Self-aware Pervasive Service Ecosystems

From Self-Organising Mechanisms to Design Patterns

Giovanna Di Marzo Serugendo
University of Geneva

Giovanna.Dimarzo@unige.ch
Outline

- Motivation: Spatial Structures and Services
- Self-Organising Mechanisms and Design Patterns
- Computational Model

- Case Study: Crowd Steering
Spatial Structures and Services
Motivation

The goal of this work is to propose an execution model for engineering self-organising application in pervasive system. The execution model should:

- Allow the (re-)use of code, easing the design and implementation of bio-inspired applications.
- Provide a set of bio-inspired mechanisms as low-level services
  - Used as a basis to build applications or high level services.
- Low-level services are requested on demand by applications or high level services
- Low-level services are embedded in the middleware using existing technologies.
Motivation

- Engineered Systems
  - Swarms of robots, unmanned vehicles
  - Information dissemination
  - P2P systems / Overlay networks
  - Immune computer
  - Trust-based access control
  - Grid computing
Motivation

- Engineering of complex systems
  - Environment
  - Software Agents / Services (autonomous entities)
  - Self-* Mechanisms
  - Middleware infrastructures

- Self-* mechanisms (bio-inspired)
  - Expressed under the form of design patterns
  - Identify limits of each pattern / mechanism
Motivation

- Spatial structures for pervasive systems
  - Self-organising / bio-inspired mechanisms
    - Decentralisation, robust, resilient, adapt to environmental changes
    - Complex behaviour with relatively simple rules

- Design Patterns’ Catalogue
  - Mechanisms recurrent and frequently involved in more complex mechanisms
  - Identification of:
    - Inter-relations between mechanisms
    - Boundaries of each mechanism
  - Classification into three levels

- Building Blocks for Spatial Structures
  - Basis for Self-Aware Services and Self-Aware Applications
  - Basis for composition of Self-Aware Services

[FMADMSC11, FMADMS+11, FMDMSM+12]
Computational Model

(a) Biological Model

(b) Computational Model
Computational Model

- **Software agents**
  - Active autonomous entities (services, information, etc.)
- **Host**
  - Computing device hosting agent execution
    - E.g. sensors, PDAs, computers, …
- **Environmental agents**
  - Autonomous entity working on behalf of infrastructure
    - E.g. evaporating pheromone, updating gradients
- **Environment**
  - Anything else on which agents have no control
    - E.g. user switching device off, temperature, humidity, etc.
Outline

- Motivation: Spatial Structures and Services
- Self-Organising Mechanisms and Design Patterns
- Computational Model
Self-Organising Design Patterns

- **Description**
  - **Abstract Transition Rule**
    \[ \text{name} ::= \langle L_1, C_1 \rangle, \ldots, \langle L_n, C_n \rangle \xrightarrow{T} \langle L'_1, C'_1 \rangle, \ldots, \langle L'_m, C'_m \rangle \]
  - **Sequence Diagram, Implementation details, Explanations**

<table>
<thead>
<tr>
<th>Name</th>
<th>The pattern’s name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliases</td>
<td>Alternative names used for the same pattern.</td>
</tr>
<tr>
<td>Problem</td>
<td>Which problem is solved by this pattern and situations where the pattern may be applied.</td>
</tr>
<tr>
<td>Solution</td>
<td>The way the pattern can solve the problems.</td>
</tr>
<tr>
<td>Inspiration</td>
<td>Biological process inspiring the pattern.</td>
</tr>
<tr>
<td>Forces</td>
<td>Prerequisites for using the pattern and aspects of the problem that lead the implementation, including parameters (trade-offs).</td>
</tr>
<tr>
<td>Entities</td>
<td>Entities that participate in the pattern and their responsibilities. Entities are agents, infrastructural agents, and hosts.</td>
</tr>
<tr>
<td>Dynamics</td>
<td>How the entities of the pattern collaborate to achieve the goal. A Typical scenario describing the run-time behaviour of the pattern.</td>
</tr>
<tr>
<td>Environment</td>
<td>Infrastructural requirements of the pattern.</td>
</tr>
<tr>
<td>Implem./Simulation</td>
<td>Hints of how the pattern could be implemented, including parameters to be tuned.</td>
</tr>
<tr>
<td>Known Uses</td>
<td>Examples of applications where the pattern has been applied successfully.</td>
</tr>
<tr>
<td>Consequences</td>
<td>Effect on the overall system design.</td>
</tr>
<tr>
<td>Related Patterns</td>
<td>Reference to other patterns that solve similar problems, can be beneficially combined or present conflicts with this pattern.</td>
</tr>
</tbody>
</table>
Self-Organising Design Patterns

