Project No. FP6-027807

CASCADAS

Bringing Autonomic Services to Life

Instrument

Thematic Priority

Work Package 3 Deliverable Month 12

Aggregation Algorithms, Overlay Dynamics and Implications for Self-Organised Distributed Systems

Period covered: from 1/1/2006 to 31/12/2006

Date of preparation: XX/XX/2006

Start date of project: 1/1/2006

Duration: 36 months

Project coordinator name: Antonio Manzalini

Project coordinator organization name: TI

Revision:

Table of Contents

Document History

Version	Date	Comment
0.1	29/11/2006	First integrated version.
0.2	08/12/2006	Draft for discussion
1.0	31/12/2006	Document's first version.
1.1	13/01/2007	Revised draft following internal review

1 Introduction

Advances in IT and ICT have resulted in an exponential growth in computing systems and applications that impact all aspects of life. Nevertheless, scale and complexity of such systems and applications represent obstacles to further developments: configuration, healing, optimization, protection and in general management challenges are beginning to overwhelm the capabilities of existing tools and methodologies, and rapidly render the systems and applications almost unmanageable, not optimised and insecure.

IBM introduced the Autonomic Computing initiative in 2001[10], with the aim of developing self-managing systems. The word autonomic is inspired by the functioning of the human nervous system and is aimed at designing and building systems that have self-configuration, -healing, -optimization and -protection features. The human body's Autonomic Nervous System is the part of the nervous system that controls the vegetative functions of the body such as circulation of the blood, heart rate, body temperature, the production of chemical 'messengers', (i.e. hormones) etc., thus hiding from the conscious brain the burden of dealing with these and many other low-level, yet vital, functions. In a similar way, autonomic computing refers to the self-managing of computing resources in order to hide complexity from operators and users. Systems make decisions, using high-level policies from operators. It will constantly check and optimize its status and automatically adapt itself to changing conditions.

With the growth of the computer industry, notable examples being efficient networking hardware and powerful CPUs, autonomic computing has been proposed as a direction to cope with the rapidly growing complexity of integrating, managing, and operating computer systems. The realization of autonomic computing is likely to result in a significant improvement in system management efficiency.

Upon launching the Autonomic Computing initiative, IBM defined four general properties a system should have to constitute self-management: self-configuring, self-healing, self-optimising and self-protecting. Since the launch of Autonomic Computing, the self-* list of properties has grown substantially. It now also includes features such as self-anticipating, self-adapting, self-critical, self-defining, self-destructing, self-diagnosis, self-governing, self-organized, self-recovery, self-reflecting, etc.

1.1 Self-organising Systems

A key challenge of the autonomic computing initiative has been to draw upon self-* properties in systems other than computational ones in order to develop new computing systems. Biology has been a key source of inspiration. Group-living animals have provided inspiration for the field of collective, or swarm intelligence [6] which models problems through the interactions of a collection of agents cooperating to achieve a common goal.

For example, eusocial insects (ants, bees, wasps, termites, etc.) form colonies presenting a high degree of social organisation. In these systems, problems are "self-solved" in real time through the emergence of the appropriate collective behaviour, which arises from the sum of all interactions occurring between the agents and with their environment.

The mechanisms that lead to self-organization in biological systems differ from those occurring in physical systems in that they are influenced by biological evolution. Thus there is the possibility of using algorithms inspired by biological evolution. These are part of what has been described as biologically-inspired or "nature-inspired" computing (the latter term chosen because of the breadth of natural complexity that does not usually fall into the area of biology). A diverse range of fields have been analysed, including evolution and genetics, bacterial adaptive mechanisms, morphogenesis and self-organising principles more broadly [12].

Another aspect of the interest in self-organisation has been the development of interest in complex networks [5] as an application area for computer science. The reason for the extraordinary interest of the computer science community in complex networks is arguably two-fold: first, the re-discovery of "small world" effects by Duncan Watts and Steven Strogatz and second, the discovery of the "scale-free" properties of many man-made physical networks (e.g. the Internet) and overlays (e.g. the World-Wide-Web).

The first experimental study of the "small world" effect was conducted by Milgram in the late 1960s [15] and demonstrated that complete strangers are often linked by a relatively short chain of acquaintances. The newfound popularity of the concept is attributable to the efforts of Watts and Strogatz to formalise it using graph theory [24, 25], which led to associating "small world" topological properties (e.g. logarithmic relationship between size and diameter) with efficiency in communication networks [21].

After Faloutsos et al. published their now famous paper [9] presenting experimental evidence that the Internet exhibited "scale-free" properties, the community realised that the work on random graphs initiated by Erdos and Renyi in 1960 [8] was a powerful tool to understand the dynamics of the Internet. Since then there has been much research effort on understanding the nature of the self-organising complexity that arises from use of the Internet, e.g., [19].

Further interest in developing self-organising systems has considered the role that decentralized control of distributed systems can play in self-organisation. Peer-to-peer networks can be used as a basis for self-organisation among elements, in order to develop complex adaptive systems [2, 3] or self-organising multi-agent systems [4, 7]. These draw upon the emergence of novel properties at the whole system level when many elements are brought together automatically, without being programmed in by system developers, and inspiration is drawn from biological systems.

Babaoglu and colleagues have developed a number of systems in this area [2, 3], all of which draw upon a dynamic network of peers with some sort of adaptive agents interacting. Babaoglu et al. [3] review these models and a variety of others all looking at how to manage systems running on unpredictable network environments drawing upon biological inspiration.

The Agent-based approach has been, and remains, a rich area for the study of the emergence of self-organisation. For example, "artificial markets" have been studied for their potential in market-based control [27]. The aspiration is that if the appropriate interaction/trading rules are encoded into a population of agents, then the agents will be able to self-organise into "useful" structures/networks, where "useful" is defined in terms of an application context e.g. Supply chains, or trading markets.

Di Marzo et al. [7] review different aspects of self-organisation in Multi-Agent Systems. They show how inspiration derives from natural systems (complex physical systems as well as natural systems). For example, the concept of stigmergy, derived from the behaviour of social insects, has also been important in inspiring the design of Multi-Agent Systems. Bernon et al. [4] review several examples of applications of self-organising multi-agent systems. They show how Multi-Agent Systems can self-organise to carry out tasks, even though individual agents have very simple properties. The emergent properties of the self-organising system support each application.

Another important area of investigation of self-organisation has focused upon how individuals can cooperate to produce self-organising behaviour. Like other aspects of the analysis of self-organisation, this has been inspired by cooperation in nature, in systems ranging from microorganisms [23] to human beings [18].

Von Neumann and Morgenstern [18] provided the foundation for such analysis through their invention of game theory, enabling the investigation of how cooperative entities could overcome possible disadvantages of cooperation in the presence of cheating and other exploitation. Maynard Smith and Price [13] applied game theory to animal behaviour through their development of the evolutionary stable strategy, and this has led to the field of evolutionary game theory [20].

Cooperation has been studied using evolutionary games, following Trivers' concept of reciprocal altruism [22] and Axelrod and Hamiltons' evolutionary game models of it [1]. They used the so-called Prisoners' Dilemma model, modelling the interaction between two prisoners in identifying the strategies they should adopt in cooperation with each other and with their jailers.

In the one-stage Prisoners' Dilemma model, the prisoners only interact once. But a more realistic and more general scenario is that interactions are repeated over time, allowing individuals to react to previous past behaviour. The Iterated Prisoners' Dilemma (IPD) implements this, and diverse strategies varying over time and space have been studied, showing how cooperation can emerge and be maintained despite the potential for cheating and other malicious behaviour.

But the conditions supporting the emergence of cooperation as found in the IPD may not always be detectable in more complex real-world scenarios, accordingly there has been interest in other techniques for the analysis of individuals in order to detect their advance potential for cooperation and participation in self-organising activities.

One such area focuses on means of determining the reputation of individual agents as a means of assessing whether interactions with them are worthwhile, e.g., [14, 16]. Other mechanisms work on the establishment of trust between potential cooperators, through links such as a "web of trust". Both these areas of activity have great potential in facilitating

the emergence of self-organisation through cooperation, and accordingly are attracting considerable interest in autonomic and other areas of distributed computing.

Overall the concept of self-organisation has motivated a great deal of research that promises to have considerable impact on emerging autonomic computing and communications technologies.

2 Aggregation algorithms and overlay topology

The first phase in the work of WP3 is to address "aggregation" which means the process by which nodes form associations ("links") with other nodes. In abstract terms we call this "clustering" and we imagine that each node has a certain "type" and prefers to have links with nodes of the same type. Hence an effective autonomous aggregation algorithm would be one which, when independently followed by all the nodes in a system, leads to an increase in the fraction of links which connect nodes of the same type. The converse ("reverse-clustering") is simply the case where the nodes prefer to have links with nodes of any type *other* than their own type.

In this section we introduce the clustering algorithm which forms the basis of our simulation work. We describe how one algorithm (termed "passive" clustering) had some of the features we desired in such an algorithm (simplicity, effective aggregation) but was unsatisfactory because of its undesirable distortion of the network topology (creating arbitrary "hubs" (highly connected nodes)). An adaptation of the "passive" clustering algorithm retains the desirable features and avoids the undesirable. We have focused our first year's work on simulations using this "on demand" clustering algorithm.

2.1 "Passive" clustering

We devised a first set of basic local rules requiring only direct interaction between first neighbours yet susceptible to give rise over time to spontaneous system-wide aggregation of elements. Because it involves two nodes being notified by a third (the "match-maker") that they are now connected in the overlay, even though those same two nodes have no part in the decision process, we refer to the corresponding logic as "passive" clustering.

The rules are as follows:

- One node, the match-maker, is randomly selected. This is equivalent to say that every node in the system has a chance of "waking-up" and initiating a rewiring procedure, provided that this procedure is brief enough (and/or infrequent enough) that a situation in which two concurrent rewiring affect the same nodes is extremely unlikely, and so every attempt can be considered an independent event.
- The match-maker randomly selects two of its own neighbours and, if they happen to belong to the same type, instructs them to link together
- If the two chosen nodes were not already directly connected (through the overlay) a new link is established between them (i.e. they become first neighbour of each other).

- If conservation of the total number of links is in force (optional) and a new connection is successfully established, the match-maker terminates one of its own links with one of its two selected neighbours.

2.1.1 Basic rewiring dynamics

We originally envisaged that relevant dynamics would only be observed in systems comprised of more than one type of such elements, so that the decision to reorganise a region of the overlay (by creating, terminating or reallocating one or more links) could be influenced by some identifiable characteristics of the nodes (or elements) involved. However, benchmarking simulations made us realise that even in a network where all nodes are identical (i.e. of the same type or "colour"), applying the passive clustering algorithm could still have a dramatic effect on topology and so that even this apparently trivial case required in-depth investigation.

We discovered that repeatedly applying the passive clustering algorithm to an initially random graph comprised of a single type of vertices resulted in the production of a different network topology, characterised by a highly heterogeneous node degree distribution.

The most surprising aspect of this result was that it contrasted with the prediction of an apparently similar abstract model, which stated that the system should retain homogeneous node degree. The contradiction revealed a subtle difference between the rewiring dynamics of the numerical experiment and those described by the abstract model.

The model can be summarised as follows:

- A graph comprised of n vertices is represented by its (symmetrical) $n \times n$ adjacency matrix M in which $M(i,j) = 1$ denotes the existence of a link between i and j and $M(i,j) = 0$ the absence of such direct connection.
- A rewiring attempt takes the form of randomly choosing three vertices i (the match-maker), j and k so that if (and only if):

$$M(i,j)M(i,k)(1-M(j,k))=1 \qquad (1)$$

  a new link is established between j and k, $M(j,k) \leftarrow 1$, and $M(i,j) \leftarrow 0$ so as to conserve the total number of links.

This effectively means that if i happens to be in the position of a match-maker (i.e. is connected to both j and k) and j and k are not already connected together, a new link is established and an old one is severed, which is *precisely* the same sequence of events that would have occurred with the same 3 vertices i, j and k in the benchmark simulation.

One immediately obvious difference though is that the probability of selecting vertices i, j and k for which condition (1) is verified is substantially different in the two procedures, as in the simulated implementation of passive rewiring, in effect, only the match-maker i is selected at random ("waking-up" probability). By contrast, j and k being chosen randomly but *among* the first neighbours of i, $M(i,j)$ and $M(i,k)$ are always equal to one. If this was the only difference however, it should only slow down the dynamics observed in the abstract model and not give rise to any qualitative difference.

The critical, less intuitive factor, turns out to be that the probability of a successful match-making is directly proportional to the number of neighbours of the match-maker in the abstract model, because a higher degree implies a higher probability that $M(i,j) = M(i,k) = 1$. Since a side-effect of rewiring is that the successful match-maker loses a first neighbour (to satisfy conservation), this creates the conditions for a negative feedback ("rich becomes poorer") in the abstract model, preserving homogeneity of node degree.

This is *not* the case in the simulated implementation of the local rules. On the contrary, every node of degree higher than one has basically an identical probability of becoming a successful match-maker (if i is the match-maker, it only depends on the probability of $M(j,k)$ being equal to zero, which in turn simply depends on overall link density) so there is no such negative feedback.

The difference between the abstract model and the simulated local rules is illustrated using a simple example in figure 1. Because a connected graph of five vertices and four edges can only exist in three different states, it is relatively straightforward to calculate the probability of transition between them and infer the invariant distribution for each of the two logics.

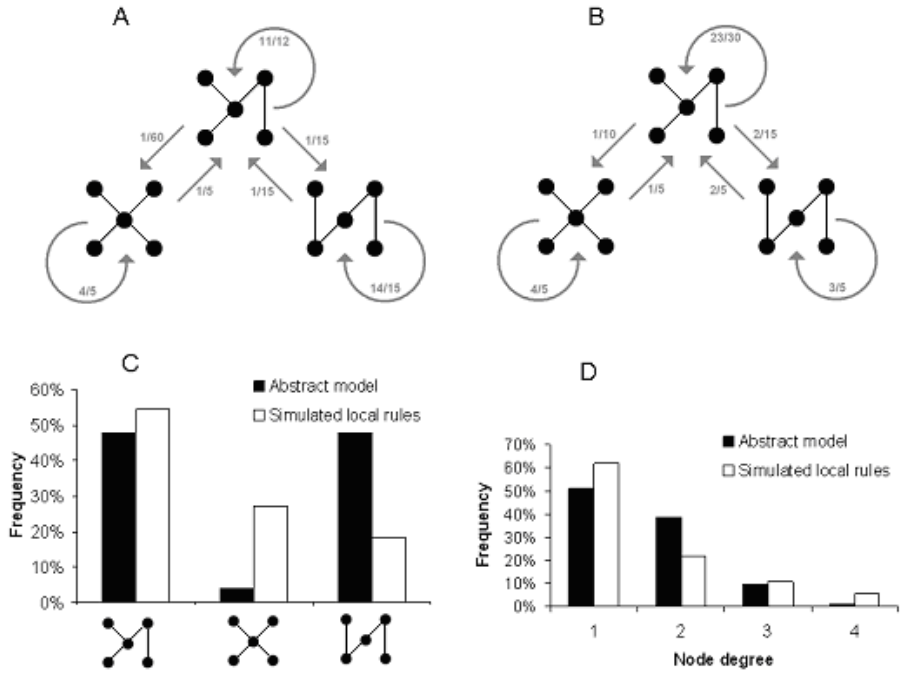

Fig. 1 Transition diagrams for the abstract model (A) and simulated local rules (B), frequency of the three possible system states in the invariant distribution (C) and corresponding signature in terms of the frequency of all possible node degrees (D).

These results can be generalised by using a Monte Carlo simulation version of the abstract model, so as to be able to compare the outputs for larger systems. As expected, the

difference between the abstract model and the simulated implementation of the local rules is maintained and becomes even more noticeable when the number of nodes and/or links increases. Figure 2 shows a comparison between the output of the two logics for a network starting as a random graph comprised of 100 vertices and 200 edges. Whilst applying the local rules gives rise to a typical power-law distribution (scale-free topology) it is obviously not the case for the abstract model (more homogeneous node degree).

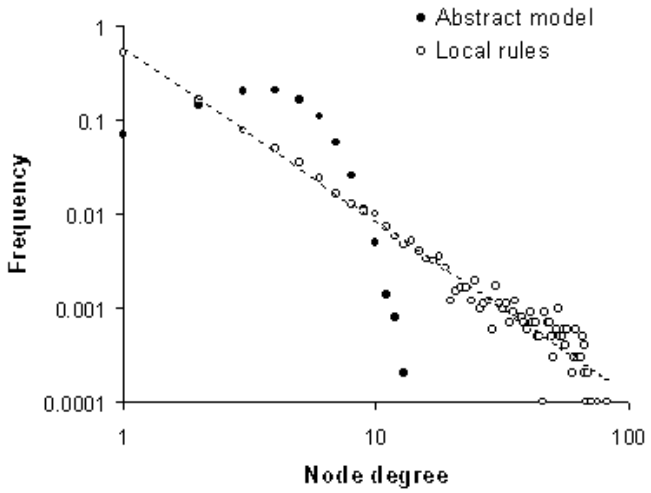

Fig. 2 Comparison of node degree distribution between the Monte Carlo simulation of the abstract model and the simulated implementation of the local rules. Initial conditions are a random graph of 100 vertices and 200 edges (100 independent realizations)

2.2 "On-demand" clustering

The results presented in section 2 clearly demonstrate that, in order to preserve homogeneous node degree in the realistic, local rules-based scenario, the rewiring procedure has to be modified so as to eliminate the indirect positive feedback leading to the emergence of scale-free topology in the passive clustering scenario. It may be objected that heterogeneous node degree can be highly beneficial to network operation if the higher connectivity of some vertices can be made to reflect their superior capability (see e.g. Jesi et al, 2006). However, in our case, such correlation is effectively absent, the emergence of hubs in the "passive rewiring scenario" resulting from the amplification of *random* fluctuations. As it cannot be guaranteed that those nodes ending up with a higher degree effectively have some specific features that designate them as efficient "super-peers", the result could be disastrous and generate critical bottlenecks, which is why we aimed at maintaining node degree as homogeneous as possible throughout the system's history.

We achieved this by distinguishing between the initiator of a rewiring procedure and the match-maker. Basically, upon "waking-up", the initiator requests a new link from one of its existing neighbours, which will then act as the match-maker. Since with this logic, the probability for a node to be appointed match-maker is obviously a direct function of its own degree (and the match-maker still ends losing one neighbour in the process of a successful rewiring operation), it introduces a negative, "rich becomes poorer" feedback similar to the one observed in the abstract model.

The detailed algorithm governing key node behaviour in the three roles of "initiator", "match-maker" and "candidate" involved in a rewiring operation following the "on-demand" clustering procedure is shown in figure 3. It involves exchanging five types of messages (plus the link termination message which isn't discussed here). The "neighbour request" (NRQ) message is sent by the initiator to the chosen match-maker and specifies the type of node desired. The "neighbour reply" (NRP) message is sent by the match-maker to the initiator to inform it of a potential candidate. The "link" (LNK) message is sent by the initiator to the candidate to ask for the establishment of a new link, which will only be effectively created if it is compatible with the goals of the candidate, as evidenced by the receipt of an "acknowledgement" (ACK) message by the initiator. Note that, for most of the results presented in this section, this will always be the case, as all nodes in the system share the same objective, i.e. they are all assumed to be simultaneously in clustering (or reverse-clustering) mode. Finally, after a successful handshake between the initiator and the candidate, the match-maker is informed via the "success" (SCC) message, so as to be able to determine whether its own connection to the candidate has to be terminated in order to conserve the total number of links.

