# Self-organising agent communities for autonomic resource management

Mariusz Jacyno*
Institute of Control and Computational Engineering,
University of Zielona Góra, 65-246 Zielona Góra, Poland

Seth Bullock
School of Electronics & Computer Science,
University of Southampton, SO17 1BJ, UK

Nicholas Geard
Melbourne School of Population Health,
University of Melbourne, Victoria 3010 Australia

Terry R. Payne
Department of Computer Science,
University of Liverpool, L69 3BX, UK

Michael Luck
Department of Informatics,
King's College London, WC2R 2LS, UK

August 20, 2012

*Corresponding author: Mariusz Jacyno, Tel: +48683282422, Fax: +446832484751, m.jacyno@issi.uz.zgora.pl, University of Zielona Góra ul. Prof. Z. Szafrana 2 65-246 Zielona Góra.

Running title: "Self-organising communities"

**Abstract**

The autonomic computing paradigm addresses the operational challenges presented by increasingly complex software systems by proposing that they be composed of many autonomous components, each responsible for the run-time reconfiguration of its own dedicated hardware and software components. Consequently, regulation of the whole software system becomes an emergent property of local adaptation and learning carried out by these autonomous system elements. Designing appropriate local adaptation policies for the components of such systems remains a major challenge. This is particularly true where the system's scale and dynamism compromise the efficiency of a central executive and/or prevent components from pooling information to achieve a shared, accurate evidence base for their negotiations and decisions.

In this paper, we investigate how a self-regulatory system response may arise spontaneously from local interactions between autonomic system elements tasked with adaptively consuming/providing computational resources or services when the demand for such resources is continually changing. We demonstrate that system performance is *not* maximised when all system components are able to freely share information with one another. Rather, maximum efficiency is achieved when individual components have only limited knowledge of their peers. Under these conditions, the system self-organises into appropriate community structures. By maintaining information flow at the level of communities, the system is able to remain stable enough to efficiently satisfy service demand in resource-limited environments, and thus minimise any unnecessary reconfiguration whilst remaining sufficiently adaptive to be able to reconfigure when service demand changes.

Keywords: Autonomic computing, networks, self-organisation, community structure, decentralised control, emergence.

# 1 Introduction

Modern software systems are among the most complex human artefacts that have been developed to date (Bullock and Cliff, 2004; Gershenson and Heylighen, 2005). Contemporary information systems depend on an increasing number of software modules, processing platforms, data sources, network connections, and input/output devices, to the extent that it is becoming impossible to predict or control their interactions. Both the scale and interconnectivity of IT systems are increasing as a consequence of their evolution, from stand-alone machines to systems of systems (comprising huge numbers of distributed and interacting components) working to provide resources on demand to a large number of users (Tanenbaum and Renesse, 1985; Forrest et al., 2005). Although modular architectures offer opportunities to tackle increasing system complexity by decomposing the overall system into specialised components, they also present challenges in terms of the maintenance of reliable and predictable operation when system objectives and structure are subject to exogenous and unpredictable change.

Consequently, it is not surprising that, several years ago, IBM released a manifesto[1] arguing that the main obstacle to further progress in the IT industry was a looming complexity crisis that would prevent reliable and cost-effective human administration of large-scale IT systems. In response, an *autonomic computing* approach was proposed, where IT systems would be capable of automatically regulating their own function (Kephart and Chess, 2003). Large-scale autonomic computing systems were expected to comprise myriads of computational elements, each acting, learning or evolving autonomously in response to interactions with other components in their local environment (Zambonelli and Parunak, 2001). System-level self-regulation would then arise as a product of these local adaptations and interactions between system elements, thereby reducing the need for manual management and control.

While the autonomic computing paradigm offered an alternative perspective on large-scale computational systems, it also presented novel challenges in understanding and managing their performance. Large-scale IT systems can easily become too complex to be managed through a 'divide and rule' analysis (Bonabeau, 2002). When a system's global behaviour results from highly non-linear interactions between many system components, the relationship between the actions of individual autonomous agents and the system level consequences of those actions can become opaque and extremely

[1] `http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf`

sensitive (Heylighen, 1991; Hogg and Huberman, 2002; Zambonelli et al., 2002). In addition to the issue of scale, autonomic systems are necessarily *open*, in that they are free to gain and lose components over time in an environment that is dynamic and unpredictable in terms of user demand, and where few *a priori* assumptions can be made (De Roure, 2003; Estrin et al., 2002; Keil and Goldin, 2004). In general, managing such systems is perhaps more akin to regulating financial markets or managing a firm, rather than supervising a local area network or traditional computer cluster.

In this paper we address the central real-world problem confronted by the autonomic computing paradigm: ensuring that large numbers of computational resources are efficiently configured and allocated in order to satisfy user demand that varies in both kind and quantity. We do so by building and exploring an agent-based model at a reasonably large scale (hundreds of interacting agents). We respect some key physical realities, e.g., interactions between components take real time and any information available to system components must be gathered, stored, and processed explicitly. However, we choose not to model a specific real-world case or application. Rather we present a relatively simple, generic model encapsulating a range of properties shared across information-driven autonomic systems consisting of a large number of autonomous and locally communicating elements. In doing so we focus on the role of information flows in maintaining efficient performance. As such, the model is intended to have heuristic value for the science and engineering of large-scale complex IT systems. In taking this approach we are adopting a well-established and explicitly recognised scientific modelling strategy in which realism is sacrificed to some extent in return for a combination of generality and precision (Levins, 1966).

More specifically, we investigate how an autonomic system's regulatory response can arise from local interactions between its elements. We present a decentralised multi-agent system in which provider agents are tasked with adaptively configuring the services that they offer, and efficiently allocating these services to the consumer agents that require them. The co-adaptive interaction between these two groups of agents is governed by simple threshold reinforcement mechanisms and localised exchange of information between each agent and its local peers. An evaluation of this approach demonstrates that when peer neighbourhoods are an appropriate size, they may exhibit self-organising behaviour resulting in the emergence of spontaneous community structures that support the requirements of their members. What distinguishes this regime is not just the *extent* to which information flows amongst system components, but the underlying *organisation* of these components and their interactions. This organisation can be described in terms

of an emergent and dynamic topology of interaction that regulates and constrains the flow of information but is simultaneously brought about by this flow. Our main contribution is to show that it is only when the right topology is achieved and maintained (perhaps requiring continuous or periodic reorganisation) that system elements exert appropriate constraints on information flow such that efficient, ongoing system performance can be achieved.

The paper is organised as follows: a description of the related work on autonomic computing, multi-agent systems and self-organising systems is presented in Section 2, which also provides the motivation and inspiration for the design of the resultant model (Section 3). The model is then evaluated in Section 4, and the strengths, limitations and implications of this model are discussed. The paper concludes in Section 5.

# 2 Related Work

## 2.1 Autonomic Computing

Autonomic systems are dynamic and complex; not only may their workflow characteristics undergo change, but the business processes that they support will be continually evolving. Such systems require administration at multiple levels and on a continual basis in order to preserve their correct operation. At a low level, dynamically changing user demand must be met through continual allocation and reallocation of computational resources. At a higher level, reconfiguration of workflows, business processes, and hardware facilities is also required. For this reason, the reconfiguration of system components and their functions is unavoidable and requires a flexible approach that maintains system efficiency despite dynamic conditions. As suggested by IBM in their manifesto, as these systems increase in scale, managing them through skilled human administration becomes impractical and autonomic solutions are required.

Much of the current research in the area of autonomic computing has addressed the problem of preserving the interoperation of existing IT systems' software modules, often encapsulating their functions in terms of autonomic managers. Techniques such as reinforcement learning (Kephart et al., 2007; Tesauro, 2007), optimal control theory (Wang et al., 2007), and maximisation of expected utility (Kephart and Das, 2007) are then exploited in order to balance power-performance trade-offs, i.e., to achieve efficient allocations of requested jobs at the same time as optimising the power consumption of unused servers.

Two kinds of control architecture tend to be employed: *centralised* and

*distributed*; both of which are illustrated schematically in Figure 1. Centralised schemes rely on a central executive to co-allocate services, schedule and plan system behaviour, etc. In contrast, distributed control schemes employ distributed protocols and focus on the design of intelligent parallel algorithms for coordinating the behaviour of agents. One example of a centralised system can be found in work by Tesauro (2007). Here, each application manager was responsible for the performance throttling of a managed server, and they shared expected utility information with a central resource arbiter. This central node then computed an optimal allocation of servers to the applications. A more distributed approach was adopted by Kephart et al. (2007) where both performance and power managers conducted their decisions based on full information about the state of parameters that define the actual functioning of individual servers.

Whilst these mechanisms are somewhat decentralised, they generally assume that up-to-date information is freely available. Thus, in order to converge on an optimal solution, such schemes require each agent to possess or have immediate access to a substantial amount of accurate and up-to-date global system information, resulting in the need for a large number of interactions in order to maintain awareness of peer goals, actions, etc. For example, although in Kephart et al. (2007) all resource allocating agents computed the optimal allocation (thus avoiding a single arbiter), they all had access to shared state information and the same environment variables indicating current system demand.

Although acting on shared global information allows autonomous system elements to make well-informed decisions in principle, it has been shown that in some cases it may lead to inefficiency and loss of stability. The El Farol Bar problem (Arthur, 1994) demonstrates that instability can emerge when independent, rational agents all have access to the same, global information. Arthur introduced the El Farol Bar problem as a game-theoretic example of the challenge facing rational agents that wish to organise themselves efficiently on the basis of shared information. In this game, every player wants to visit a specific bar only if less than 60% of the population also wish to visit the same bar — and they must make their decision without collaboration or negotiation. A rational agent that weighs up all of the evidence and decides, on balance, to visit the bar would also reasonably conclude that all other players will reach the same conclusion, since every other player is also a rational agent with access to the same information. In this case, the player should reverse its decision, as it does not want to attend a bar that is crowded with every other player. However, it should also reason that every other rational player would rationalise the same reversal, and so on.

Note that this problem can be refined in many ways and many different "solutions" to it can be sought (Challet and Zhang, 1997; Savit et al., 1999). Here we are not interested in solving the El Farol Bar problem, but rather using it as a reminder that shared information does not necessarily improve the co-ordination of decentralised decision making. In fact, the El Farol quandary rests on two symmetries: (i) every agent employs the same decision-making mechanism; and (ii) every agent reasons on the basis of the same information. Both these symmetries are typically true of the collaborative, decentralised agent systems used to explore coalition formation (Shehory and Kraus, 1998), group problem solving (Stone and Veloso, 2000) and teamwork (Pynadath and Tambe, 2002).

Collecting and processing the up-to-date information required by centralised or distributed control can also become a significant problem in large-scale deployments due to the time-delays associated with obtaining large amounts of distributed information. As a consequence, systems relying on either centralised or distributed control schemes (Figures 1a and 1b) are often vulnerable to increasing system scale and/or dynamism (Durfee, 2001).

[Figure 1 about here.]

## 2.2 Biological Analogies

While it is commonplace within multi-agent systems research to take an economic perspective on agent interactions, deriving "optimal" behavioural policies from game-theoretic considerations of strategic interaction between autonomous agents that suffer conflicts of interest, IBM's autonomic computing manifesto instead made reference to examples of natural homeostatic control that have evolved to maintain system equilibrium in biological organisations. Many impressive natural systems—for example, ecosystems (Kay, 1984), insect societies (Seeley, 2002) and biological organisms (Mesarovic et al., 2004)—have developed internal control mechanisms that allow them to organise and adapt to environmental change by relying on local interactions between the decentralised elements, rather than a more familiar control hierarchy culminating in some authoritative central executive.

For example, it is not the complex decision-making of individuals that preserves robust organisation of ant, wasp or bee colonies (Robinson, 1992; Gordon, 2002), but rather their ability to stimulate each individual's behaviour appropriately through local interactions (Ladley and Bullock, 2004, 2005; Bullock et al., in press). These, if effectively organised, facilitate flexi-

ble task allocation between colony members that is robust to environmental perturbations (Théraulaz et al., 1998). An example of such a process can be observed in an ant colony, where a potentially homogeneous population of ants, each capable of handling the same range of tasks, is differentiated into a number of distinct but organised collectives or *castes* (Bonabeau et al., 1997). Each such collective specialises in carrying out a specific task, such as food foraging, nest building, brood feeding, nest defence, etc. The survival of the colony thus depends on both the efficient handling of each system task and the adaptive division of resources (ants) into a number of such collectives responsible for these different tasks. One of the most striking aspects of such a regulatory response is its *plasticity*, a property achieved through the workers' behavioural flexibility: the ratios of workers performing the different tasks that maintain the colony's viability and reproductive success can vary (i.e., workers switch tasks) in response to internal perturbations or external challenges (Bonabeau et al., 1997). Understanding how this flexibility is implemented at the level of individual workers which certainly do not possess any global representation of the colony's needs has been addressed to some extent (Théraulaz et al., 1998; Bonabeau et al., 1997; Merkle and Middendorf, 2004). Self-regulatory colony properties appear to stem from simple threshold-based behaviours where specialisation of system elements to handle particular tasks arises as a result of reinforcement processes (Théraulaz et al., 1998).

Taking inspiration from these biological studies is attractive for several reasons. By contrast with the economic models that typically drive multi-agent systems, they foreground the ecological rationality of the agents within real-world systems (Bullock and Todd, 1999). Such agents are embedded in an environment that demands decisions be made locally, under pressure of limited time, limited computational resource, and on the basis of subjective information. These constraints place significant emphasis on the system's ability to adapt and cope rather than to achieve and maintain optimal performance, i.e., to "satisfice" (Simon, 1956) rather than to optimise.

Biologically inspired approaches also recognise the potential for system components to experience coincident, as well as conflicting, interests. For instance, like components of a data centre, cells in an organism, neurons in a brain, and termites in a colony share a common interest in achieving global system efficiency rather than conflicting interests in profit making at each others' expense. Moreover, they highlight the ability of a decentralised system composed of agents with only local knowledge to organise itself into useful structures, rather than idealising the same system in terms of a well-mixed, population of rational, fully informed agents that adapt to their

environment en masse.

