Deliverable 5.4.3
Form and function in evolving information systems

Due date of deliverable:	December 2007
Actual submission date:	December 2007
Dissemination level:	PU – public
Work Package 5.4:	WP 5.4
Participants:	UPF UniBO Telenor

Authors of deliverable: Stefano Arteconi (arteconi@cs.unibo.it)
Ozalp Babaoglu (babaoglu@cs.unibo.it)
David Hales (hales@cs.unibo.it)
Andrea Marcozzi (marcozzi@cs.unibo.it)
Ricard V. Solé (ricard.sole@upf.edu)
Sergi Valverde (svalverde@imim.es)

Abstract

This report comprises the complete D5.4.2 deliverable as specified for workpackage WP5.4 in Subproject SP5 of the DELIS (Dynamically Evolving Large-scale Information Systems) Integrated Project.

The essential goal of the DELIS project is to understand, predict, engineer and control large evolving information systems. In this workpackage we consider a broad, though often overlooked, range of network evolution phenomena that evidence multi-scale and multi-level phenomena. We believe that understanding such phenomena is key to relating form to function in complex evolving networks that display emergent properties and rich self-organised structure.

In the first section we report on our work towards the production of a group selection design pattern applicable to a number of domains within distributed information systems. We have collected together and summarised in a standard format and framework how the group selection pattern has been applied in a number of domains. We still believe however that the approach requires further work to be proposed as a coherent pattern because of a number of open issues which we discuss.

In the second section we report on a recent paper which demonstrates in simple steps how to translate biological and social evolution-type simulation models into a set of concurrent threads for distributed systems protocols based around a "tournament" selection approach. We believe this kind explanatory work can be of use to systems designers who wish to translate abstract biological and social models into experimental distributed systems protocols.

In the third section we report some work which has lead to a negative result. We investigated the viability of producing a fully distributed real-time peer-to-peer (P2P) protocol for discovering the emergent structure of it's own network by identifying the network motifs that comprise it. Some of our previous work on motif patterns in P2P protocols had indicated the potential value of such a protocol. However, we found serious limitations given only local knowledge and we discuss these to indicate why we considered this approach not viable.

Finally we report on recent work exploring multi-level processes and structures in software development and in the context of the spontaneous emergence of modularity without selective forces.[1]

[1]Most papers produced within DELIS are available from the DELIS website as DELIS Technical Reports. Where this is the case references are appended with the DELIS Tech Report number in square brackets. This indicates the paper was produced within the DELIS project, not some other project.

Contents

1 Towards a group selection design pattern

Recent models from computational social science and theoretical biology have demonstrated novel processes of group selection [17, 18, 21]. They demonstrate how selfish behavior at the individual level can, never-the-less, lead to highly co-operative and co-ordinated emergent behaviour at the collective level.

Here we discuss the formulation of a group selection "design pattern" which balances the often contradictory forces between individual goals and collective goals in distributed systems without the need for any centralised structures or control.

Previous work has shown how biologically inspired approaches can be presented in the form of design patterns [3] and, indeed, some have suggested that the "tag" approach (see later) could be formulated as a pattern [6].

The group selection approach is of great potential value for the engineering of massive, decentralized and open distributed computer systems because it embodies several attractive features: It is general, applicable to wide range of scenarios; it is scalable, allowing many millions of entities to coordinate efficiently; it is robust, meaning that entities may enter and leave the system without disrupting interaction drastically; it is self-organising requiring no central control or administration; it is resistant to certain kinds of free-riding, when entities try to exploit the system for their own benefit. Importantly, because entities only need to act on their own local performance criteria, there is no need for global information or the representation or enforcement of group or system level goals.

Group selection relies on the dynamic formation and dissolution of groups. Over time individual entities may change groups by moving to those that offer better individual performance. Interaction between entities that determine performance is mainly restricted to those sharing the same group. Essentially then, in a nutshell, groups that support high performance for the individuals that comprise them grow and prosper whereas exploitative or dysfunctional groups dissolve as individuals move away. Hence functional groups, in terms of satisfying individual goals, are selected over time.

Key aspects that define applications of group selection are: How group boundaries are formed; the nature of the interactions between entities within each group; the way that each entity calculates their individual performance (or utility) and how entities migrate between groups.

The success of any application of group selection is judged by how well the system self-organises towards achieving a collective goal - whatever that may be. Often this will be maximising the sum of individual performances but could involve other measures such as equality or fairness for example.

First we describe an abstract model which identifies, at a high level, the key aspects and emergent process that the application domain models realise. Then we present two domain models in that format. Further domain models and a detailed description of the bio- and socio-models of group selection, that inspired the pattern, can be found in [8]. This sections is a brief extract from that work.

The work presented here is a summary of [8] in which we present several models within the framework. Here we give just two examples from that work. Our aim is to move towards patterns similar to the bio-inspired approaches specified in [3].

1.1 Abstract computational model

In order to describe different realizations of the group selection mechanism in information systems applications we abstract assumptions and key aspects in the context of computational entities (e.g. agents, nodes in a network, individual processes etc.). Each instantiation of the mechanism specifies a way of implementing each aspect such that the emergent process occurs.

1.1.1 Assumptions

The group selection approach can be applied to systems where the following general assumptions hold:

- A system is composed of individual entities that can benefit from interaction with other entities
- The population of entities is partitioned into groups such that interaction is mainly limited to entities within the same group
- Entities measure their own performance periodically producing a utility value
- Entities may spontaneously change their behavior and group membership
- Entities may view and copy some state of other entities
- Entities desire to increase their performance (utility)

1.1.2 Key aspects

We have found, as will be seen in the later application examples, that any instantiation of the group selection process involves identifying the following aspects of a system - we use these key aspects as a template by which we related each application of group selection:

- Collective Goal - A desirable goal that the population of entities should attain.
- Group Boundary Mechanism (Group) - How an entity can locate and communicate with in-group members.
- Intra-Group Interaction (Interaction) - What kinds of utility effecting interactions an entity participates in with other in-group members.
- Utility Calculation Metric (Utility) - How an entity calculates a utility value based on its individual goal and in-group interactions.
- Group Migration Mechanism (Migration) - How migration between groups is performed.

