Engineering Self-organization and Emergence

Franco Zambonelli

March 6, 2006

Outline

- Part 1: Engineering Self-organization
  - Direct vs. Reverse Engineering
  - Engineering Self-organization
  - Examples
  - Evolutionary approaches
- Part 2: Engineering and Emergence
  - Micro vs. Macro Scale
  - The Meso Scale
  - Control via the Environment
- Part 3: Examples
  - Controlling Cellular Automata
  - Controlling Swarms with Co-Fields
- Conclusions
Part 1

- Engineering Self-organization

Why Engineering of Self-organization

- It appears like swarm intelligence and, in general, self-organization, may have useful applications for modern distributed systems
  - Routing, coalition formation, synchronization, etc.
  - Enforcing useful properties of self-configuration, self-management, self-adaptation, etc.

- We must turn theory into practice
  - To produce reliable distributed software systems
  - In a reproducible way
  - Applying rigorous methodologies

- That is, we need a **discipline of engineering self-organization**!
Direct vs. Reverse Engineering

- Traditionally, software development use a **direct engineering approach**
  - Start from the problem
  - Decompose it
  - Solve the various problems
  - Develop the system that solve the problem
- When getting inspiration from natural phenomena, however, we use a **reverse engineering approach**
  - Start from a phenomena which appears to solve a similar problem
  - Understand how it work (reverse engineering of the phenomena of emergent behaviors)
  - Adapt it to the problem to solve some real-world problem

Direct Engineering of Self-organization

- Self-organization typically requires a bottom up approach
  - Each component follows a set of specific rules
  - The collectivity of components following such rules determines a globally self-organized behavior
- **Direct engineering of self-organization**
  - May apply to not so complex systems and algorithms
  - In some cases, we can easily determine with "pencil and paper" the rules leading to the desired behavior
  - e.g., self-localization, time synchronization
- This is direct engineering because
  - We start from the problem
  - And can design a self-organizing algorithm that solve it
Examples of Direct Engineering

- Self-localization
  - We have already discussed about self-localization algorithms
  - Self-configuring a reference frame on a network
- This is a sort of “direct” form of self-organization
  - We can clearly understand from design
  - That the distributed algorithm will converge
  - Into a coherent reference frame
- In other words
  - The behavior of the system can predictably (deterministically) converge to a single final configuration
  - Despite the impossibility of controlling the execution of single components, i.e.,
  - Despite the intrinsic non-determinism of processes at the level of single components (i.e., despite the impossibility of controlling the exact flows of messages and activities)
- These are also often called “self-stabilizing algorithms”
  - We know the will stabilize
  - Simply disregards to control the details of how such stabilization will take place

Direct vs. Reverse Engineering

- For many other problems, it may be difficult to design a self-organizing solutions
  - Impossible to determine the set of local rules and of local interactions
  - That achieve the needed pattern of self-organization
  - This calls for reverse engineering approaches
- Emergent behaviors
  - Several self-organizing systems exhibit emergent behaviors
  - They cannot be predicted from the behavior of individuals
  - Often, systems behave in complex unexpected ways...
- Reverse Engineering of self-organization implies
  - Observing an interesting self-organized behavior (with possibly applications to distributed systems)
  - Understanding why and when such behavior arise
  - And try to reproduce and control it
Examples of Reverse Engineering

- Path finding by ants
  - And their applications in routing or task assignment
- We have seen as this behavior emerges from the system
  - Without any a priori “self-localization”
  - Without the possibility of easily recognizing “by design” that the stated behavior would have emerged from that simple local rule
  - Without convergence to a single local state, but dynamically establishing dynamic “self-organization” patterns
- In other words
  - We have reverse engineered an observed phenomenon
  - We have reproduced it in a network
  - We accept that the behavior that will emerge from the system will be non-deterministic (several equivalent configurations possible)
  - Still, any of that behavior will be useful and will be achieved in a cost effective way