Despite the advances made in understanding such biologically inspired control mechanisms, one of the key issues involved in engineering models that exploit them remains the difficulty of encouraging the 'right' local interactions and discouraging those that may frustrate and destabilise the system (Seeley, 2002).

## 2.3 Decentralised Control

A small number of studies has explored how this can be achieved in truly decentralised control schemes, without assuming that agents have access to global information (Sen et al., 1996; Hogg and Huberman, 1991; Brueckner and Parunak, 2003a; Babaoglu et al., 2002; Saffre and Shackleton, 2008; Saffre et al., 2009). In these systems, agents use decision algorithms that operate on local information. The specific problem of resource allocation (as opposed to service provision[2]) in decentralised multi-agent systems has been discussed by Sen et al. (1996), who consider a system of self-interested agents allocating resources on the basis of limited knowledge about the global system state. In this context, they investigated the effects of limiting the agents' access to both knowledge about the state of system resources, and the resulting outcome on system resource utilisation. Hogg and Huberman (1991, 2002) examined the effects of local decision making on resource utilisation within a computational ecosystem, represented by a population of resource allocating agents. In their work, the authors demonstrated how imperfect information about resource state can lead to chaotic system behaviour, and how this can be suppressed through the use of appropriate local decision-making mechanisms.

Brueckner and Parunak (2003a) present a further strategy relying on local learning mechanisms designed to establish and maintain energy-minimising resource-allocation strategies within mobile ad-hoc networks. In achieving localised strategies that reconfigure the allocation of resources in a manner that minimises power consumption, Brueckner and Parunak drew their inspiration from the self-organising properties of natural systems. One of the distinguishing features of this approach is an explicit appeal to emergent system properties where simple and local decisions conducted by system elements gave rise to system level behaviours that were not pre-programmed.

[2]By *service provision*, we refer to the problem faced by consumer agents in selecting amongst provider agents that offer services (Stein et al., 2008). By contrast, *resource allocation* also includes the complementary problem faced by provider agents in deciding which service(s) to offer.

Moreover, rather than consider these emergent properties to be problematic, here they are often expected to play key roles in achieving system-level objectives.

An analogous bottom-up approach underlies work presented by Babaoglu et al. (2002) where a self-organising approach is proposed to P2P system consisting of multiple and locally interacting nodes that are tasked to effectively provision requested resources (e.g., searched content in P2P system). A similar architecture is proposed by Saffre and colleagues (Saffre and Shackleton, 2008; Saffre et al., 2009). Here, an overlay network with self-* properties is introduced within which the nodes organise themselves to establish symbiotic relations that minimise the time and costs associated with service provisioning.

### 2.4 Contribution

The current paper focuses on achieving fully decentralised regulation of resource allocation in a model autonomic system by relying only on *local* interactions between components. More specifically, we present a decentralised multi-agent system in which provider agents are tasked with adaptively configuring the services that they offer, and consumer agents must choose from amongst these provider agents in order to efficiently secure computational resources necessary to carry out jobs.

Two aspects of this work are particularly novel. First, the problem to be solved is two-sided in that both provider and consumer agents must adaptively reconfigure their behaviour in ways that are complementary. Second, this adaptation takes place in a dynamic environment where changes in the demand for different system services may require continuous reconfiguration at the level of individual system components.

A further distinguishing feature of our work is the attempt to identify and explain the mechanism of self-organisation that underpins the relationship between the behaviour of individual agents and the effect they have in terms of system performance. We achieve this by relating the information capacity of the agents to the emergent topology of information flows between them, which self-organises in the form of functional communities.

This approach is exemplified by the work of Guerin and Kunkle (2004) in a different context: investigating the self-organising properties of an artificial swarming system represented by a simulated colony of food foraging ants. Here, the propagation of information (in this case, pathways between the ants' nest and various food sources) is achieved through the deposition of simulated pheromones within a simulated world. The studies found that

agents (representing individual ants) were locked into pathologically tight loops of behaviour when pheromone traces were too strong. As a consequence, organised but circular foraging routes emerged that were inefficient in transporting food back to the nest. In contrast, when pheromones were too weak, the system was incapable of forming any foraging structures, since information evaporated before it could be reinforced, leaving individual foragers to randomly explore the area. Between these extremes lies a regime of effective self-organised behaviour; in this case, the establishment and maintenance of efficient foraging trails. Understanding how to regulate this type of information exchange so that an autonomic computing system is maintained between both extremes, and achieving this regulation through local decision-making strategies is the main motivation for our work.

# 3 A Decentralised Resource Allocation Model

In this section, we propose a framework for a *bottom-up* resource allocation mechanism, whereby the adaptation of agents (in response to changes in the environment) is based on stimulus-response reinforcement mechanisms inspired by behaviours that encourage self-organisation within insect societies (Bonabeau et al., 1997; Théraulaz et al., 1998). In the absence of centralised controllers, the system elements need to preserve a certain degree of autonomy, allowing for local adaptation to occur given perceived changes in the environment. This architectural flexibility is provided through the use of a decentralised multi-agent system architecture. The challenge of resource allocation can be viewed as a market-based, service allocation problem, where there is a (continually changing) demand for services of a given type, and thus the market responds[3] by changing its supply of such services to satisfy the demand. As stated earlier, a multi-agent system is analogous to an autonomic system, which can be thought of as a collection of computing resources tied together to perform a specific set of functions (Kephart and Chess, 2003). These resources may be hosted in a distributed fashion by a number of servers deployed over networked machines, which provide services to each other. The framework is therefore modelled as a multi-agent system comprising a number of provider agents (*providers*) who offer services of a specific type, and consumer agents (*consumers*) who request and utilise the available services to achieve some task. We assume that both service

[3]In this context, we refer to the market as a decentralised collection of service providers, that each respond individually, based on their perception of changing service demand, rather than a single, atomic, coordinating entity.

providers and service consumers are agents running on *constrained* hardware components. Depending on the characteristics of the system, interaction between these agents may be limited by power consumption, bandwidth consumption, or time-delayed response, all of which may have associated costs if service execution is to take place "on-demand", quickly or by some deadline. In the system presented below, one aspect of such hardware limitations is represented in the form of service capacity, such that each agent may only satisfy service requests for a restricted set of service types, provided to a limited number of consumers simultaneously.

We assume an agent is capable of reconfiguring the service type it provides at run-time. This involves a significant cost in the form of down time during which various administrative tasks may be performed, such as: completing existing service commitments; removing security-compromising data from the machine state; resetting the execution stack; or loading new software modules representing the new service types. We also consider that providers increase their utility by successfully allocating service requests, and that consumers increase their utility by successfully executing services. Thus, to maximise utility, provider agents try to avoid offering services for which there is little demand (thus minimising idle time), and consumer agents try to allocate service requests efficiently by locating providers that are available and configured to offer the appropriate service. A decentralised service discovery model is assumed, whereby each agent maintains a limited registry of details regarding service providers in its environment. Consumers can discover new providers through regular dialogue with known providers that continually update and share their awareness of local service availability (see Figure 2).

[Figure 2 about here.]

The evolution of the system is therefore driven by a continually reconfiguring network of peers, as illustrated in Figure 3. Both consumer and provider agents co-adapt to each other by exchanging information, and reconfiguring their interactions; i.e., by changing what services are offered (in response to observed changes in service demand), or by changing what providers should be contacted (based on observations of the availability of different services). These local responses are driven by the decision-making mechanisms (detailed in Sections 3.1, 3.2 and 3.3, and summarised below) and the information that is propagated throughout the topology of agents as a result of their activities.

[Figure 3 about here.]

As providers have no global information regarding service demand, they utilise their own experience (based on the frequency and type of queries they receive from consumers), as well as information on service availability garnered from those consumers they interact with, to determine whether to continue offering a given service type or to *switch* to providing another service type. Consumers discover new providers through a process of social learning, where new information is acquired through "gossiping". When a consumer and provider interact, the consumer may provide details of other providers that it has interacted with. In return, the provider agent informs the consumer of other providers it is aware of (obtained through interactions with other consumers).

Thus, service management and provisioning strategies should emerge from local co-adaptations of individual agents based on observations of previous transactions. Whilst this naturally involves sharing some knowledge, the agents independently modify their individual models of the local environment. Since service availability can fluctuate as a result of several factors—including current demand, contention for services, and demand for other service types (resulting in a reconfiguration of service offerings)—it is necessary that agents maintain an accurate model of the environment by maintaining a continuous flow of pertinent information with their peers.

This notion of sharing information may initially appear counter intuitive, raising the question "why would a provider supply information on potential rival providers to its users, thus possibly reducing demand on its own services?". One answer is that all provider and consumer agents within the system might be operated by a single firm, $F$, that owns the computational infrastructure upon which they run. Each provider agent might represent a computational resource owned by $F$, or owned by clients of $F$ that employ $F$'s infrastructure to deliver their services to their customers. In either case $F$'s provider agents have an interest in collaborating with each other to achieve global efficiency and fairness rather than competing with each other to maximise 'market share' for any one client. Similarly, $F$'s consumer agents might act as brokers securing computational resource for real-world customers. Again, such consumer agents might be happy to share (suitably anonymised) information with each other in order to maximise system throughput or mean quality of service for $F$'s customers, even if these real-world customers are competitors. Such a scenario is analogous to that faced by a colony of social insects where the reproductive success of each insect is channelled through a single queen, guaranteeing that they have a shared interest in colony success and are motivated to honestly and freely share information.

However, it may also be possible to demonstrate that honest exchange of information makes sense even in competitive scenarios where different provider agents directly represent self-interested firms that are in competition with one another. We will not explore such scenarios here, but the literature on natural communication offers many illustrative suggestions as to the conditions that promote and support honest communication amongst agents with conflicting interests (Noble et al., 2001).

In general, honest information exchange provides a mutually beneficial mechanism whereby consumers can acquire a timely and accurate *model* of services in the local community, and providers can determine a realistic estimate of the service demand in the same community, and thus (if necessary) switch to improve their own utility. The neighbourhood that emerges is dependent on the size of the model that consumers retain of their peers. In addition, the stability of the neighbourhood is also dependent on this model size; the larger the model, the greater the chance of instability, as more providers may switch the type of services they offer in response to the perceived change in demand (as illustrated by the El Farol Bar problem described earlier).

The framework presented here makes the assumption that the tendency for information to be passed between agents will be influenced by the degree of *stress* that the consumers experience. This consumer stress (described more formally in Section 3.1) reflects the difficulty in locating available providers for a given service, and hence provides an indication as to whether the service supply can sufficiently meet current service demand. Whilst there is, perhaps, the opportunity for deceit in such a system, since the supply of (and demand for) services can fluctuate, maintaining an accurate model of the environment in each agent involves maintaining a continuous flow of information between agents. This issue is discussed further in Section 4.4.

By relaxing the constraints limiting the amount of knowledge held by each agent, the same model can explore scenarios in which agents have complete awareness of the current service demand. This is also equivalent to assuming the presence of a single *Matchmaker* (Sycara et al., 1997) which maintains a registry of all available service providers within a multi-agent system, and could in theory support the task of load-balancing. By providing global knowledge to both providers and consumers, the behaviour arising from using a centralised service registry would be simulated, as every consumer would rapidly acquire complete and identical models of the service landscape. Likewise, every provider would be aware of all requests from all consumers, and thus would have the same information as every other (rational) provider. Thus, the framework can be used to explore lo-

calised community behaviour by varying the size of retained knowledge, and compared to complete, global knowledge.

In summary, then, the design goals of our autonomic resource management system are threefold:

1. dynamically reconfigure service providers in order to meet the current demand for different service types;

2. dynamically maintain the awareness of each consumer such that the system maximises throughput by minimising the amount of time wasted by consumers during job allocation; and

3. dynamically reconfigure such that the system is robust to changes in supply and demand for each service type.

The remainder of this section describes service consumers (Section 3.1), service providers (Section 3.2), and knowledge exchange, i.e., "gossiping" (Section 3.3).

### 3.1 Service Consumers

In general, a single agent may be capable of both offering and consuming services. However, for the purposes of this paper, we consider the model for each behaviour as separate. Consumers are agents that request and consume services provided by provider agents. A consumer monitors the behaviour of known service providers locally, and uses this knowledge both to direct its own service requests, and to share with other agents, thereby establishing community knowledge. A consumer, $c$, maintains a local registry, $\mathcal{R}_c$ containing tuples corresponding to providers/services that the agent is aware of. Each tuple is defined as follows:

$$r_p = \langle \alpha_p, type, bias \rangle,$$

where $\alpha_p$ corresponds to the identity of a provider agent that has provided the service *type* at some point in the past, and $bias \in [0, 1]$ corresponds to a score or preference for using provider $\alpha_p$ to provide the same service in the future. As consumers will not possess complete knowledge about whether a provider is currently available, or even if it is still configured to provide the same service type, it employs a local learning mechanism to update its bias estimate. This estimate is updated based on information obtained from the different providers that it interacts with (Section 3.3).

This registry is also consulted by the consumer when it attempts to allocate a service request to a provider; the consumer ranks all the tuples in its registry that are related to the desired service type in decreasing order of *bias*, and then submits requests to providers in turn (starting with the tuple associated with the highest *bias* value), until a provider is found which can satisfy the request. Each request takes some finite time ($T_q$), and the provider will respond either to confirm that it will satisfy the request (i.e., that it is available to provide the desired service *type*), or to reject the query; either because it currently does not provide that service type, or because it is currently too busy (i.e., it does not have sufficient resource to honour the new request without compromising current commitments).

[Figure 4 about here.]

Each provider can simultaneously satisfy up to capacity $C_{type}$ service requests; therefore provided that fewer than $C_{type}$ services are being provided, a new request (of the right type) can be honoured. If a provider is capable of honouring the request, the service is executed, taking some finite time $T_e$, consuming a single service allocation resource. This resource is released once the service execution ceases. Figure 4 illustrates many of the different states that both consumer and provider agents may occupy.