1.1.3 Emergent Process

By correctly defining the key aspects above an emergent group selection occurs, achieving the collective goal, following the below general form:

- Entities are grouped in some initially arbitrary way
- Interactions between entities within groups determine entity utilities
- Based on utility comparisons between entities, and possibly randomized change, group memberships and interaction behavior (strategy) change over time
- Groups which produce high utility for their members tend to grow and persist as entities join
- Groups which produce low utility for their members tend to disperse as entities leave
- Hence group beneficial behavior tends to be selected

1.2 Selfish broadcasting

A broadcast operation involves one node in a network sending a message to the entire network population. However it is not practical in many situations for every single node to connect directly to all other nodes. A simple way of implementing a broadcast function is for each node to pass all new broadcast messages it receives to all of the other nodes it connects to (its neighbors). This way a message initiated from any node will eventually reach all other connected nodes in the network. This method is called the flood fill (FF) approach and although simple and robust it can be highly costly in terms of the total number of messages required to be passed: FF will send L messages (where L is the total number of links in the network) assuming the network is connected. This can be highly sub-optimal.

Collective Goal	Maximise the number of nodes receiving broadcast messages initiated from randomly selected nodes while minimising the number of messages sent.
Entity	Peer node - a node in a peer-to-peer overlay network. Each node stores a neighbor list (or view) and a forwarding strategy (either PASS or DROP). Periodically nodes may randomly change their strategy or copy it from other nodes, but they do not change their neighbor list.
Group	All nodes reachable via a broadcast from a given node represents the the group membership of that node.
Interaction	Interaction involves initiating broadcast messages. Receiving messages from neighbors. Passing (or not) received messages to neighbours based on the strategy (either PASS or DROP) held by the node.
Utility	Benefit (B) received upon message reception minus cost (C) due to possible message forwarding. $B > C > 0$. Hence nodes have an incentive to receive messages but no incentive to pass them on.
Migration	No direct migration (no rewiring in the network). Groups are defined by nodes receiving or not receiving messages and depends on strategy distribution among nodes. By changing a strategies nodes migrate between broadcast groups.

Figure 1: Key aspects for the Broadcast model.

The optimal broadcasting possibility in terms of messages sent, is for the message to follow a spanning tree in which each node only passes the message to those neighbors that have not already received the message. This approach requires only N messages (where N is the total number of nodes in the network).

We consider a different approach from the two limit cases above, in which nodes dynamically decide if to pass messages to their neighbors based on a simple adaptive evolutionary protocol in which there are incentives for nodes to not pass on messages.

Using simulations we found that this approach, though motivated by self-interest in the nodes, produced reasonable broadcasting performance with a total message cost significantly less than that required for the FF approach. Interestingly, we found that our simple protocol appears to evolve the network towards a critical threshold of PASS nodes and this *could* be related to site percolation theory.

The results presented here are an overview of work given in a previous paper from which more details can be obtained [1].

1.2.1 Interaction

We formulated the model in the form of a *broadcast game* in which nodes were awarded a payoff (or utility) based on the outcome of each broadcast event. A broadcast event involves randomly choosing a node from which a broadcast message is initiated. The node passes a message to each of its neighbours. In the next time-step each neighbour then decides if to pass the message to its own neighbors based on its current strategy (either PASS or DROP). This process continues until no more messages can be sent. We assume that nodes wish to receive broadcast messages (they contain useful information) but that nodes have some incentive not to pass messages on (due to a bandwidth or power cost). Hence we assume there is a benefit for receiving a message (B) and some cost for sending a message (C). We assume that $B > C > 0$ meaning a node receiving and passing a message receives positive benefit. After a broadcast event each node in the network will either receive a utility award of B, if it received but did not pass the message or $B - C$ if it received and passed the message or zero if it did not receive the message at all.

1.2.2 Migration

Migration between groups is not explicitly defined (i.e. there is no network rewiring) since only strategies are copied. The network structure itself does not change through evolution. According to the strategy distribution the network is composed of groups in the sense of connected clusters of nodes holding the PASS strategy. That is, every node i is a member of a group defined by the set of all reachable nodes S from node i via a broadcast. A group boundary is thus formed by a region of the network being bounded by nodes holding the DROP strategy - stopping any message originating inside the group from passing to other groups. Through utility comparison and strategy copying a group selection process is achieved at the level of quantity of PASS nodes in a group. Groups of nodes not receiving messages have zero utility. Hence it will copy any other nodes strategy if it received a message. This means the proportion of PASS nodes in groups receiving messages will tend to be reproduced by strategy copying. Such proportions of PASS nodes will tend toward the critical value at which groups form because by definition a group exists if it is above this critical threshold. On the other hand a node within a group receiving a message has an incentive to switch from a PASS to a DROP strategy - lowering the proportion PASS nodes in such groups. This ultimately will cause the creation of new groups not receiving messages when the PASS value falls below the critical value. Hence we see cyclical oscillations around the critical value.

1.2.3 Simulation algorithm

We implemented the broadcast model in the PeerSim platform [24]. Figure 2 shows the pseudo-code for the one active and two passive threads that form the protocol. For the simulations discussed here we selected one node randomly to initiate a broadcast message every 20 cycles. Messages were given a TTL value of 5. A cycle was equivalent to the time it took to send a message one hop in the network. Nodes maintained input buffers to store incoming messages. The results given here are for static random networks but similar results were obtained when high levels of node churn (nodes entering and leaving the network) were incorporated. For full details see [1].