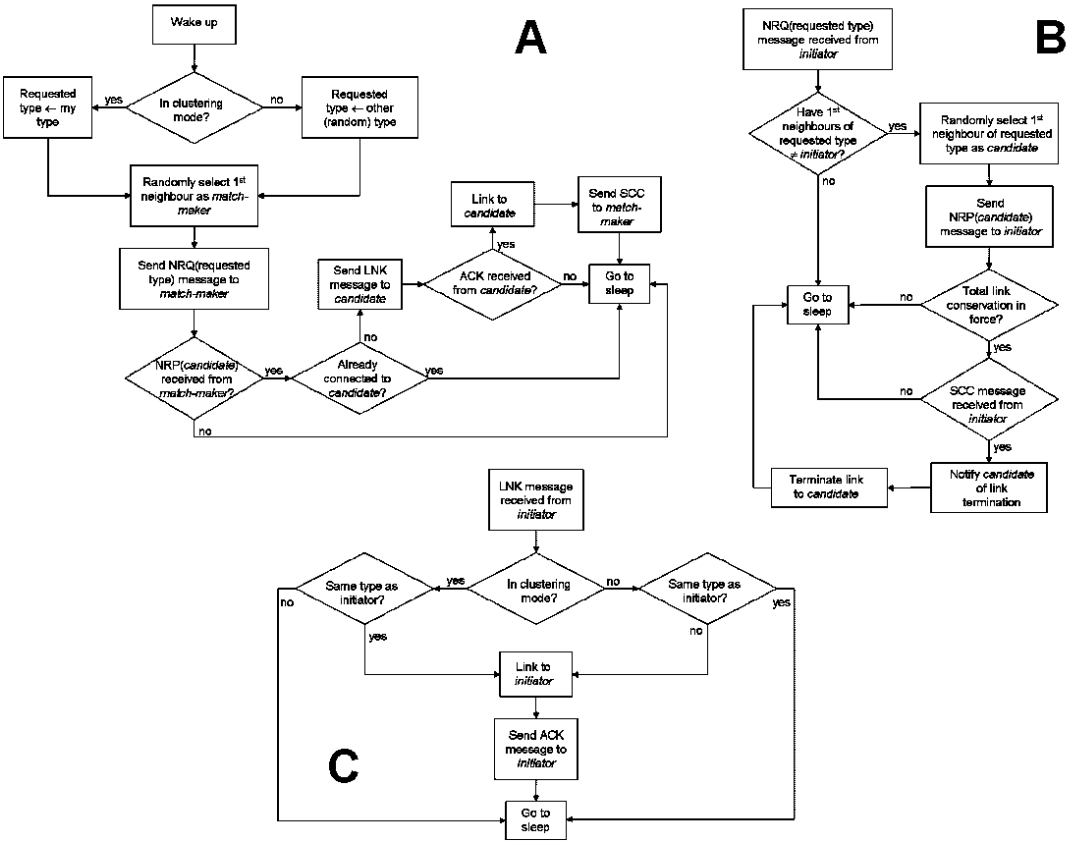

Fig. 3: The three "reasoning" loops governing node behaviour in the "on-demand" clustering scenario. Initiator rôle (A), match-maker rôle (B) and candidate rôle (C). See text for details.

2.2.1 Aggregation dynamics

In this section, we present simulation results that elucidate the collective aggregation dynamics arising in a network of peers following the local decision rules of the "on-demand" scenario, when all nodes are in "clustering" mode. We focused our investigation on two key parameters, namely the overall population size (number of peers) and the number of types involved. The average node degree (four), the topology in initial conditions (random graph) and the fraction of peers belonging to every type (an inverse function of the number of types, so that all "colours" are equally represented plus/minus one unit, due to rounding error) were maintained constant throughout the exploration of the parameter space.

We define the following additional variables to use as quantitative measurements of system state:

- Number of domains: the number of connected groups of peers belonging to the same type and exceeding a size of one (an isolated node, i.e. a node that is not

connected to any other node of the same type as its own, does not constitute a domain).

- Size of the largest domain: the "headcount" of the largest domain found in the system, notwithstanding the type of its members.
- Homogeneity: the fraction of links connecting two peers of the same type (can be for the entire system or for a specific domain, in which case the criterion is that at least one of the link's endpoints belongs to the domain).

We typically present mean values of these variables, averaged over 100 independent numerical experiments per combination of parameter values – population size and number of types – which ranged from 100 to 1000 peers (by increments of 100) and from two to ten node types respectively.

Figure 4 shows the evolution of link homogeneity (an indirect measurement of aggregation success) over simulation time. The first immediately noticeable property is scalability, with the 1000-strong population converging to similar or higher homogeneity values than the 100 nodes network. Though this is not a surprising result, a higher "surface/perimeter" ratio being *in principle* achievable in larger ensembles, it confirms that the chosen local rules are capable of supporting the self-organisation process leading to such ordered state independently of system size.

The "slowing down" effect for larger populations is attributable to the fact that a fixed number of rewiring attempts are made per time unit, independently of the number of peers. Had the probability of initiating such an attempt been fixed and identical for all nodes (assuming that possible inconsistencies resulting from concurrent modifications of the overlay can be prevented), leading to the number of rewiring attempts per time unit being statistically proportional to population size, indications are that convergence would actually be faster in larger systems.

For instance, if defining the plateau as the region in which the increase in overall homogeneity is <0.1%, it is reached in ~120 time-steps in the 100 peers/2 types scenario and in ~510 time-steps in the 1000 peers/2 types scenario, i.e. multiplying the population size by a factor 10 only translates into increasing the required number of rewiring attempts by a factor ~4. Scalability is again confirmed by the fact that the values are very similar in the case when 10 node types are involved.

Note that performance can only be measured relatively to the conditions in the initial random graph in which, statistically, the homogeneity variable is simply an inverse linear function of diversity (number of types). So for instance, reaching a situation in which ~35% of links connect peers belonging to the same type in the 1000 nodes/10 types scenario effectively means that homogeneity has increased by a factor ~3.5 compared to the initial conditions.

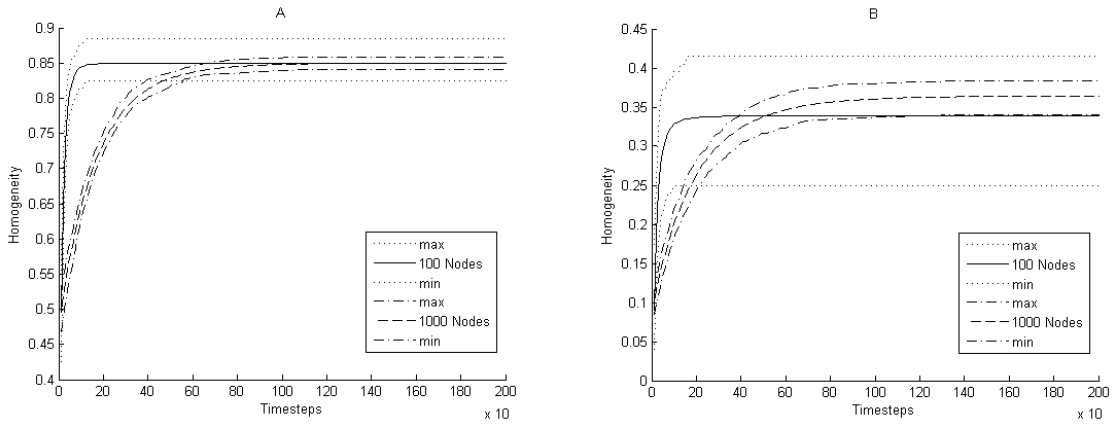

Fig. 4: Evolution of homogeneity over simulation time, for two (A) and ten (B) node types. Dashed curves indicate extreme values observed in the 100 independent realisations per combination of parameter values. See text for details.

Figure 5 shows the effect of the two key parameters (population size and number of types) on the characteristics of the domains, after 20,000 time-steps. Basically, the number of domains appears to be a power law of the diversity (number of types), the slope of which is a function of system size (number of peers). The mean size of the largest domain of each type (i.e. averaged over all 100 simulations *and* over all types) seems to be an inverse exponential of the diversity but it is also almost totally unaffected by population size (in relative terms). So in effect, the expected "headcount" of the largest domain is a linear function of system size.

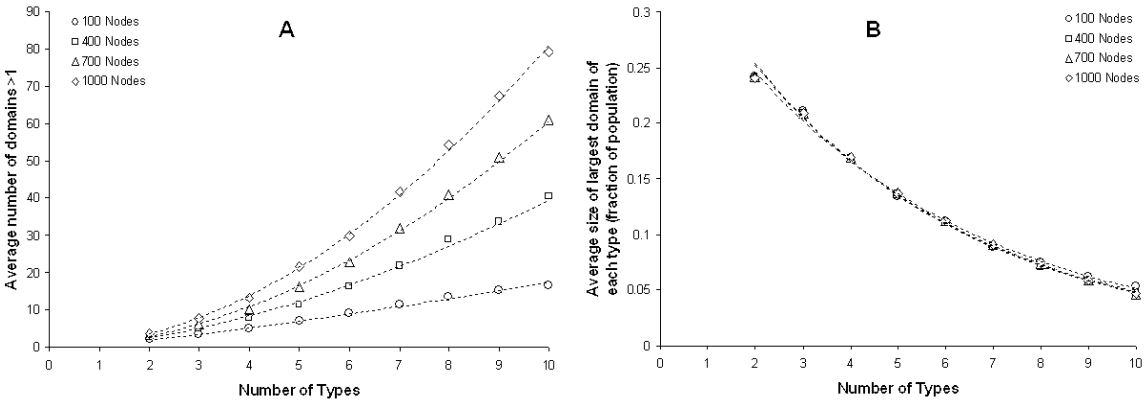

Fig. 5: Influence of the two key parameters (population size and number of types) on the number (A) and size (B) of the domains. Dashed curves in A are power-law fittings. Dashed curves in B are exponential fittings.

2.3 Spontaneous Separation of a Mixture of Behaviours

2.3.1 Mode Separation

In section 2.2 above we established that a mixed population of types of node could successfully form clusters providing all nodes obey a set of simple local rules. In many real world scenarios we would expect that some nodes would follow one set of rules and some another. A node may alter its rule set from time to time to reflect its internal state and we might even imagine that the rule set would be chosen to secure some selfish advantage for a node.

In this section we begin to explore the dynamics of a population which is a mixture of 'types' as in section 2.2 above, but where there is also a mixture of behaviours. In the work reported in this section we are concerned with only two behaviours: each node is in either clustering or reverse clustering mode. In the flow diagram of figure 3, Initiator rôle (A), a node in clustering mode will answer "yes" to the query "in clustering mode?" whereas a node in reverse clustering mode will answer "no". In other words clustering nodes will request links to new neighbours of the same type as themselves whereas reverse clustering nodes will request links to new neighbours of a type different from their own.

We were interested to see whether a population of nodes forming new links by type according to these modes would also tend to separate along mode lines, even though the modes are not themselves directly visible from one node to another. Would we tend to see highly interconnected sub-populations in the same mode?

2.3.2 Mixed Mode Simulations

In all simulations the population was formed by assigning each node the clustering or reverse clustering mode with equal probability. The mode of a node remained unchanged throughout the simulation run.

To assess the dynamics of the system we introduce two new measures, closely related to the global homogeneity as described in section 3 above.

First we have the global separation which is the fraction of links connecting nodes in the same mode. High values for this indicate that the initially mixed population (with global separation value 0.5 on average) has indeed separated such that nodes have more neighbours in the same mode (regardless of what type those nodes may be).

Secondly we have the global satisfaction which is the fraction of links which are acceptable to the nodes according to the type of the neighbour. In other words a node in clustering mode is "satisfied" with a link which connects it to a node of the same type. A reverse clustering node is satisfied with a link which connects it to a node of a different type. If only one of the nodes connected by the link is satisfied with its neighbour, the global satisfaction score for that link is halved.

The scoring system is given in table 1 below. To summarise we have three scores: homogeneity measures the extent to which nodes of the same type are clustered; separation measures the extent to which nodes in the same mode are clustered;

satisfaction measures the extent to which the nodes' preferences for neighbours of a particular type are met.

Node A Type	Node A Mode	Node B Type	Node B Mode	Homogeneity Score	Separation Score	Satisfaction Score
1	Cluster	1	Cluster	1	1	1
1	Cluster	1	Reverse Cluster	1	0	0.5
1	Cluster	2	Cluster	0	1	0
1	Cluster	2	Reverse Cluster	0	0	0.5
1	Reverse Cluster	1	Cluster	1	0	0.5
1	Reverse Cluster	1	Reverse Cluster	1	1	0
1	Reverse Cluster	2	Cluster	0	0	0.5
1	Reverse Cluster	2	Reverse Cluster	0	1	1

Table 1: Scoring system for the three measures of system organisation. The global homogeneity, global separation and global satisfaction scores are simply the mean values of these scores for all links in the system.

In the same way as in section 2.2 we saw global homogeneity rise during a simulation run, we expected to see global satisfaction rise in the mixed population because the local rules cause nodes to request nodes of a type which will "satisfy" them. We were interested to see how high satisfaction would rise and whether separation, which is not explicitly sought by the local rules, would occur.

2.3.3 Mixed Mode Results

Global separation increased over the course of each simulation run, resulting in values after 20,000 time-steps of 0.75 (+/- 0.05). The mean final separation value was little affected by the number of nodes and the number of types although the variance was larger in small networks (data not shown).

The number of types in the network did have a significant effect, however, on the number of time-steps taken to reach the final separation value. Figure 6 shows the separation of the mixed population over time. In all cases the population was 1000 nodes, with average

degree four. Each node had an equal probability of permanently assuming the clustering or reverse-clustering mode.

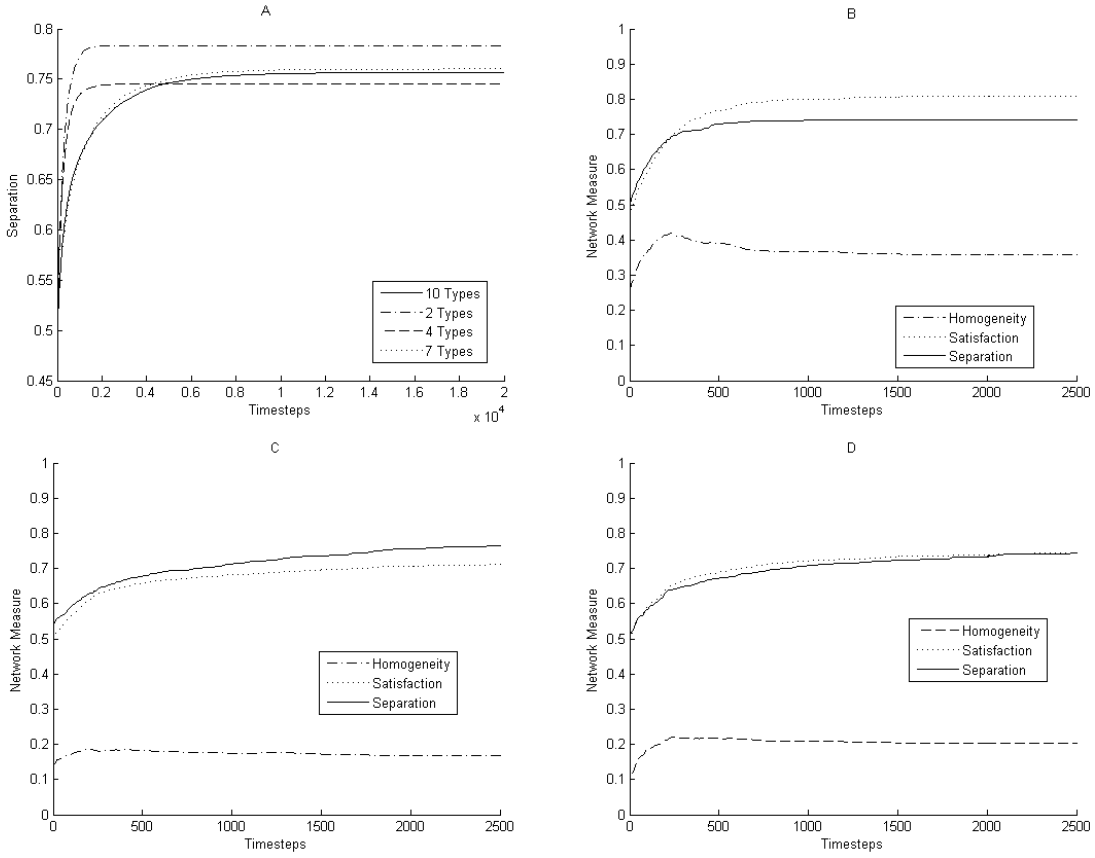

Fig 6: Evolution of network measures over simulation time. (A) Each curve shows mean separation results from 10 independent simulation runs. (B, C, D) Three examples of individual simulation runs showing evolution over time of Global Homogeneity, Global Satisfaction and Global Separation. (B) 4 Node Types, (C) 7 Node Types, (D) 10 Node Types.

The separation we see taking place is driven by the local rules through which nodes strive for "satisfactory" links. Each new link will only form if it is satisfactory to both requestor and candidate (see figure 3) and the effect of that new link on the global satisfaction measure will depend on the score for the link dropped between the matchmaker and the candidate. If the dropped link was already mutually satisfactory (scoring 1) the global satisfaction measure will not change. If the dropped link was mutually unsatisfactory (scoring 0) or only satisfactory to one of the linked nodes (scoring 0.5) the global satisfaction measure will rise by 1 or 0.5 respectively (and hence the satisfaction measure can never fall over the course of a simulation run).

This ratchet effect also applies to the separation measure. Since a candidate node will not accept a new unsatisfactory link, it is impossible for a clustering node to acquire a reverse

clustering node as a new neighbour, and vice versa. We were concerned therefore that the separation we see is no more than rising satisfaction by another name. We therefore examined the global homogeneity, satisfaction and separation measures for individual simulation runs to see what the relationship was. Figures 6b, c and d show three examples for four, seven and ten types of node. In each case there are 500 nodes in the network, permanently assigned to clustering or reverse clustering mode at time 0 with equal probability.

The plots of figure 6b, c and d are typical of individual simulation runs and show that the observed increase in separation over time is indeed closely associated with, but not identical to, the rise in satisfaction. Both measures cannot decrease during a simulation run, as explained above. Both can only rise if there is a successful new link established. Any increase in global satisfaction or global separation score resulting from that new link will depend on the satisfaction or separation score respectively of the link dropped by the matchmaker to make way for the new link.

3 Collaborative computing and load-balancing

3.1 Rationale

The purpose of developing the type of self-organising aggregation framework described in this report is to identify and document suitable rule sets capable of supporting autonomic operation of distributed peer-to-peer applications in the absence of central control. In the current trend toward service-oriented architecture (SOA), such an application would likely be designed as a collection of self-contained but mutually dependent service modules, providing elaborate "meta-services" via transparent composition and workflow management (Stal, 2002; Adam and Stadler, 2006).