Typically, once a service request has been satisfied (or if no requester could be found), the agent exchanges local knowledge with known providers (described in Section 3.3) before becoming inactive for some randomly determined period. The knowledge exchange is assumed to take some finite time, $T_i$ (irrespective of the number of providers involved), and corresponds to the process of sharing information about local service demand (and availability), and thus evolving a localised community structure. The inactive period corresponds to those periods in other scenarios (or frameworks) where agents may be performing other tasks, or interacting with a user; however, for the purposes of our framework, the agent simply becomes inactive. This period is drawn from a uniform distribution $[0, \omega]$ after which the allocation process begins again.

Consumer stress ($c_s$) is a measure of local consumer dissatisfaction, and reflects the difficulty an agent may experience when trying to provision a given service type. The agent maintains an upper limit, $f_{max}$ of request attempts for each task. The motivation is that by the time the agent has queried all relevant entries within its registry, previously unavailable providers may have become available. As the consumer may therefore traverse a registry several times whilst attempting to provision a service, an upper request limit is used to terminate the provision after $f_{max}$ failed requests.

**Algorithm 1** Consumer task allocation algorithm

**Require:** a consumer $\alpha_c$ with a need for service *type* of capacity *capacity*, and the set $\mathcal{P}$, which contains all the known providers that appear in its registry, $\mathcal{R}_c$

**Ensure:** $\alpha_c$ identifies what it judges to be the best provider of service type *type*, and $\mathcal{R}_c$ is updated through the exchange of information

1: sort ($\mathcal{R}_c$) ordered by *bias* descending
2: **for** ($q := 0$ to $f_{max}$) **do**
3: $f_q := 0$
4: $r_p := \mathcal{R}_c[q \ modulo \ |\mathcal{R}_c|]$
5: **if** typeOfTuple ($r_p$) = *type* **then**
6: $\alpha_p :=$ providerOfTuple ($r_p$)
7: $response :=$ sendRequest ($\alpha_p, \langle \alpha_c, type, capacity, c_s \rangle$)
8: **if** $response = accept$ **then**
9: $\mathcal{R}_c := (\mathcal{R}_c / r_p) \cup \langle \alpha_p, type, bias + \delta_{bias} \rangle$ {*Update bias of* $r_p$}
10: **break**
11: **else if** $response = reject$ **then**
12: $\mathcal{R}_c := (\mathcal{R}_c / r_p) \cup \langle \alpha_p, type, bias - \delta_{bias} \rangle$ {*Update bias of* $r_p$}
13: increment ($f_q$)
14: **end if**
15: **end if**
16: **end for**
17:
18: $f_{t-1} := f_t$ ; $f_t := \min\left(1, \frac{2 f_q}{f_{max}}\right)$
19: $c_s := \left(\frac{f_{t-1} + f_t}{2}\right)^2$ {*Update consumer stress*}
20:
21: **for all** $r_p \in \mathcal{R}_c$ **do**
22: $\mathcal{R}_c := (\mathcal{R}_c / r_p) \cup \langle \alpha_p, type, bias \times \delta_{decay} \rangle$ {*Update bias of* $r_p$}
23: **end for**
24:
25: **for all** $\alpha_p \in \mathcal{P}_c$ **do**
26: $\mathcal{R}'_c :=$ exchangeRegistryWithProvider ($\mathcal{R}_c, \alpha_p$)
27: $\mathcal{R}_c :=$ mergeRegistry ($\mathcal{R}_c, \mathcal{R}'_c$)
28: **end for**
29: sleep ()

[Figure 5 about here.]

During each attempt, $t$, to provision a service, the agent maintains a failure quotient, $f_t$, based on the number of requests to provider agents that have failed so far, $f_q$, where $0 \leq f_q \leq f_{max}$ and

$$f_t = \min\left(1, \frac{2f_q}{f_{max}}\right).$$

The intuition here is that the failure quotient should approach unity as the number of failed queries reaches $\frac{f_{max}}{2}$, despite the fact that it can still continue to issue requests before reaching $f_{max}$. The stress parameter, $c_s \in [0, 1]$, is generated by calculating the square of the average failure quotient for the current and previous provision (illustrated in Figure 5), as follows:

$$c_s = \left(\frac{f_{t-1} + f_t}{2}\right)^2.$$

The consumer periodically updates the ordered set $\mathcal{R}_c$ to reflect its experience in provisioning services, and to minimise the number of future rejected queries. If the request was successfully satisfied, then the tuple $r_p$ corresponding to the provider $\alpha_p$ which provided the service *type* is modified, such that *bias* is incremented by $\delta_{bias}$; otherwise it is decremented by this amount[4]. To ensure that this model of provider availability does not become stale, a decay function is used to adjust the *bias* parameter for all tuples in $\mathcal{R}_c$, by applying a *decay coefficient*[5]. After each update, the set $\mathcal{R}_c$ is then ordered with respect to *bias*, such that tuples with greater bias are placed nearer the top of the list.

The algorithm used by a consumer agent is represented in Algorithm 1. In line 1, the registry is ordered such that the highest ranked providers (according to their *bias* value) are at the top. The consumer then traverses this list, searching for providers that can satisfy its service request, until one is found (lines 2-16). If a provider is found that can offer the service, that provider's rating is incremented (line 9), otherwise it is penalised (line 12), and the failure count is incremented. Once the service has been satisfied (or the service request failed after $f_{max}$ attempts), the consumer stress is updated (lines 18-19). The ratings of all the providers are decremented

[4]The value $\delta_{bias} = 0.1$ was found empirically. The modified *bias* parameter is limited such that it does not extend beyond its defined range: $bias \in [0, 1]$

[5]The decay coefficient used in this model has the value $\delta_{decay} = 0.9$; this value was determined empirically.

using the decay coefficient (lines 21-23), before information is exchanged with them (lines 25-28). Finally, the agent sleeps until the next service invocation (line 29). Figure 6 (left) illustrates a schematic representation of the internal state of a consumer, indicating the current level of stress (given by the thermometer icon) whilst attempting to provision service of type *circle* from an internal registry of service providers.

[Figure 6 about here.]

### 3.2 Service Providers

Providers "model" the local demand for services to determine what services they should offer. Providers may only offer a single type of service at any time, despite possessing the *capability* to offer several types of services; due to limitations in physical resources (e.g., memory size, processor capacity, etc.), or based on security issues. Business sectors (such as the e-business sector) also limit the number of software modules that servers can provide at any time to avoid information leak. The suspension of availability of one service type and introduction of another can have an implicit cost, as this reconfiguration typically takes some time during which the agent cannot perform any further service execution, and thus will not obtain any utility increase. We therefore assume that each provider agent $\alpha_p$ can only offer one service type at any time, but has the capability of offering several other service types (subject to reconfiguration). The set:

$$Capability = \bigcup_{\forall \alpha_p \in MAS} Capability_{\alpha_p}$$

contains the union of all service types available from all service providers in the multi-agent system ($MAS$), whereas $Capability_{\alpha_p}$ corresponds to the set of services that $\alpha_p$ is capable of offering.[6] Thus, to determine which service type $\alpha_p$ should offer, it maintains a model $\mathcal{S}$ of current, local service demand, and determines which services to offer from that model. To achieve this, the provider maintains a number of registries corresponding to provider ratings for different service types, i.e., $\mathcal{S}_{type}$ for each registry of type *type*, as provided by consumers during information exchange.

Providers receive requests from consumers in the following form:

$$req_i = \langle \alpha_c, type, capacity, c_s \rangle$$

[6] In the results reported here $\forall \alpha_p, Capability_{\alpha_p} = Capability$.

where $\alpha_c$ corresponds to the consumer which submitted the request, $type \in Capability$ corresponds to the type of service the consumer requested, the size (in terms of *capacity*) of the task required, and the consumer's current stress, $c_s$ (defined in Section 3.1). When a new request is received that can be satisfied, the provider's registry is also updated. Each request is augmented with a *bias* rating, and stored as the tuple $r_c$, which is in a form that can also be used when exchanging information with consumers; i.e.,

$$r_c = \langle \alpha_c, type, capacity, c_s, bias \rangle$$

If the registry already contains a tuple for the consumer and service type, then the tuple in $\mathcal{S}_{type}$ is updated with the new *capacity*, and the *bias* is modified, based on the product of the consumer's stress, $c_s$ and the *update coefficient*[7], as follows:

$$bias = \min(1, bias + \delta_{update} c_s)$$

This adjustment reflects an increase in perceived demand for the service type. If no previous request exists from the requesting consumer for this service, then a new tuple is added to $\mathcal{S}_{type}$, with an initial $bias = \delta_{update} c_s$. The union of all the sets, $\mathcal{S}$, is then ordered[8] with respect to *bias*.

The provider periodically consults the ordered set $\mathcal{S}$ to determine whether or not to reconfigure its offered service. As there is no global view of current service demand, the providers have to infer this based on local demand observed from previously received requests, which can then be used as evidence for the decision to reconfigure its service offerings. If the *type* of service in the first tuple corresponds to the service that is currently being offered, then no action is taken. Otherwise, the provider performs a *switch* operation, whereby the provider changes the type of service it can provide. Whilst this switching process has no explicit economic cost, it has an implicit cost as the process takes a finite time ($T_s = 2s$), during which no other service can be provided. To ensure that the model maintained for current service demand does not become stale, a decay function is used to adjust the *bias* parameter for all tuples in the sets $\mathcal{S}_{type}$ for each type, using the decay coefficient $\delta_{decay}$.

The algorithm used by a provider agent is represented in Algorithm 2. On receiving a service request (line 1), the provider verifies that it is currently able to provide the service (in terms of both service type and capacity)

[7]The update coefficient used in this model has the value $\delta_{update} = 0.1$; this value was determined empirically.

[8]The order of equally biased tuples is arbitrary, and thus may vary whenever the set is inspected.

**Algorithm 2** Provider service provisioning algorithm

**Require:** a provider $\alpha_p$ currently offering service type $type_p$ with available capacity $capacity_p$, and a set of service types that it may offer, $\mathcal{S}$.

**Ensure:** $\alpha_p$ offers what it perceives to be the most demanded service *type* and $\mathcal{S}$ is updated through the exchange of information.

1: $req_i$ := receiveRequest() $\{where\ req_i = \langle \alpha_c, type, capacity, c_s \rangle\}$
2: **if** $(type_p \neq type \wedge (capacity_p < capacity)$ **then**
3: sendResponse($\alpha_c, req_i, REJECT$)
4: **else**
5: sendResponse($\alpha_c, req_i, ACCEPT$)
6: executeService($req_i$)
7:
8: $r_c := \langle \alpha_c, type, capacity, c_s, \_ \rangle$ $\{based\ on\ req_i;\ no\ bias\ value\ asserted\}$
9: **if** $\mathcal{S}_{type} \subseteq \mathcal{S}$ **then**
10: $\mathcal{S}'_{type} := \mathcal{S}_{type}$
11: **if** $req'_i \in \mathcal{S}'_{type}$ **then**
12: $r'_c := \langle \alpha_c, type, capacity, c_s, \min(1, bias + \delta_{update} c_s) \rangle$
13: $\mathcal{S}'_{type} := (\mathcal{S}'_{type} / r_c) \cup r'_c$ $\{Update\ bias\ of\ r_c\}$
14: **else**
15: $\mathcal{S}'_{type} := \mathcal{S}'_{type} \cup \langle \alpha_c, type, capacity, c_s, \delta_{update} c_s \rangle$
16: **end if**
17: **else**
18: $\mathcal{S}'_{type} := \{\langle \alpha_c, type, capacity, c_s, \delta_{update} c_s \rangle\}$
19: **end if**
20: $\mathcal{S} := (\mathcal{S} / \mathcal{S}_{type}) \cup \mathcal{S}'_{type}$ $\{Update\ the\ Registry\ \mathcal{S}\}$
21:
22: sort ($\mathcal{S}$) ordered by *bias* descending
23: **if** head($\mathcal{S}$) $\neq type_p$ **then**
24: PerformSwitch(head($\mathcal{S}$))
25: **end if**
26: **end if**

before going on to execute the service[9]. Once the service has been successfully completed, the provider updates its internal registry, by creating a tuple, $r_c$, based on the request (line 8). If a set of records for the requested type exists (i.e., $\mathcal{S}_{type} \subseteq \mathcal{S}$ in line 9), then either the new tuple is added to the set (line 15), or if an appropriate tuple exists, it is updated (line 13). Otherwise, a new set $\mathcal{S}_{type}$ is created with the new tuple as its only element (line 18). The provider then sorts all its sets of tuples $\mathcal{S}$ into descending order, to determine if it should change the type of service it currently offers (lines 22-25). In the example depicted in Figure 6 (right), provider E is most likely to continue to offer services of type *square*, but might conceivably switch to offering services of type *hexagon* in the near future, since this is also associated with moderately high demand.

## 3.3 Information Exchange

To facilitate the migration of knowledge regarding the availability of services and current service demand, both providers and consumers share knowledge before revising their respective models. This process is initiated by consumers, and occurs each time a consumer completes a transaction with a provider.

### 3.3.1 Sharing knowledge with Providers

Each consumer shares all of the tuples contained in its local registry, $\mathcal{R}_c$ and its current stress level, $c_s$, with each of the providers that are listed in the registry. Each provider then uses an integration policy to incorporate this knowledge into its own local registries (i.e., $\mathcal{S}_{type}$ for each of the service types the provider knows about). This integration policy limits the number of tuples merged from the consumer's knowledge for each given type (i.e., tuples from $\mathcal{S}_{type}$) with its own knowledge, based on $c_s$. This stress level indirectly represents the quality of knowledge the consumer possesses; low $c_s$ values suggest that the tuples provide an accurate representation of the current availability of services, whereas high $c_s$ values suggest that the knowledge is poor or that the supply of that particular service type is low, thus leading to difficulties in provisioning services. The maximum number of information tuples the provider is willing to substitute ($t_n \in \mathbb{Z}$) for each service type

[9]If the provider is in the process of reconfiguring or *switching*, all requests are rejected until any currently executed services have been completed, and the provider has successfully changed its current service offering.

