of global evidence. Such ambiguity cannot be resolved by this scheme and other methods must be used.

One way is to select a single ambiguous labeling and split the matching problem into several sub-problems identical, except that in each the ambiguous labeling is resolved in a particular way. If the original problem had a solution, then it must exist in one of the sub-problems, and the relaxation process can be applied to them, with recursive use of the splitting technique if any of the sub-problems remain ambiguous after applying the scheme.

We now give a larger example to illustrate the above concepts.

Example 2

The World W and Model M for this example are shown in Figs, 4a, and 4b. The results after the initial labeling in Fig. 4c, and the results after the first and second iterations are shown in Figs. 4d, and 4e. Notice that this final labeling is the union of the three overlapping instances of M in W so that a subsequent case analysis would be needed to separate these three instances.

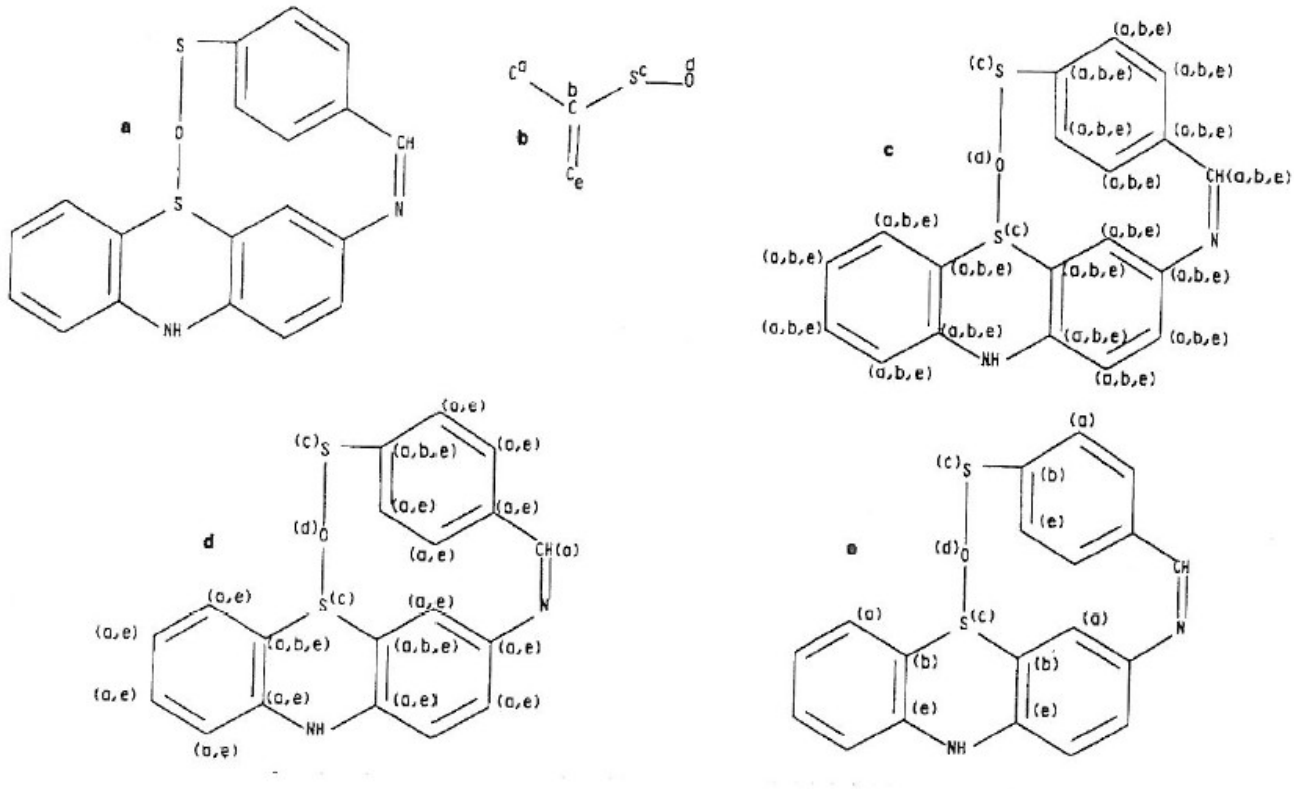


Fig. 4(a). World structure (W) for matching; (b) model M with atoms arbitrarily labeled, (c) world structure W initially labeled; (d) world structure W after first iteration (e) world structure after second iteration W