- High-Level Patterns:
  - Flocking
  - Ant Foraging
  - Quorum Sensing
  - Chemotaxis
  - Morphogenesis

- Composed Patterns:
  - Digital Pheromone
  - Gradients
  - Gossip

- Basic Patterns:
  - Repulsion
  - Evaporation
  - Aggregation
  - Spreading
Spreading

http://ergodd.zoo.ox.ac.uk/eurasia/Eurasian%20Street%20Web/Studies/animations/Sim_30_608_393_smoothed.gif
Spreading

- Problem:
  - in systems, where agents perform only local interactions, agents’ reasoning suffers from the lack of knowledge about the global system.

- Solution:
  - a copy of the information (received or held by an agent) is sent to neighbours and propagated over the network from one node to another. Information spreads progressively over the system and reduces the lack of knowledge of the agents while keeping the constraint of the local interaction.

- Entities – Dynamics - Environment

\[
\text{spreading} :: \langle L, C \rangle \xrightarrow{t_{spr}} \langle L_1, C_1 \rangle, \ldots, \langle L_n, C_n \rangle \\
\text{where } (L_1; \ldots; L_n) = \nu(L), (C_1; \ldots; C_n) = \sigma(C, L)
\]
Spreading

- Implementation

Start

input event?

No

Yes

Stop

Is the same than the value broadcast before?

yes

no

Broadcast the inf. received
Aggregation
Aggregation

- **Problem:**
  - Excess of information produces overloads. Information must be distributively processed in order to reduce the amount of information and to obtain meaningful information.

- **Solution**
  - Aggregation consists in locally applying a fusion operator to synthesise macro information (filtering, merging, aggregating, or transforming)

- **Entities – Dynamics - Environment**

\[
\text{aggregation} :: \langle L, C_1 \rangle, \ldots, \langle L, C_n \rangle \xrightarrow{r_{aggr}} \langle L, C'_1 \rangle, \ldots, \langle L, C'_m \rangle \\
\text{where} \quad \{C'_1, \ldots, C'_m\} = \alpha(\{C_1, \ldots, C_n\})
\]
Aggregation

- Implementation

Start

input event?

No

yes

Apply aggregation operator

Stop

Host

Infrastructural Agent or Agent

data_req()

send_data()

Apply Aggregation

send_data_aggr()

Store Aggregated Data
Evaporation
Evaporation

Problem:
- Outdated information cannot be detected and it needs to be removed, or its detection involves a cost that needs to be avoided. Agent decisions rely on the freshness of the information presented in the system, enabling correct responses to dynamic environments.

Solution
- Evaporation is a mechanism that periodically reduces the relevance of information. Thus, recent information becomes more relevant than older information.

Entities – Dynamics – Environment

\[ \text{evaporation} :: (L, C) \xrightarrow{r_{\text{ev}}} (L, C') \]

\[ \text{where} \quad C' = \epsilon(C) \]
Evaporation

- Implementation

```
Start

Rev? 
No  Yes

Rel(Inf) > 0 ?
No  Yes

Stop  Apply Evap.
```

Apply Evap. Process

get_rel(Inf)
set_rel(Inf)
Repulsion
Repulsion

- **Problem:**
  - Agents’ movements have to be coordinated in a decentralised manner in order to achieve a uniform distribution and to avoid collisions among them.