*type* is defined as follows:

$$t_n = (1 - c_s) * m_c$$

where $m_c$ is the size of the set of tuples provided by the consumer for a given service type. The provider selects the highest ranking $t_n$ tuples (ordered by each tuple's *bias* parameter) for integration into its registry. Each provider maintains a limit ($m_p$) on the number of tuples it stores, which determines which of the consumer's tuples are retained.

### 3.3.2 Tuple Integration

There are three possible ways that each new tuple may be integrated into the provider's registries, based on whether the provider has existing information on $\alpha_p$ (specified by the tuple), namely: *add*, *substitute* or *update*. If none of the tuples in $\mathcal{S}_{type}$ refer to this provider, then the tuple is added or substituted, using the following policy: if $|\mathcal{S}_{type}| < m_p$ then the new tuple is simply *added* to $\mathcal{S}_{type}$. Otherwise, a tuple for some other provider is potentially removed to allow the new tuple to be stored. The way this is done is based on the *bias* value; the provider inspects those tuples for which the *bias* is less than that in the new tuple, and randomly selects one of these to be *substituted*. If none are found, then the new tuple is not introduced. This mechanism guarantees that only knowledge that has an equal (or higher) *bias* than that existing within a provider's memory will be introduced.

The third integration mechanism, *update*, modifies an existing tuple maintained by the provider. As this new tuple represents an additional, subjective evaluation of provider $\alpha_p$, the new *bias* is calculated by averaging the new and previous tuple.

### 3.3.3 Sharing knowledge with Consumers

Providers maintain *bias* rankings for other provider agents that they may be aware of (through exchanging information with consumers), and organise these with respect to service type (as illustrated in the schematic representation in Figure 6 (right)). Thus, a provider may appear within more than one set of tuples $\mathcal{S}_{type}$, depending on the knowledge that was acquired from different consumers during the last information sharing phase (e.g., in Figure 6, provider $E$ believes that provider $B$ offers services of type *hexagon* and *circle*). Providers can therefore deal with incoming information in a manner

that is sensitive to the type of service required by the consumer they are interacting with. If provider $E$ interacts with a consumer attempting to secure service type *pentagon* and, during this interaction, learns that the consumer associates provider $G$ with a high *bias* estimate, this new information will be used to update $E$'s *bias* estimate for provider $G$ as a provider of service type *pentagon*.

Hence, each provider shares all the knowledge it possesses about other providers (i.e., $\mathcal{S}$), based on its aggregated knowledge shared by different consumers. As consumers will have existing (if somewhat limited) knowledge of different service types, each consumer organises its knowledge, $\mathcal{S}$, into two distinct subsets: $\mathcal{S}^{act}$ and $\mathcal{S}^{inact}$, prior to sharing with different providers. The set $\mathcal{S}^{act}$ contains all the tuples $r_p \in \mathcal{S}$ where the *type* of $r_p$ corresponds to that known by the consumer (i.e., *type* appears in $\mathcal{R}_c$, and thus is considered *active information*). The remaining tuples are considered *inactive* information, shared (indirectly) between different consumers through common providers.

The proportion of data retained from each of these new sets is determined by the current stress of the consumer, such that the higher the value of $c_s$, the greater proportion of tuples from $\mathcal{S}^{inact}$ will be retained. The consumer orders each set (based on *bias*), and then selects $\max(0, |\mathcal{S}^{act}| - 2c_s)$ tuples from $\mathcal{S}^{act}$, and $\min(2c_s, |\mathcal{S}^{inact}|)$ tuples from $\mathcal{S}^{inact}$ for retention. The top ranking $t_n$ tuples are then integrated into the consumer's local registry, $\mathcal{R}_c$ using the same mechanism as that described in Section 3.3.2. The only difference is that the *bias* value is not aggregated, but rather is calculated by averaging the new and previous tuple. In addition, the *bias* of all $r_c \in \mathcal{S}^{inact}$ are simply replaced by $bias = 1$.

This process facilitates the discovery of new services from a group of providers that may not offer a desired service type, but through consecutive interactions, may reconfigure to satisfy new consumers. Thus, consumers are willing to retain a small number of tuples pertaining to these providers (i.e., those that appear in the set $\mathcal{S}^{inact}$) within their local memory only when their allocative stress is high. As this, from a global system point of view, may indicate change in the demand occurring within the system, this mechanism allows providers to 'migrate' to a community with increased service demand. However, introduction of information into the consumer's registry does not guarantee that the provider will change its current service offering.

# 4 Experimental Evaluation

In this paper we explore two hypotheses that underlie the design of the model presented in Section 3; namely:

**Hypothesis 1** A stimulus-response mechanism that facilitates the propagation of limited knowledge regarding resource availability results in the formation of stable communities, whereby roles are well established and correspond to the needs of the community.

**Hypothesis 2** When information flow between peers is limited, this mechanism can deal with significant fluctuation in the demand for different resources by smoothly reorganising agent communities.

[Table 1 about here.]

In Section 4.2, the performance of the decentralised resource allocation model is compared to that of a system employing centralised control and analysis of the system's capacity to form local communities for a variety of service scenarios is conducted. The registries managed by each agent are examined to visualise the network of connectivity between different agents and to verify the formation and structure of communities corresponding to homogeneous service usage (thus supporting Hypothesis 1). This is followed, in Section 4.3, by an investigation of the stability of the emergent communities when service demand is dynamic. The results demonstrate that in the cases where the size of the service registries is limited, providers provide a stable supply of relevant services to the fluctuating market, whereas when the service registry size is large or unlimited (thus simulating the case where services can be discovered through a centralised registry, such as a matchmaker middle-agent (Sycara et al., 1997)), the supply of services degenerates and becomes chaotic (thus supporting Hypothesis 2).

The following experiments were carried out using a variety of model configurations. Table 1 lists the default parameter values (introduced in Section 3) used for each of the evaluations; in the case where a parameter may be different, the new parameter values are explicitly defined. For example, the empirical evaluation in Section 4.2 (Hypothesis 1) uses $C_{type} = 5$, whereas in Section 4.3 (Hypothesis 2), $C_{type} = 2$. Note that all time measurements are quoted as *simulation time* seconds, and do not relate to real time. Each evaluation was performed 10 times, and the results were tested for statistical significance.

Before either of these studies is presented, in Section 4.1 we first introduce an assortativity metric that will be used to evaluate the community structure exhibited by a population of software agents.

## 4.1 Assortativity Metric

An *assortativity metric* has been defined to evaluate the structure of the information flow between individual agents. This measure represents the strength of the community structure in a population of consumers interested in securing the same service type.

In order to calculate assortativity for a particular service type, we first identify the set of consumers interested in securing that service, $\mathcal{C}$. We then determine the complete set of providers known to this set of consumers, $\mathcal{P}$, and populate a square matrix $M$ of size $|\mathcal{P}| \times |\mathcal{P}|$, where $|\mathcal{P}|$ corresponds to the number of unique providers in $\mathcal{P}$. Each element of $M$, $M_{i,j}$, corresponds to the number of consumers in $\mathcal{C}$ that simultaneously know both provider $i$ and provider $j$. Note that some but perhaps not all of the members of $\mathcal{P}$ will be configured to offer the service type sought by members of $\mathcal{C}$.

We can interpret $M$ as specifying a network of undirected connections between members of $\mathcal{P}$. In the simplest case, we might determine that an edge $(i, j)$ exists only where $M_{i,j} > e$. If most network edges connect providers that offer the same type of service then consumer agents are aware of appropriate providers and share a consistent collective memory.

The assortativity metric, $H$, can be defined as follows:

$$H = \frac{1}{n} \sum_{i}^{n} d_i$$

where $i$ is one of the $n \leq |\mathcal{P}|$ providers in $\mathcal{P}$ that have at least one network neighbour, and $d_i$ is the proportion of these neighbours that offer the same service *type* as $i$.

In a random system, the proportion of network edges connecting providers offering the same service will converge to $\frac{1}{T}$, where $T$ is the number of service types. The baseline assortativity, $H$, of communities in such a disorganised system will therefore also tend to be $\frac{1}{T}$.

## 4.2 Demand variety

In this analysis, the system's ability to form local communities was examined over a range of scenarios with differing numbers of service types. The

community structure was investigated, to explore the types of links that emerge through sharing limited information between peers.

### 4.2.1 Scenario Configuration

To explore the effect that different combinations of service types may have on demand, and hence on community organisation, the number of unique service types demanded by consumer agents was varied, such that the *demand variety* is defined as $V = |Capability| \in [1, 6]$ (see Section 3.2). Each consumer was configured to require exactly one *type* of service from the pool of available service types, such that the demand for each service type was equal. Thus, when $V = 1$, all consumers shared the same type of service requirements, whereas for $V = 2$, the consumer population comprised two equally-sized sub-populations that differed in the service they required. Based on the perceived service demand, providers can choose to offer a single service selected from the set $Capability$, as defined in Table 2.

[Table 2 about here.]

In each case, a provider is configured to offer no more than $C_{type}$ "units" of the same service at any one time (i.e., $C_{type}$ is its *service capacity*). The maximum number of tuples in each consumer's memory is defined as $m_c = 10$; i.e., each consumer will remember details of up to 10 providers after each information exchange phase (Section 3.3). On initialisation, consumers are given a random selection of information about the existence of different providers. As the providers have no preference for providing any service, the selection of services they provide is initially randomised. To facilitate faster convergence on the creation of communities, only information residing in the $\mathcal{S}^{act}$ set (Section 3.3.3) was used during the first 200s of simulation time[10].

### 4.2.2 Evaluation and Analysis

Before analysing the way in which the decentralised resource allocation model operates, we first contrast its performance with that of an idealised system reliant on centralised control, described below. This comparison is performed over a range of scenarios that vary in terms of the demand variety, $V$, exhibited by consumers.

[10]The value of 200s was found empirically for a range of experiments and values of $m_c$. Tests on unlimited memory or local registry size (i.e., $m_c = \infty$) failed to converge, despite a range of bootstrapping values tested.

By contrast with the decentralised model, the centralised model utilises a single centrally maintained ideal registry that is divided into sub-registries, each of which corresponds to one of the service types demanded by consumer agents. Upon initialisation, the tuple associated with each provider is allocated to the sub-registry associated with the service that it is configured to provide.

Every simulated second, each sub-registry is updated instantaneously with truthful information about the state of all providers currently configured to offer the associated service. Should a provider switch which service it provides, its associated tuple will switch to the appropriate sub-registry. Each of these sub-registries is sorted in order of the currently available spare capacity offered by each provider such that providers with most spare capacity are at the head of the list. Consumers no longer make use of any local memory, but instead rely on the central registry, consulting the contents of the sub-registry associated with the service that they require and querying providers in order of their spare capacity.

Under realistic conditions, centrally maintaining information on providers would require interactions between the centralised registry and every provider agent, each of which would consume time and computational resource. Here, we do not explicitly model these costs, but simply impose a one second latency between consecutive instantaneous updates.

For the centralised model, Figure 7 shows that system performance (or *throughput*, calculated as the ratio of successful service allocations to overall service requests) is low across all scenarios and lowest for scenarios where consumers exhibit low demand variety. This is a consequence of the congestion created by herding which is maximised where all consumers demand the same service and have access to the same information on providers, and minimised to the extent that consumers are divided into many groups with different service requirements. In the limit where each consumer requires a unique type of service provided by a unique provider, near optimal throughput could be achieved by the centralised model.

By contrast, the system performance of the decentralised model is not unduly affected by varying demand variety, remaining close to the optimum. For the case where $V = 1$, all consumers issue requests for the same service type (i.e., type $A$), encouraging all providers to switch to offer this type of service. Once this has been achieved, there is no pressure for any provider to switch, and the only service allocation failures that may occur are due to requests being sent to providers that are busy. Those consumers stressed by such failures are more likely to update their registries. Thus, the only type of organisation occurring within the environment is due to consumers

replacing knowledge of over-committed providers. Since there is sufficient aggregate supply to meet consumer demand and consumers have different assessments of provider utility, eventually a stable assignment of consumers to providers is able to form.

[Figure 7 about here.]

The task of achieving a stable supply of services becomes more challenging for cases where $V > 1$, since providers may choose to switch the service that they offer on the basis of the (local) consumer demand that they experience. However, the performance of the decentralised model was near-optimal for all scenarios where $V \in [2, 3, 4, 5, 6]$, reflecting the fact that the organisation of information across the agent communities was effective, allowing consumers to establish and interact with an appropriate set of providers (with respect to the desired service type), and thus allowing providers to stabilise their selection of offered services, without the need to continually *switch*. Since each agent is capable of utilising only a subset of the locally available information due to the limited size of their local service registries, this stability emerges from the efficient organisation of shared information that evolves across the different agent communities.

[Figure 8 about here.]

[Figure 9 about here.]

This efficient organisation is captured by the *assortativity* metric described in Section 4.1 which summarises the way in which the contents of agent registries correlated with their service demands. This metric distills information on which providers are known (and utilised) by which consumers into a measure of the coherence of a system's communities. Low values suggest a lack of any community structure, whereas high values reflect distinct communities with very few links between each community.

Strongly assortative community structure was achieved by the decentralised model independent of the number of different service types required by the consumer community. Figure 8 illustrates that this level of assortativity is achieved rapidly in the decentralised model and at a rate that is independent of demand variety, $V$. Hence, unlike the centralised model, decentralised system performance scales well with demand variety.

The resulting network topologies are illustrated in Figure 9(a) for the scenario $V = 3$. In this topology, nodes represent providers, and edges represent the case where both providers are known by at least one consumer,

and the weight of this edge represents the cardinality of the set of consumers that know the two providers. At $t = 20s$ there is little evidence of any community structure, and the links between providers offering different types of services suggests that there is little organisation in the information being shared by the consumer population. This is supported by the fact that the edges between providers are generally weak, suggesting that provider pairs are known by comparatively few consumers.