Engineering such an advanced self-managing distributed service provision framework is the focus of much attention, particularly from industry (Kephart and Chess 2003). However, in most cases, it relies on centralised monitoring and implementation. We take the opposite view that effective system agility requires individual components to be fitted with the basic "reasoning" and decision-making abilities required to build, maintain and recycle collaborative relationships "on-the-fly" in a dynamic environment featuring a wide variety of perturbation factors (from variable resource availability to largely unpredictable demand patterns) (Montresor et al., 2002; Nakrani and Tovey, 2004).

In this section, we demonstrate how the local "on-demand" clustering rules described in section 3 can be used to generate and maintain a collaborative overlay connecting elements with complementary abilities, improving the performance of the system as a whole. Depending on individual circumstances, this can mean teaming up with processing nodes hosting the same service but facing different workloads (in which case aggregation can be used for load-balancing purposes, via transparent redirection of requests) or, on the contrary, with elements performing completely different functions (in which case one such element can act as an access point to a service that it cannot offer by itself, possibly as part of providing a composite "meta-service").

The overall objective is to dispense from centralised orchestration, in a manner somewhat similar to the one we proposed in previous work (Saffre et al., 2006b; Saffre et al., 2007), but with the important difference that the topology of the overlay is not fixed. On the contrary it is used to connect pre-specialised service components and doesn't involve (re-)allocation of local resources.

3.2 Simulated implementation

In order to experiment with the collaborative computing and load-balancing abilities of our overlay construction and maintenance framework, we devised a typical distributed processing scenario, as well as a modified set of local rules to inform the decision of requesting a neighbour of a particular type.

At initialisation, we compute a global workload distributed across the system according to a number of criteria (tuneable parameters). We then measure the global effect of rewiring on co-operative processing in terms of the evolution of that workload over time (no additional requests are made at run-time). This is of course meant as a demonstrator, featuring an easily measurable variable, i.e. the monotonically decreasing fraction of the initial workload still waiting to be processed. In practice, the same decision rules would take as their input the local structure of the demand in terms of service requests (type and arrival rate), not the properties of a static workload.

The two key parameters basically determine whether the demand is homogeneously distributed or not and what fraction of individual components receive requests for services that they do not provide directly. Defining heterogeneity of the workload is done by arbitrarily designating a specified fraction of nodes as being "overloaded". This simply means that, in initial conditions, the number of requests queuing at these nodes is multiplied by a factor 10. A second parameter determines what fraction of the nodes are designated "specialised" entry points. A generic entry point starts with all service types equally represented in the local load, whilst all requests queuing at a specialised entry point correspond to its particular type. Figure 7 shows an example of each of the four possible initial states ("normal/generic", "normal/specialised", "overloaded/generic" and "overloaded/specialised").

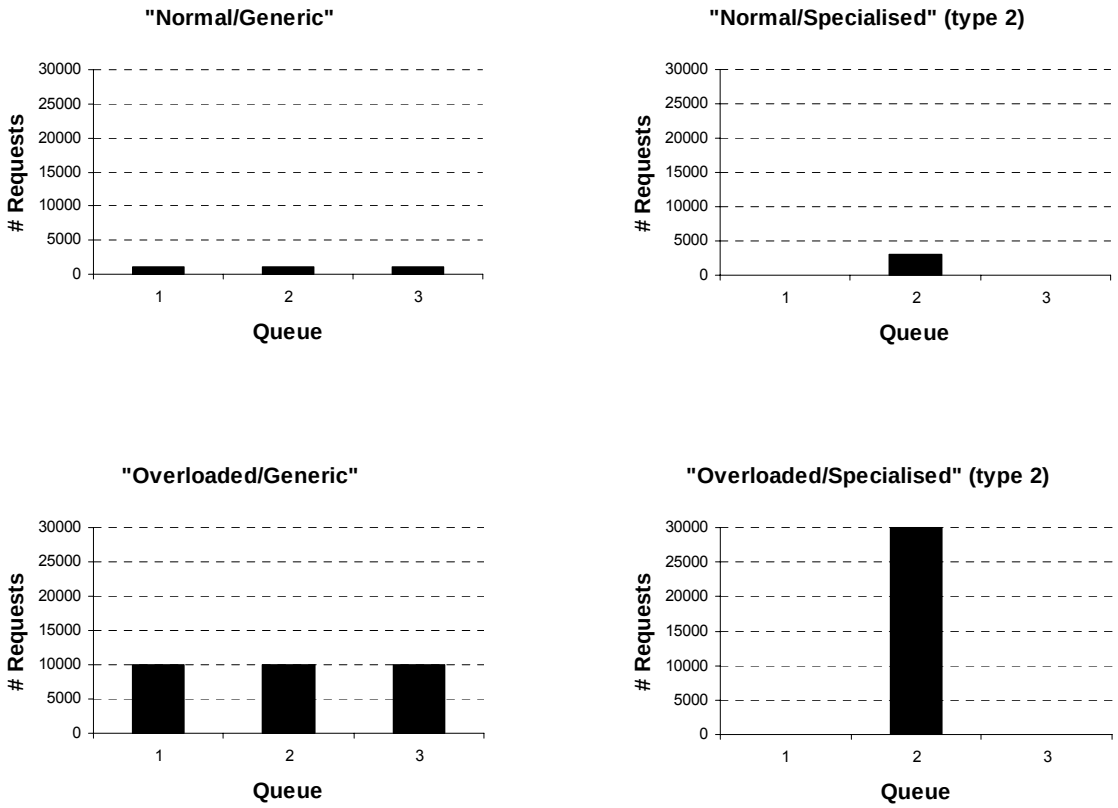

Fig. 7: The four possible initial node states, with three different service types.

The load-balancing logic is as follows. At every time-step, every node goes through the entire list of its first neighbours. For every one of them, it checks whether the queue maintained by that neighbour for the service that it provides is shorter than the corresponding local queue. If it is, one request is transferred (one step toward evening the load). This is done independently of whether both nodes are of the same type or not. Obviously, if the node initiating the queue length comparison is a specialised entry point for a service that is different from the one provided by that particular neighbour, no transfer can occur (as the local queue will be empty).

The rewiring process is as follows. At every time-step, every node has a fixed probability of initiating a neighbour request. If it does, it follows the same procedure as the on demand clustering described in section 2 above apart from two modifications:

- The requested type is not based on the initiator being arbitrarily set in “clustering” (same type) or “reverse-clustering” (any other type) mode. Instead, it is designed to reflect the current workload. Basically, one request is drawn at random among all queues and determines the desired type. This ensures that the probability of requesting a neighbour providing a given service is linearly proportional to the relative length of the corresponding queue (which mimics a rational decision by the initiator to try to establish the most useful collaborative link). It should be noted that

this logic also implies that a node without any requests queuing does never initiate a rewiring attempt.

- To keep things relatively simple at this stage, the candidate proposed by the match-maker does not perform any compatibility test: accepting the rewiring is automatic and mandatory (on the candidate's side).

At initialisation, all nodes are embedded in a random graph so as to reach a specified target average node degree. In all the simulations presented here, the population size is 1000 and the average node degree is four. The diversity (number of service types) is set to ten.

3.3 Results

In order to present results in the clearest possible way, we introduce the notion of a "cumulative penalty", which is incremented by one unit at every time-step for every request still queuing in the entire system. Obviously, this is an abstraction: in practice, a real penalty should only be incurred if queuing time exceeds an acceptable limit, probably defined as part of a service level agreement (SLA). However, it is a simple and convenient variable to use for the purpose of comparing global performance between rules and scenarios (i.e. "the lower, the better").

Figure 8 shows individual simulation traces for multiple combinations of the most relevant parameter values, namely the fraction of "specialised" access points, the fraction of overloaded elements and the probability (per time-step) that a node initiates a "neighbour request" procedure (P).

The overall trend is that the advantage of allowing the overlay to self-organise (reduced penalty) is maximised when all nodes act as generic access points (i.e. must handle requests for all services). This is to be expected in the sense that, with an average degree of four, finding suitable neighbours to process all ten request types effectively *requires* rewiring during the course of the simulation. Conversely, if all nodes are specialised access points (i.e. only receive requests for the service they can perform in isolation), the effect of rewiring is improved load-balancing (i.e. overloaded elements can more efficiently delegate processing if their neighbours belong in majority to the same type, which is highly unlikely in the initial random graph conditions). Indeed, in this scenario (100% specialised), the entire workload can in principle be processed locally (without request transfers) at the cost of substantially increased delays (which is the reason why all three curves, including P = 0, eventually saturate in fig. 8b, indicating that there are no requests left queuing in the system).

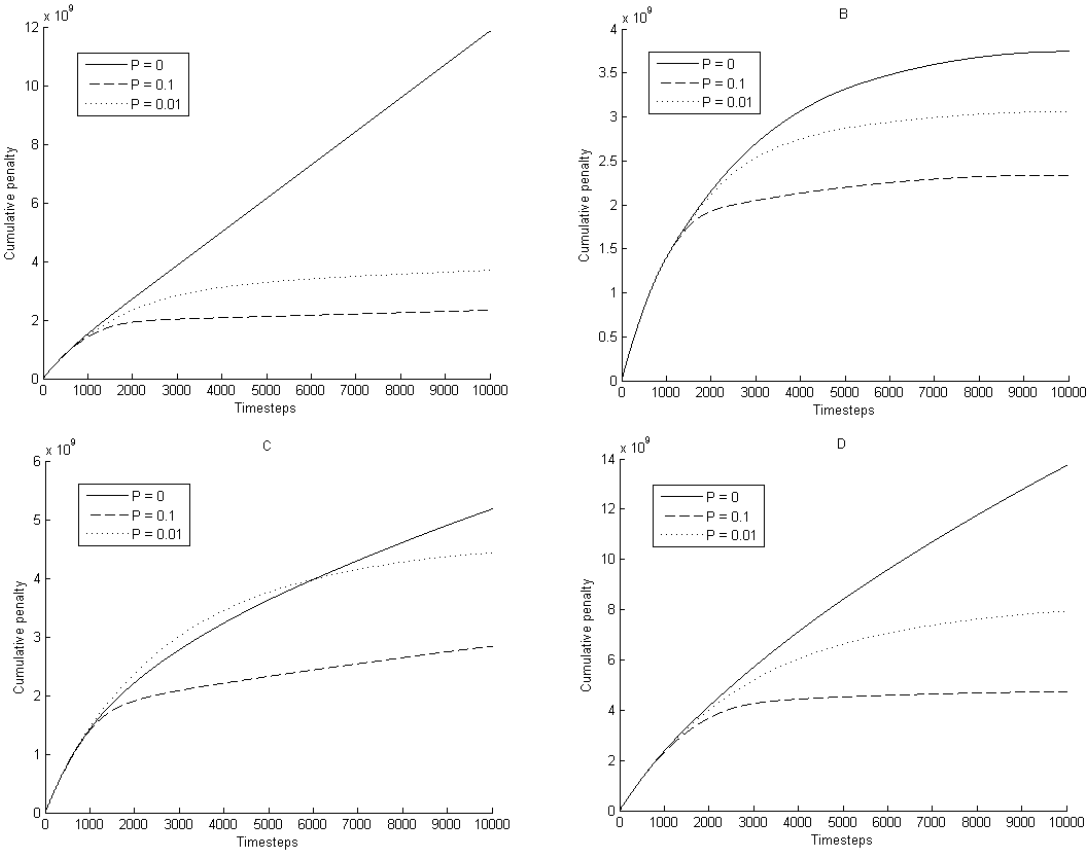

Fig. 8: Evolution of the cumulative penalty over simulation time, for three different "neighbour request" probabilities (1000 nodes, 10 types). (A) 0% specialised, 10% overloaded. (B) 100% specialised, 10% overloaded. (C) 50% specialised, 10% overloaded. (D) 50% specialised, 20% overloaded. See text for details.

The intermediate case (50% specialised) is probably the most interesting and realistic, as it implies the presence of nodes with different and potentially conflicting objectives (e.g. a generic access point could be teamed up with an overloaded specialist of a different type, resulting in neither of them being able to use each other's service). The expectation here is that the continuous rewiring process can implicitly deal with such a situation, statistically resulting in the establishment of more efficient partnerships in the long term. The evolution of the cumulative penalty (a measure of global success) suggests that the higher the fraction of overloaded nodes, the more beneficial the self-organisation of the overlay, which confirms that load-balancing is the clearest advantage of rewiring (compare fig. 8c and 8d). However, this macroscopic trend doesn't convey any useful information about the local, microscopic dynamics.

These are best explained by tracking the evolution of node degree over time, decomposing the signal based on node characteristics (normal versus overloaded, specialised versus generic). Results show that overloaded elements effectively succeed in surrounding themselves with "helpers" (independently of whether they are specialised or not), resulting in higher average degree (see fig. 9).

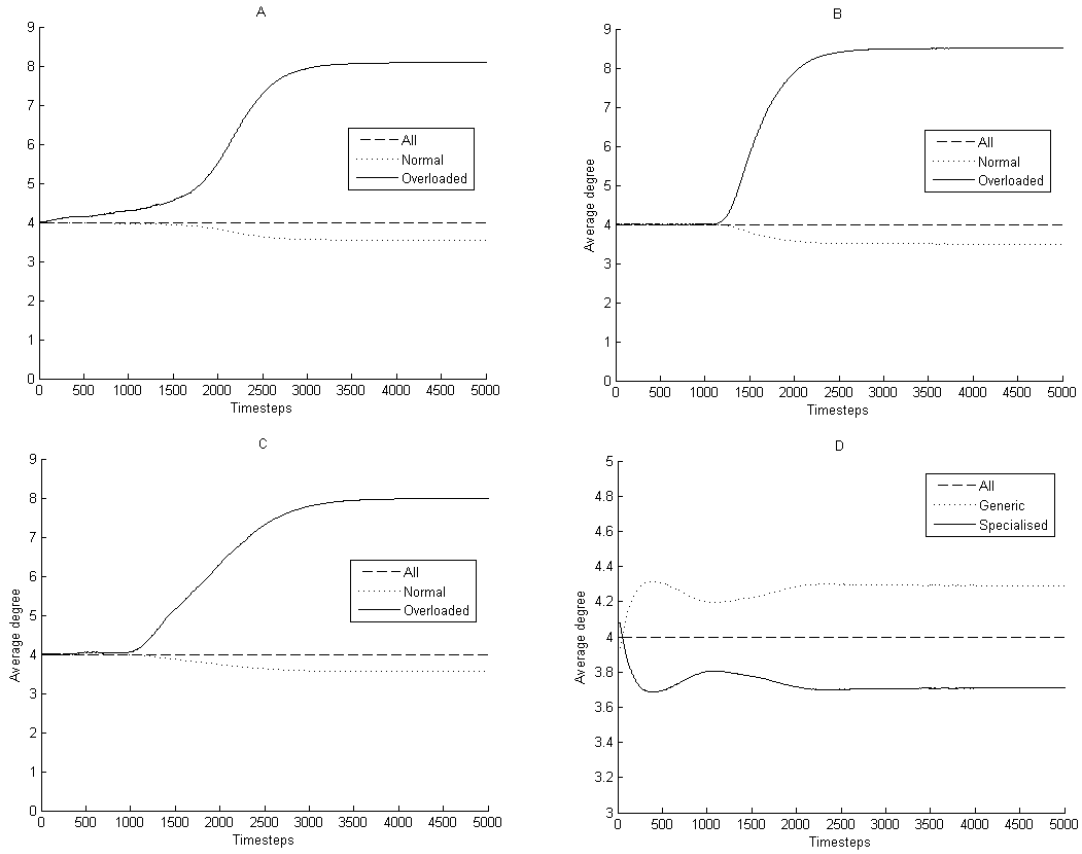

Fig. 9: Evolution of average node degree, split by node category (1000 nodes, 10 types, 10% overloaded, 100 independent realisations). (A) 0% specialised. (B) 100% specialised. (C and D) 50% specialised. See text for details.

The fact that the separation appears earlier in the 0% specialised scenario (t ~ 100) than in the 100% specialised scenario (t ~ 1000) reveals that the reason why overloaded nodes gradually acquire a higher degree is because they continue requesting new neighbours after their "normal load" counterparts have finished processing all their local requests and so have effectively stopped doing so (resulting in the asymmetry). Indeed, when specialised, a normal node has 1000 requests in its single populated queue at t = 0, versus 100 in every one of its ten queues for a generic access point, resulting in a 1000 versus 100 time-steps delay before normal nodes start idling and effectively become available as "helpers" (1 request processed per time-step). Note however that a generic access point can be idling and still have requests queuing (and so initiate rewiring attempts), as long as the local queue corresponding to its own type is empty. So even in the 0% specialised scenario, normal nodes are only gradually stopping to act as initiators after t = 100, hence the "slow start" noticeable on fig. 9a.

Fig. 9d is shown to indicate that, for an identical load, generic access points are statistically more successful at acquiring new neighbours than specialised ones. This effect is explained by the fact that, in a scenario featuring ten different types, the probability of a specialised node repeatedly asking for a single kind of neighbour resulting in a successful handshake diminishes rapidly as the local supply for the requested type gets depleted.

4 Impact of selfish behaviour on aggregation dynamics

Models of aggregation dynamics in networks are often based on the assumption all nodes perform according to their expected behaviour, that is to say cooperatively with other nodes. But this can't always be assumed in the real world. Accordingly we have to consider the effect of selfish behaviour by nodes on aggregation dynamics.

4.1 Benchmark: influence of permanent cheaters

A possible benchmark scenario is to consider the situation where a subset of nodes cheat all the time, that is they supply misleading information when acting as matchmakers irrespective of whether they will gain benefits from this cheating.

The potential impact of this cheating can be studied by modifying the simulation models of clustering described above.

Initial results are based on making a varying proportion of nodes fail to act as matchmakers whenever requested (proportion of cheaters in figure 10 below) and measuring the effect this has on convergence to global homogeneity.

4.1.1 Results

As expected, the presence of some nodes which refuse to act as matchmakers under any circumstances has an effect on aggregation. Global homogeneity, a measure of the degree to which clusters form, rises under all circumstances, but reaches lower plateaus as the percentage of cheaters increases. Further work is needed to explore the effect of permanent cheaters as different parameters are varied.