- **Solution**
  - The Repulsion Pattern creates a repulsion vector that guides agents to move from regions with high concentrations of agents to regions with lower concentrations. Thus, after a few iterations, agents reach a more uniform distribution in the environment.

- **Entities – Dynamics – Environment**

\[
\text{repulsion} :: \langle L, C \rangle, \langle L_1, C_1 \rangle, \ldots, \langle L_n, C_n \rangle \xrightarrow{rev} \langle L', C \rangle, \langle L_1, C_1 \rangle, \ldots, \langle L_n, C_n \rangle
\]

where \( L' = \rho(\{\langle L, C \rangle, \langle L_1, C_1 \rangle, \ldots, \langle L_n, C_n \rangle\}) \)
### Implementation

- **Start**
- **SendPositionRequestToNeighbours()**
- **Received Positions?**
  - No
  - yes
  - **Calculate Repulsion()**
  - **CalculateDesired Position()**
  - **MoveToHostClosest ToNewPosition()**

**Signal Flow Diagram**

- **Agent**
  - **SendPositionRequest()**
  - **CalculateRepulsion Vector()**
  - **CalculateDesired Position()**
  - **MoveToHostClosest ToNewPosition()**

- **Host**
  - **SendPosition**
  - **SendPositionRequest**

**Neighbouring Hosts**

- **Host Agent Neighbouring Hosts**

Self-Organising Design Patterns

High-Level Patterns
- Flocking
- Ant Foraging
- Quorum Sensing
- Chemotaxis
- Morphogenensis

Composed Patterns
- Digital Pheromone
- Gradients
- Gossip

Basic Patterns
- Repulsion
- Evaporation
- Aggregation
- Spreading
Gossip
Gossip

- **Problem:**
  - in large-scale systems, agents need to reach an agreement, shared among all agents, with only local perception and in a decentralised way

- **Solution**
  - information spreads to neighbours, where it is aggregated with local information. Aggregates are spread further and their value progressively reaches the agreement

- **Entities – Dynamics - Environment**

\[
\begin{align*}
\text{spreading} & : \langle L, C \rangle \xrightarrow{r_{\text{spr}}} \langle L_1, [\text{Recd}, C] \rangle, \ldots, \langle L_n, [\text{Recd}, C] \rangle \\
\text{where} & \quad \{L_1, \ldots, L_n\} = \nu(L)
\end{align*}
\]

\[
\begin{align*}
\text{aggregation} & : \langle L, C_1 \rangle, \ldots, \langle L, C_m \rangle, \langle L, [\text{Recd}, C_{m+1}] \rangle, \ldots, \langle L, [\text{Recd}, C_n] \rangle \\
\xrightarrow{r_{\text{aggr}}} & \langle L, C'_1 \rangle, \ldots, \langle L, C'_k \rangle \\
\text{where} & \quad \{C'_1, \ldots, C'_k\} = \alpha(\{C_1, \ldots, C_n\})
\end{align*}
\]
Gossip

- Implementation:

- Start
- Input events?
  - Yes
    - Same value broadcasted before?
      - Yes
        - Apply aggregation operator
      - No
        - Broadcast aggregated Information
  - No
- Stop

Flowchart:
- Agent or Infras. Agent
- Host
  - Send(inf)
  - Apply aggr. operator
  - Send(aggr)
- Neighbour Hosts
  - Send(inf)
  - Send(aggr)
Gradient
Gradient

- Problem:
  - agents belonging to large systems suffer from lack of global knowledge to estimate the consequences of their actions or the actions performed by other agents beyond their communication range

- Solution
  - information spreads from the location it is initially deposited and aggregates when it meets other information. During spreading, additional information about the sender’s distance and direction is provided: either through a distance value (incremented or decremented); or by modifying the information to represent its concentration (lower concentration when information is further away).