However, the rapid increase in community strength over time in Figure 8 demonstrates that this initial system instability recedes due to the formation of strong local communities which are homogeneous with respect to service type. By $t = 80s$, the scenarios generally converge to stable configurations, where providers rarely *switch* to other service types, and consumers resolve any request conflicts by identifying those providers that are in less demand within the community. The corresponding network topology at time $t = 250s$ is also illustrated in Figure 9(b), where three distinct and highly connected communities have formed. The strongest edges (corresponding to cardinalities approaching 40 pairs of providers) within the network are those between providers that offer the same service type (typically found within a local community), whereas the weakest connections mainly exist between communities. In the absence of external perturbation (e.g., changes in consumer demand, or removal of providers) community structure remains stable, i.e., communities do not change in size, providers do not reconfigure, and consumers interact with the appropriate community.

[Figure 10 about here.]

The rapid community convergence observed for those scenarios where $V$ was high (Figure 8) suggest that there may be processes that catalyse the efficient organisation of interactions between consumers and providers, such that the allocation of services to the most appropriate providers emerges to be both stable and efficient at the global level. To explore this, an analysis of community strength was conducted for *both consumer and provider* service registries for the scenario where $V = 5$. The network topology for consumers is presented in Figure 10(a), where five separate communities (corresponding to the five available service types) are clearly visible, with strong edges within the communities (where the providers have a mean assortativity value of 0.94), and weak edges between communities.

Providers organise their information corresponding to service type (see Section 3.2), which can result in information corresponding to those service types not currently offered by a stable provider becoming stale. This is

due to the fact that providers will rarely interact with consumers desiring alternate service types. Thus, to analyse the providers' registries (i.e., $\mathcal{S}$), the assortativity metric should be modified, so that only information that is recent is considered by the modified metric. The resulting network topology for providers is presented in Figure 10(b), which reveals the existence of five distinct community structures (with mean strength equal to 0.83).

## 4.3 Dynamic Service Demand

The analysis in the previous section confirms the hypothesis that stable community structures form such that the supply of services can organise to satisfy service demand (provided that sufficient resources are available). However, this assumes that the demand is static and fixed. In this section, the validity of Hypothesis 2 is explored, whereby the stability of communities that emerge from the stimulus-response behaviour of the model is explored in a highly dynamic environment where service demand is continually changing.

### 4.3.1 Scenario Configuration

To investigate the performance of limiting the size ($m_c$) of consumer service registries (and hence information flow), and compare it to the case where complete information is available, the scenario was modified to facilitate changes in service demand over time. For all providers in this scenario, $Capability = \{A, B\}$, and thus $V = |Capability| = 2$ for all providers in this scenario. The service capacity for each provider is restricted to satisfying two service requests at any time; i.e., $C_A = C_B = 2$ (Table 3).

[Table 3 about here.]

Consumer demand varies exogenously, with demand for service type $A$, oscillating in anti-phase with the demand for service type $B$. This variation is implemented by altering the average sleep period that consumers undergo between successive service requests. Thus, when the average sleep period for consumers requesting $A$ is longer than that for consumers requesting $B$, then the demand for providers of service type $B$ will effectively be greater. The sleep period is defined as:

$$sleep_{type} = rand + \delta_{type}$$

where $rand$ is a random value drawn from the range $[0, \omega]$, and $\delta_{type}$ is derived from the sinusoidal function of time (illustrated in Figure 11). Thus,

by varying $\delta_A$ and a corresponding $\delta_B$ which is 180° out of phase, a symmetrical change in demand can be achieved for the two services. To allow the system to achieve a steady state, demand for both types of service is equal and constant for the first $200s$[11].

Within each evaluation, several simulation runs were performed with different service registry, or *memory* sizes, $m_c$, to evaluate the model with limited information retention, and these results were contrasted with a *global* information model.

[Figure 11 about here.]

[Figure 12 about here.]

### 4.3.2 Evaluation and Analysis

Figure 12 illustrates the mean performance of the model with respect to $m_c$. The graphs are normalised with respect to the optimal system performance experienced by the system in equilibrium during steady state (i.e., when service-demand is satisfied by supply such than no reconfiguration of providers is necessary). An analysis of the model's efficiency in successfully satisfying service requests for different sizes of memory (Figure 12) reveals that the system efficiently satisfies service requests only for certain cases where $4 < m_c < 12$. In addition, the level of information flow between consumers is low under such conditions, suggesting that the distribution of knowledge across different consumers regarding local providers is relatively stable. However, for memory sizes outside this range, performance falls to a level below that achieved if global knowledge was available ($m_c = \infty$), where the flow of information between consumers is maximised.

Three general types of behaviour can be observed, based on the way in which the model responds to changing conditions for different memory sizes. These are captured in Figure 13, which illustrates the behaviour emerging from three representative cases (i.e., $m_c \in \{2, 5, 20\}$). When $m_c = 2$ (Figure 13 empty rectangles), the performance degrades as a result of the consumers' inability to resolve service request conflicts in a timely manner. As a result, the stress level grows within an increasing number of unsatisfied consumers. This catalyses the re-organisation and sharing of the consumers

[11] Whilst this scenario is relatively simple compared to real patterns of demand, the intent is to evaluate the behaviour of the system (and resulting communities) whilst maintaining full control of the demand dynamics. More complex scenarios (and real-world case studies) could be explored in future work.

local knowledge, resulting in the acquisition of knowledge about new providers which are likely to be more suitable to the consumer's tasks. However, given the high volume of shared information from stressed consumers, providers become selective regarding the information they retain. Since the number of retained types is so small, information loss occurs, resulting in the formation of isolated groups of agents aware of only small groups of providers, which are unable to propagate this information to other, similar groups.

[Figure 13 about here.]

[Figure 14 about here.]

The model for $m_c = 5$ exemplifies a scenario in which efficient performance is exhibited (Figure 13 solid rectangles). In this case, the model responds smoothly to changes in demand, due to the fact that the consumers' service registries are large enough to retain some information about local consumers despite the change in demand, but small enough that little of the new information exchanged is retained, and hence does not compromise future service provision. As a consequence, there are comparatively few consumers that experience stress high enough to necessitate a continual reorganisation of the local service registry (as in the case when $m_c = 2$), and thus require new information (less than 10%). This also limits the number of providers that observe large changes in service demand, and thus reduces the number of providers that subsequently *switch* services. The way in which providers *switch* services in response to the changes in demand is illustrated in Figure 14. During the initial 200s, local communities start to form corresponding to the two service types, with roughly equal numbers of providers supporting each community. However, as the demand for one service type increases, the level of reorganisation experienced by the consumer population is limited, and thus the number of providers that *switch* matches the resulting demand, yielding a stable, but dynamic environment.

As the number of known providers increases (e.g., in the case where $m_c = 20$), the performance becomes degenerate (empty circles in Figure 13). In this case, as consumers become stressed due to increases in service demand, they exchange and retain larger quantities of information regarding new providers. This compromises their ability to successfully locate service providers, as there will be greater competition for the highest ranking ones. This leads to an increase in the stress of the consumers, which in turn exerts unnecessary pressure on providers to *switch* due to a continuous and

elevated exchange of information. Thus the equilibrium destabilises, as providers fail to respond to changes in the service demand (as illustrated by the empty rectangles in Figure 14), and the community structure is lost.

[Figure 15 about here.]

The case where $m_c = \infty$ corresponds to that where global information is available to all agents (solid circles on Figure 13). In this case, *herding* occurs, where a large number of consumers ends up crowding for the most attractive providers. As the number of requests that can simultaneously be satisfied is small ($C_{type} = 2$ for these experiments), a substantial number of requests are rejected, resulting in large numbers of new queries being issued. This results in an increase in stress experienced by a large number of consumers, which increases the pressure on providers to continually *switch*, thus further destabilising any community structure. The resulting behaviour of providers (which offer services of a given type) varies in a similar way to that observed when $m_c = 20$ (empty rectangles in Figure 14).

A system that effectively manages service provision can be characterised by a community structure that is stable enough to sustain internal reorganisation. Therefore, to better understand the relationship between the model performance and the manner in which agents share local information, an analysis of the community strength (using the assortativity metric defined in Section 4.1), was used. Figure 15 plots the assortativity strength of consumers against time for two different scenarios; where $m_c = 5$ (Figure 15(a)) and $m_c = 20$ (Figure 15(b)).

As the consumers experience increased stress due to rising demand for a given service type, they exert increasing pressure on the providers to satisfy this demand, which in turn strengthens the community structure. In the scenario explored within Figure 15(a), the community strength approached 0.9 as the demand for a service type reached its peak; reflecting the fact that the majority of knowledge retained by the consumers was correctly identifying relevant providers. Correspondingly, as the service demand falls, the strength of the community decreases accordingly. Figure 15(b) illustrates the pathological case, whereby larger memory size ($m_c = 20$) causes providers to *switch* far too frequently, compromising the ability of consumers to effectively locate relevant providers. In this case, the community strength becomes erratic, and no longer varies in phase with changes in service demand.

These results suggest that the formation of strong and adaptive communities (i.e., those that exhibit high assortativity) is accompanied by good

system performance and a robust response to varying demand. Recall that service provision depends on the co-adaptive stability between agents responding to locally perceived changes. The formation of a strong community by agents interested in a particular service type yields an exchange of information that is *constrained*. If a community is strong, knowledge passed to consumers can be exploited to induce a small subset of providers to *switch* service types, minimising the risk of destabilising the availability of services to other consumers in other local communities. This raises the question of how strong should a community be in order to support this process optimally? A tightly linked community (where community strength approaches one) could prevent reorganisation when there are changes in demand. However, if the community is too weak, then the availability of services will be destabilised. A second question thus arises: under what conditions is such a system capable of forming strong communities? As demonstrated in this paper, simply varying the amount of information available to agents is sufficient to bring about changes in system behaviour: from being too disconnected, to effectively evolving self-organising communities, and eventually to a configuration where agents regress into a chaotic flux.

[Figure 16 about here.]

Figure 16 plots the community strength as a function of memory size. In those cases where the size is too small or too big, degenerate behaviour results, similar to that illustrated in Figure 15(b). For those cases where $m_c \in \{2, 3\}$, consumers retain insufficient knowledge to successfully facilitate a useful flow of information as the environment changes. In contrast, when the memory size is large (i.e., $m_c > 15$), consumers retain more information about the environment, and consequently can better stimulate change in the provider population. As levels of stress increase, there is a greater chance that larger numbers of consumers will compete for the most favoured providers, resulting in a feedback loop that escalates stress levels, which increases the pressure on providers to change, yielding further destabilisation. It is only within the intermediate range of cases (i.e., $4 \leq m_c < 15$) that the community strength increases. Here, each agent is able to adapt to changes in the environment effectively, without necessarily being able to identify the globally optimal providers. The close agreement between Figures 16 and 12 demonstrates that it is the strength of the communities formed through information flowing through the system that directly underpins and accounts for system performance.

## 4.4 Discussion

When first confronted by the problem of configuring computational resources and allocating them in response to varying consumer need, it seems likely that smart enough agents employing probabilistic and possibly collaborative reasoning of some kind might be able to circumvent the "herding" problem. This may indeed be the case, but instead it may be that, as Hogg and Huberman (1991) concluded, the smarter the agents, the more likely they are to suffer from some kind of co-ordination problem. Our approach here has been to explore whether these potentially intractable co-ordination problems can be avoided by employing much simpler agents and relying on asymmetries in their knowledge state that arise naturally under conditions of only local information flow.

A critical factor in achieving the efficient performance is the organisation of knowledge across the consumer and provider populations. A system where consumers successfully organise their local knowledge supports ongoing interaction between providers that can supply the services that the corresponding consumers require. Moreover, such an organisation also has the ability to smoothly reconfigure the supply of services in response to changes in demand. By contrast, operating on the wrong information will result in degraded performance due to high numbers of rejected queries and time spent on needlessly reconfiguring service providers. Populations of agents that possess only incomplete local knowledge must therefore rely on the appropriate flow of information in order to keep this knowledge up to date, and coherent enough to be of use. This must be achieved in the absence of any global flow-facilitating mechanisms that determine *what* information should flow to *which* agent.

An analysis of the information flow that emerges from the evaluation of our model can provide a number of insights into the dynamics of self-organisation, and in particular, how self-organisation can emerge from localised decision-making within the context of resource management problems involving configuration of providers to offer services based on locally perceived demand. Although results demonstrate that the size of the local registry containing provider information is one of the critical factors influencing the flow of information between agents in the neighbourhood, and thus the cohesion and stability of the community as a whole, there are other interdependent variables that play an important role in the process. Such variables include: the frequency of information exchange, the quantity of information exchanged in a single exchange transaction, and thus the level of influence an agent has on another agent's decision making. These variables

have been demonstrated to play a key role in achieving self-organisation and adaptation for a number of other decentralised architectures (Guerin and Kunkle, 2004; Brueckner and Parunak, 2003b; Packard, 1988). In the model presented in this paper, the extent of exchanged information was dynamically affected by consumer *stress* ($c_s$) which determined the amount of information that both consumer and provider agents were willing to accept. Likewise, the level of consumer stress was an important factor in determining the impact of the consumer, whilst querying the provider for a given service type. Finally, this stress level was also significant in establishing the demand for a scarce resource, and hence encouraging providers to switch the type of service they currently provided. As all of these factors have an impact on how agent interactions evolve, and thus how information flows through the system, we plan to i) conduct a further investigation to analyze how these parameters affect each other in order to formally identify the synergistic relationship between these parameters and their influence on the dynamic stability of the system, and ii) to identify efficient techniques for automatically *self-regulating* them in a decentralised manner based on information available locally to each agent.

