# Autonomic Computing

## A Multiagent and Distributed AI perspective

Sumit Chachra

April 2, 2004

Intelligent Systems Laboratory

# Presentation Outline

- Introduction
  - What is Autonomic Computing ?
  - What is an Agent ?
  - MAS & DAI
  - Why MAS & DAI ?
  - Issues in MAS
  - Distributed CSP / COP
  - Agent Oriented Programming / Environments
- Related Work / Literature Survey
- Solution Approaches
  - Resource Allocation
    - Sensor Networks
  - Distributed Breakout
  - Partial Constraint Satisfaction
- Conclusions and Future Work

# Introduction - What is Autonomic Computing ?

Phrase coined by Dr. Alan Ganek, Vice President of Autonomic Computing. According to IBM an Autonomic Computing System:

- needs to “know itself”
- must configure and reconfigure itself under varying conditions
- never settles for the status quo - it always looks for ways to optimize its workings
- must perform something akin to healing
- must be an expert in self-protection

## Contd . . . Introduction - What is Autonomic Computing ?

- must know its environment and the context surrounding its activity, and act accordingly
- cannot exist in a hermetic environment
- will anticipate the optimized resources needed while keeping its complexity hidden

# Introduction - What is an Agent ?

An **autonomous agent** is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future . . . S. Franklin and A. Graesser, “Is it an agent, or just a program?: A taxonomy for autonomous agents,” in *Proceedings of the Third International Workshop on Agent Theories, Architectures and Language*. Springer-Verlag, 1996

An agent is a **computational entity** such as a software program or a robot that can be viewed as perceiving and acting upon its environment and that is autonomous in that its behavior at least partially depends on its own experience . . . G. Weiss, Ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999

## Contd . . . Introduction - What is an Agent ?

Agents are:

**Computational Entities** Physically exist in the form of programs that run on computing devices

**Autonomous** To some extent they have control over their behavior and can act without the intervention of humans or other systems.

**Interacting** They may be effected by other agents or by humans in pursuing their goals and executing their tasks.

# Introduction - MAS & DAI

- **MAS** stands for Multi-Agent Systems or environments where more than one agent exists.
- At times the number of agents may be too numerous to deal with them individually. It is then more convenient to deal with them collectively, as a society of agents.
- Distributed AI (**DAI**) is the study, construction and application of multiagent systems, that is, systems in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks.
- DAI focuses on coordination:
  - Cooperation
  - Competition

# Introduction - Why MAS & DAI ?

The current computing systems/infrastructure lend themselves well to be modelled as MAS:

**Inherent Distribution** - In the sense that data and information to be processed arise at geographically different locations/times etc. For example computers in a network/wireless computing devices etc.