- Entities – Dynamics - Environment

\[
\text{spreading} :: \langle L, [D, C] \rangle \xrightarrow{r_{spr}} \langle L_k, [D \pm \Delta D, C] \rangle \\
\text{where} \quad L_k = \text{random}(\{L_1, \ldots, L_n\})
\]

\[
\text{aggregation} :: \langle L, [D_1, C] \rangle, \ldots, \langle L, [D_n, C] \rangle \xrightarrow{r_{aggr}} \langle L, [D', C] \rangle \\
\text{where} \quad D' = \min/\max(\{D_1, \ldots, D_n\})
\]
Gradient

### Implementation:

1. **Start**
   - **Input events?**
     - No
     - **Distance is lower than local one?**
       - No
       - **Stop**
         - yes
         - **Aggregated inf.**
           (The one with lower distance stays stored)
         - **Broadcast inf. incrementing counter**
       - yes
       - **send(inf, d)**
         - **check inf and Aggregate.**
         - **send(inf, d+1)**
         - **send(inf, d+1)**
2. **Host**
   - send(inf,0)
3. **Neighbour Hosts**
   - send(inf,0)
4. **Agent or Infras. Agent**
   - send(inf,d)
   - **check inf and Aggregate.**
   - **send(inf, d+1)**
5. **Host**
   - send(inf,d)
Digital Pheromone
Digital Pheromone

- **Problem**: coordination of agents in large scale environments using indirect communication

- **Solution**: digital pheromone provides a way to coordinate agent’s behaviour using indirect communication in high dynamic environments. Digital pheromones create gradients that spread over the environment, carrying information about their distance and direction. Thus, agents can perceive pheromones from the distance and increase the knowledge about the system. Moreover, as time goes by digital pheromones evaporate, providing adaptation to environmental changes.
Digital Pheromone

- Entities – Dynamics – Environment

\[
\text{spreading} :: \langle L, [PhV, C] \rangle \xrightarrow{t_{spr}} \langle L_k, [PhV - \Delta PhV, C] \rangle \\
\text{where} \quad L_k = \text{random}(\{L_1, \ldots, L_n\})
\]

\[
\text{aggregation} :: \langle L, [PhV_1, C] \rangle, \ldots, \langle L, [PhV_n, C] \rangle \xrightarrow{t_{aggr}} \langle L, [PhV_i, C] \rangle \\
\text{where} \quad PhV_i = \max(\{PhV_1, \ldots, PhV_n\})
\]

\[
\text{evaporation} :: \langle L, [PhV, C] \rangle \xrightarrow{t_{ev}} \langle L, [PhV', C] \rangle \\
\text{where} \quad PhV' = PhV \times Ev_{factor}
\]
Self-Organising Design Patterns

- High-Level Patterns
  - Flocking
  - Ant Foraging
  - Quorum Sensing
  - Chemotaxis
  - Morphogenesis

- Composed Patterns
  - Digital Pheromone
  - Gradients
  - Gossip

- Basic Patterns
  - Repulsion
  - Evaporation
  - Aggregation
  - Spreading
Ant Foraging
Ant Foraging

- **Problem:**
  - large scale optimisation problems that can be transformed into the problem of finding the shortest path on a weighted graph.

- **Solution**
  - the Ant Foraging Pattern provides rules to explore the environment in a decentralised manner and to exploit resources

- **Entities – Dynamics - Environment**

\[
\begin{align*}
\text{up\_move} :: & \langle L, [PhV_1, C] \rangle, \ldots, \langle L_n, [PhV_n, C] \rangle \xrightarrow{\text{r\_move}} \langle L_i, [PhV_i, C] \rangle \\
& \text{where } PhV_i = \max(\{PhV_1, \ldots, PhV_n\})
\end{align*}
\]

\[
\begin{align*}
\text{random\_move} :: & \langle L, C \rangle \xrightarrow{\text{r\_move}} \langle L_i, C \rangle \\
& \text{where } L_i = \text{random}(\{L_1, \ldots, L_n\})
\end{align*}
\]
Ant Foraging

- Implementation
Morphogenesis
Morphogenesis

- **Problem:**
  - in large-scale decentralised systems, agents decide on their roles or plan their activities based on their spatial position