VI. APPLICATION TO CLOUD COMPUTING SERVICES

Recently, Garcia and Sim [3] have described the application of self-organization for service composition in cloud computing. The basic idea here is assign special purpose agents for web-services, resources, service-providers, brokers and clients (consumers). These agents have well defined acquaintances (connected neighbouring agents) and all the agents use the contractual paradigm to execute any task. That is, such special agents form a cluster or a specific pattern that can be used for failure-free composition for cloud service requirements using the contractual paradigm.

Certain common behavioural patterns of workflows occur in Commerce and control applications, Jezequel et al.[5], Ramirez and Cheng [15], Shalloway, and Trott [16]. A Pattern enforces a problem solving discipline for a design architecture. It consists of three parts: a context, a problem and a solution. The context is a description of the environment before the pattern is applied. It outlines the preconditions under which the problem and its solution appear. The context shows where and when the pattern will work. It is usually introduced with the help of a scenario. Problem describes the goals and objectives of a situation. Solution describes the means to achieve the goals and objectives through a suitable protocol. In cloud computing, it is possible to choose clusters or patterns of various types for resource providers, service providers, brokers and clients that can avoid node failures, increase throughput, and enable quick connections based on the property of the individual networks. This is a meeting point between large scale graph networks and cloud computing, and a potentially useful application area for cloud computing services and multimedia web services under random failures or deliberate attacks ,Murthy and Krishnamurthy [12].

The various types of task patterns that arise in E-business e.g. Purchasing, manufacturing and negotiation) require a "*what if*" programming approach consisting of intention and actions for trial-error design, before a commitment is made. This approach enables us to take care of the unpredictable nature of connectivity among the devices and the networks and also provide for the trial and error program design. We now describe several agent-based task patterns that arise in E-business and how the contract-based workflow patterns is useful for modeling and realising cloud computing services; there can be other workflow patterns which are combinations of these patterns.

We will formalize the description of the patterns using set-theoretic notation.

The behaviour of the workflow depends upon two factors: the control and dataflow between each other. In general, workflows can be sequential, repetitive, concurrent or a combination of these depending upon the problem, as described below:

a. *Conventional service transactions with ACID properties*: These occur at the *lowest level of a workflow*.
b. *Supply- chain service pattern*: Here the context is *a sequence of conventional transactions between a customer and a seller through a set of intermediary agents*.
This pattern can arise in manufacturing problem too where a workstation can request different other workstations for parts and the movement of these parts are transactions. Here the workflow is successful, if and only if each individual transaction in a chain is successful. An abort in any transaction will be considered unsuccessful, restoring the consistency of the initial state. Here the total task is partitioned into mutually disjoint sub-workflows each with its own pre and postconditions. The total workflow is successful, if and only if, each sub-workflow satisfies the precondition and post condition. Also for each subworkflow W(j) its predecessor W(j-1)'s post condition is included in the precondition of W(j). Thus any abort in the sequence W(1),W(2),...W(n) restores the system back to its consistent state . Also let A(i) and C(i) refer to boolean flags for abort and commit respectively for each workflow W(i).Then there is a control flow between two successive workflows W(j) and W(j+1); if W(j) is aborted then W(j+1) does not proceed.
Then we may write formally the supply chain workflow pattern by three conditions:
(i) $W = W(1) \cup W(2) \cup W(3).,.\cup W(n)$ where
$W(j) \cap W(j+1) = A(j)$ or $C(j)$ for $1 \leq j \leq n$.
(ii) Pre (W(j+1)= Post W(j) **and**
(iii) the boolean OR of flags A(i) satisfies:
[A(1)OR A(2).OR .A(n)] = 0 ;or the boolean AND of boolean flags C(i) satisfies:
[C(1) AND C(2)...AND...C(n)] = 1
at the completion of every workflow W(i) in the sequential composition..