**Inherent Complexity** - Too large tuo be solved by a single agent

**Natural View of Intelligent Systems** - Intelligence and interaction are deeply coupled

**Speed-up and Efficiency** - Agents can operate asynchronously and in parallel

## Contd . . . Introduction - Why MAS & DAI ?

**Reliability** - Failure of 1 or several agents does not make overall system useless

**Scalability** - New agents can be added or removed easily

**Costs** - More cost effective than a centralised system

# Introduction - Issues in MAS

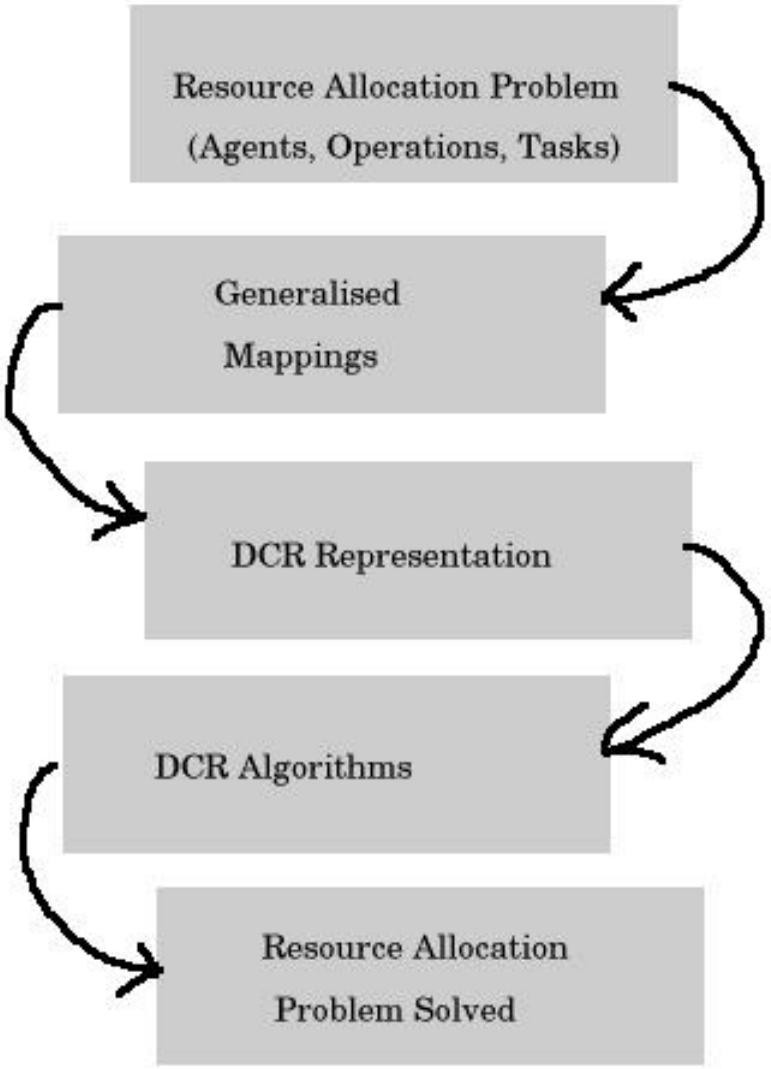
- How to enable agents to decompose their goals and tasks, allocate sub-goals and sub-tasks to other agents, and to synthesize partial results and solutions
- How to enable agents to negotiate and contract, and which protocols should they use
- How to reconcile disparate viewpoints and conflicts and synthesize views and results
- How to enable agents to form and dissolve organizational structures – teams, alliances etc.
- How to enable agents to reason and represent other agents
- How to enable them to find out whether they have achieved progress in their coordination efforts

# Introduction - Distributed CSP / COP

- A distributed CSP is a CSP in which variables and constraints are distributed among autonomous agents.
- Each variable $x_j$ belongs to one agent $i$, represented as $belongs(x_j, i)$.
- Constraints are also distributed among agents, and the fact that an agent $l$ knows a constraint predicate $p_k$ is represented as $known(p_k, l)$.
- We say that a distributed CSP is solved iff the following conditions are satisfied:

  - $\forall i, \forall x_j$ where $belongs(x_j, i)$, the value of $x_j$ is assigned to $d_j$, and $\forall l, \forall p_k$ where $known(p_k, l)$, $p_k$ is true under the assignment $x_j = d_j$.

# Contd . . . **Introduction - Distributed CSP / COP**



# Introduction - Agent Oriented Programming / Environments

The Foundation for Intelligent Physical Agents (FIPA) is an international standards body working for interoperability between agents and agent platforms, has defined specifications for agents, agent management services and agent communication languages. There are a number of environments available for programming agent-based software:

**ABLE** - Better known as the Agent Based Learning Environment. Developed by IBM, this is a toolkit for building multiagent autonomic systems. Easy to build hybrid intelligent agents which draw on the strengths of each technology while compensating for any weaknesses.

**JADE** - The Java Agent Development framework is another FIPA-compliant multiagent toolkit

ZEUS, FIPA-OS, JatLite and AgentBuilder are a few others.

# Presentation Outline

- Introduction
  - What is Autonomic Computing ?
  - What is an Agent ?
  - MAS & DAI
  - Why MAS & DAI ?
  - Issues in MAS
  - Distributed CSP / COP
  - Agent Oriented Programming / Environments
- Related Work / Literature Survey
- Solution Approaches
  - Resource Allocation
    - Sensor Networks
  - Distributed Breakout
  - Partial Constraint Satisfaction
- Conclusions and Future Work