1.2.4 Results

We tested the broadcast game on a random network topology of degree 20 where all links were symmetric. The system shows an interesting behaviour. A typical run of the algorithm is shown in Figure 3. In this experiment a single message is generated every 20 simulation cycles and spread starting from a randomly chosen node, while in each cycle each node compares its utility with a random node with 20% probability.

Active "replication" thread for node (i):

```
Periodically do
    j ← selectRandomNode()
    IF Utility_j > Utility_i THEN
        Strategy_i ← Strategy^j
        With probability m change Strategy_i
        Utilityi ← 0
    END IF
```

Passive "initiate broadcast" thread for node (i):

```
When requested to broadcast message (msg) do
    LOOP for each neighbour node (j) in view
        Send(j, msg)
    END LOOP
```

Passive "receive message" thread for node (i):

```
When receive message (msg) from neighbour (k) do
    IF new unseen message THEN
        Increment Utility_i by B (benefit)
        IF (Strategyi == PASS) and (TTL of msg > 0) THEN
            Decrement TTL of msg
            Decrement Utility_i by C (cost)
            LOOP for each neighbour node (j) in view except (k)
                Send(j, msg)
            END LOOP
        END IF
    END IF
```

Figure 2: Pseudo-code for the Broadcast protocol comprising one active and two passive threads. The topology was initialised to a random network with degree 20. All links between nodes were symmetric.

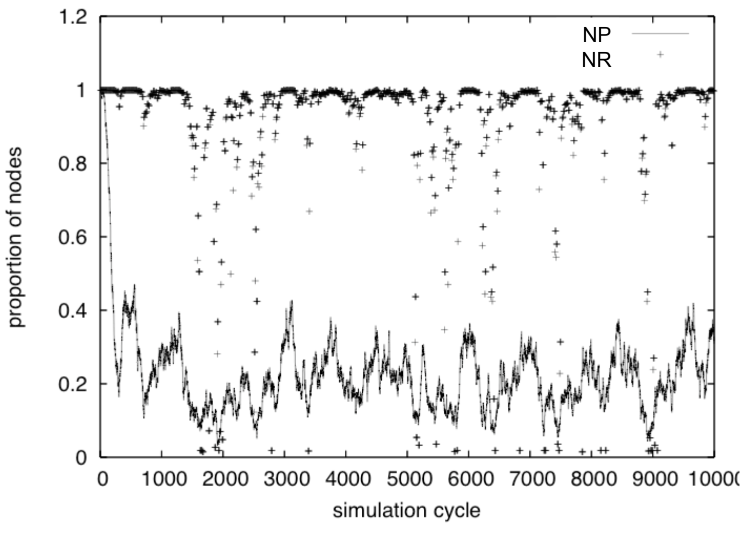

Figure 3: A single run of broadcast game. The solid line (NP) represents the ratio of nodes with PASS strategy, while each dot (NR) represents a message and the quantity of nodes that received it. Initially NP drops dramatically but it meta-stabilises at a critical threshold below which messages are not spread and above which messages become redundant.

We initialised all nodes with the PASS strategy (NP=1). Hence at the start of the run flood fill is performed and there are a lot of redundant messages. If some node stops passing messages it gets a higher utility and gets copied, so initially a dramatic drop in NP occurs as nodes discover how to get higher utility by not passing the message. Eventually too many nodes will adopt the DROP strategy and the message will fail to be spread throughout the network. At this stage the network is divided into a groups receiving messages and groups not receiving anything. Nodes in non receiving groups will tend to reproduce the NP value of the receiving cluster and will eventually be able to receive messages again. Hence a group-like selection is created between clusters receiving messages and clusters not receiving anything, and the strategies are distributed oscillating around a critical value above which there are redundant messages and below which messages can not be spread throughout the network.

These dynamics are very complex and subtle, we only hinted at them here. For a fuller discussion see [1].

1.3 Content replication

Nodes in a network often store and serve content to other nodes. However, each has a finite capacity and if requests for content exceed the capacity then queries fail. It is generally not possible for *a priori* predictions of load demand because at given times some content may suddenly become popular and at others hardly requested at all. Hence over a given time period a population of nodes has a certain total capacity to serve requests (the sum of all individual node capacities) and some demand load (queries going to the nodes). Assuming nodes can replicate content and redirect queries we present a simple node level protocol that self-organises nodes into cooperative clusters leading to efficient outcomes in some simple scenarios. CacheWorld is a simulation model that applies to group selection pattern to this replication sceanrio. The key aspects are shown in figure 4. This section is an overview previous work where further details can be found [9].

Collective Goal	Maximise the total number of queries served by harnessing unused capacity in underloaded nodes.
Entity	Peer node - a node in a peer-to-peer overlay network with the ability to receive and serve queries, for a content item, from clients external to the overlay network. Each node has a maximum capaciy limiting number of queries serviceable over a time period. Each node can be thought of as a web server, for example, and stores is own content item and a replicated copy of each of its neighbours content items.
Group	The neighbour list (or view) of a node defines its group
Interaction	Receiving redirected queries from overloaded nodes or conversely redirecting queries to a random neighbour when overloaded. When a node makes a connection to a new neighbour both nodes mutually replicated their contents.
Utility	A simple binary satisfaction function: if all queries received by a node are eventually served then the node is satisfied otherwise it is unsatisfied.
Migration	Periodically, unsatisfied nodes move randomly in the network. But a node will only accept an incoming connection from a moving node if it is in a receptive state. A node is only receptive if it has spare capacity or is itself unsatisfied.

Figure 4: Key aspects for the CacheWorld model.

1.3.1 Interaction

We assume a population of N server nodes which form a P2P overlay network. In addition to being part of the overlay, each node functions as a server responding to requests (queries) from clients outside of the overlay. An example could be that each node is a web server with the overlay linking the servers and clients being web browsers on user machines. Servers store a copy of their own content item (e.g. a website) and have additional storage for k replicated items from other servers. The overlay links servers bidirectionally if they mutually replicate content. In our website example this would mean two linked servers hold a copy of the others' site. We also assume servers have access to three services: a replication service that copies items between servers; a peer sampling service that supplies a random server from the overlay; a content server that serves or redirects queries as required.