Examples of Direct vs. Reverse Engineering

- Synchronization in sensor networks
- There, we have two possible solutions
- A normal self-stabilizing algorithm
  - See e.g., synchronization on sensor networks
  - Neighbor nodes synchronize with each other
  - And the process propagates in the whole network
  - And eventually it converges into a global synchronization
- An emergence algorithms
  - E.g., a model of “firefly synchronization”
  - In which simple local rules
  - Are shown to make a global synchronization of activities emerges
- What choice to make depends on the systems’ constrains, on the costs of the approaches, and on simplicity of deployment
Methodology of Reverse Engineering

- Traditional methodology for direct engineering
  - Analysis
  - Design
  - Implementation
  - Test
- Reverse engineering methodology
  - Observe phenomena
  - Map it into a real-world problem
  - Simulate and tune
  - Reproduce

Reverse Engineering of Self-Organization: the Methodology

- Here’s a typical process of reverse engineering
- Simulation plays a very important role
  - To prove the applicability of concepts!
Example of Reverse Engineering

- "Hey, ants find shortest paths in a very effective and adaptive way"
  - After all, I have to find adaptively shortest path in a network...Then...
- Let’s reverse engineer this
  - Understand how ant foraging works (done!)
  - Apply metaphors (environment = ants = control packets; receiver = nest; sender = food; pheromones = routing tables)
  - Tune parameters (number of ants, evaporation of routing tables)
  - Simulate on a network simulator
  - Deploy as a real routing algorithm

Patterns of Self-Organization

- For several problems
  - There exist a variety of self-organizing phenomena
  - Already studied and deeply analysed
  - That can be exploited as a “ready-to-use” solution
- This define a “pattern-based” solutions
  - Given a problem that correspond to a specific pattern
  - Have a look at a “catalogue” to see if some self-organizing algorithm exist that solve the problem
- Babaoglu 2005, Parunak 2005 propose such an approach
  - But of course, one must be very lucky, and this is not a general purpose solution...
Use Self-organization with Caution!

- Again, as it is the case with any new technology
  - It may work, but it may sometime goes wrong
  - Too expensive, too slow, not leading to the exact same behavior as needed, etc.
  - Enthusiasm may be dangerous ("OK, now let's re-write the whole system in swarm intelligent terms!")
- So what?
- Use swarm intelligence prudently
  - Build system in traditional ways
  - Enrich them with moderate amounts of swarm intelligence
- Unfortunately
  - A general methodology to properly mix traditional behavior (e.g. rational agents) and swarm intelligence (e.g. ants) is missing
  - Still, it is possible, and nature does that every day!

Where to Put Intelligence

- When faces with swarm intelligent systems and multiagent systems
  - The issue arise on "where to put intelligence" in our systems
- Should we rely on rationale "intelligent" agents
  - And have them understand and reach goals, and understand and adapt to situations as individuals
- Or should we relay on "stupid" agents
  - And rely on the swarm to achieve global goals and adaptive behavior?
- In general
  - What to put in agents and what in the system
- This is a very sensible design choice
- How did nature decide?
Evolution in Nature: Individuals vs. Societies

- Evolution has exploited both directions
  - Smarter species have evolved from non-smart ones
    - With a better fitness to survive, e.g., to solve problems, as **individuals**
  - Social insects with swarm intelligent behavior have evolved from non-social insects
    - With a better fitness to survive, e.g., to solve problems, as **swarms**
- In addition, species have evolved that have both social and individual capabilities
  - E.g., humans!
  - Which are intelligent per se
  - And which are intelligent in terms of societies!