# Related Work / Literature Survey

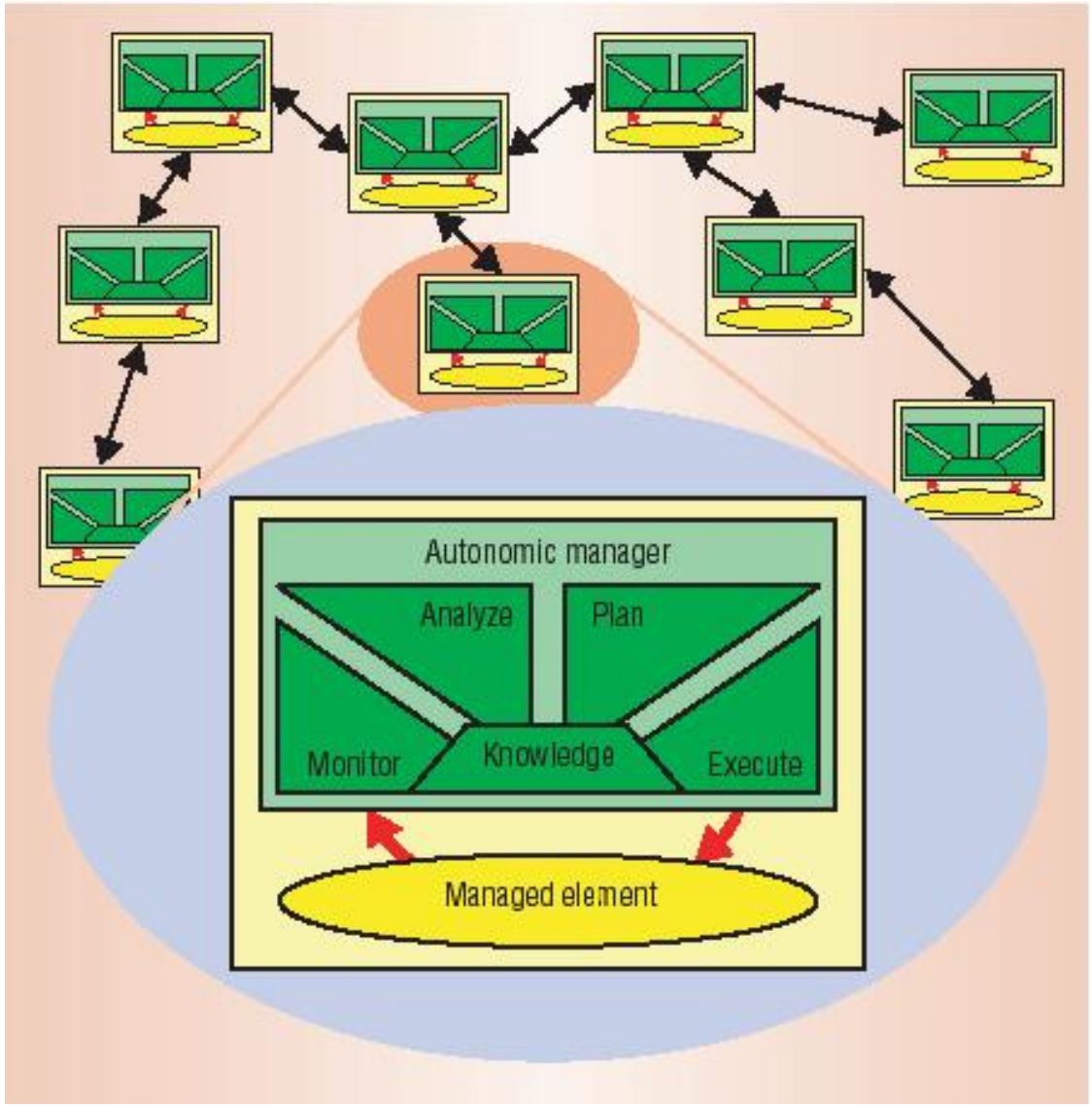
J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, pp. 41–50, Jan. 2003

The authors envisage that pervasive computing will become a nightmare given the constant increase in complexity and interconnectivity of systems, hence systems need to be made autonomic and be able to make decisions on their own, eliminating the human from the loop.

Some challenges which have been outlined:

- Machine learning in MAS environments not well studied
- Standardization of directory services, expression of system capabilities etc.
- Understanding of how business level policies/goals effect lower level performance and vice-versa
- Development of negotiation mechanisms

Contd . . . J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *IEEE Computer*, pp. 41–50, Jan. 2003



# Related Work / Literature Survey

N. R. Jenning, “On agent-based software engineering,” *Artificial Intelligence*, vol. 117, pp. 277–296, 2000

A qualitative analysis of the merits and demerits of agent-based software engineering and its comparison with the current object oriented paradigm.

Two central arguments of this paper can be expressed as:

**The Adequacy Hypothesis -** Agent oriented approaches can enhance modelling/design capabilities of making complex and distributed software systems.