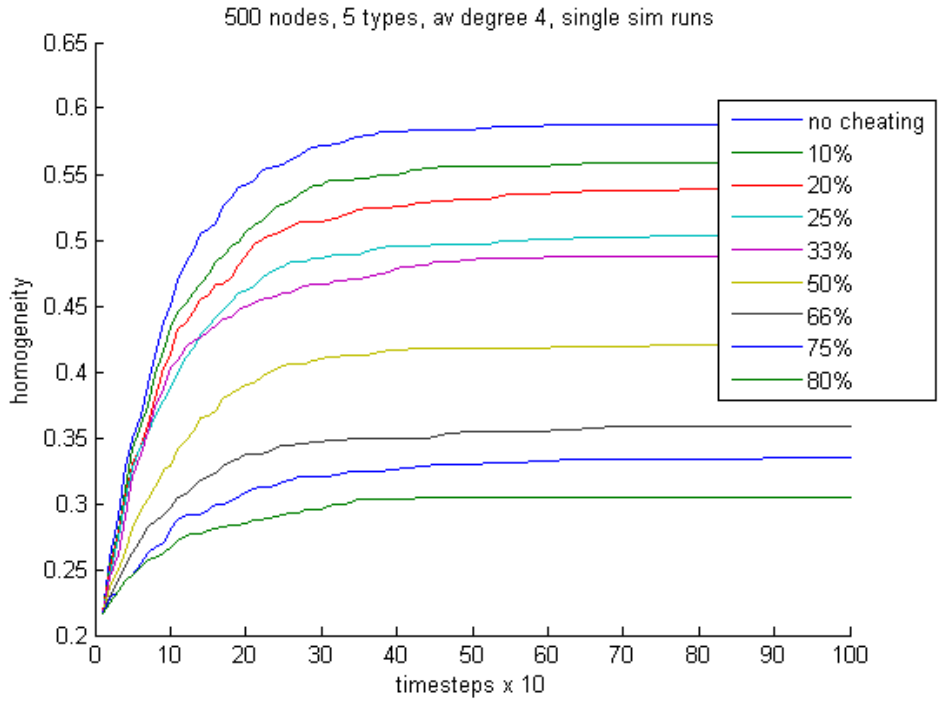

Figure 10. Effect of permanently cheating nodes on global homogeneity in clustering algorithm. 1000 links, randomly connected network.

4.2 Rational selfish behaviour

An appropriate extension of this approach is to consider the situation when some nodes cheat under certain conditions, but not all the time. Analysis of this can be assisted through the assumption that nodes are acting rationally, so as to gain selfish benefits. Here both simulation and game theory can be used to analyse the outcome of aggregation dynamics.

4.2.1 Degree-based

One approach to the investigation of rational selfish behaviour in aggregation algorithms is to consider the consequences of selfish cheating behaviour on average degree, that is, will aggregating nodes alter their behaviour selfishly in order to gain advantage through their links, and what effect will this have on the overall degree distribution?

We are particularly interested in seeing what the effect will be on global system behaviour when a subset of the nodes follows a 'cheating' strategy which might be intended to give them some advantage over their 'honest' neighbours. The first example of this, for which results are not yet available, considers the situation where nodes might wish to maintain an artificially high number of preferred neighbours (i.e. neighbours of the same type if the node is in clustering mode) by acting as an 'honest' matchmaker only when the requestor is of a different type.

4.2.2 Objective-based: selfish overlay creation for load-balanced service composition

4.2.2.1 Introduction

The Internet has evolved to become a commercial infrastructure of service delivery instead of merely providing host connectivity. Recently, different forms of overlay networks have been developed to provide attractive service provisioning solutions, which are difficult to be implemented and deployed in the IP-layer.

We term the logical connections among service provider nodes a *service overlay*. Services are deployed across the Internet and managed by individual, uncoordinated service providers.

In this context, service composition is the process of assembling independent, reusable service components to construct a richer application functionality.

When service components are deployed by multiple providers, issues of scalability, load-balancing across service instances must be addressed. Figure 11 captures the scenario under consideration: several replicas of different services are deployed on overlay nodes at different Internet locations. A client session is formed by choosing a specific set of service instances.

The focus of this section (4.2.2) is on the mechanisms to construct and maintain the service overlay taking into account the client request load that characterizes overlay nodes: overlay connections reflects the need for a node to distribute its load across replicas of the overloaded service. We use the term re-wiring to indicate overlay maintenance.

In this section we show that the problem of service overlay construction can be cast as a network creation game, first defined in Fabrikant (2003). We have extended the basic model to accommodate the service load experienced by the nodes and we study the equilibrium overlays that are built through their uncoordinated interaction. Our experiments on the *service overlay creation game* are hindered by the fact that computing the best response in this game is NP-hard. Hence, we present a randomized local algorithm to study service overlay creation with a realistic size.

Our preliminary results show that selfishly created overlays can scale well and allow for efficient load allocation.

The problem of load balancing for service overlays has been studied for example by Raman (2003), where obedient nodes perform overlay routing to balance client requests. In contrast, the study of uncoordinated and selfish load balancing ineluctably leads to the framework of congestion games (see for example Suri 2004, 2004a). The framework of congestion games offers tools to match client-server machines with the goal of sharing server work-load, neglecting the graph connectivity required by service overlay to achieve service composition.

Instead, in our work load sharing is considered as an additional constraint when building the service overlay; however, load balancing decisions cannot disrupt the normal overlay operation.

The remainder of this section is structured as follows: in Section 4.2.2.2 we detail the system model and architecture and in Section 4.2.2.3 we define the problem of the service overlay creation. In Section 4.2.2.4 we describe the numerical evaluation of our model and present some preliminary results. We illustrate related works in Section 4.2.2.5 and conclude in Section 4.2.2.6.

4.2.2.2 System model

In this section (4.2.2) we focus on service overlay networks, wherein overlay nodes provide not only application-level data routing but also value-added services, such as media compression/transcoding, language translation, encryption or decryption services, etc... Each service might have multiple instances that can be composed on-the-fly into a *service-level path* that provides new application functionalities.

We address a scenario where independent service providers deploy and manage their service instances at multiple locations on the Internet. Other third-party portal providers might compose these for end-users.

Figure 11 shows an example of a service overlay network, consisting of several overlay nodes deployed at different Internet locations. Individual service providers deploy their service components at these nodes (that they might own), which autonomously form an overlay network.

Service-level paths are constructed by choosing a set of required service instances and forming a path in the overlay network. The origin of a service-level path lies at overlay nodes that we term *exit nodes* for the particular client session. The exit node receives all client requests for a service or a set of services. Our system architecture is inspired by the work in Raman (2003).

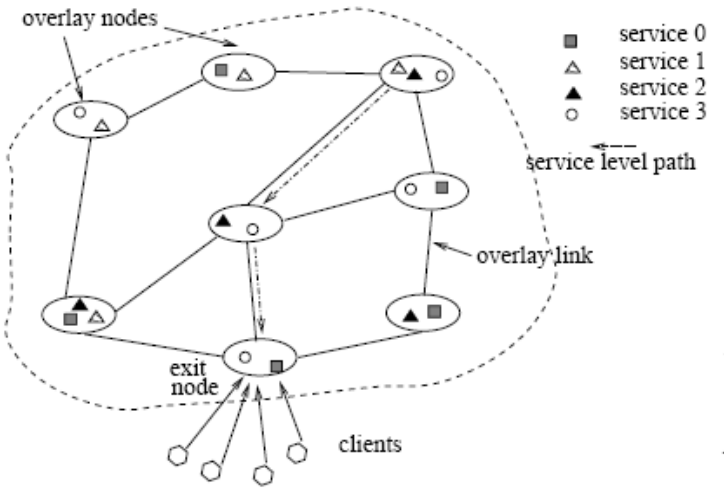

Figure 11. System architrecture under consideration with example local service path.

Unlike data routing, composing an end-to-end service path requires not only the overlay network connectivity, but also the satisfactions of various user quality-of-service requirements, such as response time and delays.

In this preliminary work we neglect the impact of the underlying physical network on the performance of the service overlay.

While there are several challenges in the context of service composition, in this section we focus on the construction and maintenance of the service overlay network. Service overlay creation is achieved by the uncoordinated interaction of nodes that wish to minimize the cost incurred in the creation and maintenance of overlay links. Node selfishness is expressed in terms link, communication and load costs.

We arbitrarily define an underlying economical model to explain the functioning of the service overlay. Service providers have a (monetary) gain per-request-served that arrive at exit nodes. However service capacity is finite and the latency (response time) experienced by end-users grows as the *p*th power of the number of requests-per-service at the exit node. This assumption is in line with related work such as Awerbuch (1995), Alon (1997), Avidor (2001) and Suri (2004a).

Excess service load can be shared (or traded in exchange of monetary compensation) between overlay nodes that have the same service replicas.

The closer a service replica to the overloaded exit point, the lower delay experienced by end-users: indeed, requests do not have to follow long service-level paths to be satisfied.

4.2.2.3 Uncoordinated service overlay creation

Given a generic unstructured overlay network that defines the logical connectivity among nodes, the problem we address in this section (4.2.2) is to find stable overlay

configurations that take into account load and connectivity constraints. As outlined in Section 4.2.2.2, load constraints imply that overloaded (exit) nodes seek at establishing direct links with other nodes offering the same service so as to share end-user requests, while at the same time being constrained by connectivity requirements. Note that also non-overloaded nodes are involved in the service overlay creation process.

Understanding the overlay network characteristics is the domain of our study in this section. Unless a centralized entity with a global vision of the overlay network dictates how to setup links in the overlay so as to satisfy performance and load constraints, each node needs to determine neighbors to establish links with. Our goal is to characterize the service overlay when nodes select links selfishly. To improve the tractability of the problem, we limit the scope of our study to physical topologies in which every node can communicate with every other node, that is the *communication graph* (i.e., the underlay) is connected. In the scenarios we investigate, we assume that an underlay routing protocol exists.

4.2.2.3.1 Assumptions

With the aim of simplifying the problem formulation we assume that only one service instance is deployed on each overlay node. Hence, in the following a node is equivalent to a service.

We define a *type* t_i associated with a node *i* as the encoded description of the service that node can provide.

Every node *i* is characterized by a *nominal capacity* λ_{ni} which defines the end-user request load a node can handle; λ_{ci} identifies the current load experienced by node *i*. In the following we assume that:

$$\Lambda_n = \sum_{i=1}^{n} \lambda_{n_i} , \Lambda_c = \sum_{i=1}^{n} \lambda_{c_i} , \Lambda_c \leq \Lambda_n \qquad (1)$$

Equation 1 simply states that the system is able to support the requested load, that is, the system does not saturate.

Note: because of the document format used, subsubscripts are not shown in the text in some cases where they should be. They are shown in the numbered equations.

4.2.2.3.2 Overlay creation game

We model overlay creation as a non-cooperative game with *n* nodes (i.e, players) whose strategies are to select which nodes to connect to. Our model extends the one presented by Fabrikant et. al. in Fabrikant (2003).

Formally, there is a finite set of players $N = \{1,...,n\}$, a finite set of player types

$T = \{t_1, t_2 ,...,t_n\}$ and the strategy space of player $i \in N$ is the list of other players to connect to, i.e., the set $S_i = \{(s_{ij})_{j \neq i} | s_{ij} \in 0,1\}$ where $|S_i| = 2^{(n-1)}$. Players simultaneously

announce the list of other players with whom they wish to be connected. Their decisions generate an undirected graph $G(s) = (N, A(s))$ as per the linking rule of the game. Note that this is a *single-stage* game with *simultaneous* announcements. Our game model requires complete information of the service overlay graph as well as node types and assumes players to be computationally unbounded.

In our game, $A(s) = \{(i,j) : i \neq j, s_{ij} = 1 \vee s_{ij} = 1\}$. Therefore, a link (i,j) is formed if either participant in the link decides to establish the connection. In the game, each player selects its strategy to minimize its cost.

The cost incurred by player i when all players adopt strategy s is additive in the cost l_j of a link to players of different type, in the cost l_j of a link to players of the same type modulated by a threshold function based on load information, as well as in the sum of the costs of reaching all other players:

$$c_i(s) = \alpha \sum_{j \in NB_i} (1 - \delta_{t_i,t_j}) l_j + \alpha\, \sigma(\lambda_{c_i}, \lambda_{n_i}) \sum_{j \in NB_i} \delta_{t_i,t_j} l_j + \\ + \sum_{j=1}^{n} d_{G(s)}(i,j) \quad (2)$$

where NB_i is the set of overlay neighbors for which a direct link exists, $d_{G(s)}(i,j)$ is the shortest path distance from node i to node j in the graph $G(s)$, and $\delta_{ti,tj}$ is the Dirac-Delta function $\delta_{ti,tj} = 1$ if $t_i = t_j$ and 0 otherwise.

The connection cost α represents the relative importance of player *i*'s direct links to others and is the only parameter in the model. Following the definition of the set $A(s)$, if player i establishes a link to player j, player i only will pay the cost for this action.

The function $\sigma(\lambda_{c_i}, \lambda_{n_i})$ is defined as follows:

$$\sigma(\lambda_{c_i}, \lambda_{n_i}) = 1 - \frac{1}{1 + e^{k(\lambda_{c_i} - \lambda_{n_i})}} \quad (3)$$

$\sigma(\lambda_{c_i}, \lambda_{n_i})$ is used to modulate the linking cost to a node running a replica of the service instantiated in i. When $\lambda_{ci} \leq \lambda_{ni}$, linking costs can be easily reduced to the model defined by Fabrikant (2003). However, when $\lambda_{ci} \geq \lambda_{ni}$ link costs to other nodes with the same instance of the overloaded service in i significantly decrease. Hence, as long as node i can support its current load, it will seek to minimize communication and linking costs; when the requests load λ_{ci} approaches and exceeds the threshold λ_{ni}, node i will prioritize the creation of links to overlay nodes that can support requests for the overloaded service.

The total cost of the graph G is then defined as:

$$C(G) = \sum_{i=1}^{n} c_i(s) \quad (4)$$

We now define what constitutes a solution of the game, that is, which overlay topologies result from the overlay creation game. When networks arise from the unilateral action of players, standard Nash equilibrium analysis can be informative about the structure of the networks that emerge. Let $s = s_N = (s_i, s_{N\setminus i})$ and let ζ designate the set of all undirected networks on N.

Definition 1: A network $G(s) \in \zeta$ is a Nash equilibrium network if there exists a strategy s that supports $G(s)$ where

$$c_i(s) \leq c_i(s'_i, s_{N\setminus i}) \forall i \in N \text{ and } s'_i \in S_i$$

4.2.2.3.3 Game optimization process

We study the equilibrium graphs created by the service overlay creation game through a numerical analysis. We use an (iterative) procedure in which players make decisions based on their current view of the network in order to select which actions to perform (change links).

The initial condition of a game is a connected random graph (see details in Section 4.2.2.3.4) and players change their link configuration to minimize the cost they bear, as given by Equation 2. We uniformly distribute the types to the nodes in the overlay.

We use two variants of this process in our numerical analysis. The first is characterized by an exhaustive search of the best response strategy, that is we compute the Nash equilibrium of the *one-shot game*. This approach limits the size of the network that we can study, as the time complexity to find all possible strategies a node can have is exponential with the number of nodes (the problem is somehow similar to that studied by Fabrikant (2003), where it is shown that computing the Nash equilibria for the network creation game is NP-hard).

We thus propose an alternative approach wherein we restrict the set of actions available to a player.

It should be noted that this alternative approach is a stand-alone algorithm and not an alternative method to compute Nash equilibria. These procedures are described in detail below.

4.2.2.3.3.1 Exhaustive search, Dynamic Best-response

Ideally, all of the strategy space available to a node should be examined to determine the action(s) yielding the lowest cost, as defined in Equation 2.

Our algorithm enumerates for every node *i* all the connection vectors that are within the feasible region of the optimization problem and associates a cost computed using Equation 2. Node *i* will select the connection vector (the strategy) that minimizes the cost. Note carefully that during each round, players are not aware of the moves of other players. This means that though we use a fixed ordering of actions, our numerical analysis is in line with the *simultaneous announcements* assumption of the game. If this was not the case, a node

could strategize by observing what other nodes chose before, whilst being in the same simulation round.

In this work we used the dynamic best response algorithm that is described, together with a possible optimization, in Sureka (2005). The simple exhaustive algorithm can also be improved using integer linear programming methods.

4.2.2.3.3.2 Randomized local search

We now describe an alternative algorithm for the overlay formation. As opposed to the exhaustive search technique, we limit the set of actions available to each player in the following way: during each round, a node can perform only one link addition and one link drop operation. The addition of a link is restricted to the 2-hop neighborhood of a node whilst a link drop is restricted to direct neighbors only. In practice, we introduce a ``mediator'' role: node *i* explores all possible mediators, which are all direct neighbors. For each mediator *j*, node *i* queries for a node *k* of the same type ($t_i = t_k$) that is not in *i*'s neighborhood. A randomized selection is performed in case of ties. When a link addition (*i,k*) is performed, a link drop (*i,j*) follows, where *j* is the node that offered node *k*.

Note carefully that a ``mediator'' *j* that satisfies $t_i = t_j$ is ruled out as a possible option. It should be noted that we allow links to be established and dropped thus the initial node degree remains constant or increases: players cannot strategize on the number of links.

Note that in this algorithm, link creation is not dictated by Equation 2 but only depends on the initial service graph. Hence the parameter α has no influence on the final overlay graphs. We keep, however, the legend describing α in the plots in Section 4.2.2.4 to identify results for different experiment runs.

4.2.2.3.4 Discussion

The service overlay creation game produces graphs that can potentially absorb a heterogeneous workload incoming at different entry points. However, our model is not concerned with the mechanism used to allocate the load, as it is done in traditional and selfish load-balancing. An interesting area of future research is to couple our overlay creation game with a mechanism used to trade excess load between service replicas owned by independent providers. A good candidate would be a distributed VCG auction scheme (Parkes 2004), in which excess load is allocated to those replicas that (truthfully) declare that they have spare capacity to offer.

4.2.2.4 Numerical evaluation

In this section we present the experimental setup used to characterize service overlays created by selfish nodes. Our goal is to compute the Nash equilibria topologies of the

overlay creation game and the stable topologies of the randomized local search algorithm using the methods described in Sections 4.2.2.3.3.1 and 4.2.2.3.3.2 above. The overlay network graph *G*(*s*) defined in Section 4.2.2.3.2 is represented by an adjacency matrix.

4.2.2.4.1 Experimental set-up

The initial conditions of the optimization problem are given by a set of *N* nodes, a node type vector *T* that identifies which services are instantiated on the *N* nodes and a randomly generated service overlay.

We use the Erdos-Renyi model of random graphs to generate the initial overlay topology, with a link probability defined by the parameter *p*. We define a maximum number $\hat{T}$ of different services available in the overlay and uniformly distribute them on the nodes.

In this work we assign equal nominal service capacities to each node, that is

$$\bar{\lambda_{n_i}} = \bar{\lambda_n} \forall i \in N.$$

The request load λ_{ci} assignment is described in Section 4.2.2.4.3 below. We also set the linking cost defined in Equation 2 to $l_j = 1$.

Every experiment is executed 50 times and the plots showed in Section 4.2.2.4.3 are an average over the number of experiment runs. For every simulation run we randomize on the initial service overlay.

4.2.2.4.2 Metrics

In this preliminary work we are interested in studying the graph properties of the equilibrium service overlay.

We evaluate the node degree that characterize each node. Note that although the graphs obtained with the two procedures described in Sections 4.2.2.3.3.1 and 4.2.2.3.3.2 are undirected by construction, in the resulting service overlay links are considered bidirectional. Hence, the adjacency matrix that represents the overlay graph is the symmetric version of the matrix representing the outcome of the optimization process.

For the Nash equilibrium topologies calculated with the dynamic best response algorithm we also show the total cost of the network as defined in Equation 4. The total cost is computed over the undirected equilibrium graph which is the outcome of the overlay creation game. As defined in Section 4.2.2.3.2 costs are not shared by the end-points of an edge but they are paid by the node that created the edge.