c. *Mutually exclusive- collectively exhaustive service pattern*:
Here, the context is *a single buyer trying to acquire goods from several suppliers to complete a given task through several mutually exclusive transactions* and collectively exhaustive transactions. This pattern is required in control and manufacturing where a total assembly of parts are required from several component parts. As the name indicates there is no control or dataflow among the different workflows. This is different from supply chain in the sense, there need not be a sequential or chain condition imposed among the workflows, namely each post condition of a preceding workflow is included in the precondition of the subsequent workflow as they are time independent. However, all the workflows need to be committed successfully eventually after a finite time of completion. If any one is aborted this workflow is unsuccessful and the initial states are restored.
Thus we may formally write the mutually exclusive-collectively exhaustive pattern by:
(i) $W = W(1) \cup W(2) \cup W(3).,.\cup W(n)$ where
$W(i) \cap W(j) = \varnothing$ for $1 \leq i,j \leq n$.
and
(ii) the boolean OR of flags A(i) satisfies:
[A(1)OR A(2).OR .A(n)] = 0 ;or the
boolean AND of boolean flags C(i) satisfies:
[C(1) AND C(2)...AND...C(n)] = 1 at eventual completion of all the workflows W(i)

d. *Negotiated choice service pattern- inviting tenders)*:
Here, the context is that *a Single customer bargains and negotiates with several suppliers* simultaneously to obtain a particular product at a bargain price, usually a function of the price and quality of the product (optimal choice). Here, the competition among the suppliers do not arise, since each one does not know the prices quoted by others. In manufacturing sector, this pattern arises, when a workstation is negotiating with other workstations for parts at different levels of completion. This workflow is successful, if and only if one of the workflows is committed and soon after that time the rest of the workflows are aborted respecting the contract. In the latter case the states are properly restored.
The negotiated choice pattern can be formalized by:
(i) $W = W(1) \cup W(2) \cup W(3).,.\cup W(n)$ where
$W(i) \cap W(j) = \varnothing$ for $1 \leq i,j \leq n$.
and
(ii) Optimal W(i) = C(i)
(iii)after committing W(i) all other W(j), $j \neq i$ are aborted.
e. *Auction Pattern*
Auction process is *a repetitive or iterative controlled competition among a set of clients and a Single auctioneer* for selling a specific object, coordinated by the auctioneer. In the auction pattern, the workflows happen between an auctioneer and several clients through communication and these are successively revised. The difference between the auction process and the negotiated choice is that in the auction process all the clients know each others bid, while in negotiated choice the sellers do not know other sellers quotes. Thus in auction there is dataflow among the successive workflows, namely the price quoted at the earlier bid.
The rules of the English auction pattern are as follows:
1. The auctioneer begins the process by a bid W(bid i=1) with a price P(1) and opens the auction.
2. At every succeeding bidding step $2 \leq i \leq M$, decided by a time stamp i , only one of the clients k among the N clients ($1 \leq k \leq N$) is permitted to bid with a price P(i,k) ; the auctioneer relays this information.
The bidding client k is called active and this client becomes inactive until a new round begins.
3.Then the auctioneer relays the information and opens a new bid; receives a response from another (or same) client j who bids a price P(i+1,j) strictly greater than a finite fixed amount of the earlier bid. P(i,k). (This is English auction; it can be modified for other auctions) .
4.If within a time-out period no client responds, the final bid is committed for the sale of the goods. The auction is closed by aborting earlier bids.
The English auction pattern can be formalized by workflow W(bid i)