**The Establishment Hypothesis -** This approach will succeed as a main stream software engineering paradigm.

# Related Work / Literature Survey

D. D. Roure, M. A. Baker, N. R. Jennings, and N. R. Shadbolt, "The evolution of the grid," *Grid computing: making the global infrastructure a reality, Wiley*, pp. 65–100, 2003

The approach to computing, using geographically distributed resources for parallel and distributed applications has several names - metacomputing, scalable computing, global computing, internet computing and lately as grid computing. The evolution has been divided into three generations:

**First Generation** Characterised by projects such as FAFNER and I-WAY. Basically task-farmed a large number of fine-grain computations or connected existed high bandwidth networks.

**Second Generation** Represented the development of middleware, software infrastructure and toolkits. Key grid technologies such as CORBA, Jini & RMI, peer-to-peer computing etc. emerged.

**Third Generation** There is a strong sense of automation in the third generation systems, for eg. when humans can no longer deal with the scale and heterogeneity, but delegate to processes to do so. Configuration and repair cannot remain manual tasks.

# Related Work / Literature Survey

J. Sabater, C. Sierra, S. Parsons, and N. R. Jennings, "Engineering executable agents using multi-context systems," *Journal of Logic and Computation*, vol. 12, no. 3, pp. 413–442, 2002

They introduce three new ideas over previous work on advantages of multi-context systems:

- grouping of contexts together into modules, giving another level of abstraction
- idea of bridge rules which delete formulae from certain contexts (as opposes to just introducing them), which allows the modelling of consumable resources.
- time delay in the execution of bridge rules in order to allow inter-context synchronization.

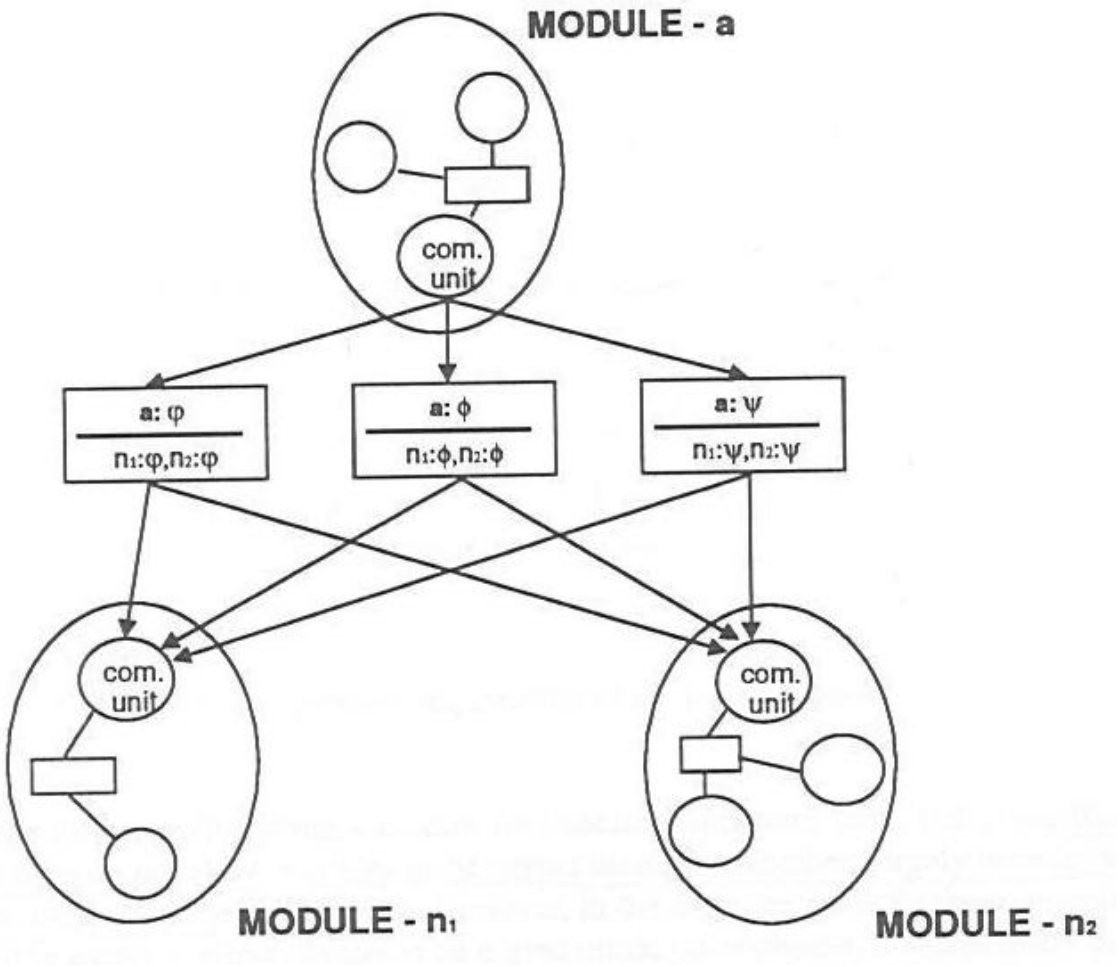
Work gains importance since it allows different agents to follow different logics as long as there exists a common one for communication. This can easily be extended and understood in the real world domain, where each company/server/software follows its own particular logic.