Over a given time period nodes receive queries (load) from clients to serve their content item. Each node has a fixed capacity, C, specifying the total queries it can serve in the given time period. If the load exceeds capacity then the nodes is said to be "overloaded". Overloaded nodes redirect queries to randomly selected neighbors. If a neighbor is not itself overloaded it will serve the query from its local content replica (or cache), otherwise it will ignore the query. The essential idea therefore is that overloaded nodes will have neighbors that are not overloaded and can serve queries.

1.3.2 Migration

Each node maintains an estimate of the proportion of queries for its own content that were actually served (ps). A node is said to be satisfied when $ps \geq t$, where t is some threshold value. Here we set $t = 1$ for all nodes, meaning nodes are only satisfied if 100% of their queries are served. Periodically nodes attempt to change their neighbors (move) in the overlay network if they are not satisfied. When a node i moves, it drops all its current links and selects a random node, j, from the population. i then attempts to link to j. The link is accepted if j is in a *receptive* state. A node is defined as receptive if it is currently not satisfied or if it has spare capacity which is unused. This captures the notion that a node only wants new connections if it is either not satisfied or if it has spare capacity to offer. If a receptive node accepts an incoming link but already has the maximum k links then it drops a randomly selected old link to make space for the new link.

1.3.3 Simulation algorithm

We implemented the CacheWorld model in the PeerSim platform [24]. Figure 5 shows the pseudo-code for the active and passive threads that form the protocol. The passive thread is activated when a node receives a query, the active thread is activated periodically. For the simulations discussed here we selected a simple loading scenario on the nodes in which half the nodes were overloaded and half were underloaded such that an optimal network structure exists in which all queries could be answered. Details of this and other load scenarios can be found in [9].

1.3.4 Results

Figure 6 shows results from three different experiments (isolated, random and dynamic) for the simple half overloaded / half underloaded scenario. Here we show the simplest case where number of replicas $k = 1$, meaning that nodes essentially form pairs rather than complex network structures. The *Isolated* experiments gives a performance baseline by running the simulation with all CacheWorld services turned-off: no links between nodes, no query redirection and no movement. This captures the situation where nodes simply answer their own queries and do not know about other nodes. The *Random* experiments give a secondary baseline. Here the overlay is initialized to a random topology (degree k) that is fixed - i.e. no node movement is possible. However, nodes my redirect their queries

```
Active "node movement" thread for node:

Periodically do
    IF not satisfied
        drop all neighbour links
        j ← selectRandomNode()
        IF j is receptive then link to j
    END IF

Passive "receive query" thread for node:

When receiving a query (q) do
    IF not overloaded, service q directly
    ELSE IF neighbours > 0 and q is not already a redirected query
        j ← selectRandomNeighbour()
        redirect q to j
    END IF
```

Figure 5: Pseudo-code for the CacheWorld protocol. It comprises one active and one passive thread. These were simulated within the PeerSim environment. Notice that only one hop of query redirection is allowed.

to neighbors if they become overloaded. In the *Dynamic* experiments the CacheWorld protocol is fully enabled allowing for dynamic node movement based on satisfaction, as previously described. By comparing *Random* and *Dynamic* experiments we can determine how much extra performance the dynamic movement produces over just a fixed random overlay. By comparing *Isolated* and *Dynamic* experiments we can determine the overall increase in performance obtained by using CacheWorld over letting servers deal with all their own queries individually.

1.4 Summary

In this section we have given an overview of two existing application models that have been inspired by group selection approaches - for further details and models see [8]. We identified some key aspects that define such models and expressed them in a standard template. However, there are still open issues before we can present our approach as a design pattern in a principled way. Firstly, all the applications so far discussed have only been tested at the level of simulation in which many potentially significant effects are not modelled. For example, we have not tested the systems against several kinds of malicious attacks. Also there is an assumption in many of the models that nodes will, at some level, supply valid information (for example for utility comparison or link exchanges) and malicious nodes may not do this - we have addressed some of these issues in previous studies [2, 16] but there is more work to do here. We believe that to produce a credible design pattern we need to report on experience with deployed and tested implementations. We feel that the work presented here moves us closer to that possibility but, again, there is work to do.

The group selection approach may be just that: a design *approach* rather than a design *pattern.* Our experience has been that, given a co-ordination problem in a distributed system, the group selection approach is more like a general methodology than a solution. Rather like functional decomposition within structured programming, or inheritance hierarchies in object orientated design, the approach requires one to identify what might be the natural structures of the interactions within the system - i.e. you have to answer the questions: Where are the groups here? What are the strategies or relevant individual behaviour choices? How can migration occur? What is the collective goal? What are the individual goals? Producing answers to those questions is not easy and often requires experimentation. Yet if crisp definitions can be found for each of these aspects then the group

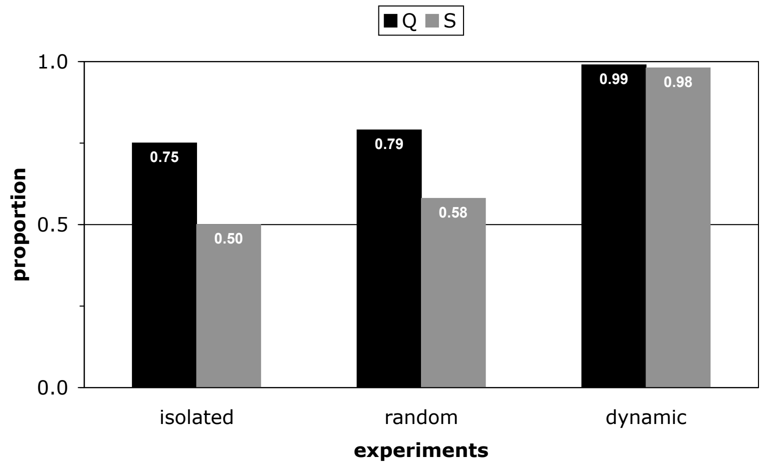

Figure 6: Results from the Content Replication model. Isolated shows performance when the protocol is tuned off. Random shows the performance when there is no node movement. Dynamic shows results when the protocol is fully enabled. Q = proportion of queries answered, S = proportion of nodes satisfied.

selection process is relatively straightforward to apply.