- **Solution**
  - specific agents spread morphogenetic gradients. Agents assess their positions in the system by computing their relative distance to the morphogenetic gradients sources

- **Entities – Dynamics - Environment**

\[
\text{state\_evolution} :: \langle L, [D, \text{State}, C] \rangle \xrightarrow{r_{\text{move}}} \langle L, [D, \text{State}', C] \rangle \\
\text{where} \quad \text{State}' = \pi(D)
\]
Morphogenesis

- Implementation:

  - Start
  - Gradients Received? (Yes)
    - Estimate relative position based on received gradients
    - Change role according to the relative position
  - Gradients Received? (No)

Diagrams:

- Host
  - GradInf()
  - Estimate relative position
  - Change Agent's role

- Neighbour Hosts
  - GradInf()
Chemotaxis
Chemotaxis

- **Problem:**
  - decentralised motion coordination aiming at boundaries of events

- **Solution**
  - agents locally sense gradient information and follow the gradient in a specified direction (i.e. follow higher gradient values, lower gradient values, or equipotential lines of gradients).

- **Entities – Dynamics - Environment**

\[
\text{move} :: \langle L, [D_1, C]\rangle, \ldots, \langle L_n, [D_n, C]\rangle \xrightarrow{r_{\text{move}}} \langle L_i, [D_i, C]\rangle \\
\text{where } D_i = \text{min/max/equal}(\{D_1, \ldots, D_n\})
\]
Chemotaxis

- Implementation

Start

- Send concentration request to n neighbouring hosts
  - yes
  - Inf. Received?
  - yes
  - Move to host with highest gradient

No

Agent

- GradRequest()
- Send_grad()

Host

- GradRequest()
- Send_grad()
- Choose host with highest concentration
- MoveToHost(h)

Neighbour Hosts

Choose host with highest concentration
MoveToHost(h)
Quorum Sensing

- **Problem:**
  - collective decisions in large-scale decentralised systems, requiring a threshold number of agents or estimation of the density of agents in a system, using only local interactions.

- **Solution**
  - the Quorum Sensing Pattern allows to take collective decisions through an estimation by individual agents of the agents’ density (assessing the number of other agents they interact with) and by determination of a threshold number of agents necessary to take the decision

- **Entities – Dynamics – Environment**
  - Gradient, Morphogenesis

\[
\pi(D) = \begin{cases} 
\text{State} & \text{if } D \leq \text{threshold} \\
\text{State'} & \text{if } D > \text{threshold}
\end{cases}
\]
Quorum Sensing

- Implementation

Start

- Gradients Received?
  - No
  - yes
    - Gradient's concentration higher than threshold?
      - no
      - yes
        - Trigger collaborative task

Agent

Host

if Grad > thre
trigger task

Neighbour Hosts

Gradients Received?

yes

if Grad > thres
trigger task
Flocking
Flocking

- Problem:
  - Dynamic motion coordination and pattern formation of swarms

- Solution
  - The Flocking Pattern provides a set of rules for moving groups of agents over the environment while keeping the formation and interconnections between them

- Entities – Dynamics – Environment
  - Cohesion (group)
  - Separation
  - Alignment (velocity, direction)
  - Group objective (target)
Outline

- Motivation: Spatial Structures and Services
- Self-Organising Mechanisms and Design Patterns
- Computational Model
BIO-CORE

Core Services

- Gossip
- Quorum Sensing
- Chemotaxis
- Morphogenesis
- Foraging

Gradients

CORE

- Aggregation
- Spreading
- Evaporation
BIO-CORE is an execution model based on: shared space technology, and rules.

BIO-CORE implements a set of:

- **low level core services**, providing basic self-organising mechanisms (e.g. Aggregation, Spreading, Evaporation, Gradients)
- requested on demand by applications or high level services.
BIO-CORE

Structure

BIO-CORE is composed of:

Core Data space: Shared space located in nodes, where data reside.

External Interface: Provides a set of primitives to allow applications and high level services to inject, retrieve or remove data.

