Bio-inspired Self-organization Methods and Models for Software Development

Daniel J. Dubois

Advisor: Prof. Elisbetta di Nitto

Dipartimento di Elettronica e Informazione
Politecnico di Milano

GIIS 2009 – Salerno, Italy – Sep 14th, 2009
Context and Problem

- **Context**: decentralized peer-to-peer collaborations of services/components.
  - Suitable for pervasive systems, often obtained as collaboration of pre-existing components.

- **Problem**: how to enable self-organization without increasing complexity of computing systems.
Objectives

- Existing solution:
  - Autonomic computing: systems that self-configure, self-heal, self-repair, and self-optimize [IBM manifesto].
  - Superimposing on a software system a control layer (example: the MAPE cycle).

- Our Goal:
  - Provide a more decentralized solution based on self-organizing logic
    - use analogies taken from the biological world to accomplish complex tasks using simple rules.

- Simple elements
- No centralization / hierarchies
- Spontaneous behavior
- High number of elements
- High uncertainty
Self-Organization

The spontaneous evolution of a system into an organized form in the absence of external pressures.

• Self-organization algorithms are able to guide the evolution of the system toward a specific goal with no or minimal user intervention.

• Example: Vineyard control system.
Example: Vineyard control system
Bio-inspired Self-Organization: forms of self-organization that come from observed phenomena of the natural world.

- Examples from the natural world: finding the optimal path in ant colonies, fireflies synchronization, etc.

• Advantages
  – Scalability,
  – Robustness,
  – Ability to handle malicious or malfunctioning components,
  – …

• Disadvantages
  – We miss proper design approaches supporting their application.
Applying Self-organization: Ingredients

- Self-organization principles.

- Algorithms.
  - May combine bio-inspired principles.

- Development guidelines
  - List of steps to be executed to move from the model of algorithms/principles to the actual implementation.

- Validation aspects:
  - Is the application problem solved?
  - Does the algorithm converge?
  - What about transitory behavior?
Bio-inspired Principles

• Derived from the observation of natural phenomena.

• They may be composed together and translated into real deployable algorithms.

• Most important principles:
  – Noise
  – Emergence
  – Diffusion
  – Stigmergy
  – Evolution

Sep 14th, 2009

Bio-inspired Methods and Models for Software Development
Noise Principle

“a strategy of errors”

Definition: the use of some kind of perturbations to move a system away from its expected goal in the short term, but increasing the probability to improve it in the long term.
Emergence Principle

“a team is better than the sum of its individuals”

Definition: capability of a system composed by multiple components to reach (with a certain probability) a global goal by achieving local goals at component level.
Communication principles: Diffusion and Stigmergy

**Diffusion**

"gossiping is not necessarily a bad habit"

**Definition**: A method of disseminating information among many interconnected components. Messages are sent to all known nodes (flooding) or random nodes (gossiping).

**Stigmergy**

"leave a note and I’ll read it"

**Definition**: A method of communication that stores information in the environment.
Evolution Principle

**Definition**: capability of the system to improve itself using a natural selection process among its components.

- Best components tend to survive, worst components tend to die.
Stochasticity Property

- Bio-inspired principles apply in a probabilistic way.
- Increase the probability that the system reaches a desired state, but they do not assure that the system actually reaches it.
- The question is no longer whether systems work, but what is the probability that they work.

- All properties of the system (functional and non-functional) are expressed in stochastic terms:
  - system keeps working in presence of unexpected behaviors
  - absence of guarantees on when a specific goal is reached.
Example: Traffic Reduction in Publish-Subscribe Systems

- **Characteristics of pub/sub systems:**
  - Many components and a dispatcher;
  - Components may send messages and subscribe to some classes of messages;
  - The dispatcher forwards received messages to the correct subscribers;
  - Asynchronous (loosely-coupled) communication model.

- **Problem:**
  - The dispatcher may be a single point of failure.
  - Need of a dispatcher composed of multiple elements (called brokers).
  - Finding the optimal overlay network for these brokers to reduce the total traffic.
Idea: Overlay Self-Organization

- Finding a method to self-optimize the broker overlay network spontaneously.
- Use only simple local rules and the limited local knowledge of each broker.

Before Self-organization:
- Broker 1: Publish: soccer, Subscribe: sport
- Broker 2: Publish: soccer
- Broker 3: Subscribe: sport

After Self-organization:
- 1 saved message
- Broker 1: Publish: soccer
- Broker 2: Publish: soccer
- Broker 3: Subscribe: sport
Overlay Self-Organization Algorithm

Phase 1: Training

• Collect traffic information.
• Count how much traffic belonging to a certain class (subscription) has been received/forwarded to neighbor brokers.

Phase 2: Reconfiguration

• Use information collected during Phase 1 to predict the effects of a reconfiguration using a utility function.
• Evaluate such function for a significant amount of possible reconfigurations and apply non-deterministically a random one or the one that maximizes it.
Results

- What we have learned from experimental results:
  - The algorithm scales well;
  - Convergence is preserved also in case of churn and selfishness;
  - Artificially added noise to increase the optimality of the solution.
Implementation Issues

• The proposed model does not capture:
  – Conflicts when handling concurrent iterations.
  – Activation/Termination problem.
  – Choice of interval between algorithm iterations.

• Possible solutions:
  – Locking mechanisms.
  – Keeping the algorithm always active VS finding a propagation strategy.
  – Heuristics for automatically determining iteration intervals.

• Stochasticity:
  – No guarantees that the optimal solution is obtained, however the probability may be estimated.
Creating Self-org algorithms for Software Systems

- Identify principles/algorithms:
  - Analyze the problem/requirements.
  - Choose the principles/algorithms that are most suitable for that class of problem.
Creating Self-org algorithms for Software Systems

• **Building Model:**
  - Compose and instantiate principles/algorithms identified so far.
  - Prove that the model satisfies its requirements.

---

**The Problem of Validation**

• Validation techniques from:
  - System theory;
  - Optimization theory;
  - Game theory;
  - Statistics;
  - Theoretical Computer Science.
Creating Self-org algorithms for Software Systems

- **Implementation:**
  - Translate the model into a deployable form.

Principles/Algorithm identification

Build a model and perform a preliminary validation

Implementation

Identify and solve real-systems problems using a formal and/or an experimental approach
Creating Self-org algorithms for Software Systems

- Identify and solve problems not captured by the model:
  - Race conditions.
  - Component synchronization.
  - Frequency of iterations.
  - Amount of communication.
  - …
Conclusions

• We explore the problem of applying bio-inspired self-organization techniques to real systems.

• **Successful applications:**
  – Distributed Decentralized Load Balancing in Heterogeneous Networks.

• **Future work:**
  – A complete systematic methodology consisting of:
    • a list of steps and design patterns based on bio-inspired principles.
    • heuristics that can be used to calibrate the algorithms.
  – Identify suitable validation approaches.
References

QUESTIONS?