This research highlights *functional substitutability*, i.e., the ability of a component to change its behaviour in response to changes in demand. This is desirable with many scenarios where there is a risk of failure in one component, such as within robotic rescue scenarios (Kitano, 2000) whereby a robot may be able to perform several different tasks (e.g., sensing, moving, performing actions etc), but only a few of these are needed for any given scenario. However, changes in the environment may necessitate corresponding changes in the roles (at run time) that those robots perform. Similar behaviours have been observed within biological systems. In natural ant colonies, it has been observed that within what is often ascribed as a homogeneous community of ants sharing the same behavioural repertoire, there arise leaders that strongly stimulate the actions of other ants by their more frequent activity (Dobrzanski and Dobrzanska, 1987). Such *hyperactive* ants stimulate the workload of the colony in every aspect (food foraging, nest building, brood feeding, etc.). Thus, *leader* and *follower* roles arise under certain conditions (rather than being inherent, programmed behaviours). However, a complete understanding of this social stimulation process is still unclear.

## 5 Conclusions

In this paper, we have investigated how a simple stimulus-response mechanism inspired by a study of emergent behaviours within biological systems can be used to facilitate the emergence of stable behaviour in a large multi-agent system. We argue that this approach or something like it is necessary to support collaborative behaviour of large-scale, decentralised IT systems, such as in autonomic computing systems. The paper's contribution is to demonstrate that constrained information flow can enable simple agents to solve a decentralised co-ordination problem through the self-organisation of agent communities. These communities arise spontaneously as a consequence of local gossiping. They reflect the nature of the pressure experienced by the system, are stable, and are capable of smoothly reconfiguring to cope with changes in consumer demand.

Future work will explore scenarios in which the demand for computational resource may outstrip supply (or vice versa), the challenge of combining power management considerations with performance targets in a single autonomic scheme, and the robustness of decentralised behaviour to systemic parameters and the types of system noise and error that characterise real-world environments.

**Acknowledgements**

Thanks to the SENSe and AIC research groups at the University of Southampton for discussion of these ideas.

## References


Arthur, W. B. (1994). Inductive reasoning and bounded rationality. *American Economic Review*, 84:406–411.

Babaoglu, O., Meling, H., and Montresor, A. (2002). Anthill: A framework for the development of agent-based peer-to-peer systems. Technical Report UBLCS-2001-09, University of Bologna, Italy. Presented at the 22nd International Conference on Distributed Computing Systems.

Bonabeau, E. (2002). Predicting the unpredictable. *Harvard Business Review*, 80:1–9.

Bonabeau, E., Sobkowski, A., Théraulaz, G., and Deneubourg, J. (1997). Adaptive task allocation inspired by a model of division of labor in social insects. In Lundh, D., Olsson, B., and Narayanan, A., editors, *Biocomputing and Emergent Computation*, pages 36–45. World Scientific.

Brueckner, S. and Parunak, H. V. D. (2003a). Self-organizing MANET management. In Marzo, G. D., Karageorgos, A., Rana, O. F., and Zambonelli, F., editors, *Engineering Self-Organising Systems*, pages 1–16. Springer.

Brueckner, S. A. and Parunak, H. V. D. (2003b). Information-driven phase changes in multi-agent coordination. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 950–951, NY, USA. ACM Press.

Bullock, S. and Cliff, D. (2004). Complexity and emergent behaviour in ICT systems. Technical Report HP-2004-187, Hewlett-Packard Labs.

Bullock, S., Ladley, D., and Kerby, M. (in press). Wasps, termites and waspmites: Distinguishing competence from performance in collective construction. *Artificial Life*.

Bullock, S. and Todd, P. (1999). Made to measure: Ecological rationality in structured environments. *Minds and Machines*, 9:497–541.

Challet, D. and Zhang, C. (1997). Emergence of cooperation and organization in an evolutionary game. *Physica A: Statistical and Theoretical Physics*, 246:407–418.

De Roure, D. (2003). On self-organization and the semantic grid. *IEEE Intelligent Systems*, 18:77–79.

Dobrzanski, J. and Dobrzanska, J. (1987). About the joint action by ants: Collaboration or participation? *Panstwowe Wydawnictwo Naukowe*, 36:61–75.

Durfee, E. H. (2001). Scaling up agent coordination strategies. *IEEE Computer*, 34(7):1–8.

Estrin, D., Culler, D., Pister, K., and Sukhatme, G. (2002). Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69.

Forrest, S., Balthrop, J., Glickman, M., and Ackley, D. (2005). Computation in the wild. In Park, K. and Willins, W., editors, *The Internet as a Large-Scale Complex System*, pages 203–227. Oxford University Press.

Gershenson, C. and Heylighen, F. (2005). How can we think the complex? In Richardson, K., editor, *Managing the Complex*, volume 1 of *Managing Organizational Complexity: Philosophy, Theory and Application*, chapter 3, pages 1–14. Information Age Publishing.

Gordon, D. (2002). The organization of work in social insect colonies. *Complexity*, 8(1):43–46.

Guerin, S. and Kunkle, D. (2004). Emergence of constraint in self-organizing systems. *Journal of Nonlinear Dynamics, Psychology, and Life Sciences*, 8:2:131–146.

Heylighen, F. (1991). Modelling emergence. *World Futures: The Journal of General Evolution*, 31:89–104.

Hogg, T. and Huberman, B. A. (1991). Controlling chaos in distributed systems. *IEEE Transactions on Systems, Man and Cybernetics*, 21:1325–1332.

Hogg, T. and Huberman, B. A. (2002). Dynamics of large computational ecosystems. Technical Report HPL-2002-77, Information Dynamics Laboratory, Hewlett-Packard Laboratories, Palo Alto.

Kay, J. J. (1984). *Self-Organization In Living Systems*. PhD thesis, Deparment of Systems Design Engineering, University of Waterloo.

Keil, D. and Goldin, D. (2004). Modelling indirect interaction in open computational systems. In *Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 371–377, Washington, DC, USA. IEEE Computer Society.

Kephart, J., Chan, H., Das, R., Levine, D., Tesauro, G., Rawson, F., and Lefurgy, C. (2007). Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*, page 24, Washington, DC, USA. IEEE Computer Society.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *IEE Computer*, 36(1):41–50.

Kephart, J. O. and Das, R. (2007). Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1):40–48.

Kitano, H. (2000). Robocup rescue: A grand challenge for multi-agent systems. In *ICMAS '00: Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, pages 5–12, Washington, DC, USA. IEEE Computer Society.

Ladley, D. and Bullock, S. (2004). Logistic constraints on 3d termite construction. In Dorigo, M., Birattari, M., Blum, L. M., Mondada, F., and Stützle, T., editors, *Proceedings of the Fourth International Workshop on Ant Colony Optimisation*, pages 178–189. Springer, Berlin.

Ladley, D. and Bullock, S. (2005). The role of logistic constraints on termite construction of chambers and tunnels. *Journal of Theoretical Biology*, 234:551–564.

Levins, R. (1966). The strategy of model building in population biology. *American Scientist*, 54:421–431.

Merkle, D. and Middendorf, M. (2004). Dynamic polyethism and competition for tasks in threshold reinforcement models of social insects. *Adaptive Behavior*, 12(3-4):251–262.

Mesarovic, M., Sreenath, S., and Keene, J. (2004). Search for organizing principles: Understanding in systems biology. *Systems Biology*, 1(1):19–27.

Noble, J., Di Paolo, E. A., and Bullock, S. (2001). Adaptive factors in the evolution of signalling systems. In Cangelosi, A. and Parisi, D., editors, *Simulating the Evolution of Language*, pages 53–78. Springer, Heidelberg.

Packard, N. H. (1988). Adaptation toward the edge of chaos. In Kelso, J., Mandell, A., and Shlesinger, M., editors, *Dynamic Patterns in Complex Systems*, pages 293–301. World Scientific.

Pynadath, D. and Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423.

Robinson, G. E. (1992). Regulation of division of labor in insect societies. *Annual Review of Entomology*, 37:637–665.

Saffre, F. and Shackleton, M. (2008). "embryo": An autonomic co-operative service management framework. In Bullock, S., Noble, J., Watson, R., and Bedau, M. A., editors, *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, pages 513–520. MIT Press, Cambridge, MA.

Saffre, F., Tateson, R., Halloy, J., Shackleton, M., and Deneubourg, J.-L. (2009). Aggregation dynamics in overlay networks and their implications for self-organized distributed applications. *Computer Journal*, 52(4):397–412.

Savit, R., Manuca, R., and Riolo, R. (1999). Adaptive competition, market efficiency, and phase transitions. *Physical Review Letters*, 82:2203–2206.

Seeley, T. D. (2002). When is self-organization used in biological systems? *Biological Bulletin*, 202:314–318.

Sen, S., Roychowdhury, S., and Arora, N. (1996). Effects of local information on group behavior. In *Proceedings of the Second International Conference on Multi-Agent Systems*, pages 315–321. AAAI Press, Menlo Park, CA.

Shehory, O. and Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101:165–200.

Simon, H. A. (1956). Rational choice and the structure of the environment. *Psychological Review*, 63(2):129–138.

Stein, S., Jennings, N. R., and Payne, T. (2008). Flexible service provisioning with advance agreements. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, volume 1, pages 249–256.

Stone, P. and Veloso, M. (2000). Layered learning and flexible teamwork in RoboCup simulation agents. In Veloso, M., Pagello, E., and Kitano, H., editors, *RoboCup-99: Robot Soccer World Cup III*, pages 495–508, London, UK. Springer, Berlin.

Sycara, K., Decker, K., and Williamson, M. (1997). Middle-agents for the internet. In Pollack, M. E., editor, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 578–583. Morgan Kaffman.

Tanenbaum, A. and Renesse, R. V. (1985). Distributed operating systems. *ACM Computing Surveys*, 17(4):419–470.

Tesauro, G. (2007). Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11(1):22–30.

Théraulaz, G., Bonabeau, E., and Deneubourg, J. (1998). Response threshold reinforcement and division of labour in insect societies. *Proceedings of the Royal Society of London, Series B*, 265:327–332.

Wang, M., Kandasamy, N., Guez, A., and Kam, M. (2007). Distributed cooperative control for adaptive performance management. *IEEE Internet Computing*, 11(1):31–39.

Zambonelli, F., Mamei, M., and Roli, A. (2002). What can cellular automata tell us about the behavior of large multi-agent systems? In Garcia, A. F., de Lucena, C. J. P., Zambonelli, F., Omicini, A., and Castro, J., editors, *Software Engineering for Large-Scale Multi-Agent Systems, Research Issues and Practical Applications*, pages 216–231. Springer.

Zambonelli, F. and Parunak, H. V. D. (2001). Signs of a revolution in computer science and software engineering. In *Second Italian Workshop on Objects and Agents*, pages 1–12. Springer, Heidelberg.

# List of Figures

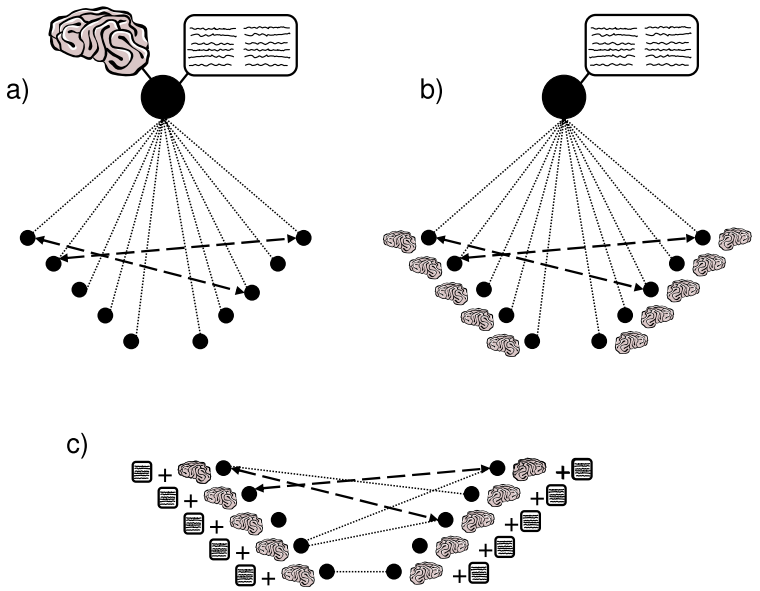



Figure 1: Three classes of control organisation: a) centralised control, b) distributed control reliant on consensual, up-to-date, global information, and c) fully decentralised control. Service providers and consumers are represented by small circles, central executives or central repositories by large circles. Agents may store information (lozenges) and/or execute co-allocation algorithms (brains). Dotted lines connote information exchange, whereas dashed lines connote the pairing of services and resources achieved by the co-allocation process. Hybrid classes of control architecture may combine elements of centralised, distributed and decentralised control.

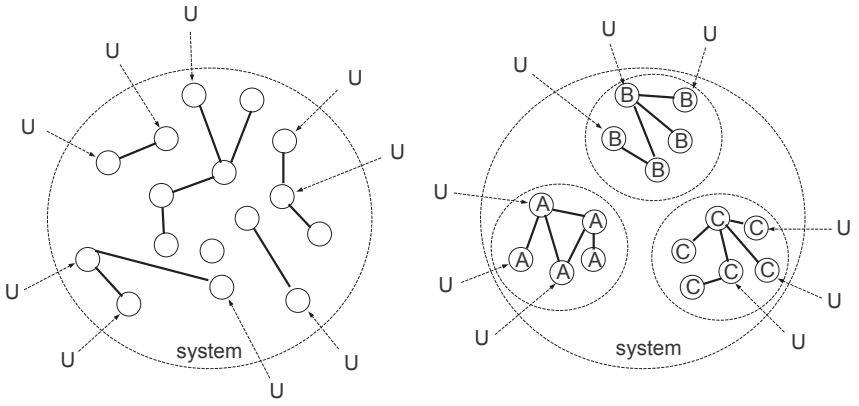



Figure 2: An overview of the resource management organisation process. Consumers ($U$) impose a demand for different types of resources (service requirements) on resource providers (small circles). The inefficient disorganised configuration (left) represents the case in which providers have no knowledge of what services are in demand, and consumers don't know which providers offer their desired services. The organised configuration (right) represents the case in which service demand is satisfied by local supply.