4.2.2.4.3 Preliminary results

In this Section we present some preliminary results on the graph structures obtained with the two optimization procedures. We restrict our attention to small service overlays, where N = 9 peers host one service each.

The initial overlay that we use as a seed for the optimization process is obtained using the Erdos-Renyi model with p = 0.4. We perform a connectivity check before running a simulation. We set the maximum number of service instances to $\hat{T} = 3$ and uniformly distributed service replicas (t_i) on every node: there are three replicas per service on different nodes. The nominal capacity allocation used in our experiments is λ_{ni} = 1 for all i. Figure 12 shows an instance of the initial and the best-response service overlays.

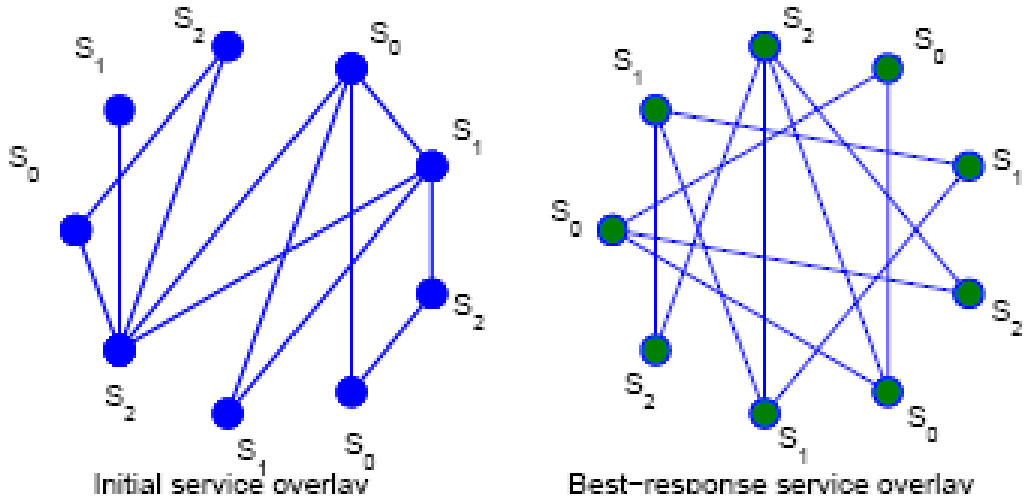

Figure 12. Two instances of service overlays. Using the initial service overlay, nodes run the dynamic best-response algorithm and rewire the overlay.

In this preliminary work we study two settings.

- Case 1: The end-user request load λ_{ci} is concentrated on three nodes (that are overloaded) hosting different services, whereas all other nodes are idle; $\lambda_{ci} = \lambda_{cj} = \lambda_{ck} = 3$ where $t_i \neq t_j \neq t_k$.
- Case 2: The end-user load is more balanced: for each service type one replica receives λ_{cl} =2, another λ_{cm} =1 while the third replica receives λ_{cn} =0, where $t_l \neq t_m \neq t_n$.

We study the impact of the only parameter of the model α that takes the following values: $\alpha = \{1, 10^2, 10^3, 10^4, 10^5\}$.

4.2.2.4.3.1 Results for Case 1

Figure 13 shows the tail of the node degree distribution for the equilibrium graph that results from overlay creation game, where the cost of a player follows Equation 2.

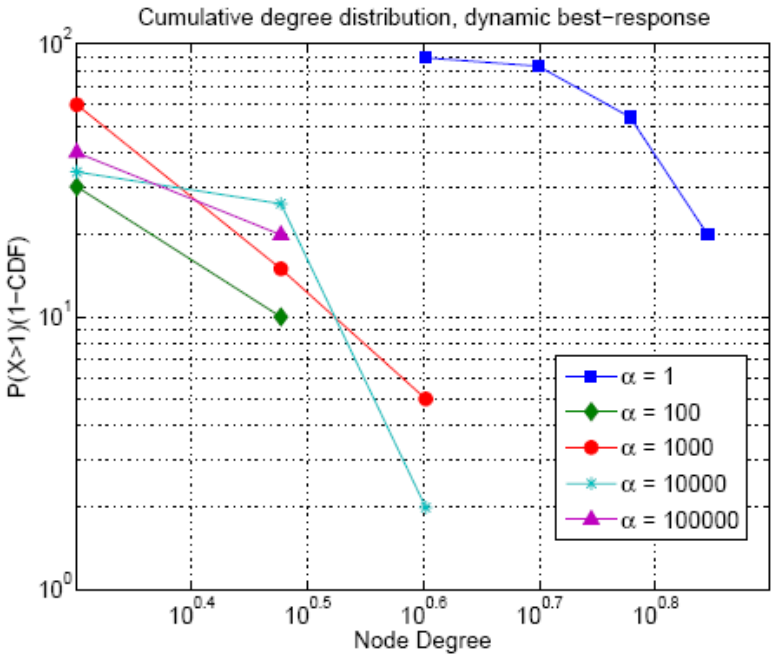

Figure 13. Case 1: Tail of the distribution of node degree of the equilibrium graph, overload scenario. For a low value of *α*, the tail has exponential decay, while for a high value of *α*, the decay approximately follows a power-law.

The graph is shown in a log-log scale and the curves indicate, for a given node degree *d*, the fraction of nodes with degree greater than *d*. For $\alpha = 1$, that is for a small cost associated to the creation of a link with respect to communication costs (in terms of hop distance), the distribution is not very skewed and its tail decays in a similar fashion to the exponential distribution. When $\alpha \geq 100$, the decay in the distribution is more steep, and exhibits similar behavior to that of a power-law.

Figure 14 shows the total cost of the equilibrium graph (see Equation 4) as a function of *α*. We note an exponential trend of the curve.

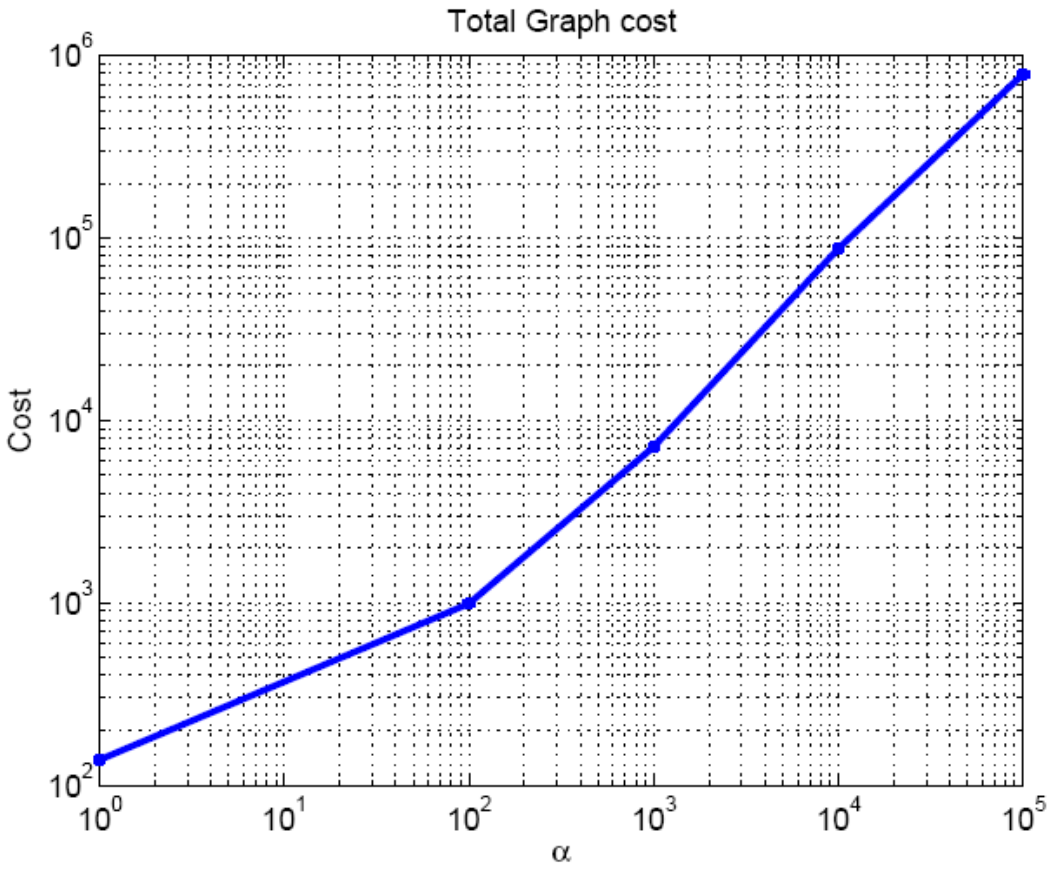

Figure 14. Case 1: Total cost of the equilibrium graph. Note how costs increase with the parameter *α*.

Figure 15 shows the tail of the node degree distribution for the randomized local search algorithm. Note that in this case the parameter *α* has no influence on the results, while the difference in the curve shapes is explained by different initial conditions of the service overlay.

The graph indicates an exponential decay of the degree distribution. This is a consequence of the re-wiring strategy of the local algorithm: the initial degree of a node can only increase due to the ``switch'' strategy when selecting which link to add and which to drop.

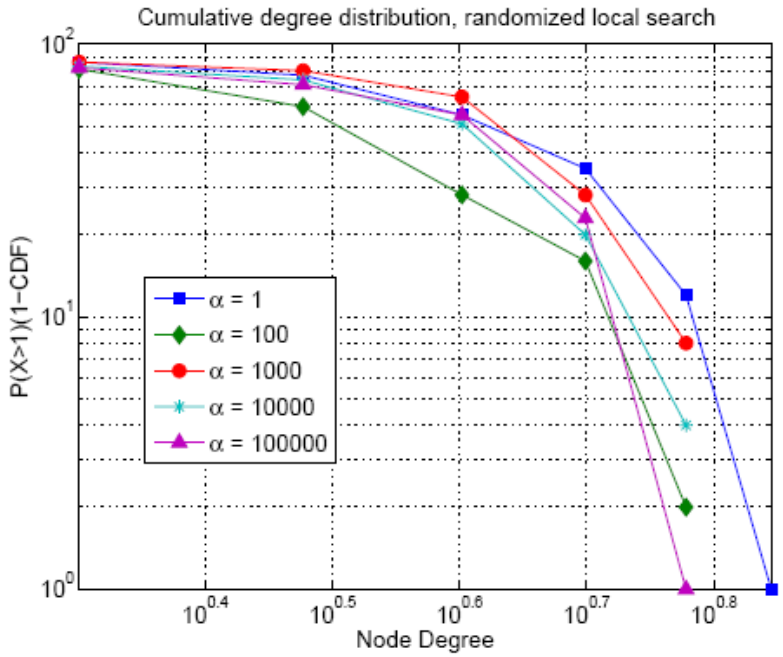

Figure 15. Case 1: Tail of the distribution of node degree of the graph resulting from the randomized local search, overload scenario. The parameter *α* has no impact by construction: the tail of the distribution of node degree follows an exponential decay, while variations depend on the initial random overlay.

4.2.2.4.3.2 Results for Case 2

In the following we present our results for a scenario in which the end-user request load comes at two entry points for every different service. We do not show the total cost of the equilibrium graph nor the degree distribution of the randomized local algorithm since similar observations as for Case 1 can be made.

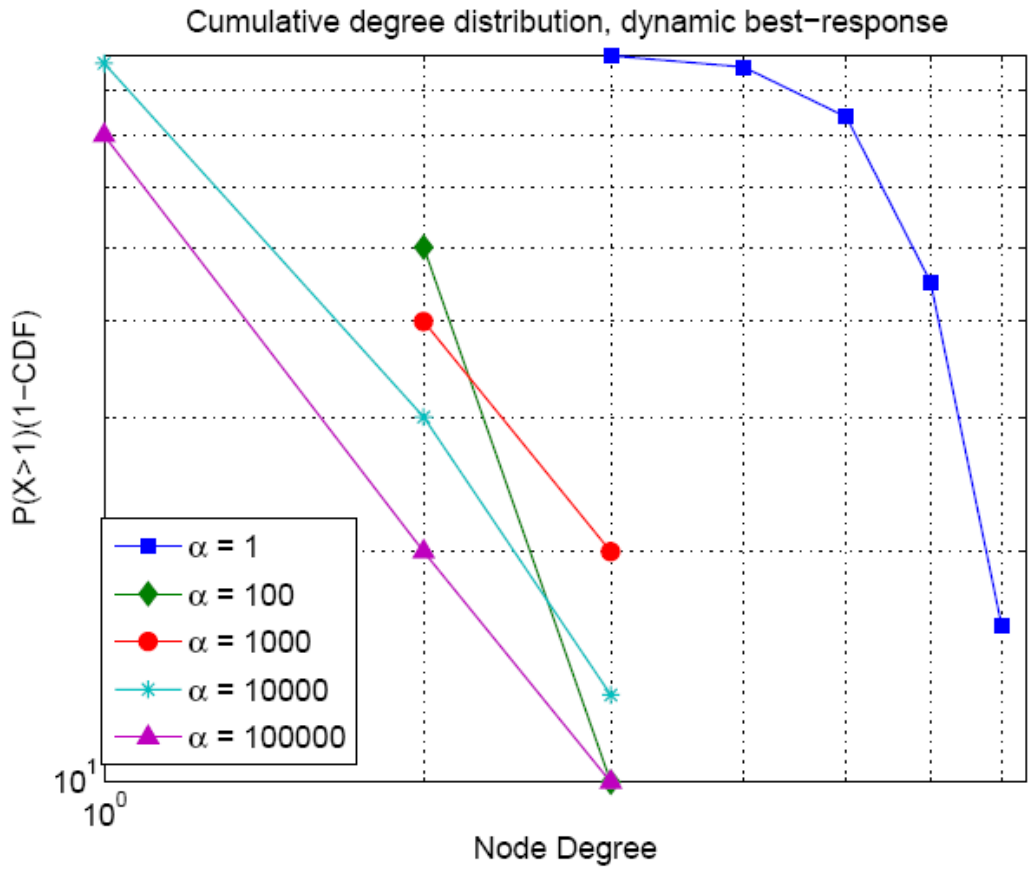

Figure 16. Case 2: Tail of the distribution of node degree of the equilibrium graph, overload scenario. For a low value of *α*, the tail has exponential decay, while for a high value of *α*, the decay is much more compact than in Case 1.

Figure 16 shows the tails of the degree distribution for the equilibrium graphs. While similar observations as for Case 1 can be made, we notice that the degree distribution span is reduced as compared to the previous scenario. This is a consequence of the presence of two entry points in the service overlay together with the request load distribution. Two out of three overlay nodes hosting a replica of the same service reached or passed the threshold (see Equation 3) that reduces the linking cost to nodes of the same type.

The important finding that we have obtained in this work is that our game model can be seen as a way of obtaining power-law graphs supporting a heterogeneous workload. We are currently investigating on a refinement of the randomized local search algorithm to drop unnecessary links so as to obtain degree distributions that decay approximately as a power-law.

4.2.2.5 Related work

Traditional load balancing has a long history in distributed systems literature. A small sample of previous results, that focus on coordinated and often centralized solutions, includes the following: Philips (1993) investigates the online assignment of unit length jobs under the L_∞ norm; Alon (1997) considers offline assignments of unit length jobs; Avidor (2001) consider greedy assignments of weighted jobs under the L_p norm, where the client-server graph is complete bipartite while Awerbuch (1995) considers arbitrary client-server graphs and uses the L_2 norm.

Uncoordinated load balancing has been studied using game theory for example by Suri (2004, 2004a). Congestion games are at the heart of these works: uncoordinated, selfish clients compete to select servers providing the lowest response time (latency). Client latencies are related to the server loads. Server response time is assumed to be inversely proportional to the speed of the server, but grows with the *p*-th power of the number of users connected to the server. As we point out in Section 4.2.2.1 this framework is not suitable to meet the connectivity requirements of the service overlay.

The work presented in Raman (2003) from which we borrow the system model achieves load balancing among service replicas through overlay routing. The link metric adopted to select overlay service paths is inversely proportional to the capacity available at nodes on the service path. Note that load information is disseminated in the overlay so as to update the link cost metric.

The works that are closely related to ours are presented in Fabrikant (2003), Chun (2004), Corbo (2005). Fabrikant et al. first present the network creation game that they use to characterize Internet topologies (Fabrikant 2003). Their model accounts for link creation and communication costs and the resulting graph is generated by the unilateral interaction of selfish players. This model is extended in Corbo (2005), where the authors consider equilibrium topologies that arise from the bilateral interaction of selfish players; in their model, link cost is paid by both vertices of an edge in the equilibrium graph. Chun (2004) presents an extension of the basic model in Fabrikant (2003) wherein an application-level routing overlay is constructed by selfish overlay nodes.

4.2.2.6 Conclusion and future work

In this work we study service overlays created by the uncoordinated action of selfish nodes deployed across the Internet. Instead of routing in the overlay excess user requests that would overload a node, we allow nodes to re-wire the overlay taking into account both their uncoordinated, selfish behavior and load requirements.

Services are instantiated on the nodes of the overlay by independent, selfish providers who seek at maximizing their gain by serving as many end-user requests as possible. We model the creation and maintenance of the service overlay using an extension of the network creation game originated by Fabrikant (2003) that takes into account link, communication and load costs.

The ultimate goal of the overlay creation process is to render the service-overlay capable of absorbing a heterogeneously distributed workload that would otherwise result in some nodes with a specific service being overloaded and others remaining idle.

Our preliminary results show that varying the α parameter in the cost model produces service overlays with degree distributions with tails ranging from exponential to power-law distributions.

We also propose a randomized algorithm to achieve the same goal without the need for global knowledge of the overlay graph. The algorithm converges to stable overlay configuration with degree distributions with exponential tails.

In our future work we will design a mechanism to allocate load across service replicas, based on the selfishly constructed overlay. With the ultimate goal of studying the ``Price of Anarchy'' Koutsoupias (1999) we will compare the load distribution computed by a centralized optimal load balancing algorithm and the worst case equilibrium obtained with our solution.

5 Self-aggregation and ACE specification

This section is focused on how to apply self-aggregation algorithms within the context of the ACE model being developed by WP1.

Sub-section 5.1 describes the ACE architecture proposal from WP1. Sub-section 5.2 shows how the aggregation algorithms described in section 2 can fit into the ACE architecture.

Sub-section 5.3 describes the architecture of the prototype being realized at DEI and provides an initial discussion on the evaluation of its performance. In the prototype we experiment the application of the aggregation algorithms in the context of a simplified version of the ACE model. Our ACEs are distributed entities interacting through an event-based middleware. For the moment they do not perform any application-specific action, but they are able to enact the self-aggregation algorithms in a distributed setting.

Sub-section 5.4 closes the chapter by presenting a preliminary evaluation of the approach.

5.1 ACE Architecture Proposal from WP1

An ACE consists of a common part and a specific part. The common part is identical for each type of ACE and contains a minimum set of fundamental capabilities. In contrast, the specific part contains additional functionality that is required for solving particular tasks and may be different for each type of ACE [1].

Starting from this definition and from the Conceptualisation elaborated in the first project year, Figure 17 shows the WP1 proposal for the Ace structure [2]. In the following we

shortly describe the elements related to WP3 aggregation algorithms. For a complete description the reader can refer to [2].