![Evolution in Nature: Individuals vs. Societies](image)

Evolution in Nature: Individuals and Societies

- In several cases,
  - The capabilities of individuals
  - Co-exists with the capabilities of the swarm
- Humans are the most representative example
  - Individual behavior
  - We indeed are "intelligent" and behave, in most of the cases "rationally"
  - Using direct interactions/communications
- Yet, several aspects of our lives are rules by "swarming" behavior
  - We forget our rationality (or at least it is less apparent)
  - And act/interact based on what we feel on the environment
  - In somewhat unconscious ways
Examples of Swarming in Humans

- Following trends
  - Are we 100% confident that we like current way of dressing?
  - Or it is rather a form of “aggregated” behavior driven by the mass behavior?
- Synchronized clapping
  - Are we looking for synchronization?
  - Or does this emerge naturally and inconsciously?
- Social conventions
  - We act socially, based on conventions which we did not decide
  - And that in several cases we never perceive
  - E.g., driving on the right, walking without hurting, tuning the volume of our voice depending on context, etc.
  - Emergent footpaths
  - Walked grass act as pheromones
  - And we tend to follow pheromones
- Emergent footpaths (see lecture on swarm intelligence...)

Universality of Swarm Intelligence (1)

- After all, it appears like phenomena of swarm intelligence are rather common
  - Not only in “stupid” animals
  - But also in intelligent, rational, animals
- Simply
  - In intelligent animals, the perception of individual intelligence dominates over the behavior of the swarm
  - In stupid animals the intelligence of the swarm is much more evident and surprising
- However, this consideration suggest that the presence of “swarm intelligence” may be a rather universal property of complex systems of autonomous entities
Universality of Swarm Intelligence (2)

- For all types of swarm intelligence we have seen that
- There are values of the parameters by which “nothing happens”
  - E.g., pheromones do not diffuse or evaporate too fast and no global coherent behaviors emerge, the system is in thermodynamic equilibrium, with all ants wandering randomly
- There are values of the parameters for which the system ends up in chaotic state
  - E.g., pheromones diffuse very fast but evaporate fast too, thus inducing a chaotic, non meaningful, behavior in ants
- In between, there are values of parameters that enables
  - The system to stay out of equilibrium
  - And self-organize in global patterns
- Again, we are dealing with systems “at the edge of chaos”!

Inventing Swarm Intelligence Phenomena

- So far, we have seen how to use observed phenomena in the real world
  - That works in several cases
  - However, it also introduced the risk of having “solutions in search of a problem”
  - Which is often dangerous for science as well as for real business!
- What we should ask is
  - Has nature already played all its cards?
  - Or there could be swarm intelligent behaviors
  - Not exhibited by nature
  - That we could build and exploit?
- Can we be “gods” of our own artificial ecology
  - Inventing our own insects
  - Our own laws of interactions
  - That lead to the needed solutions (robust and reliable)
  - How can we do? This is by no means easy…
Evolutionary Approaches

- It is difficult to "invent" from scratch new swarm intelligent phenomena solving a specific problem
  - Only a few genius can "see" them
- A possible approach to "discover" new swarm intelligent system by getting inspiration from evolution
- Start with a population with a randomly selected behavior
  - Or with some forms of randomly selected behavior
  - Or both randomly selected
  - Simulate the ecology
  - Measure its "fitness" in solving the specific problem via self-organization
- Mutate the system by creating new "species" with new forms of interactions
  - Simulate mutated ecologies
  - And have only those ecologies that behave better survive
    - Reproduce with each other
    - Mutate again
    - And so on recursively...

Part 2

- Engineering Emergence
The “Micro” Perspective

- Traditional Mainstream (Software) Engineering adopts a “Micro” approach
  - Focus on individual components and their interactions
  - Full predictability at each level
  - Controlled non-determinism
  - Direct engineering
- And this is here to stay for a multiplicity of applications
  - B2B, workflow systems, safety-critical systems
- Though getting more and more “autonomic”
  - Multiagent systems, automated negotiation, environmental dynamics, internal control loops, etc.
  - But this is far from emergence and self-organization...