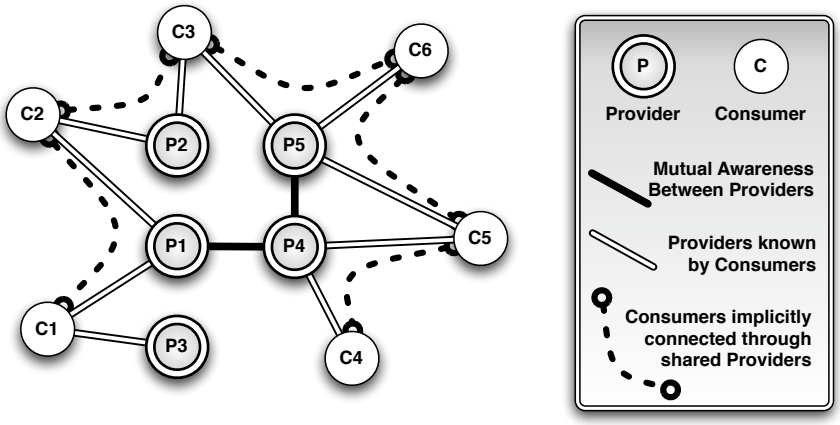



Figure 3: The system viewed as a network of peers limited in knowledge about other agents by the maximum size of their memory (in this example each peer is aware of only two other agents). Consumers do not interact directly with other consumers; however, they can indirectly affect each other's behaviour through the information they rely on and share with commonly known providers. During the exchange of information between consumers and providers, a network reorganisation process occurs that may lead to the removal or addition of new connections between agents that may consequently become aware of each other's existence, or conversely, loose knowledge about each other. For example, *C1* and *C2* know about the same provider *P1*, through which information about the existence of provider *P3* can be passed to *C2*, thus forming two new links between *P3* and *P1* and between *P3* and *C2*.

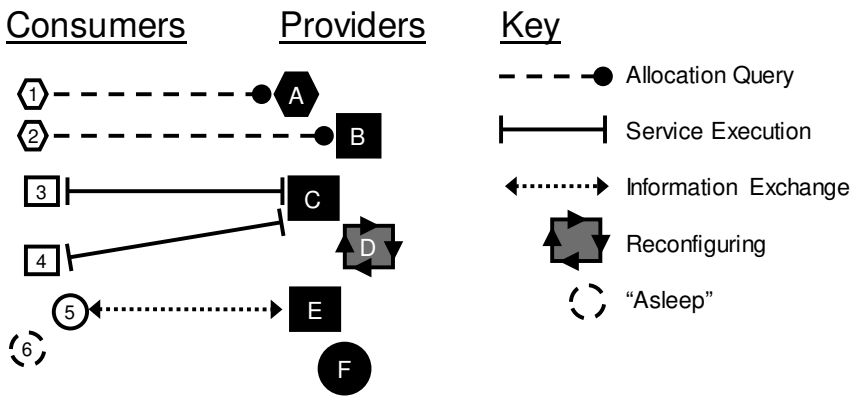



Figure 4: Several different types of service (represented by the three different polygons: *square*, *circle*, and *hexagon*) are required by service consumers (open polygons numbered *1* to *5*) and offered by service providers (solid polygons labelled *A* to *F*). Consumers *1* and *2* query providers *A* and *B*, respectively, requesting the same type of service (i.e., type *hexagon*). Provider *B* is not configured to offer this service type (it currently provides *square* services), whereas provider *A* could satisfy the request, provided it is available and has spare capacity. Consumers *3* and *4* have provisioned tasks to provider *C*. *D* is busy whilst reconfiguring to offer a different type of service. Consumer *5* is exchanging information with provider *E*, whereas consumer *6* is "sleeping" between jobs. Finally, provider *F* is currently unoccupied, but available to offer services of type *circle*.

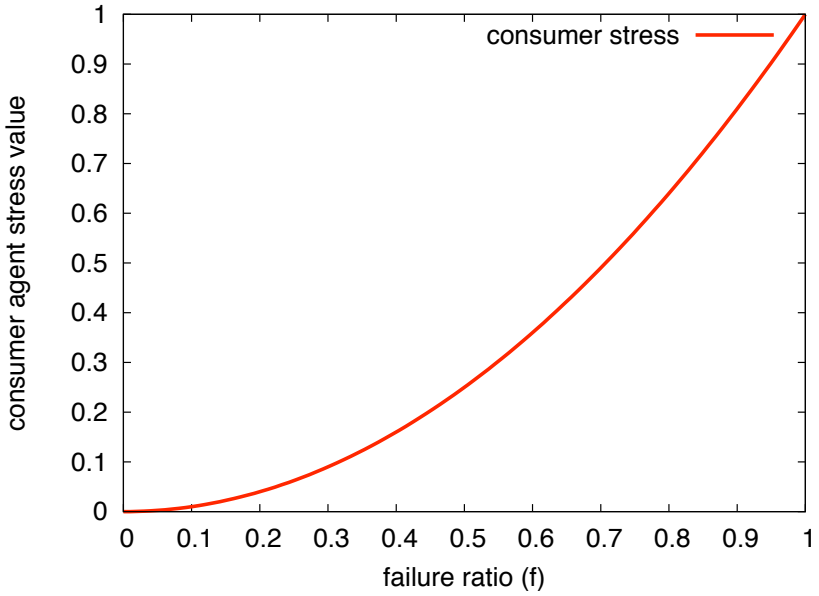



Figure 5: Relation between consumer stress and the failure ratio (f).

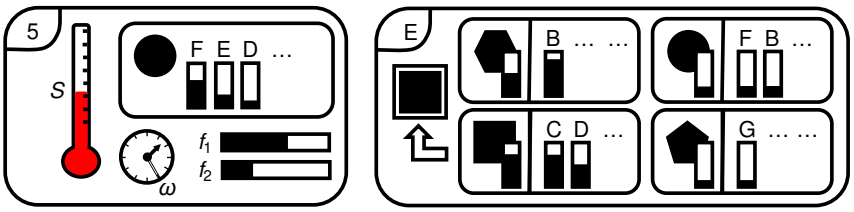



Figure 6: Schematic representation of the internal state held by Consumer *5* (left) and Provider $E$ (right). Consumer *5* experiences a current level of stress, $S$, based on the number of failed queries whilst trying to provision the previous two services (*f1* and *f2*, respectively). Consumer 5 also retains rankings (i.e., *bias*) for $m_c$ known providers (in this case, the providers $D$, $E$ and $F$, all of which are believed to provide services of type *circle*). Between jobs, Consumer 5 sleeps for some random period drawn from the uniform distribution $[0, \omega]$. Provider $E$ is currently configured to perform service (*square*), and maintains estimates of the demand for all known service types, plus *bias* estimates for all known service providers, organised by type of service offered.

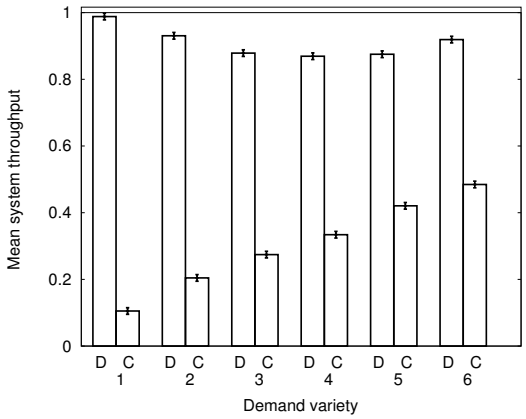



Figure 7: Mean throughput of jobs as a proportion of optimal throughput (horizontal line) for decentralised (D) and centralised (C) resource management with differing levels of demand variety, $V$. For the centralised resource management strategy, throughput declines with decreasing $V$ as a consequence of the increasing congestion created by herding. For decentralised resource management, throughput is near optimal for $V = 1$ and remains high where more than one variety of service is required by the consumer population. Each bar represents the mean throughput for the final 600 seconds of 10 independent simulation runs. Error bars represent standard deviations about the mean.

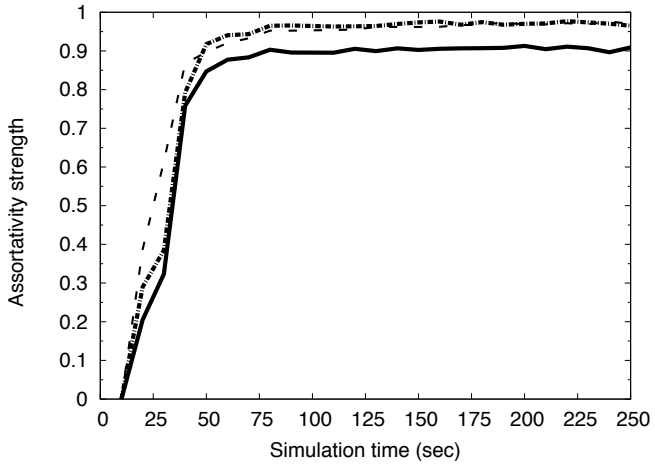
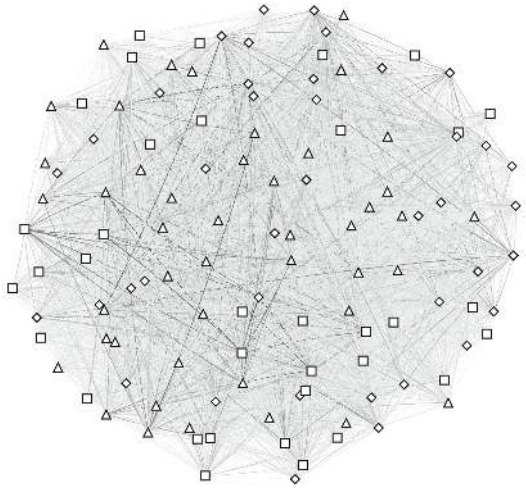
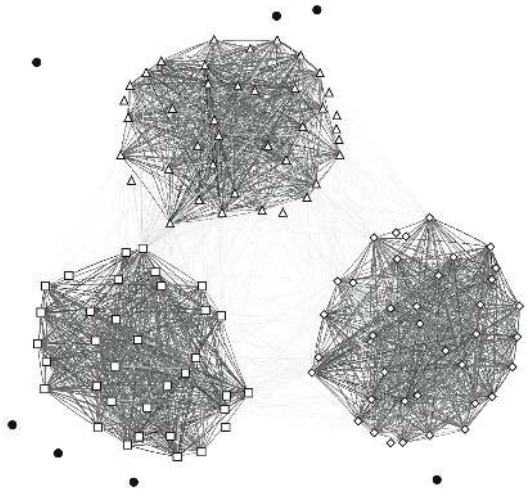



Figure 8: Strongly assortative community structure is achieved rapidly. The rate of convergence is independent of demand variety. Each curve represents a single representative run: where $V = 3$ (dotted line), $V = 4$ (dashed line) and $V = 6$ (solid line).

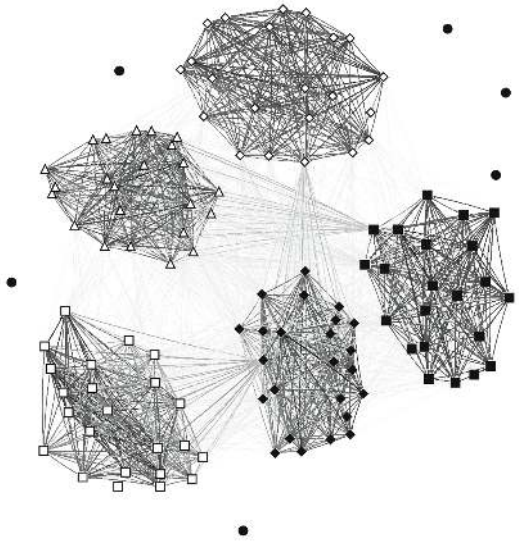


(a) At t=20s, there is no convergence

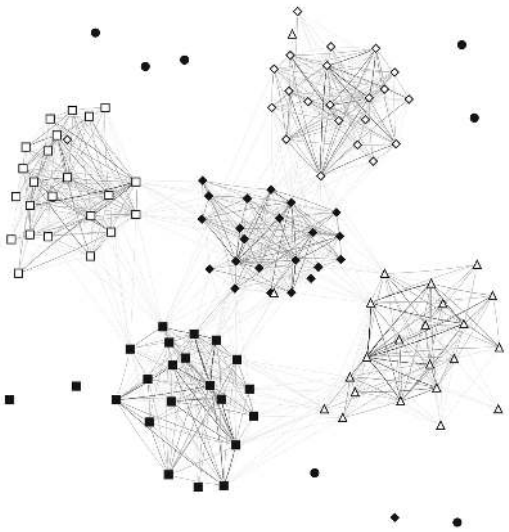


(b) At t=250s, communities start to converge

Figure 9: Network representing the organisation of a model with $V = 3$ at different times in the simulation, illustrating the formation of communities. Links indicate relations between providers offering the following service types: $A$ (box), $B$ (diamond), and $C$ (triangle). Solid circles are singleton nodes, i.e., each is a provider that is only known by one consumer.



(a) Consumer Information Only



(b) Provider Information Only

Figure 10: Networks representing the organisation of a model with $V = 5$ at $t = 250s$, based on (a) the consumers' service registries ($\mathcal{R}_c$), and (b) the provider's registries ($\mathcal{S}$). The links indicate relations between providers offering the following service types: $A$ (box), $B$ (diamond), $C$ (triangle), $D$ (solid box) and $E$ (solid diamond). Solid circles are singleton nodes, i.e., each is a provider that is only known by one consumer.