The key point is that by answering these questions one is forced to think about the runtime distributed *social dynamics* of a system at a level of abstraction much higher than specific implementation issues, static structures or purely individual properties. By doing this one is forced to *think socially* and is freed from the need to design centralised structures to impose the collective goal.

If group selection in distributed information systems can be developed as a general design methodology then perhaps this could have a wide general impact in the emerging are of socially intelligent systems design.

2 From evolutionary simulations to distributed systems protocols

Translating evolutionary algorithms into parallel distributed P2P protocols is a relatively simple process. Generally this involves a parallel and asynchronous copying of application behaviors between pairs of processing entities based on a utility measure. In order to illustrate this translation process we give a set of pseudo-code template algorithms, starting with the kind of evolutionary algorithms given in biological and social simulation work and ending with an outline of a set of threads that could be the basis of a protocol design for a distributed P2P system. The work in this section is brief summary of [12] which was indeed for a an audience unfamiliar with co-evolutionary models and how to translate them.

2.1 A generic co-evolutionary algorithm

Figure 7 shows an outline of a typical co-evolutionary algorithm. We do not show here the particular way that entities interact to gain fitness (utility) or the specific reproduction method used. The reproduction phase may be implemented in many ways. A common approach in biological models is to use so-called “Roulette Wheel” selection [7]. This is a probabilistic approach requiring access to all the finesses of the the entire population. A simpler approach, from the point of distributed implementation, is to use a Tournament Selection approach.

```
Initialize some population P of N entities
loop for some number of generations
   entities in P interact in some way and obtain a fitness (utility)
   reproduce a new population P2 by replicating entities from
      P in proportional to fitness
   apply mutation to each entity in P2 with some low probability
   P = P2
end loop
```

Figure 7: A generalized synchronous evolutionary algorithm. Similar to the kind of algorithms used in biological and social simulation work.

```
Initialize some population P of N entities
loop for some number of cycles
   select some entities in P to interact in some way and obtain utility
   loop for some number of reproductions
     select a random pair (i,j) of entities from P
     if Utility(i) > utility(j) then
       copy behavior of i to j
       apply mutation to j with low probability
       utility(j) = 0
     end if
   end loop
end loop
```

Figure 8: A generalized asynchronous evolutionary algorithm using Tournament Selection method during reproduction.

```
Active application thread for node i:
do forever:
  Engage in application level interaction with other nodes using Si
  Update utility value Ui

Active reproduction thread for node i:
do forever:
  wait(delta)
  j = selectRandomNode()
  receive(Uj, Sj) from node j
  if Uj > Ui then
    Si = Sj
    with low probability Mutate(Si)
    Ui = 0
  end if

Passive reproduction thread for node i:
do forever:
  send (Ui, Si) to requesting node j
```

Figure 9: A generalized Tournament Selection approach represented as three concurrent threads assumed to be running in each node over a population of nodes forming a peer-to-peer system. Here Si representing an application behavior (or strategy) of node i and Ui represents the utility.

2.2 Translation to tournament selection

Figure 8 shows the same general outline algorithm but with the reproduction phase expanded with a simple tournament selection approach. Some number of reproductions are performed in which random pairs of entities are selected, utilities are compared and the entity with the lower utility copies the behavior of the node with the higher utility - meaning the behavior of the higher utility node is effectively *reproduced.* After reproduction and with low probability some "mutation" is applied to the behavior, meaning that some randomized change is made in behavior.

2.3 From tournament selection to threads and protocols

The benefit of working with evolutionary algorithms is that, although algorithms are often presented as sequential and synchronous which aids simulation and analysis on a single machine, by their nature they should be easily translatable into distributed implementations because evolutionary processes are fundamentally distributed. Sequential evolutionary algorithms are a simulation abstraction of a parallel process. Figure 9 gives a simple example of a set of threads that would need to be executed by each node in a peer-to-peer system such that it would implement the same tournament based selection.

2.4 Summary

In this section we gave a brief and practically illustrate of how to take an evolutionary algorithm and translate it into a P2P network protocol in general. Of course for specific application domains specific details dictate this process. In the previous section 1 we showed applications of the group selection approach. There can be seen the final protocol threads produced from a translation processes starting with an evolutionary algorithm for group selection but applied to the specific contexts.

3 Viability of real-time distributed motif analysis

As shown previously [11] network motif analysis (indemnifying and counting all 4-node undirected motifs) may be used to detect the status of the evolution of a network topology and possible anomalies derived from attacking or cheating nodes. Figure 10 shows all possible 4-node subgraphs.

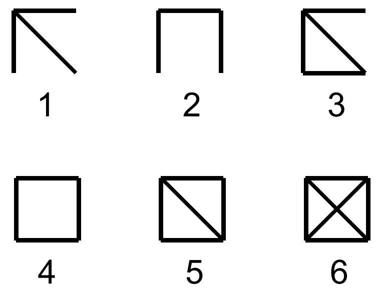

Figure 10: All possible undirected 4-node subgraphs. We aimed to assess the viability of searching and aggregating these in a distributed way using only local node information and processing.