(i) W(bid (1)) for price P(1) and await response ; then for from client k with a price P(i, k).
(ii) Then for each $2 \leq i \leq M$ and $1 \leq k \leq N$, the auctioneer examines P(i,k) , aborts W(bid (i-1)) and opens a new bid W (bid (i+1), and waits. If no response, commits W(bid (i) at price P(i,k) and ends.
Thus
(i) W* = iterative workflow = W(1)followed byW(2) followed by W(3). .....followed by. W(i), $1 \leq i \leq M$.
(ii) Bid W(i) = P(i,k)
(iii) Post W(i)= Pre W(i+1)= P(i,k)
(iv) If $P(i,k) \geq P(i-1,j)$ abort client j of W(i-1)
(v) bid W(i)+1)= P(i,k);
(vi) If no response after W(i+1),commit W(i) at price P(i,k) and end.
Table 1 summarizes all the above patterns.

TABLE 1: WORKFLOW PATTERNS IN COMMERCE AND CONTROL

Name of Pattern	Properties
Conventional Transaction	**Has ACID properties: *one customer, and one supplier* ; commitment. or abort.**
Supply Chain	**Workflow between *a customer and several sellers consisting of a set of transactions through intermediary agents*. The workflow is successful if and only if each individual transaction in the chain is successful.**
Mutually Exclusive-Collectively Exhaustive	**Workflow consisting of several *mutually exclusive and collectively exhaustive transactions between a customer (workstation) and several suppliers* (workstations) (to acquire different items(during manufacture). The workflow is successful if and only if all the transactions are committed.**
Negotiated Choice	**Workflow takes place between a *customer and several suppliers or peer a workstations negotiating for parts at different stages in manufacturing; the customer bargains and negotiates with several suppliers* simultaneously to obtain a particular product at a bargain price. This workflow is successful, if and only if one of the transactions is committed.**
English Auction	**A *controlled competition among a set of clients and an auctioneer* coordinated by the auctioneer. Here the last bid with the highest value is chosen for the sale of the goods and the auction is closed with one successful bidder.**

VII. MULTI-AGENT TOOLKITS

Shakshuki et al. [18], evaluate multiagent tool kits, such as: Java Agent development framework (JADE), Zeus Agent building toolkit and JACK Intelligent Systems. They consider Java support, and performance evaluation. The number of agents they consider is of the order of 32. For the implementation of the paradigm described here, further developments are needed in Agent technology, since we need a very large number of agents to simulate many real-life scientific applications.
Gorton et al [4] have evaluated agent architectures: Adaptive Agent architecture (AAA), Aglets developed by IBM, and the Java based architecture Cougaar. The paradigm described here is well-suited for implementing in Cougaar, a Java based agent architecture, since Cougaar is based on human reasoning. A Cougaar agent consists of a blackboard that facilitates communication and operational modules called plug-in that communicate with one another through the blackboard and contain the logic for the agent's operations. The use of blackboard and direct communication are useful for simulating the problems in Synthetic biology. Many other recent developments include Repast and other agent based software tools, Adamsky and Komosinski [1]. Repast is object oriented and has a discrete event scheduler, 2D visualization, and can be used with a variety of languages: java, C#, managed C++, Prolog etc and is available for several Platforms. Swarm Software is a mixture of Object oriented C and Java and can be very useful for swarming and related simulations.
Yet another tool is Star Logo, in [1]. Eiffel, Java , and UML are powerful languages to implement Mobile Object Programming Systems. They provide for software contract that captures mutual obligations through program constructs such as:" require [else]" for precondition and "ensure [then]"for post condition, assertions, invariants. Eiffel provides for a good semantics for exception handling through a "rescue" clause and "retry" clause for dealing with the recovery and resumption .The tool called "iContract", Kramer [7], provides developers with supports for design by contract using Java.
Jini and JXTA are currently being studied as useful to support agent based computing (see Web).