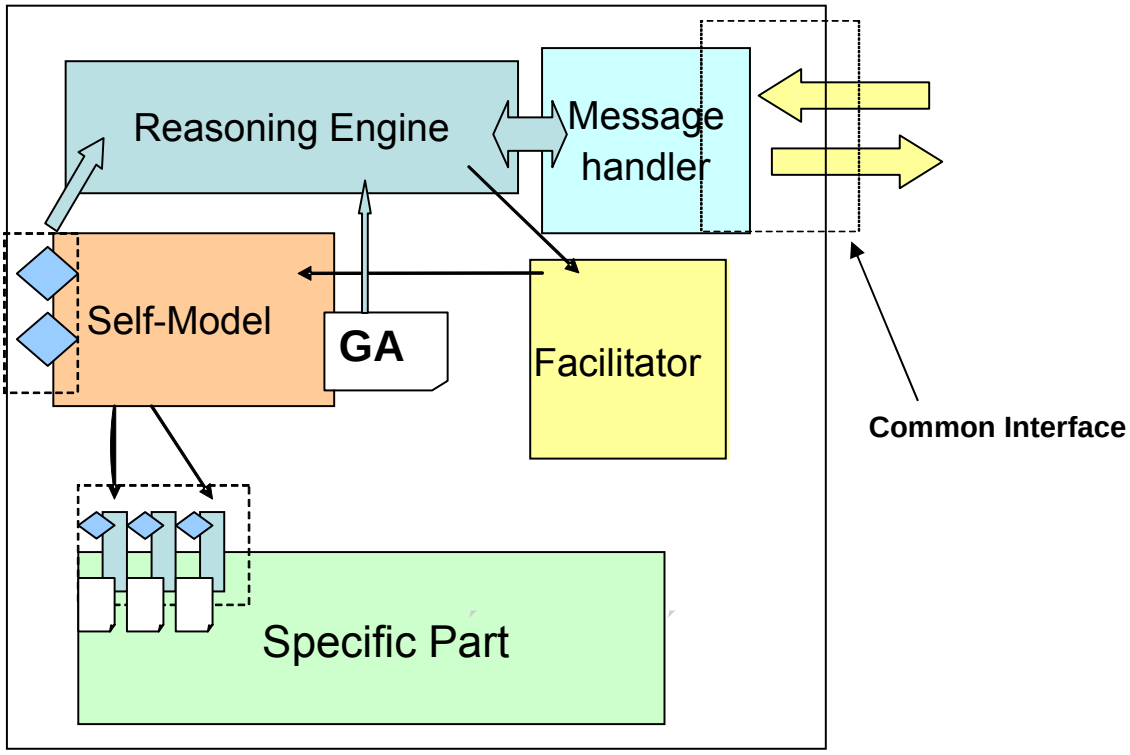

Figure 17. ACE conceptual model [2].

5.1.1 Common Part

The Common Part defines the way ACEs behave, communicate and interact with the outside world (i.e., other ACEs or the environment). The communication is message-based and then the **Common Interface** is implemented by a **Message Handler** which is able to understand a fixed set of messages which implement the way ACEs collaborate and communicate.

5.1.2 Common Interface

The set of messages addressed by the Common interface are:

- Goal needed (GN): a sort of request, with a semantic description attached, which specifies what kind of functionalities the ACE needs from other ACEs, to achieve its goals. This implies that any ACE should be able, given a GN, to semantically match it with its Goal Achievable (GA) (see next item) in order to properly answer to the received GN.
- Goal Achievable (GA): this message is used by an ACE to state what kind of task it is able to provide. It is also a semantic description and typically is used to reply to a GN request any time the ACE is able to satisfy it.

5.1.3 Specific Part

The Specific Part contains the ACE specific functions. It exposes these functions through the Specific Interface.

5.1.4 Specific Interface

The Specific Interface contains a description of each function contained in the ACE specific part. For each function a semantic description of the **Goal Achievable** (i.e. the job the ACE is able to do) and the **Preconditions** (i.e. indispensable and essential action, conditions) needed for the function execution are specified.

5.1.5 Self-Model

The Self-Model describes the possible states for the ACE and the possible transitions between pairs of states. In other terms, it could be defined as a state machine and then

the Self-Model is a description of the steps the ACE will execute to achieve its goal.

Any state is described by a semantic description used by the ACE to reason about its current state (see Reasoning Engine).

The transition functions are the specific functions: they indicate a state change and are described by an event, a condition (Precondition) that would need to be fulfilled to enable the transition, and then the specific function to be executed.

The ACE self-model is published by using the above described GN – GA protocol.

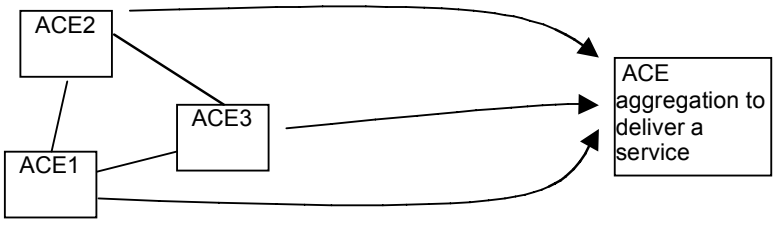

Figure 18. Ace aggregation.

5.1.6 Reasoning Engine

The reasoning engine executes (the implementation of) the self-model and its main role is to keep trace of:

The state reached in the Self-Model execution. Eventually, it may take trace of the history, storing the previous states of the self-model.

The environment: any GA or GN coming from other ACEs.

Mainly, it is able to run the state machine used to describe the self-model. It checks if a transition may take place, runs the transition, invoking the proper specific features if specified, and proper represents the semantic description of the new reached state.

5.1.7 Facilitator

The Facilitator is the core autonomic part of the ACE, adapting its behaviour to the changed conditions, situations, fault etc…

The behaviour adaptation in this model means changing the self-model state machine. In order to achieve this, the self-model "developer", or any autonomic mechanism able to shape the self-model, has to insert in the original self-model some additional transitions to so called "variation states" necessary to adapt the ACE behaviour, such additional transitions are called *Join Point*.

The facilitator main execution cycle is the following:

The Reasoning Engine will notify any changes perceived in the environment or in the ACE's internal state (self-model) to the Facilitator. The Facilitator is able to match such changes to a template and, any time a match is found, the proper Join Point is activated in order to adapt the current self-model to a modified one needed to face the changed condition.

5.2 Self-Aggregation

As discussed in the previous section, within WP1 autonomic behaviour are understood as the possibility for an ACE to change its self-model. This change is triggered by the Facilitator that is able to identify the situations that determine the change.

From the perspective of WP3, aggregation algorithms can be seen as another way for the ACE of being autonomic. In this case, the change does not affect the self-model of the ACE, but the relationship with its neighbors. This can still be considered a form of evolution to be handled by the Facilitator subcomponent on the basis of the algorithms presented in the previous sections of this report.

The reasons for the Facilitator to trigger aggregation can be of various kinds. For instance:

- The Facilitator realizes that the load of the ACE to achieve its goal (GA) should be split among various others in order to improve the overall performance of the system.
- The ACE already belongs to an aggregation but the Facilitator realizes that it cannot reach the ACE neighbors. In this case, the aggregation algorithms need to be started in order to recreate the aggregation itself.

- The Facilitator realizes that, in order to achieve its current goal (GA), the ACE needs a number of goals (GNs) that cannot achieve by itself. In this case, the aggregation algorithm can be started to establish a stable relationship with those neighbors able to offer the required goals (GNs).

Various other reasons for aggregation can be identified depending on the specific application the ACEs are built for. The Facilitator will offer a language for expressing the rules triggering the execution of the aggregation algorithms. This language will be used by the developer to properly define the autonomic behaviour of ACEs.

For performance reasons, it is likely that the aggregation algorithms do not run forever when started. The Facilitator will incorporate some heuristics to determine when to stop aggregation still obtaining a reasonable grouping of ACEs.

As discussed in the previous sections, the aggregation algorithms that are currently available are of two kinds, those supporting the aggregation of neighbours of the same type (clustering) and those supporting the aggregation of neighbours of different types (reverse clustering). It is reasonable to think that, in a given situation, an ACE would need to enact both of them to establish different overlay networks of neighbours (possibly mapped on the same physical network) coexisting within different conceptual planes. So, for instance, an ACE will participate in a cluster group for a certain purpose and in a reverse cluster group for a different purpose. Being these groups defined in different overlay networks, they will not interfere with each other.

From the perspective of WP1, a network of ACE neighbors can be seen as an *aggregated ACE* whose goal (GA) is obtained by composing the goals (GAs) of the component ACEs. With this respect, the result of an aggregation algorithm on a network of ACEs can be seen as a change in the Self-Model of the corresponding aggregated ACE. This allows us to re-conduct all extensions to the Facilitator needed to support aggregation into the framework designed by WP1 that considers any autonomic behavior as a change into the Self-Model.

In the next months we will concentrate on the requirements for the language for triggering aggregation, on the heuristics for stopping it, and on the possibility of supporting the coexistence of different aggregations at different planes.

In the next sections we describe a first proof of concept prototype of distributed ACEs able to perform some variations of the aggregation algorithms presented in the previous sections of this report.

5.3 Architecture of the distributed prototype of self-aggregating ACEs

The aim of our prototype is to implement a simplified population of ACEs each of which is able to execute the the clustering and reverse clustering aggregation mechanisms. ACEs are distributed and can communicate by sending and receiving events according to a publish/subscribe style. Each ACE for the moment is very simple. It has a type (colour in the WP3 terminology) and knows its neighbors. Moreover, it is able to understand if another ACE is of the same type or of a different type. For the moment the prototype is able to manage simple types. The management of complex types requires the definition of proper type matching mechanisms.

We have chosen a publish/subscribe interaction paradigm because of its ability to support multicast interaction, decoupling between senders and receivers of events, easy plugin/out of components. To implement such a paradigm we have exploited the REDS middleware [3,4] that we shortly present in Section 5.3.1. It offers a MessageBroker that is the logical component in charge of both receiving subscriptions and events and delivering the events to those that have subscribed to them. This logical entity can be installed as a centralized component or as a distributed set of brokers. The middleware is in charge of supporting the synchronization among them in order to guarantee that all subscribers receive the events even if these are published by component connected to a different broker.

In order to support the deployment and the startup of the system, we have developed a utility component called Configurator. Its goal consists in the instantiation of the distributed ACEs components and of the communication middleware.

The figure below summarizes the various components of the architecture.

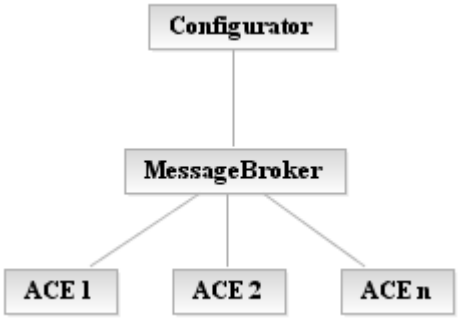

Figure 19. Main components of the prototype architecture.

In the following we give some details on each prototype component.

5.3.1 ACE Component

The ACE component contains the following classes (see Figure 20):

- The class **AceLocalNode** contains all data needed for the correct execution of an ACE. Among the other data, it owns a list of neighbor nodes each one represented by an instance of AceNode.
- The abstract class **AceNode** contains the *type* and the *identification* of an ACE.
- The interface **AceMessageHandler** is the class that manages the interaction with the other ACEs. It is a REDS client that allows the ACE to connect and to use the MessageBroker.
- The logic of self-aggregation algorithms is defined in the **AggregationAlgorithm** class (and in its subclasses).

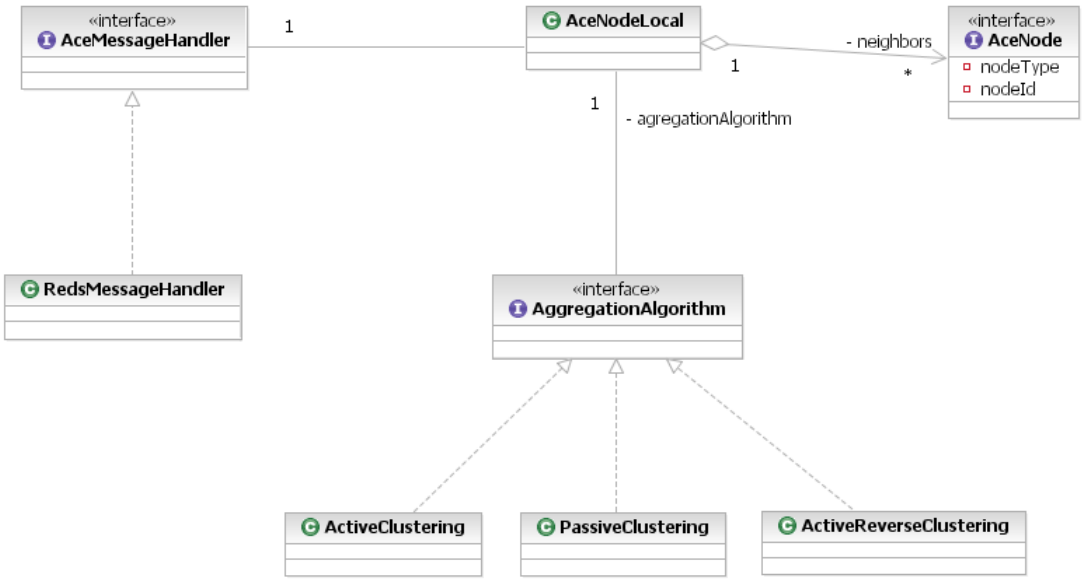

Figure 20. Main elements of an ACE.

Referring to the WP1 ACE model, AceLocalNode and AceNode are to be considered part of the ACE Self-Model, while the AggregationAlgorithm is the part of the Facilitator in charge of managing autonomic reconfiguration of the relationships between the ACE and its group.

5.3.2 Dynamic view

The diagrams in Figure 21 and Figure 22 represent the interactions between a triple of ACEs to implement the active and passive clustering algorithms.

Figure 21 defines the steps to be followed to execute the passive algorithm:

1. The **initiator,** at a certain point in its life decides to start the (reverse) clustering algorithm. This decision can be taken as a result of a change in its internal state or as a result of the observation of the external environment. To start the algorithm the initiator sets its state to busy and chooses two of its neighbors. These neighbors must be compatible with each other, i.e., they have to be of the same type in case clustering is going to be executed, of different types otherwise.
2. The **initiator** node sends a startAggregation request to **firstNeighbor** and **secondNeighbor** and waits for the responses. If one of the two is busy, **initiator** stops the execution of the algorithm. Otherwise it executes the next steps.
3. **initiator** sends a *connect()* message to the **secondNeighbor** that executes an *addNeighbor(**firstNeighbor**).* **initiator** node waits for the response message from **secondNeighbor.**

4. After receiving the confirm message, **initiator** node sends a *requestSwitch()* to the **firstNeighbor**. The **firstNeighbor** node executes a *removeNeighbor(**initiator**)* and an *addNeighbor(**secondNeighbor.**)*
5. **initiator** waits for the reply message from **firstNeighbor**.
6. **initiator** informs the two neighbors that the switching operations have been terminated
7. Finally, **initiator** removes the firstNeighbor from its neighbor list.

Please notice that setBusy and setIdle operations are used to avoid that more than one (reverse) clustering activity is performed at a time.

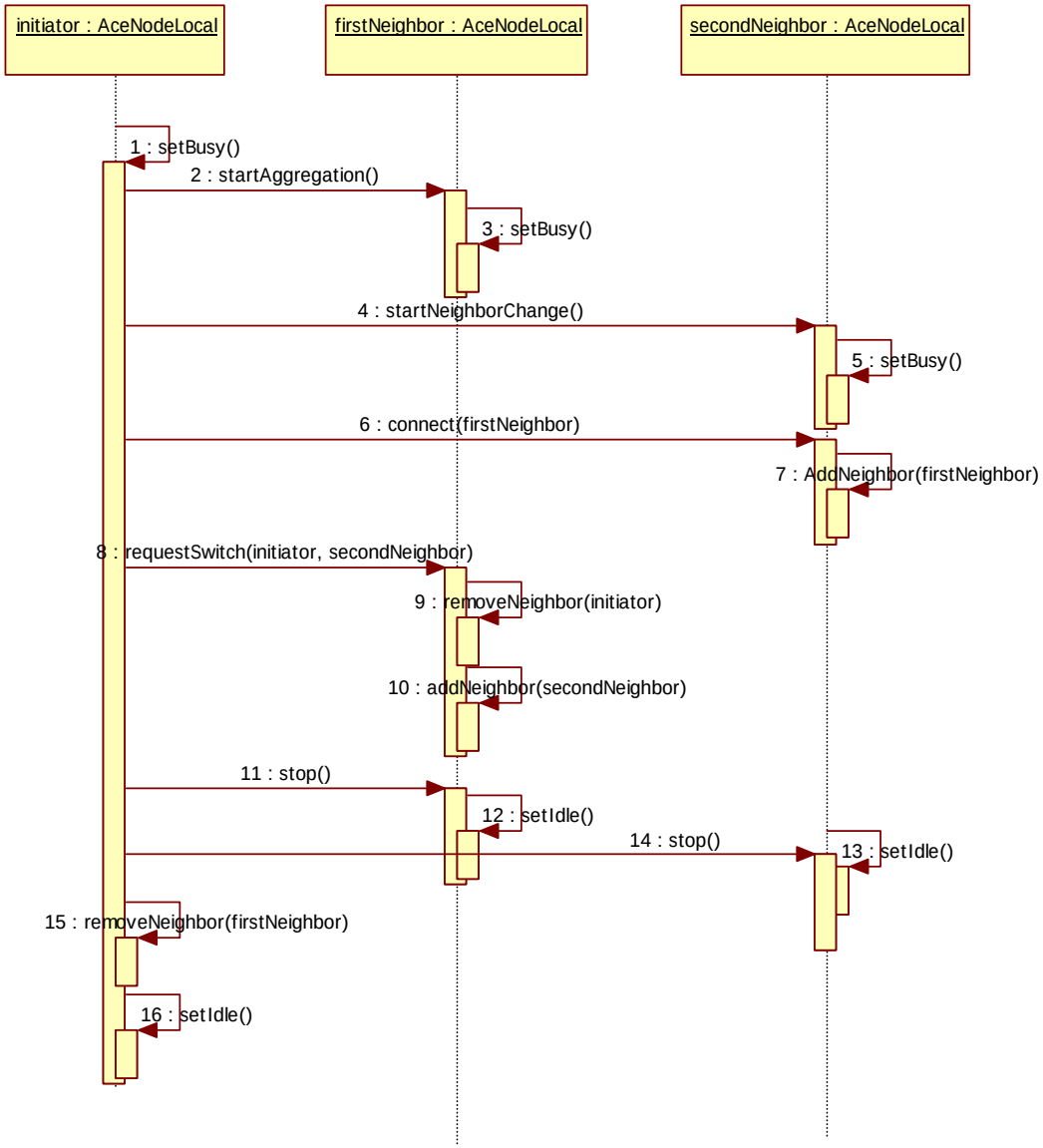

Figure 21. Passive (reverse) clustering algorithm.