The “Macro” Perspective

- Dealing with large-scale distributed systems
  - Dynamic P2P networks, Wireless sensor networks, multiagent systems ecologies, self-assembly
- Global and emergent behavior
  - Observing AND/OR Enforcing
  - Reverse engineering of self-organization
- Focus on “macro” aspects
  - No control over single components
  - Non-determinism
  - Local rules → global behavior
- This is where current research is
But “Micro” and “Macro” are not Independent worlds…

- In most of real-world situations
  - Micro systems are developed
  - And situated in an operational environment on which we have no full control
  - And which cannot be "stopped"
- In other words:
  - Micro-scale systems are immersed into macro-scale one
- And this is indeed always the case for
  - Networking
  - Service-oriented computing
  - P2P Computing
  - Pervasive Computing

The “Meso” Scale

- The micro and the macro scales co-exist and influence each other
  - How the local behavior affects the local one
  - And vice versa
- We must take both into account in system design and management
  - How can we predict at both levels?
  - How can we enforce properties at both levels?
More General “Meso” Scale Scenarios

- We can also consider “macro” into “macro”
  - A “self-*” system which we know well
  - Interact with another one
  - E.g., Gnutella into the Internet

- Problems
  - How do we know the two systems preserve their own properties?
  - How can we re-tune them to ensure properties at both levels?

The Problem of Emergent, Unexpected Behaviors

- Very complex systems (i.e., macro-scale self-organizing systems), unlike simple ones
  - Exhibits non-linear relationships between structure and behavior
- Changes in structure can
  - Do nothing or
  - Dramatically affect the behavior
Examples (non computational)

○ Traffic Management
  - We know well how a roundabout (a sharp example of self-organizing system) works per se
  - But what about its impact in a complex network of streets?

○ Ecology
  - We may know a lot about a specific ecosystems and about specific species
  - But what about the introduction of a new species into an ecosystem?
  - The Internet

Examples (computational)

○ Cellular Automata Networks
  - Upon small perturbations on the lattice (e.g., re-wiring)
  - Global changes in the CA dynamics

○ Routers instability
  - Upon topology changes or relevant traffic changes
  - Some router may fail to sustain updates
  - With waterfall congestion effects at the global level

○ Computational markets
  - Upon the insertion of agents with differentiated strategies in markets
  - Emergence of war-prices, cyclic phenomena, inflationary processes
Engineering vs. Emergence

○ If we work at the micro scale only
  - With traditional “mainstream” direct engineering techniques
  - With a “design” approach
  - We miss the global perspective

○ If we work at the “macro” scale only
  - We can achieve global properties by emergence and reverse engineering
  - But may miss local goals
  - And may miss control

But why this is important?

○ It is not only the recognition of a basic need
  - Most real cases face such issues

○ More important, knowing what happens when we have a new system into an existing one
  - Can form the basis for new forms of decentralized control
  - And a new approach to engineer emergent systems

○ Have a complex self-org macro system and
  - Know what happens when acting on it
  - Knowing how to enforce a specific behavior on it

○ Shifting from
  - Local Rules → Hopefully Useful Global Behavior (pure macro-scale reverse engineering approach) TO
  - Local Rules + Decentralized Control → Engineered Purposeful Emergence
Decentralized Control via External Perturbation

- Try to
  - Disturb the system where and how you can
  - Attempting to identify perturbation patterns that
  - Affect as needed the emergent system behavior
- Example with Cellular Automata
  - Introduce a moderate degree of stochastic behavior in cells
  - And have global (otherwise non-emerging) patterns emerge

Decentralized Control via Components Injection

- Try to
  - Inject new components/subsystem
  - That interact with the systems so as to
  - Affect as needed the emergent system behavior
- Example with Cellular Automata
  - Inject a few percentage of cells with modified rules
  - And have peculiar needed patterns (very unlikely attractors) emerge
Engineering Emergence at the Meso Scale

- What is needed to advance knowledge in decentralized meso-scale control?
  - So as to produce a set of practical tools
  - Enabling the engineering and control of complex self-org systems
  - In a methodical and repeatable way?
- Conceptual advances in modeling
  - Discrete vs. Continuum Computing
  - Logics vs. Physics
  - Genotypes vs. Phenotypes
  - Design vs. Intention
  - Topology vs. Dynamics
- Or maybe the adoption of more usable abstractions other than those of local rules and local interactions?