A main requirement for such techniques to be applied on systems like P2P networks is decentralization: since P2P networks are strongly decentralized and nodes only have partial knowledge of the entire system, it is not possible to assume global knowledge of the system when performing decentralised and distributed motif analysis. Unfortunately existing techniques[25] need a full description of the network topology to work. Our aim, which we report on here, was to propose a method to perform motif analysis on a local scale using only nodes partial knowledge (views). However, we abandoned this line of work due to the problem of uniqueness that we encountered (see section *** below).

Despite this negative result we give a brief overview of the line of work taken and why we considered the task not viable.

3.1 Basic architecture

The main idea behind the decentralisation technique is to split global motif analysis into two main steps:

- Local motifs evaluation performed by single nodes independently.
- Global aggregation of local statistics.

The first step can be achieved through node view exchanges and analysis. A node receiving the views of its entire neighborhood has enough information to locally detect all the directed 4-nodes motifs. Once each node has accurate statistics regarding local motifs they can be easily aggregated and a global count can be achieved using gossip-based aggregation techniques[14]. A system architecture that could support these two basic steps is illustrated in figure 11.

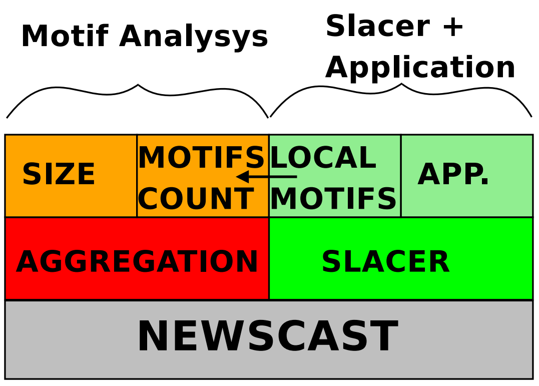

Figure 11: The system is composed by two different applications running in parallel: Slacer and aggregation framework. A common modlue is the random peer sampling service (i.e. Newscast[15]), on top of which both Slacer (right-hand side) and aggregation (left-hand side) call. On top of Slacer local motif evaluation could run along with some specific application. Local motifs statistics could then be passed to the aggregation framework for global aggregation.

3.2 Adjacency matrix

To evaluate 4-nodes undirected motifs each node needs to collect all of its neighbors views and then check for overlaps to look for simple local structure. In example if a node A receives the views of three neighbors B, C and D and realizes they are connected, a clique (motif6) is discovered. Since many checks of this kind has to be performed, an adjacency matrix structure is built by nodes when they receive their neighbors views.

When all the neighbors' views have been received the matrix can be used to easily check for the presence of local motifs. A simple example of a small network and a related adjacency matrix are given in figure 12 and table 1.

In the adjacency matrix of a generic node N each column (row) represents a neighbor, while each entry (i, j) represents a possible edge between two neighbors of N and is set to 1 if the edge is present or to 0 otherwise. Using this structure it is quite easy to find the 6 possible 4-nodes motifs shown in figure 10 as discussed below. In the following A, B, C, D represents upper-left, upper-right, lower-left and lower-right nodes respectively, and matrix of node A is examined.

- **Motif 1**: Column B has 2 entries set to 0 [(B, C) and (B, D)]
- **Motif 3**: Column B has 2 entries set to 0 [(B, C) and (B, D)] and the entry between the two nodes [(C, D)] is set to 1
- **Motif 5**: Column B has 2 entries set to 0 [(B, D) and (C, D)] and the entry between the two nodes [(B, C)] is set to 0
- **Motif 6**: Column B has 2 entries set to 1 [(B, C) and (B, D)] and the entry between the two nodes [(C, D)] is set to 1

For motif2 and motif4 the adjacency matrix does not contain enough information since independently from the node taken into account, at least one node at distance 2 need to be reached, hence other than the adjacency matrix built locally the full views previously received from the neighbors are needed as well. Using both the matrix and the whole neighbors' views motif2 and motif4 can be detected in the following way:

- **Motif 2**: One node between B and C has a neighbor D that is not a neighbor of A
- **Motif 6**: Nodes B and C both have D as a neighbor but D is not a neighbor of A

Example Looking at the network in figure 12, using the adjacency matrix in table 1 constructed by node B gathering the views of all its neighbors, B can easily evaluate the motifs he is part of. It is clear from figure 12 that among the others, B is part of an instance of motif3 (nodes B, C, E, F) and an instance of motif6 (nodes A, B, D, E). These motifs are easily detectable applying the rules above to the adjacency matrix illustrated in table 1.

To detect motif3 it is enough to notice that column C has two 0 entries in rows E and F, and entry (E,F) is set to 1. The same condition holds for many other instances of motif3 that are not clearly visible from the figure, like the one given by nodes (A, B, D, F), in fact looking at the matrix column F has rows A and D set to 0 and the entry (A,D) is set to 1.

Motif6 is detected by node B looking at column A having entries D and E set to 1 and the entry (D, E) set to 1 as well.

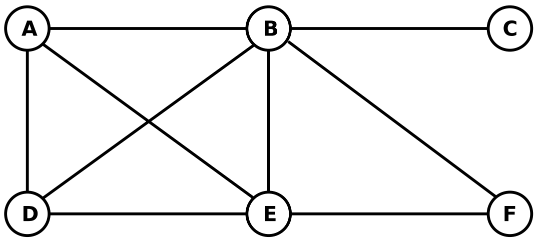

Figure 12: Simple example network to show local motif analysis through adjacency matrixes. Node B's matrix is illustrated in table1

	A	**C**	**D**	**E**	**F**
A	-	0	1	1	0
C	0	-	0	0	0
D	1	0	-	1	0
E	1	0	1	-	1
F	0	0	0	1	-

Table 1: Adjacency matrix of node B from figure 12. For simplicity the whole matrix is shown even though since network links are undirected.

This approach makes it relatively easy to count motifs locally, but before feeding such counts to the aggregation part an agreement between the nodes on how to count (and who should count) motifs has to be reached. These problems are discussed in the following sections.