Contd . . . J. Sabater, C. Sierra, S. Parsons, and N. R. Jennings, "Engineering executable agents using multi-context systems," *Journal of Logic and Computation*, vol. 12, no. 3, pp. 413–442, 2002

Bridge rules can be understood as rules of inference with premises and conclusions in different units. For instance:

$$\frac{u_1 : \Psi, u_2 : \varphi}{u_3 : \Theta} \quad (1)$$

$$\frac{u_1 > \Psi, u_2 : \varphi}{u_3 : \Theta} \quad (2)$$



# Related Work / Literature Survey

M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara, "The distributed constraint satisfaction problem: Formalization and Algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 5, September/October 1998

**General assumptions:**

- Each agent has exactly one variable
- All constraints are binary
- Each agent knows all constraint predicates relevant to its variables

This work is focussed on discussing 2 algorithms for solving DCSP problems:

**Asynchronous Backtracking** Allows agents to run concurrently and asynchronous Directed constraint graph which decides the flow of messages.

**Asynchronous Weak-Commitment Search** Uses the min-conflict heuristic as a value ordering heuristic. Abandons partial solutions and restarts search process if it is a *nogood*.

# Related Work / Literature Survey

P. Scerri, J. Modi, W. Shen, and M. Tambe, “Are multiagent algorithms relevant for real hardware? A case study of distributed constraint algorithms,” in *Proceedings of the Eighteenth Annual ACM Symposium on Applied Computing*, Mar. 2003

- Aim was to do efficient and optimal resource allocation in sensor networks so as to track a particular target.
- Applied a particular distributed resource allocation algorithm (Adopt-SC) developed for an abstract coordination problem in a real hardware application.
- Probabilistic representation of resources and tasks, to deal with uncertainty and dynamics.
- Task uncertainty, real-time constraints and tast dynamism are the “domain details” dealt by using a two-layer architecture.

# Presentation Outline

- Introduction
  - What is Autonomic Computing ?
  - What is an Agent ?
  - MAS & DAI
  - Why MAS & DAI ?
  - Issues in MAS
  - Distributed CSP / COP
  - Agent Oriented Programming / Environments
- Related Work / Literature Survey
- Solution Approaches
  - Resource Allocation
    - Sensor Networks
  - Distributed Breakout
  - Partial Constraint Satisfaction
- Conclusions and Future Work

# Solution Approaches - Resource Allocation

A distributed resource allocation problem can be represented as a triple $< A_g, \omega, \theta >$ where:

- $A_g = \{A_1, A_2, \ldots, A_n\}$ is a set of agents
- $\omega = \{O_1^1, O_2^1 \ldots, O_p^i, \ldots, O_q^n\}$ is a set of operations, where operation $O_p^i$ denotes the $p^{th}$ option of agent $A_i$.
- $\theta = \{T_1, T_2, \ldots, T_n\}$ is a set of tasks, where each task $T \in \theta$ is a set of sets $\{t_1, t_2, \ldots, t_n\}$ and each $t_i \in T$ is a set of operations. Each $t_i$ is called a *minimal set*.

## Contd . . . **Solution Approaches- Resource Allocation/Sensor Networks**[1]

Sensor networks are the ideal domain for application of resource allocation algorithms/constraint satisfaction/optimization techniques.

As computer networks (and computational grids) become increasingly complex, the problem of allocating resources within such networks, in a distributed fashion, will become more and more of a design and implementation concern . . . V. Lesser, C. L. Ortiz, and M. Tambe, Eds., *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer Academic Publishers, 2003

Hence techniques applied to solve sensor network problems(using simple Doppler radar sensors/DARPA challenge problem) can easily be extended to apply computer systems. The major change is in the kind of abstractions that are applied to the real-world problem.