The Role of the Environment

- Let us assume the system is (or can be abstracted as) immersed in an environment
  - The physical environment (or a pervasive network infrastructure)
  - A manageable representation of a network environment (e.g., a structured overlay)
- And that
  - Interactions occur via the environment \textit{(stigmergy)}
  - The environment reifies in the form of specific properties of it (artifacts or distributed states) the actual state of the system
- Then
  - Observing the environment means observing the system
  - Controlling the environment (i.e., controlling its properties) implies controlling the systems
  - Conceptual shift from controlling the system to controlling the environment
From Engineering Systems to Engineering Environments

- Once an environment abstraction is properly enforced
  - We may know how the system structure/dynamics reflect in the environment
  - We may know how to inject properties in the environment or how to perturb the properties of the environment

- Then we can study how
  - Given a complex systems of interacting components
  - A specific global behavior of the systems can be affected/influenced by the environment
  - A specific behavior can be enforced by acting on the environment

Spatial Environments

- The environment abstraction must be simple and usable
  - Enable an easy understanding of its structure
  - Enable an easy modeling of its properties and dynamics

- Environments as **metric spaces**
  - To apply concepts of coordinates and distances
  - To apply standard dynamical systems modeling

- Examples
  - Mobile Ad-Hoc Networks and Geographical Routing
  - Self-Assembly and Modular Robots
  - Pervasive Computing and Logical Spaces
  - P2P Structured Overlays (e.g., CAN, Chord)

- Open Question: can other types of systems tolerate a suitable mapping in metric spaces? (e.g., complex social networks)
Cognitive Stigmergy

- Most phenomena and systems relying on spatial stigmergic interactions
  - E.g., ant colonies and hormones in self-assembly
  - Assumes that components/agents simply react to properties in the environment
- However
  - It is possible to make the properties of the environment more “semantic”
  - E.g., not simply pheronomes but more complex artifacts and data structures
- And have components/agents “reason” about what they perceive
  - So that decentralized forms of control can be enforced directly into the system
  - “Cognitive self-management” achieved through controlled self-organization

The TOTA Approach at UNIMORE

- Attempting to exploit Stigmergic + Spatial + Cognitive Self-Organization In pervasive computing scenarios
  - A simple mechanisms for field-based stigmergy
  - Supported by a simple and highly usable API
- Spatial self-organization
  - The components of the systems (robots, mobile users)
  - Live and execute in space (or in a network lattice)
- Stigmergic self-organization
  - All interactions occurs via computational fields that diffuse
  - And that are locally sensed by agents
- Cognitive self-organization
  - Agents can perceive properties associated with fields
  - They are not necessarily merely “reactive”
The Model: Co-Fields

- Agents (or the environment itself) spread fields across the environment
  - Field-specific (app. specific) propagation rule
  - Local sensing of fields
- Global behavior and self-organization
  - Perception by agents of “coordination fields” as combinations of individual fields
  - Action driven by local shape of coordination fields
  - Preserve the possibility of field-awareness
- Example: flocking

The Infrastructure: TOTA

- “Tuples On The Air” relies on a distributed tuple-based coordination model
  - Distributed tuple structures propagated across a network environment (or parasitically in an RFID-enriched env)
  - Locally read by agents
  - Providing context-awareness and field-based structures
- Each tuple characterized by $T = (C,P,M)$
  - $C$ = content, associated with it, possibly changing during propagating
  - $P$ = propagation rule, how the tuple propagates and how it changes $C$ while propagating
  - $M$ = maintenance rule, how the tuple structure re-shape upon network changes
- Application agents can
  - Inject any application-specific tuple structure (field)
  - Read locally read available tuples
  - React, it they think so, and be affected by these tuples
A Simple Example
injection of a simple tuple
\( C = (\text{int } v = 0) \)

propagation of the tuple
\( P = (\text{spread everywhere, inc } v \text{ at each hop}) \)
A Simple Example
other agents can locally sense the tuple
i.e., the local value deriving from propagation
A Simple Example
what if the network topology changes?
e.g., due to mobility of source node