3.2.1 Counting

Because of their symmetry some motifs are detected by more than one of the nodes composing them, hence counted more than once, some mechanism to avoid multiple counts of the same motif instances is needed. A very simple solution to this problem consists in implicitly agreeing on each motif instance to be counted by the highest ID node that detects it. Looking at figure 10 motif1 and motif3 are detected only by the upper-left node, so motif1 and motif 3 can be immediately count when detected. Motif2 and Motif5 are detected by 2 different nodes (respectively, upper-left upper right, and upper-left lower-right), since the nodes detecting the motif are aware of each other the one with highest ID will count for the motif and the other one will do nothing. The same rule applies to motif3 and motif6 which are detected by all the 4 nodes composing them.

3.2.2 Submotifs

In many cases some motifs include other motifs as a part of them (submotifs). In example motif3 includes an instance of motif1 and motif 4 includes 4 instances of motif2. Either that these sub-motifs have to be counted or not the way to go is defining an inclusion relation and an order relation from it. When detecting motifs locally, bigger ones have to be searched first, when a big motif is found it is possible both to stop searching for smaller motifs (if submotifs have to be ignored) or automatically count for submotifs with no need to detect them (i.e. if a node detects motif3 it can automatically count for a motif1 as well).

3.2.3 Uniqueness

Uniqueness of a motif is defined as the number of times a subgraph appears in the network with completely disjoint groups of nodes. In dense networks there are many overlapping motifs and evaluating uniqueness is a hard task. Supposing a node is part of many different motif instances and each of this instances is formed by node that are part of other motifs as well, which nodes/motifs should be chosen to have an accurate uniqueness value estimate? This is not a trivial problem, algorithm working on the whole network topology return approximated results as well (i.e. mfinder[25] manual states that the returned uniqueness value for a given motif is just a lower bound to the real value).

Using the distributed mechanism presented so far it is practically impossible to have an accurate estimation of uniqueness. The main reason for this lies in the fact that 2-hop distance knowledge of the topology may not be enough to decide if a given motif instance should be counted for uniqueness or not. In fact choosing a given instance motif might start a chain reaction leading to many different instances more than 2-hops away not to be counted. Unfortunately to increase the estimation of uniqueness value more information than 2-hop neighborhood is needed. Gathering more information (n-hops) would improve uniqueness accuracy, but on the other hand it would increase enormously both network traffic and local computation. Facing uniqueness evaluation in a distributed fashion turns out to be a hard task and raises doubts about the feasibility of a good distributed motif analysis. An example of bad uniqueness estimation is given in figure 13

3.3 Summary - not viable in current framework

We discussed briefly in this section why we considered a P2P distributed motif analysis to be not viable for the approach and architecture considered. The main problem being accurate uniqueness estimated. This is not to say that some other approach may not yield better results but we did not pursue this aspect further.

4 Multi-level structures in software and networks

4.1 Software development

We have investigated the origin of fluctuations in the aggregated behaviour of an open-source software community in [22].

In a recent series of papers [5, 4, 23], de Menezes and co-workers have shown how to separate internal dynamics from external fluctuations by capturing the simultaneous activity of many system's components. In spite of software development being a planned activity, the analysis of fluctuations

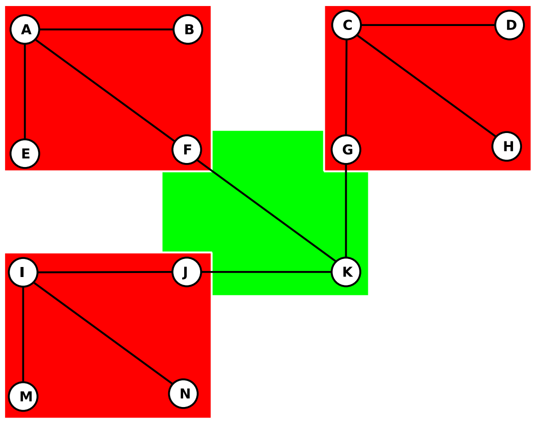

Figure 13: Example of uniqueness evaluation. When evaluating uniqueness value of motif1 if the instance composed by nodes (F, G, J, K) is chosen uniqueness would be 1. In this case the exact uniqueness value for motif1 is 3 and it is obtained evaluating the distinct instances composed by (A, B, E, F) (C, D, G, H) and (I, J, M, N)

reveals how external driving forces can be only observed at weekly and higher time scales. Hourly and higher change frequencies mostly relate to internal maintenance activities. There is a crossover from endogenous to exogenous activity depending on the average number of file changes. This new evidence suggests that software development is a non-homogeneous design activity where stronger efforts focus in a few project files. The crossover can be explained with a Langevin equation associated to the cascading process, where changes to any file trigger additional changes to its neighbours in the software network. In addition, analysis of fluctuations enables us to detect whether a software system can be decomposed into several subsystems with different development dynamics.

4.2 Modularity for free

In [19] we investigated the spontaneous emergence of modularity in networks without selection or design . Modularity is known to be one of the most relevant characteristics of both biological and technological systems and appears to be present at multiple scales. Given its adaptive potential, it is often assumed to be the target of selective pressures and design principles. Under such interpretation, selection would be actively favouring the formation of modular structures, which would specialize in different functions. Here we show that, within the context of cellular networks, no such a selection pressure is needed to obtain modularity. Instead, the intrinsic dynamics of network growth by duplication and diversification is able to generate it for free and explain the statistical features exhibited by small subgraphs.

5 Conclusion

We have presented an overview of a group selection based design pattern which makes use of novel models of multi-level topology evolution to co-ordinate co-operative interactions between computer nodes in a network. We have also briefly presented a method for translating evolutionary simulation models, often found in within the biological and sociological research literature, into concurrent threads that run on peer nodes. We discussed the non-viability of distributed motif analysis in a peer-to-peer system using only local knowledge. Finally we report on recent work exploring multi-level processes and structures in software development and in the context of the spontaneous emergence of modularity without selective forces.