VIII. CONCLUSION

We described an agent based programming paradigm for self-organization and autonomy in computational networks. With the presently available tools in agent technology and large communication networks, several important advances can be made in choosing suitable topology of various types of networks (or patterns) used to achieve reliable communication and computation that are

failure proof, robust against attacks and capable of adaptation and self-organization. In our examples we illustrated only static examples. Time varying examples need further research and belong to the area of complex systems.

REFERENCES

[1] Adamsky,A and Komosinski, M. (2006), Artificial life Models in Software, Springer, New York.
[2] Dobson, S et al., (2010), Fulfilling the vision of Autonomic computing, IEEE Computer,January, pp.35-41.
[3] Garcia, J.O-G., and K-M Sim,(2010) Self-organizing agents for service composition in cloud computing, IEEE International Conference on Cloud computing Technology and Sciences, IEEE Computer Society, pp.59-66.
[4] Gorton,I, et al., (2004), Evaluating agent Architectures: Cougaar, Aglets and AAA, Lecture Notes in Computer Science, Vol.2940, pp.264-274, Springer Verlag, New York
[5] Jezequel, M., Train,M, and Mingins,C.(2000). Design Patterns and contracts, Reading: Addison Wesley.
[6] Kephart,J.O and Chess,D.M (2000),The vision of Autonomic computing, IEEE Computer, January, pp. 41-50
[7] Kramer,R (1998).“iContract-The java Design by contract tool,, 26 th Conference on Technology of object oriented Systems,(TOOLS USA’98) Santa Barbara.
[8] Little, M, Transactions and Web services (2003), Communications of the ACM, Vol.46, No 10, pp.49-54,., 2003.
[9] Meyer, B. (1992). Applying design by contracts, IEEE Computer 25(10), 40-52.
[10] Murthy, V.K. (2005), Contextual-knowledge management in peer to peer computing, International Journal of Knowledge-Based & Intelligent Engineering Systems, 9, pp.303-314.
[11] Murthy,V.K. and Krishnamurthy, E.V. (2005), Contextual information Management using Contract-based workflow, Proc. ACM Computing Frontiers, CF’05, Iscia, Italy.
[12] Murthy,V.K., and Krishnamurthy,E.V (2009). Multiset of Agents in a Network for Simulation of Complex Systems, Chapter 7 in Recent advances in Nonlinear dynamics and synchronization(NDS-1)-theory and applications**,** Eds.Kyamakya , K, et al, Springer Verlag, New York
[13] Odell,J.J., Objects and agents compared, (2002) , J. Object technology, Vol.1, No.1, pp, 41-53,May-June .
[14] Prehofer ,C and Bettstetter,C (2005), Self-organization in communication networks: Principles and design paradigms, IEEE Communication Mag,July,pp.78-85.
[15] Ramirez,A,R and Cheng, B.H.C,(2009), Design Patterns for developing dynamically adaptive systems,ICAC 09, Spain
[16]Shalloway, S and Trott,J.R. (2002),Design patterns Explained, Addison Wesley, New York.
[17] Smith,R.G., (1980),The Contract net Protocol:High level communication and Control in a distributed Problem Solver,IEEE Transactions on Computers, Vol.C29, 1104-1113.
[18] Shakshuki,E and Jun,Y,(2004), Multi-agent development toolkits: An Evaluation, Lecture notes in Artficial intelligence, Vol.3029, pp.209-218,Springer Verlag, New York
[19] Woolridge, M. (2002), An Introduction to Multi-Agent Systems, New York, John Wiley.

Professor E.V. Krishnamurthy is with the Computer Sciences Laboratory, Australian National University. He is the author of several books and papers in Computer Science and Information technology.

Address:

Australian National University,Canberra,ACT,0200,Australia

email: Evk.Krishnamurthy@anu.edu.au