Figure 22 describes the steps to be followed to perform the active aggregation algorithm. In this case, the initiator looks for a matchmaker among its neighbor. This identifies, among its neighbors, a candidate for aggregation and then triggers the aggregation mechanism. In our implementation, only a neighbor that doesn't belong to the right type (i.e., different from initiator in clustering and same as initiator in reverse clustering) can be selected as matchmaker; at present, the extension to different cases is under study.

More in detail, the steps being executed are the following:

1. The **initiator** chooses one of its neighbors among those having a type that is not compatible with its own type. This (in)compatibility relationship depends on the aggregation approach (clustering or reverse clustering) being executed. If, for instance, reverse clustering is executed, then any neighbor having the same type fo the initiator is not compatible with it. **initiator** elects this node as **matchMaker** and sends to it a *matchRequest().* If the node accepts to be a matchmaker, it sets itself to busy.
2. The **matchMaker** identifies a new **neighbor** to connect to the **initiator**. Then it sends both to the **initiator** and the other **neighbor** a *startNeighborChange()* request. If the two nodes are currently idle, they set themselves to busy and signal their ability to participate to the switch.
3. **matchMaker** sends a *connect()* to the **neighbor** that executes an *addNeighbor(**iniator**).* **matchMaker** waits for the response from the **neighbor** node.
4. The **matchMaker** sends a *requestSwitch()* to the **initiator**. The **initiator** executes the methods *removeNeighbor(**matchMaker**)* and *addNeighbor(**neighbor**).*
5. Finally, the **matchMaker** sends a *stop()* request to the other parties and then it removes the initiator from its list of neighbor.

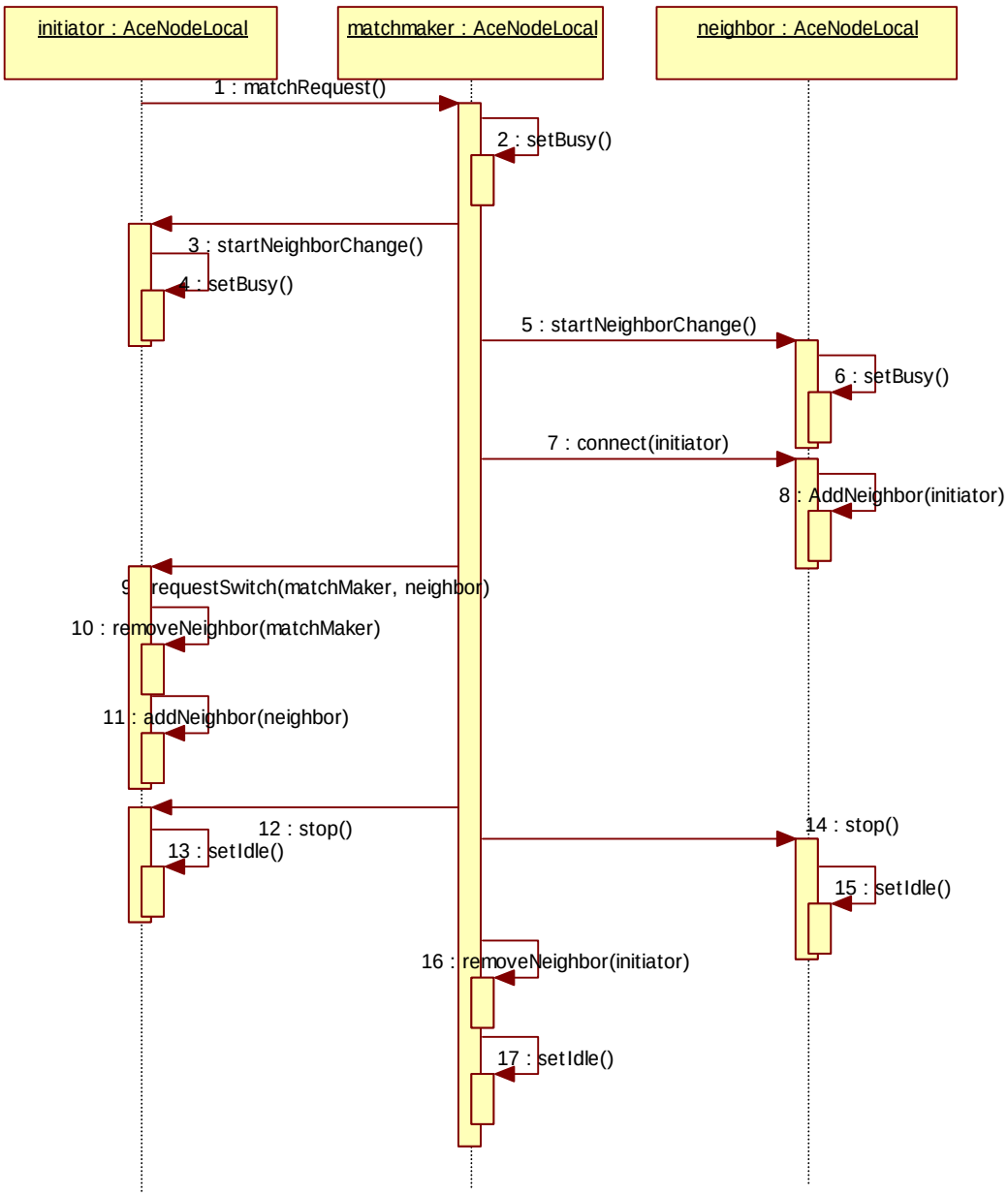

Figure 22. Active (reverse) clustering algorithm.

5.3.3 Configurator component

The main steps performed by the Configurator component are:

- The instantiation of the MessageBroker.
- The creation of the initial population of ACEs and their connection to the middleware

- The start up of the clustering algorithm execution.

The diagram in Figure 23 describes the main classes belonging to the Configurator (except for the ACE component that is shown only for clarity reasons).

AceConfigurator is the core of the component: it exploits the information within AceTopology (and its subclasses) to define the topology of the system. During its lifecycle, each ACE stores the real connections between the ACE nodes as a list of neighbors.

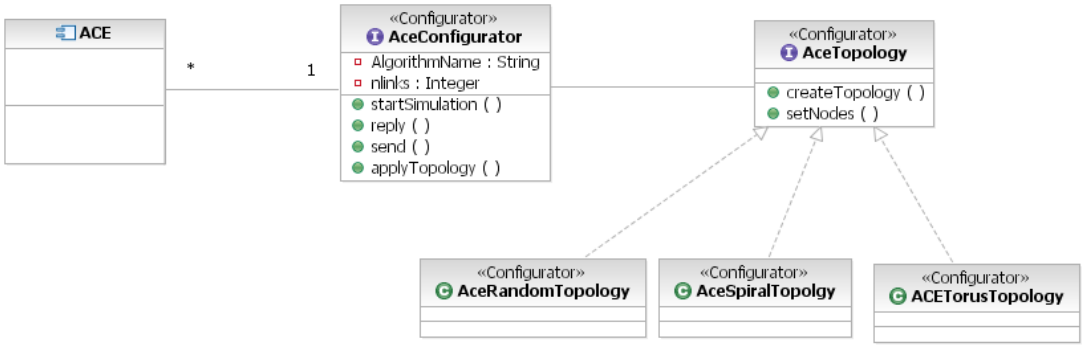

Figure 23. Main elements of the Configurator component.

5.4 REDS

In this section we provide an overview of the REDS middleware we exploit to support event-based communication [3, 4].

REDS (**RE**configurable **D**ispatching **S**ystem) is a framework of Java classes to build publish-subscribe applications for large, *dynamic* networks.

Distributed publish-subscribe applications are organized as a collection of components, which interact by *publishing* messages and by *subscribing* to the classes of messages they are interested in. Publish-subscribe applications are built around a publish-subscribe middleware, which provides a *dispatcher*, responsible for collecting subscriptions and forwarding messages from publishers to subscribers, and a library to access the services of the dispatcher and implement application components.

REDS provides the client API to write publish-subscribe applications and defines a general framework of components, with clearly defined interfaces, to build a distributed dispatcher organized as a set of *brokers* linked in an *overlay dispatching network*, which collaborate to route messages from publishers to subscribers.

With respect to other publish-subscribe middleware REDS provides several innovations:

- Its modular architecture allows system integrators and middleware programmers to easily adapt REDS to their needs, e.g., by defining their own format for messages and filters, the mechanisms used internally by each broker to match and forward messages, and the strategy for routing messages, just to mention the most relevant aspects.

- REDS is the first publish-subscribe middleware expressly designed to support arbitrary topological reconfigurations of the message dispatching network. Each REDS broker includes two modules, the TopologyManager and the Reconfigurator, which embed the logic used to manage run-time reconfiguration of the dispatching network, either to react to changes in the underlying phisical network or to adapt it to the application's needs, e.g., to balance the traffic load, or to change the number of brokers and their connectivity.

- Finally, REDS natively supports replies to messages, thus naturally providing bidirectional communication within the framework of content-based publish-subscribe communication. REDS brokers keep track of the transit of messages tagged as Repliable and store routes followed back by replies. By supporting replies natively, REDS brokers are able to track the number of expected replies for each message and check if and when all of them have been received---a feature that cannot be obtained by implementing replies at the application level.

5.4.1 Reds in a nutshell

Components of an application built using REDS access the publish-subscribe services provided by REDS through an object that implement the DispatchingService interface. It

provides methods to subscribe and unsubscribe to classes of messages, to publish messages, to receive subscribed messages, and to reply to them. REDS provides several implementations of the DispatchingService interface, which differ for the protocol used to access the REDS dispatcher. Developers using REDS may define their own format of messages and filters (used to define the classes of messages application components want to subscribe to) by extending the Message class and implementing the Filter interface. Otherwise, they can use the messages and filters already provided with REDS.

The REDS dispatcher is organized as a set of brokers linked together in an *overlay dispatching network*. Each REDS broker is internally structured in a set of modules grouped in two layers: the *Overlay* and the *Routing* layer. The former manages the overlay network that connects brokers, while the latter is in charge of routing messages on top of such overlay. Figure 24 shows the internal structure of a REDS broker.

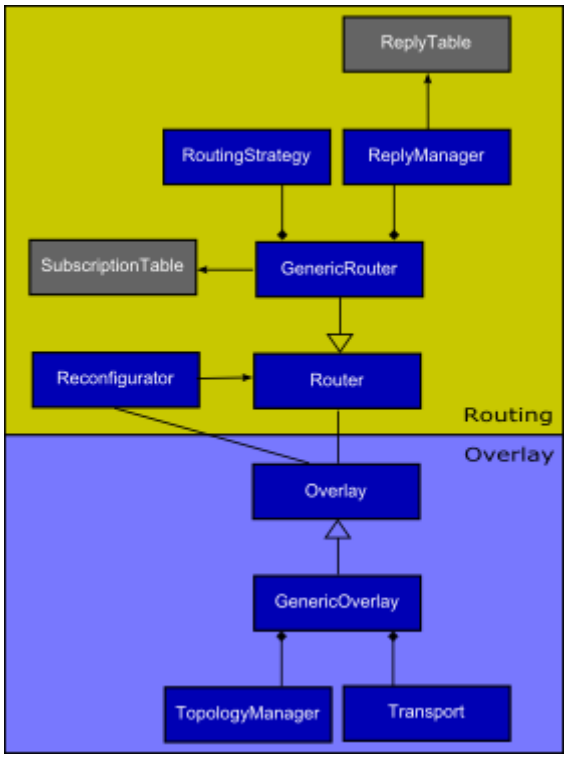

Figure 24: the internal structure of a REDS broker

The goal of the **Overlay** layer is to implement the mechanisms to manage the overlay network that connects REDS brokers and to exchange messages on top of it. In particular, it embeds the protocols that keep the overlay network connected when the topology of the underlying network changes (e.g., because some peers leave a peer-to-peer network or because of mobility in MANETs).

The **Routing** level offers the very publish-subscribe functionalities: it stores clients' subscriptions and uses the Overlay to route messages from publishers toward subscribers.

Programmers using REDS may adapt the behavior of the system to their needs by implementing new versions of the REDs components in case the available implementation does not fit their specific needs.

As an example, one could define a new component which realizes a new routing strategy for messages and subscriptions. If correctly implemented, such component can be combined with existing ones to implement a fully functional, innovative dispatching network with a minimal effort. (Details on REDS can be found in [3,4]).

5.5 Performance analysis: preliminary results

In this section we show the first results we have obtained using our prototype to execute the ACE self-aggregation algorithms. Specifically, we illustrate some simple performance results for the clustering (and reverse clustering) algorithms, by considering both active and passive clustering and different topologies (Random, Torus, Spiral).

All results we have collected so far refers to a configuration with 100 nodes, 1000 links and 5 different node types (colors).

To give an idea of the algorithm performance we have collected information about the number of exchanged messages (to measure the traffic over the network) and the level of node *homogeneity* achieved by the algorithms. By level of homogeneity we mean the percentage of nodes having neighbors of the same type. Clearly, a high level of homogeneity is our goal in case we are enacting a clustering approach. In case we refer to a reverse clustering mechanism, then we would need to tend to keep the level of homogeneity very low.

All measures have been taken assuming specific times for the termination of the aggregation (see 3rd column in all the following tables).

More precisely, for the clustering algorithms we have observed results close to an asymptote when the simulation reaches the 100 seconds. For this reason, we show the obtained results for time intervals of 25 seconds starting from 0 second and ending at 100 seconds.

For the reverse clustering algorithms we have observed quite stable results also ending the simulation at 20 seconds. On this basis, we show the obtained results starting from 0 second and ending at 20 seconds with time intervals of 5 seconds only.

At present, a more thorough analysis is in progress to give statistical soundness and validity to the results. To this end we are trying to set up a measurement framework where the experiments are performed in a controlled way, so to take into account the variables that affect the results (e.g., the number of nodes and link and for the number of different node types). Besides, the experiments should be replicated so that a set of statistical operations (e.g., measures of central tendency and data distribution) can be done on collected data. The obtained results can then be used to decide which algorithm is most appropriate for addressing a particular problem or situation.

5.5.1 Passive clustering

Topology	Messages (10^3)	Time (s)	Homogeneity
Random	0	0	19,40%
Random	19	25	55,80%
Random	36	50	67,40%
Random	52	75	69,90%
Random	66	100	72,90%
Torus	0	0	17,10%
Torus	19	25	35,30%
Torus	36	50	47,00%
Torus	51	75	53,80%
Torus	68	100	63,80%
Spiral	0	0	17,90%
Spiral	19	25	32,60%
Spiral	34	50	42,80%
Spiral	51	75	49,10%
Spiral	67	100	53,90%

Table 2 Passive clustering.

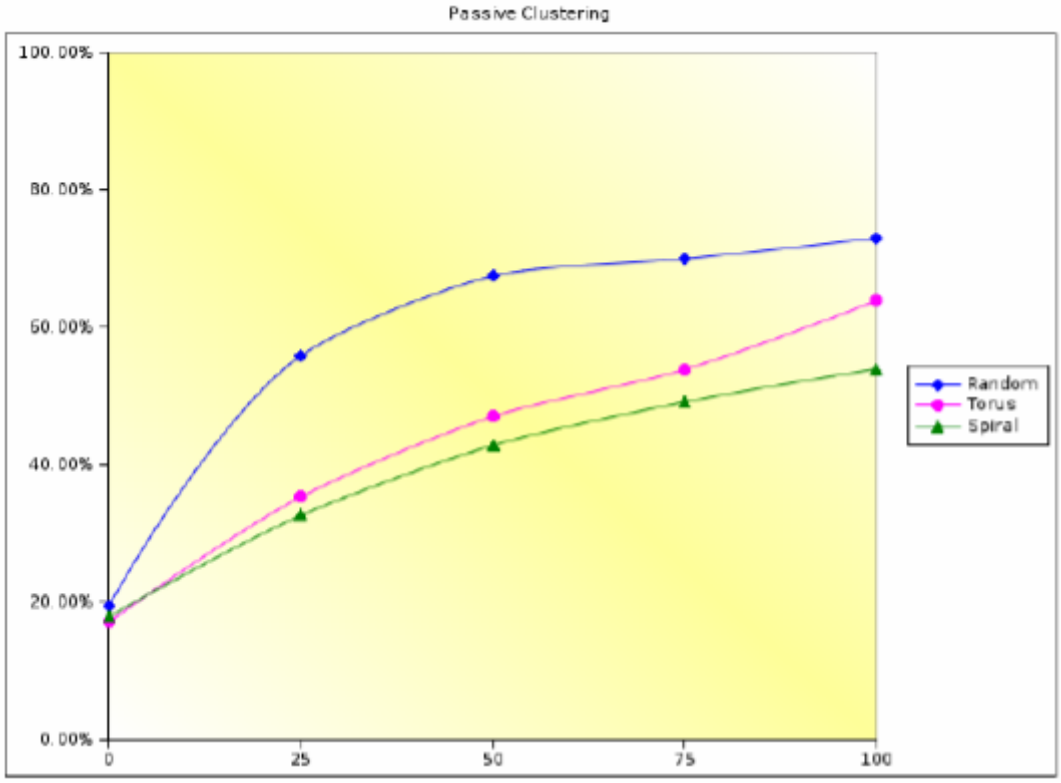

Figure 25. Passive Clustering: Homegeneity level vs algorithm completion time.

The results in table 2 and Figure 25 seem to indicate that the RANDOM topology is the best suited to achieve a high level of homogeneity (goal of the clustering) with a substantial equivalence with the other topologies in the number of exchanged messages.

5.5.2 Active clustering

Topology	Messages (10^3)	Time (s)	Homogeneity
Random	0	0	21,40%
Random	19	25	66,10%
Random	35	50	70,60%
Random	50	75	71,00%
Random	64	100	72,40%
Torus	0	0	17,70%
Torus	21	25	30,90%
Torus	37	50	34,10%
Torus	55	75	35,90%
Torus	72	100	36,40%
Spiral	0	0	18,30%
Spiral	20	25	30,30%
Spiral	36	50	33,50%
Spiral	55	75	34,50%
Spiral	72	100	35,30%

Table 3. Active clustering.

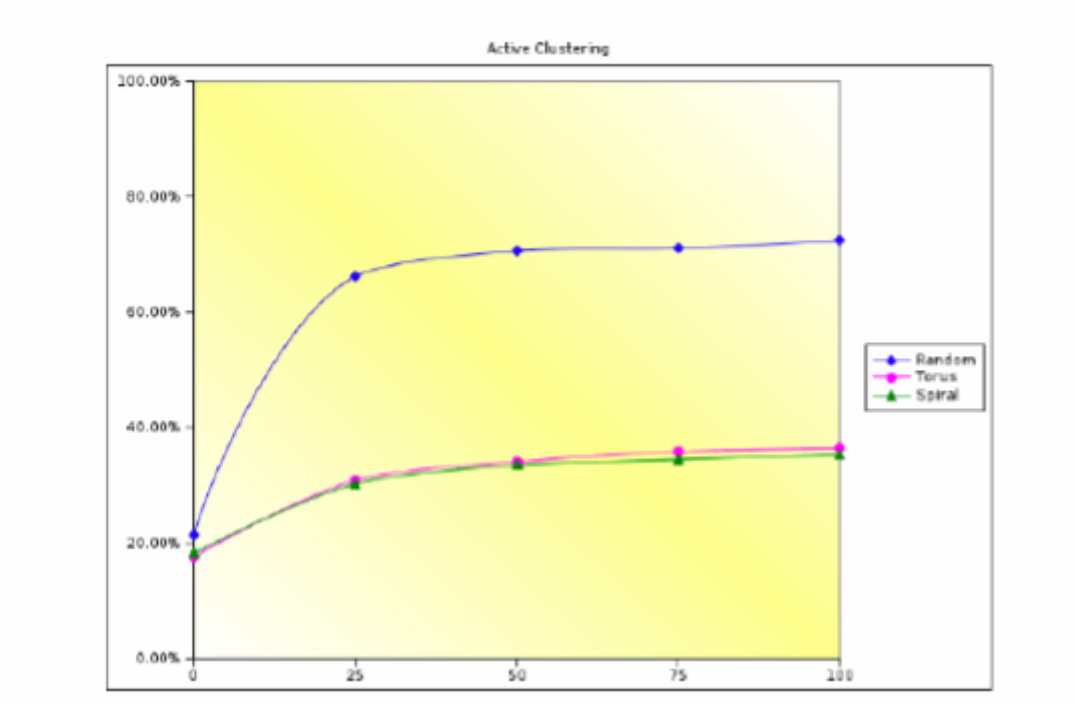

Figure 26. Active Clustering: Homegeneity level vs algorithm completion time.