A Simple Example
a maintenance rule can specify how to act
e.g., $M = \text{(react to changes, restructure tuple)}$
Engineering Emergence in TOTA

- **Micro-scale**
  - Exploit fields as a sort of distributed shared memory for contextual interactions
  - Tolerating network and environmental dynamics

- **Macro-scale**
  - Exploit fields to re-produce known phenomena of self-organization
  - Or to invent new (we can invent our own laws for field propagation)

- **Meso-scale**
  - When a specific (micro) field-based application is immersed in a macro-scale scenario
  - Proper combination of fields can accommodate both
    - The needs of the micro-scale
    - The needs of the macro-scale
  - In any case, new fields can be injected for the goal of enforcing the needed controls

---

**Engineering Emergence in TOTA: An Example (1)**

- **Micro-scale**
  - Users visiting a museum
    - Where are you? → PRESENCE fields
  - Meeting by tourists
    - Tourists following each other’s PRESENCE fields
  - Flocking by museum guards
    - As in the flocking example
  - All of these realized by application-specific and independent fields
Engineering Emergence in TOTA: An Example (2)

- **Macro-scale**
  - Diffusive Load Balancing of Crowd
    - Global diffusion of Presence fields
    - Weighted with data expressing room capacity
  - Users behavior
    - Can follow suggestions
    - Can analyze what’s happening (cognition!)
  - Overall good load balancing even if a limited percentage of users follows the fields suggestions

---

Engineering Emergence in TOTA: An Example (3)

- **Meso-scale**
  - Flocking + Load Balancing
  - How can we conciliate?
  - Approach:
    - Have flocking agent perceive (and react upon) the following field:
      \[ \text{Coord} _\text{Field},(x,y,t) = \text{Flock} _\text{Field},(x,y,t) + \mu \cdot \text{LB} _\text{Field}(x,y,t) \]
    - And tune \( \mu \) (shape perceived fields) as needed
  - \( \mu = 0 \)
    - Ignore load balancing, and do pure flocking
    - Small decrease in load balancing quality
  - \( \mu > 0 \)
    - Flock accounting for crowd
    - Decrease in the accuracy of the flock formation
  - In any case, each single agent can “see” the individual fields and take actions accordingly
The CASCADAS Project

- Integrated Project to be funded by EU under FET “Situated and Autonomic Communication” Initiative
  - “Component-ware for Autonomic and Situation-Aware Communication Services”
  - 13 Partners
  - Starting January 2006
- General objective
  - Identify models and tools for a new generation of adaptable, self-organizing, context-aware communication services
  - Based on the central abstraction of ACE “autonomic communication element”, as the basic building block for complex service networks
  - Exploiting biologically and socially-inspired self-organization and self-management phenomena

The CASCADAS Approach

- Goes in the already stated directions
  - Level of services
  - Level of control
  - Exploiting sorts of shared knowledge networks for stigmergic cognitive interaction
- Several interesting activities at UNIMORE
  - Thesis, post-laurea fellowships
Conclusion and Open Issues

- Engineering self-organization and emergence is definitely a challenge
  - Producing self-organizing systems in a repeatable and measurable way (via reverse engineering methodologies)
  - Controlling the continuous evolution and increase of complexity of existing systems (via decentralized control)

- Possible promising approaches include
  - Focusing at the “meso” scale
  - Promoting stigmergy and environment engineering (as in Co-Fields and TOTA, and as in the “knowledge networks” of CASCADAS)
  - Controlling the environment to control emergence