[1] Work being done along with Ted

# Solution Approaches - Distributed Breakout

- The breakout algorithm is an iterative improvement algorithm
- All variables have tentative initial values
- Each constraint is assigned a weight(initally 1)
- The *flawed* solution is revised by using a hill-climbing search
- Occasionally they will be trapped in a *local-minima*
- The summation of the weights of violated constraints is used as an evaluation function

## Contd . . . **Solution Approaches - Distributed Breakout**

When trapped in a local-minimum, the breakout algorithm increases the weight of violated constraints in the current state by 1 so that the evaluation value of the current state becomes larger than those of the neighboring states.

```
procedure breakout
      until current-state is solution do

      if current state is not a local-minimum
            then make any change of a variable value that
            reduces the total cost

            else increase weights of all current nogoods
      end if; end do;
```

## Contd . . . Solution Approaches - Distributed Breakout

**2 problems while applying the `procedure breakout` to distributed CSP's:**

- If 2 neighboring agents are allowed to change their values at the same time(to make use of parallelism), the evaluation value may not be improved, and oscillations may occur.
- To detect that the agents as a whole are stuck in a local-minimum, agents have to exchange global information among themselves.

**These problems can be solved by the following ideas:**

- Only that agent among the neighboring agents, which can maximally improve the evaluation value is given the right to change its value.
- Each agent instead detects that it is in a quasi-local-minimum, which is a weaker condition and requires local communication only.

## Contd . . . **Solution Approaches - Distributed Breakout**

***NOTES:***

- The agents exchange *ok?* and *improve* messages amongst their neighbours iteratively.
- A *termination_counter* is maintained to detect termination of algorithm. If its value is $d$, it means that every agent whose distance from agent $x_i$ is within $d$ satisfies all its constraints.
- When *termination_counter* of some agent becomes equal to *max_distance*, agents terminate the execution of the algorithm.
- Most CSP algorithms can be converted to COP algorithms. For example, in the above an appropriate cut-off(value of *max_distance*) would do the trick.
- Other DCSP algorithms such as 'asynchronous backtracking' and 'asynchronous weak-commitment search' work in a similar manner.

# Solution Approaches - Partial Constraint Satisfaction

When real-life problems are modelled as a CSP, the resulting CSP is often over-constrained and almost all conventional CSP algorithms do not produce a solution.

A partial CSP is formall described as:

$$< (P, U), (PS, \leq), (M, (N, S)) > \quad (3)$$

- $P$ is an original CSP
- $U$ is a set of 'universes'
- $(PS, \leq)$ is a problem space, where $PS$ is a set of CSP's and $\leq$ is a partial order over $PS$
- M is a distance function over the problem space
- $(N, S)$ are necessary and sufficient bounds on the distance between P and some soluble member of PS

## Contd . . . **Solution Approaches - Partial Constraint Satisfaction**

**Extensions:**

- Maximal CSP's
- Hierarchical CSP's - Constraints are divided into several groups, which are ordered according to the importance value of constraints within them
- Soft and hard constraints
- Weighted CSP's - Each constraint has a weight. Become similar to what we discussed before

# Solution Approaches - Other approaches

1. Market Based Approaches, Auctioning, Contract Net Architecture, Blackboard systems

2. Methods of modelling Agent Architectures - Reactive, BDI (Belief Desire Intention), Layered etc.

3. Distributed planning and problem solving techniques

4. Learning techniques such as Q-Learning and Reinforcement learning in general can be applied to MAS for iterative performance improvement.

5. Learning from own and other agents experience.

# Presentation Outline

- Introduction
  - What is Autonomic Computing ?
  - What is an Agent ?
  - MAS & DAI
  - Why MAS & DAI ?
  - Issues in MAS
  - Distributed CSP / COP
  - Agent Oriented Programming / Environments
- Related Work / Literature Survey
- Solution Approaches
  - Resource Allocation
    - Sensor Networks
  - Distributed Breakout
  - Partial Constraint Satisfaction
- Conclusions and Future Work

# Conclusions and Future Work

- Viewing autonomic elements as agents and autonomic systems as multiagent systems makes it clear that agent-oriented architectural concepts will be critically important.
- Machine learning and specifically reinforcement learning lends itself well to multiagent systems applied for autonomic applications.
- Simulation of autonomic components and systems, trying to model them in the best possible manner along with abstracting them so as to be able to apply problem solving techniques easily
- Identify real world constraints and issues in computer systems, which come under the purview of problems autonomic computing aims to solve
- Working on applying DCOP algorithms to simulations of a particular Doppler radar sensor networks

www.ece.arizona.edu/∼chachra/autonomic

**Username:** isl

**Password:** autonomic