In this case we observe again a superiority of the RANDOM topology with respect to the others. In particular, the RANDOM topology for active clustering offers results that are comparable with the ones obtained for the Passive Clustering. Instead the Torus and Spiral topologies reach for active clustering a lower level of homogeneity with respect to the case of passive clustering.

5.5.3 Passive Reverse Clustering

Topology	Messages (10^3)	Time (s)	Homogeneity
Random	0,0	0	18,50%
Random	2,9	1	17,40%
Random	5,6	5	16,50%
Random	9,2	10	15,00%
Random	15,8	20	8,60%
Torus	0,0	0	18,10%
Torus	2,8	1	17,80%
Torus	5,4	5	16,60%
Torus	9,0	10	15,50%
Torus	15,8	20	10,50%
Spiral	0,0	0	19,20%
Spiral	2,9	1	18,80%
Spiral	5,3	5	15,90%
Spiral	8,9	10	15,00%
Spiral	15,6	20	13,70%

Table 4. Passive reverse clustering.

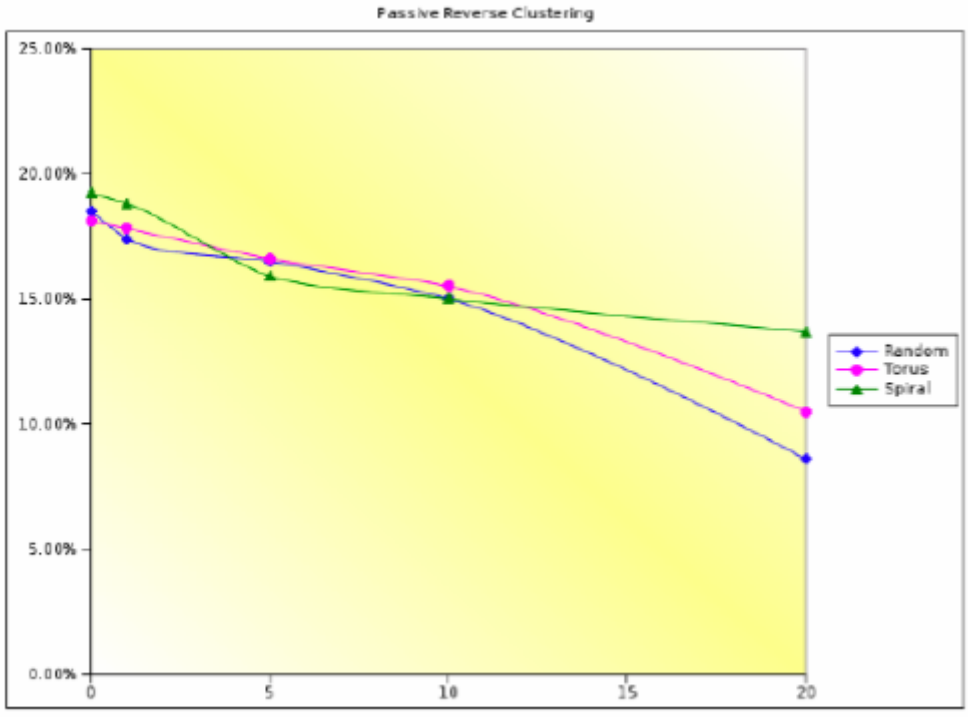

Figure 27. Passive Reverse Clustering: Homegeneity level vs algorithm completion time.

With this kind of clustering, the three different topologies present similar performance in terms of both homogeneity level (in this case it should be as low as possible) and number of exchanged messages with a slight dominance of RANDOM topology.

5.5.4 Active Reverse Clustering

Topology	Messages (10^3)	Time (s)	Homogeneity
Random	0,0	0	19,40%
Random	3,0	1	15,30%
Random	5,1	5	2,50%
Random	5,8	10	0,00%
Random	6,5	20	0,00%
Torus	0,0	0	18,30%
Torus	2,9	1	16,70%
Torus	5,6	5	12,10%
Torus	8,9	10	8,20%
Torus	11,8	20	0,60%
Spiral	0,0	0	17,80%
Spiral	2,9	1	16,60%
Spiral	5,7	5	12,30%
Spiral	9,2	10	10,00%
Spiral	13,1	20	4,10%

Table 5. Active reverse clustering.

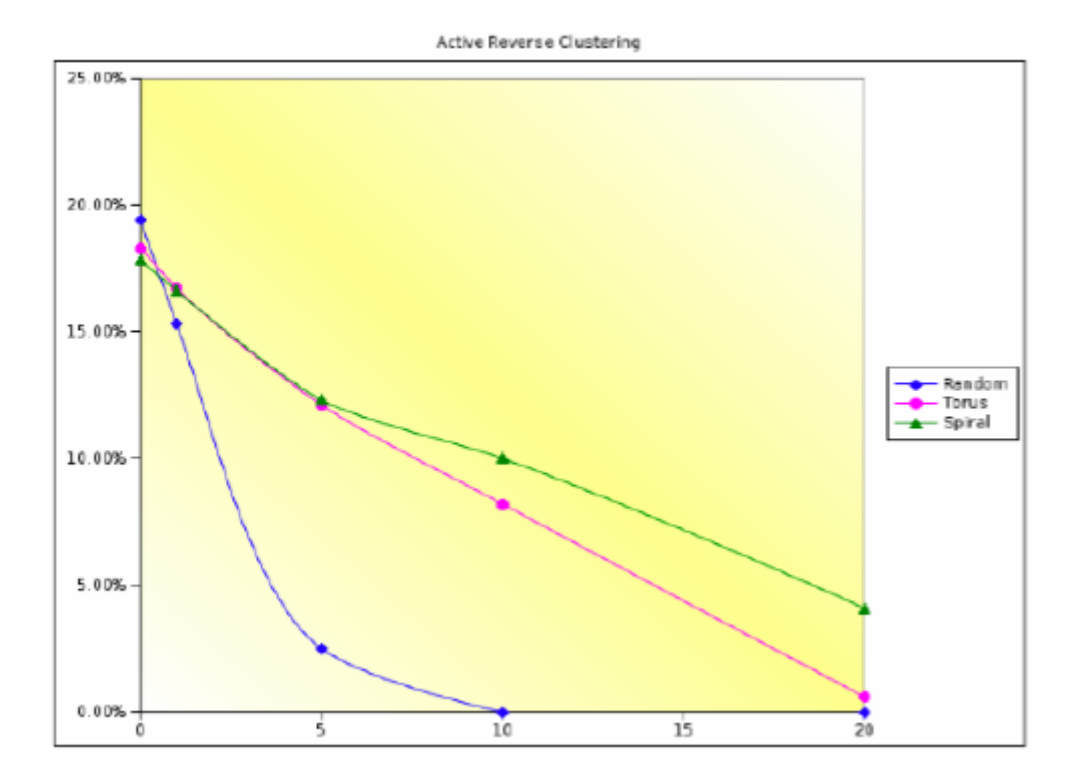

Figure 28. Active Reverse Clustering: Homegeneity level vs algorithm completion.

With this algorithm we observe a complete convergence (0% of homogeneity level) reached with the RANDOM topology in a very short time (10 seconds). Torus and spiral topologies exhibit good performances as well (even if not optimal), which confirm a superiority of the active versus the passive version of the algorithm.

5.5.5 Findings

By summarizing, we can roughly say that for clustering algorithms the RANDOM topology achieves a good level of homogeneity both with active and passive version, while torus and spiral topologies show lower performances that slightly decrease in the case of active clustering.

For reverse clustering algorithms, instead, it is possible to clearly indicate the RANDOM topology and the active version of the algorithms as the combination that shows the best performance: complete convergence with a low number of messages and with a short completion time.

Moreover, for each algorithm we have also shown, for each completion time, the number of exchanged messages that roughly measures the traffic over the network. In such a way the selection of the aggregation algorithm can be performed basing not only on the reached homogeneity level but also on the cost paid in terms of generated network traffic.

6 Conclusions

This first deliverable presents the first year's most significant results with respect to one of the high-level objectives of WP3: to investigate the suitability of specific algorithms for the self-aggregation of Autonomic Communication Elements (ACEs), an enabler for self-organized service creation, deployment, regulation and life-cycle management. We followed a two-tier approach, using modeling and simulation to investigate fundamental systemic properties (sections 2-3) and their sensitivity to the most relevant source of perturbation ("selfish" decision-making, section 4), then developing a prototype implementation of the preferred rule-set using a pre-existing middleware framework (proof-of-concept, section 5).

We found that even extremely simple candidate local rules had "hidden" properties susceptible to lead to inoperable global configuration, supporting our initial hypothesis that a principled study of system properties has to be an integral part of the process of engineering the behavioural repertoire of individual components. Our main conclusion is that, if the objective is to design a set of local rules capable of scaling up to a large, fully decentralised population of such components, failing to conduct such a preliminary, model- or simulation-based investigation can have catastrophic consequences, even when small-scale experimental deployment revealed no unwanted properties.

Our results lead us to recommend using the "on-demand" clustering algorithm whenever constraints dictate that only strictly local messaging between first neighbours (i.e. information transfer without any dedicated messaging infrastructure - either centralized or distributed - or forwarding capability) is available. This will of course not always be the case, but can be regarded as the most challenging scenario for a rule-set relying on self-organization to promote the emergence of the desired system configuration (due to the strict locality of information and absence of explicit long-range interactions) and so constitutes the ultimate test of robustness.

In practice, this means that whenever operating under these extreme conditions, an ACE following the "on-demand" clustering algorithm would initiate a rewiring procedure as soon as it detects a discrepancy between its "Goal Needed" (GN) and "Goal Achievable" (GA) lists of required/available functionalities. This situation can result from many different events like, e.g., the breaking of an existing collaborative link (GA $\rightarrow$ GN), the submission of a new type of request (additional GN), a change in the local load (GA $\rightarrow$ GN, due to a surge in demand leading to the current collaborative relationships being no longer able to absorb the corresponding workload)…

Depending on the circumstances, the initiator can choose one or more of its first neighbours (ACEs with which it has an existing relationship) as (a) match-maker(s), and the constraint on the conservation of the total number of links can be relaxed or not. But fundamentally, our work demonstrates that successful self-organization would take place, at a predictable rate, provided that well-identified conditions are met (most importantly in

terms of the diversity of "goals types", which must be low compared to population size and of the same order of magnitude as the average node degree).

In the course of this work, we also found that some aspects of the self-aggregation process were more closely related to the activities of WP4 than originally envisaged. This is because "selfish" decisions motivated by the nodes' supposed goal of maximizing their own efficiency can directly impact on system dynamics by interfering with the normal execution of the clustering algorithm(s).

We have investigated methods to quantify the impact of such a "cheating" behaviour and applied other techniques to determine what topological reorganization would likely occur in a population of "selfish" nodes. Whether "selfish" and/or "rational" decision-making can play a constructive part in the self-organization process (as it does in natural ecosystems and markets) or must somehow be kept under control by appropriate disincentives has still to be determined and will be part of future work.

[RICHARD, PAUL, PIETRO – PLEASE MODIFY /APPEND THE LAST TWO PARAGRAPHS AS YOU SEE FIT]

The objective of the prototype implementation was two-fold: (1) to verify that the abstract algorithms used in simulation could actually be transposed into a functional interaction framework applicable to a real distributed system, and (2) to facilitate integration of WP3's outputs into ACE design by combining the terminology of WP3 and WP1 into a coherent whole, thus providing a common reference.

The successful implementation of the various rule-sets demonstrated that translating the abstract models into concrete sets of interaction patterns was eminently feasible. By investigating multiple starting topologies (random, spiral, torus), our experiments helped to quantify some non-intuitive advantages of the random graph. Finally, the prototype was also used to produce a first batch of results relating to the "messaging cost" and convergence time associated with the various clustering algorithms

References

Sections 2 and 3:

Aberer, K., Datta, A. and Hauswirth, M. 2005 *P-Grid: Dynamics of Self-organizing Processes* in Structured P2P Systems in: Peer-to-Peer Systems and Applications, Lecture Notes in Computer Science vol 3485 Springer

Adam, C. and Stadler, R. 2006 *Implementation and Evaluation of a Middleware for Self-Organizing Decentralized Web Services* Second IEEE International Workshop on Self-Managed Networks, Systems & Services (SelfMan 2006), June 2006,Dublin, Ireland.

Babaoglu, O., Jelasity, M., Montresor, A., Fetzer, C., Leonardi, S., van Moorsel, A. and van Steen, M. (2005) *Self-Star Properties in Complex Information Systems*, Lecture Notes in Computer Science, Hot Topics, vol 3460, Springer

Barabasi A. L., Albert, R. and Jeong, H. (1999) *Mean-Field Theory for Scale-Free Random Networks*. Physica A 272, 173-187.

Camazine S., Deneubourg, J. L., Franks, N. R., Sneyd, J. Theraulaz, G. And Bonabeau, E. (2001) *Self-Organization in Biological Systems*. Princeton University Press.

Gelenbe, E., Liu, P. and Lainé, J. (2006) *Genetic Algorithms for Autonomic Route Discovery* in Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and its Applications (DIS'06)

Jelasity M., Babaoglu, O. (2005) *T-Man: Gossip-based Overlay Topology Management*. Proceedings of the 3rd International Workshop on Engineering Self-Organising Applications.

Jesi, G. P., Montresor, A. and Babaoglu, O. "Proximity-aware Superpeer Overlay Topologies" Second IEEE International Workshop on Self-Managed Networks, Systems & Services (SelfMan 2006), June 2006,Dublin, Ireland.

Kephart, J. O. and Chess, D. M. (2003) *The Vision of Autonomic Computing* IEEE Computer

Manzalini A. and Zambonelli, F. (2006) *Towards Autonomic and Situation-Aware Communication Services: the CASCADAS vision*. IEEE Workshop on Distributed Intelligent Systems, Prague, June 2006.

Marrow, P. and Manzalini, A. (2006) *The CASCADAS Project: A Vision of Autonomic Self-organizing Component-ware for ICT Services* International Transactions on Systems Science and Applications Vol. 2 No. 3 pp 303 -308

Montresor, A., Heing, M. and Babaoglu, O. (2002) *Messor: Load-Balancing through a Swarm of Autonomous Agents* in Proceedings of the 1st International Workshop on Agents and Peer-to-Peer Computing, Bologna, Italy, July 2002.

Nakrani, S. and Tovey, C. 2004 *On honey bees and dynamic server allocation in internet hosting centers* Adaptive Behavior, Vol. 12, No. 3-4 pp 223 - 240

Saffre F. and Ghanea-Hercock, R. (2003). *Simple Laws for Complex Networks*. BT Technol. J. 21:2, 112-119.

Saffre, F., Jovanovic H., Hoile, C. And Nicolas, S. (2006a) *Scale-Free Topology for Pervasive Networks*. In: Intelligent Spaces, The Application of Pervasive ICT, Steventon, A. and Wright, S. Eds. Springer-Verlag.

Saffre F., Halloy, J and Deneubourg J. L. (2006b) *Steering and Evaluating Autonomic Deployment of Service Components in a P2P Network*. Int. Trans. Syst. Sc. App. 2:3, 315-318.

Saffre F., Halloy, J., Shackleton, M. and Deneubourg J. L. (2007) *Self-Organised Service Orchestration through Collective Differentiation*. To appear in IEEE Transactions on Systems, Man and Cybernetics-Part B (in press).

Stal, M (2002) *Web services: beyond component-based computing* Communications of the ACM, vol 45, no. 10

Section 4:

[1] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *Proc.ofSODA*, 1997.
[2] A. Avidor, Y. Azar, and J. Sgall. Ancient and new algorithms for load balancing in the *lp*norm. *Algorithmica*, 29:422–441, 2001.
[3] A. Awerbuch, A. Azar, E. F. Grove, P. Krishnan, M. Y. Kao, and J. S. Vitter. Load balancing in the *lp*norm. In *ProcofIEEEFOCS*, 1995.
[4] B-G. Chun, R. Fonseca, I. Stoica, and J. Kubiatowicz. Characterizing selfishly constructed overlay routing networks. In *Proc.IEEEINFO-COM*, 2004.
[5] J. Corbo and D. Parkes. The price of selfish behavior in bilateral network formation. In *Proc.ACMPODC*, 2005.
[6] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and

S. Shenker. On a network creation game. In *Proc.ofACMPODC*, 2003.

[7] E. Koutsoupias and C. H. Papadimitriou. Worst-case equilibria. *LectureNotesinComputerScience*, 1563:404–413, 1999.

[8] D. C. Parkes and J. Shneidman. Distributed implementations of vickrey-clarke-groves mechanisms. In *Proc.ofAAMAS*, 2004.

[9] S. Philips and J. Westbrook. Online load balancing and network flow. In *Proc.ofACMSTOC*, 1993.

[10] B. Raman and R. H. Katz. Load balancing and stability issues in algorithms for service composition. In *Proc.ofIEEEINFOCOM*, 2003.

[11] A. Sureka and P. R. Wurman. Using tabu best-response search to find pure strategy nash equilibra in normal form games. In *Proc.ofACMAAMAS*, 2005.

[12] S. Suri, C. D. Toth, and Y. Zhou. Selfish load balancing and atomic congestion games. In *Proc.ofSPAA*, 2004.

[13] S. Suri, C. D. Toth, and Y. Zhou. Uncoordinated load balancing and congestion games in p2p systems. In *Proc.ofHOT-P2P*, 2004.

Section 5:

[1] Cascadas description of work on line at: https://cascadas-project.org:444/repositories/cascadas/trunk/Technical_Annex_and_CPF/CASCADAS_DoW.pdf

[2] Cascadas WP1 Deliverable 1.1 v.0.1 on line at https://cascadas-project.org:444/repositories/cascadas/trunk/wp1/deliverables/M12/Del1.1_v01.doc

[3] "REDS: A Reconfigurable Dispatching System" G. Cugola, G. Picco, In *Proceedings of the 6th International Workshop on Software Engineering and Middleware (SEM 2006)*, Portland (OR, USA), November 2006.

[4] REDS web page: http://zeus.elet.polimi.it/reds/