References

[1] Arteconi, S., Hales, D. (2006) Broadcasting at the Critcial Threshold. October 2006, Technical Report UBLCS-2006-22, University of Bologna, Dept. of Computer Science. [DELIS-TR-0373]

[2] Arteconi, S.; Hales, D., Babaoglu, O. (2007) Greedy Cheating Liars and the Fools Who Believe Them. In Engineering Self-Organising Systems, Proceedings of the 4th International Workshop, ESOA 2006, Hakodate, Japan, May 9, 2006, Revised and Invited Papers. Brueckner, S.; Hassas, S.; Jelasity, M.; Yamins, D. (Eds.), Lecture Notes in AI , Vol. 4335, Springer. [DELIS-TR-0397]

[3] Babaoglu, O., Canright, G., Deutsch, A., Di Caro, G., Ducatelle, F., Gambardella, L., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., and Urnes, T. (2006) Design Patterns from Biology for Distributed Computing. In *ACM Transactions on Autonomous and Adaptive Systems, vol. 1, no. 1, 26–66.*

[4] de Menezes M. A. and Barabsi A.-L., (2004) Phys. Rev. Lett., 93, 68701.

[5] de Menezes M. A. and Barabsi A.-L., (2004) Phys. Rev. Lett., 92, 28701.

[6] De Wolf, T. and Holvoet, T (2007) Design Patterns for Decentralised Coordination in Self-Organising Emergent Systems. In Engineering Self-Organising Systems, Proceedings of the 4th International Workshop, ESOA 2006, Hakodate, Japan, May 9, 2006, Revised and Invited Papers. Brueckner, S.; Hassas, S.; Jelasity, M.; Yamins, D. (Eds.), Lecture Notes in AI , Vol. 4335, Springer.

[7] Goldberg, David E. Genetic Algorithms in Search, Optimization and Machine Learning Addison-Wesley Pub. Co. 1989.

[8] Hales, D., Arteconi, S., Marcozzi, A., Chao, I. (2007) Towards a Group Selection Design Pattern. Technical Report UBLCS-2007-25, University of Bologna, Dept. of Computer Science. [DELIS-TR-0568]

[9] Hales, D., Marcozzi, A., Cortese, G. (2007) Towards Cooperative, Self-Organised Replica Management. Proceedings of the First International Conference on Self-Adaptive and Self-Organizining Systems (SASO2007), July 2007, Boston, MIT. IEEE Press. [DELIS-TR-0532]

[10] Hales, D. and Arteconi, S. (2006) SLACER: A Self-Organising Protocol for Coordination in P2P Networks. *IEEE Intelligent Systems* 21(2):29-35. [DELIS-TR-0370]

[11] Hales, D. and Arteconi, S. (2007) Motifs in Evolving Cooperative Networks Look Like Protein Structure Networks. Proceedings of the European Conference on Complex Systems 2007, Dresden [DELIS-TR-0533]

[12] Hales, D. (2007) Applying Evolutionary Approaches for Cooperation. In F.H.P. Fitzek and M. Katz. (eds) Cognitive Wireless Networks: Concepts, Methodologies and Visions. Springer. [DELIS-TR-0530]

[13] Hales, D. and Edmonds, B. (2005) Applying a socially-inspired technique (tags) to improve cooperation in P2P Networks. *IEEE Transactions in Systems, Man and Cybernetics - Part A: Systems and Humans, 35(3):385-395.* [DELIS-TR-0111]

[14] Jelasity, M., Montresor, A. and Babaoglu, O. (2005) Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst., 23(1):219-252*

[15] Jelasity, M., Kowalczyk, W. and van Steen, M. (2003) Newscast Computing *Technical Report IR-CS-006*, Vrije Universiteit Amsterdam, Department of Computer Science, November 2003, Available at: http://www.cs.vu.nl/globe/techreps.html#IR-CS-006.03

[16] Jesi, G. P, Hales, D., van Steen, M. (2007) Identifying Malicious Peers Before Its Too Late: A Decentralized Secure Peer Sampling Service. Proceedings of the First International Conference on Self-Adaptive and Self-Organizining Systems (SASO2007), July 2007, Boston, MIT. IEEE Press. [DELIS-TR-531]

[17] Riolo, R., Cohen, and M.D., Axelrod, R. Cooperation without Reciprocity. *Nature* 414, 441-443, (2001).

[18] Santos F. C., Pacheco J. M., Lenaerts T. (2006) Cooperation prevails when individuals adjust their social ties. *PLoS Comput Biol 2(10): e140. DOI: 10.1371/journal.pcbi.0020140*

[19] Sol, R. V. and Valverde, S. Spontaneous emergence of modularity in cellular networks. Journal of the Royal Society Interface 5, 129–133.

[20] Sun, Q. and Garcia-Molina, H. (2004) SLIC: A Selfish Link-based Incentive Mechanism for Unstructured Peer-to-Peer Networks. In Proc. of 24th IEEE Int. Conf. on Distributed Systems, IEEE Computer Society.

[21] Traulsen, A.; Nowak, M. A. (2006). Evolution of cooperation by multilevel selection. *Proceedings of the National Academy of Sciences* 130(29):10952-10955.

[22] Valverde, S. (2007) Crossover from Endogenous to Exogenous Activity in Open-Source Software Development. Europhysics Letters 77, 20002.

[23] Yook S. H. and de Menezes M. A. (2005) Europhys. Lett., 72, 541.

[24] Peersim Peer-to-Peer Simulator, Available at: http://peersim.sf.net

[25] mfinder network motif detection tool software,
available at: http://www.weizmann.ac.il/mcb/UriAlon/groupNetworkMotifSW.html