Engine: The engine processes the data located in the Core Data Space and executes rules according to the data properties.
Execution Model and Core Services

- **BIO-CORE**
  - Execution model for Core mechanisms (ready to use) as Core Services
  - **Decoupling**
    - Functionality of Self-Aware Applications from Environmental Tasks
      - Spatial structure maintenance (e.g. Gradient spreading, Evaporation, …)
    - Mechanisms for Spatial Structure made available to applications as Core Services
  - **Reuse of code**
    - Same Core Service used by many Self-Aware Applications or Services
    - Encapsulation of specific mechanisms (one at a time)

[FMDMSM11]
### Core Services

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Unique identifier</td>
</tr>
<tr>
<td>EVAP</td>
<td>Activate the Evaporation Service. Automatically this property sets up the Relevance attribute (REL)</td>
</tr>
<tr>
<td>AGGREGATE</td>
<td>Activate the Aggregation Service</td>
</tr>
<tr>
<td>SPREAD</td>
<td>Activate the Spreading Service</td>
</tr>
<tr>
<td>GRADIENT</td>
<td>Activate the Gradient Service. Automatically this property also enables the SPREAD and AGGREGATE properties and sets up the Distance attribute (DIST)</td>
</tr>
<tr>
<td>REPULSE</td>
<td>Activate the Repulsion Service</td>
</tr>
<tr>
<td>DATA</td>
<td>Actual information stored in the LSA</td>
</tr>
</tbody>
</table>
Execution Model and Core Services

- **Spreading Service**
  \[
  \langle \text{ID, SPREAD, DATA} \rangle \xrightarrow{r_{spr}} \langle \text{ID, DATA} \rangle, \text{bcast}\langle \text{ID, SPREAD, DATA} \rangle
  \]

- **Evaporation Service**
  \[
  \langle \text{ID, EVAP, DATA} \rangle \xrightarrow{r_{ev}} \langle \text{ID, EVAP, 1, DATA} \rangle
  \]
  \[
  \langle \text{ID, EVAP, REL, DATA} \rangle \xrightarrow{r_{ev}} \langle \text{ID, EVAP, REL} * f_{evap}, \text{DATA} \rangle
  \]
Composition of Core Services
- Design time composition, late binding
Regional Leader Election

- The goal of this simulation is:
  - To implement the Regional Leader Election example.
  - To exploit the notion of BIO-CORE and Core Services.

- In this simulation we will see:
  - Progressive building of disjoint groups
  - A movie, showing the implementation results.

- This simulation shows that:
  - How Core Services enacting basic mechanisms are implemented using universal eco-laws.
  - Core services can be requested on demand by applications or high-level services, by injecting LSAs, thus easing the design and reusing code.
  - How applications and high-level services can be performed by composing lower level services.
Regional Leader Election

- **Goal:** To split the network into disjoint groups, each led by one node.

**LSA**
- **LSA**\(_1\): \(<\text{ID}, \text{EVAP}, \text{REL} = \text{random}(0,100), \text{DATA}>\)
- **LSA**\(_2\): \(<\text{ID}, \text{SPREAD}, \text{DATA} = \text{leaderID}>\)

**EC**
- \(\text{EC}_1\): \(<\text{ID}, \text{EVAP}, \text{REL}, \text{DATA}> \rightarrow <\text{ID}, \text{EVAP}, \text{REL} \times f_{\text{evap}}, \text{DATA}>\)
- \(\text{EC}_2\): \(<\text{ID}, \text{SPREAD}, \text{DATA}> \rightarrow <\text{ID}, \text{DATA}> \text{bcast}( <\text{ID},\text{SPREAD},\text{DATA}>\)

**Regional Leader Election**

1. **createNewLSA (LSA**\(_1\)**)**
2. **injectLSA(lsa)**
3. **readLSA()**
   - If noRole, LSA(Spread leaderID)
   - If noRole, value=0
4. **injectLSA(lsa)**
5. **createNewLSA (LSA**\(_2\)**)**
Thank you

Questions?