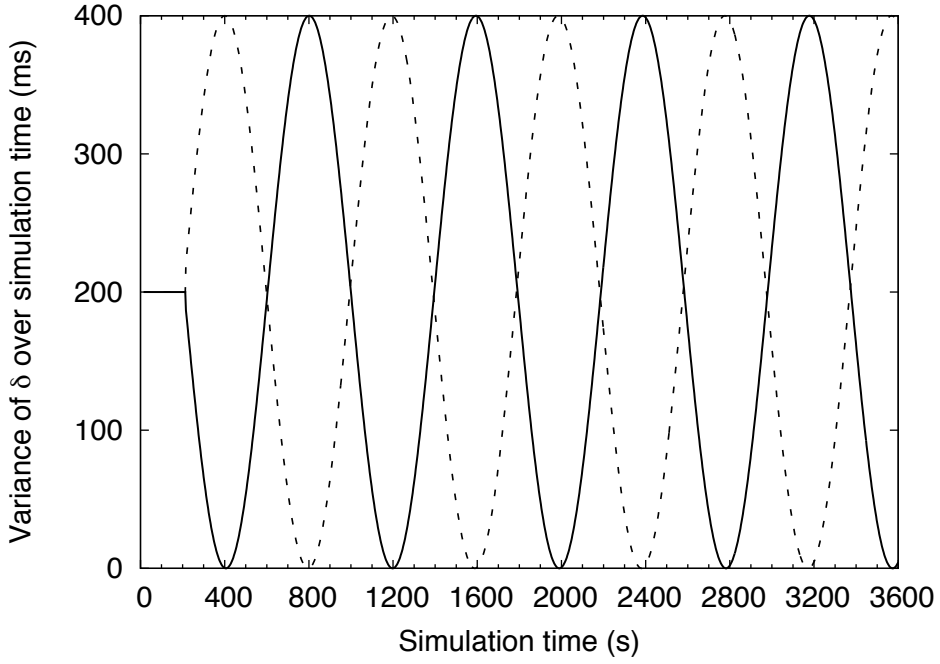



Figure 11: Variance of $\delta_{type}$ parameter stimulating the change in demand of services, for type $A$ (dotted line) and $B$ (continuous line).

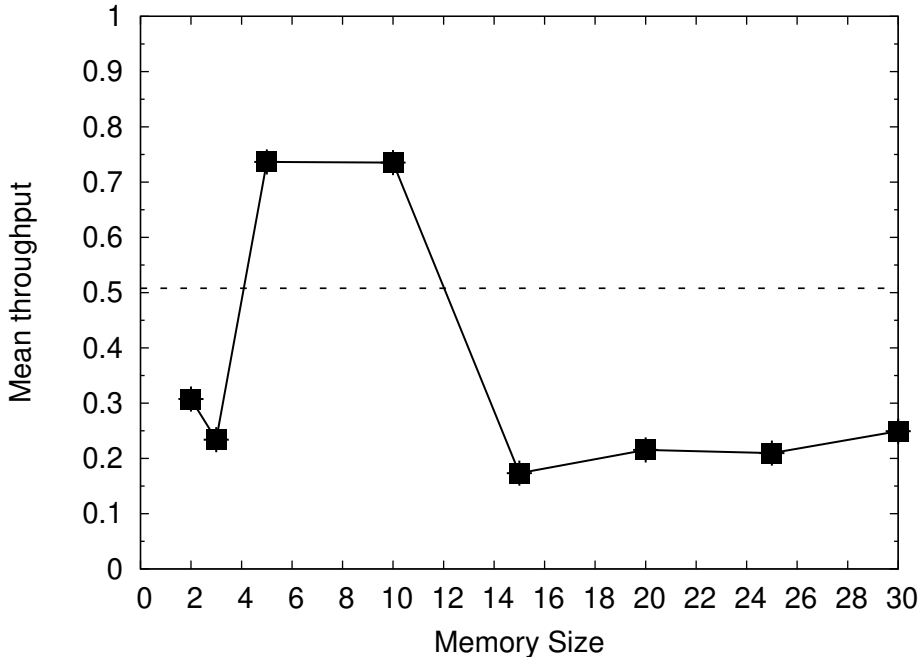



Figure 12: The mean system throughput for different memory retention sizes (i.e., varying $m_c$). The dotted line corresponds to the case where $m_c = \infty$.

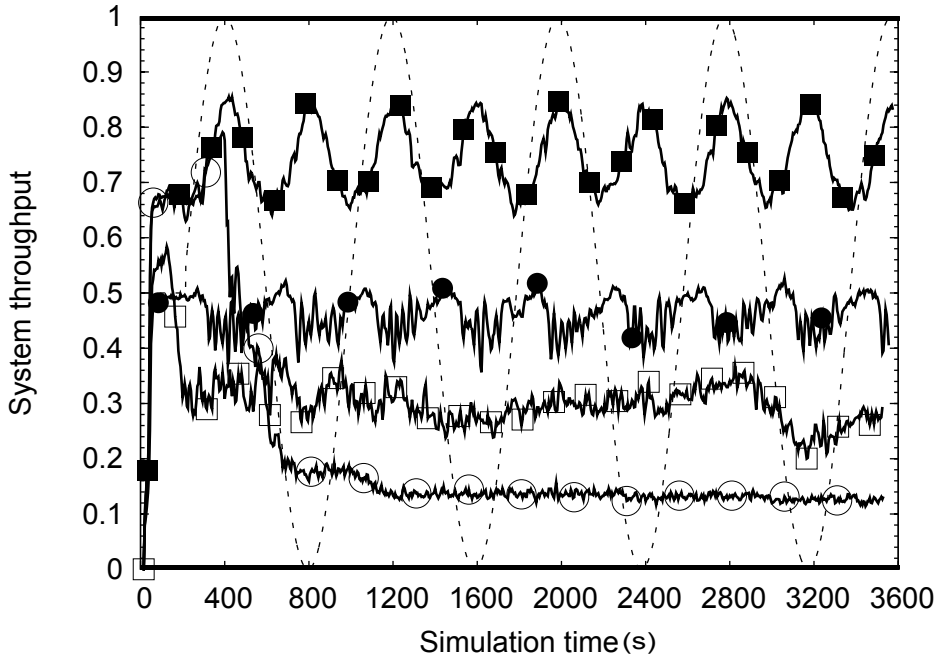



Figure 13: The performance (i.e., system throughput) of four system configurations with differing memory sizes: $m_c = 2$ (empty rectangles), $m_c = 5$ (solid rectangles), $m_c = 20$ (empty circles), and $m_c = \infty$ (solid circles).

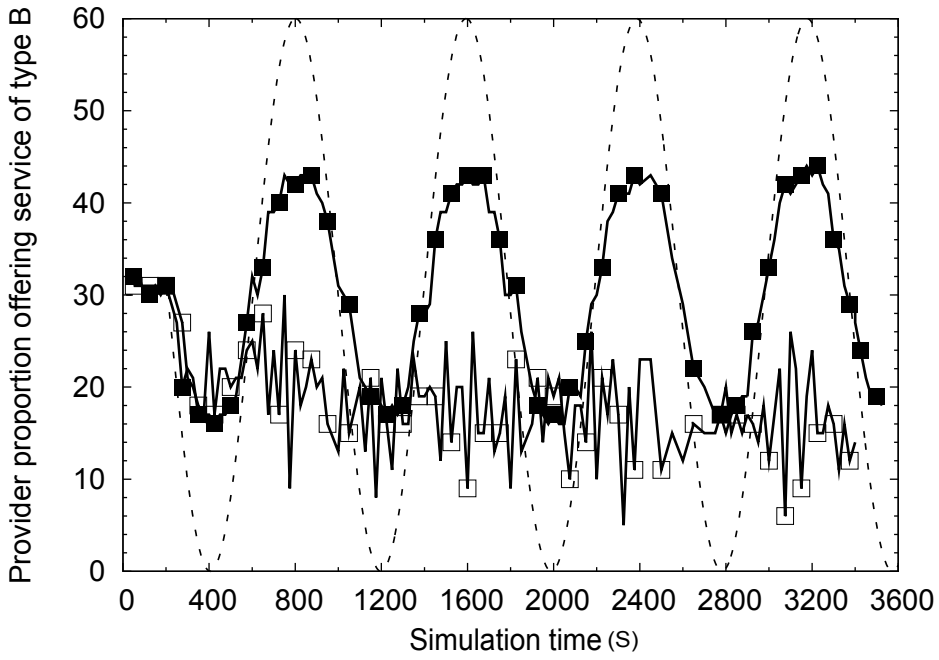
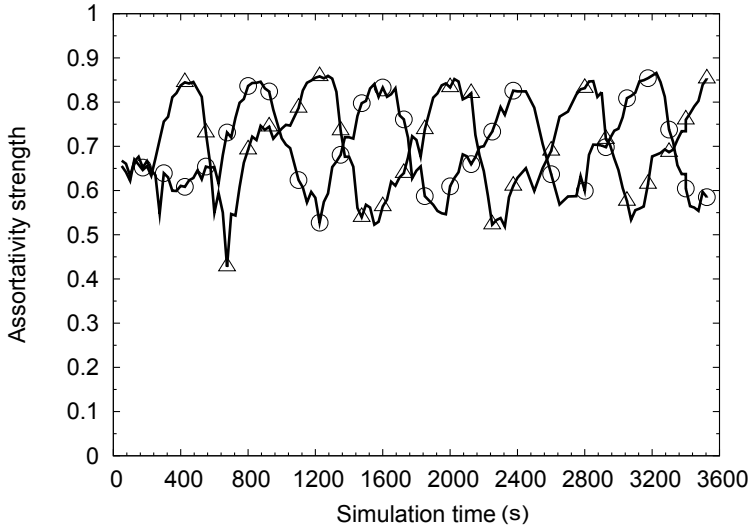
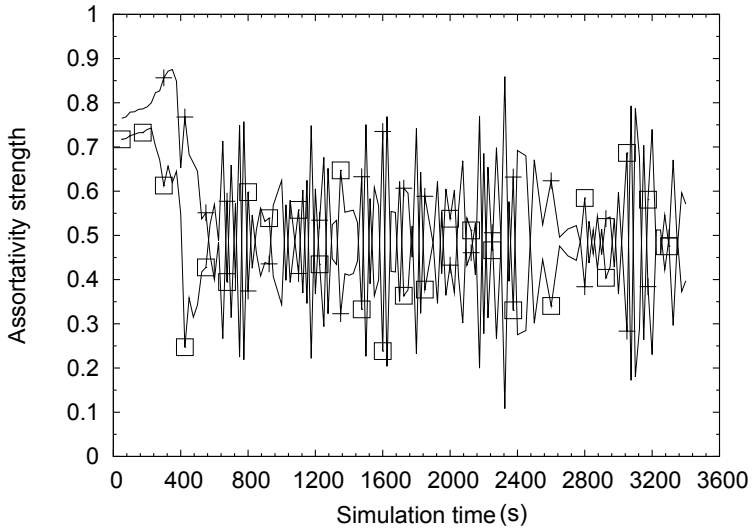



Figure 14: The number of resources currently offering service of type $B$ with respect to changing demand for $B$ (dotted lines) for two system configurations: solid rectangles ($m_c = 5$) and empty rectangles ($m_c = 20$).




(a) Small memory model ($m_c = 5$)




(b) Large memory model ($m_c = 20$)

Figure 15: The assortativity strength for communities providing service types $A$ and $B$ over time. The triangles represents community strength for service community $A$ (where $m_c = 5$), whereas the circles represent the corresponding community strength for community $B$. The rectangles and crosses denote the corresponding populations (for $A$ and $B$) when ($m_c = 20$).

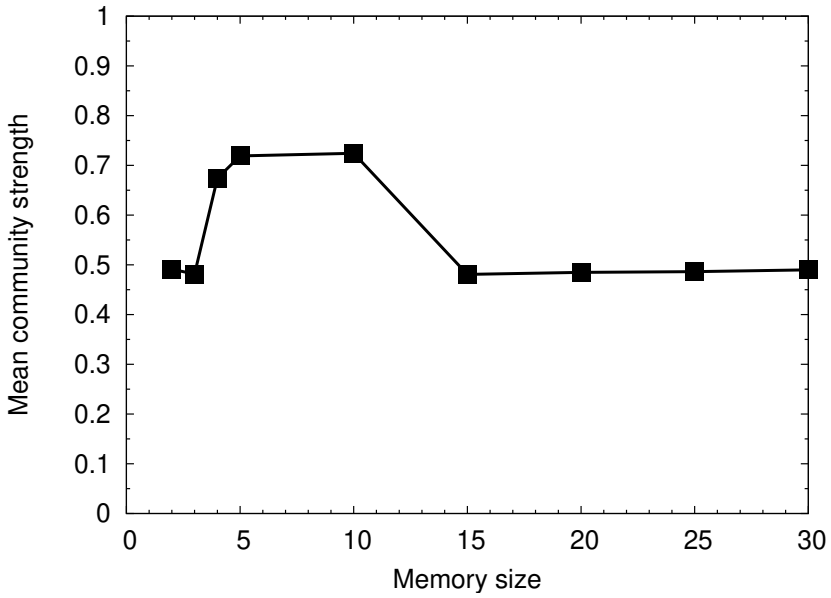



Figure 16: The mean community strength as a function of memory size.

# List of Tables

| Parameter | Value | Description |
|---|---|---|
| $T_q$ | 50ms | Time taken to query a provider |
| $T_i$ | 50ms | Time taken to complete each knowledge exchange phase |
| $T_e$ | 500ms | Time taken to satisfy a service request |
| $T_s$ | 2s | Time taken for a provider to perform a *switch* operation |
| $\omega$ | 50ms | Typical upper bound for the sleep distribution |
| $C_{type}$ | 2 or 5 | Number of services that can simultaneously be honoured |
| $f_{max}$ | $2 \times m_c$ | maximum number of query attempts a consumer will make when provisioning a service |
| $m_s$ | $m_c$ | Number of tuples retained by a provider for each service type |
| $\delta_{bias}$ | 0.1 | The *bias* increment coefficient |
| $\delta_{decay}$ | 0.9 | Decay coefficient is used to allow stale information to decay |
| $\delta_{update}$ | 0.1 | Update coefficient, used to update provider bias based on $c_s$ |

Table 1: The parameters used to configure the evaluation (unless otherwise stated).

| $V$ | Set of service types ($Capability$) | Number of Consumers | Number of Providers | Service Capacity $C_{type}$ |
|---|---|---|---|---|
| 1 | $\{A\}$ | 600 | 120 | 5 |
| 2 | $\{A, B\}$ | 600 | 120 | 5 |
| 3 | $\{A, B, C\}$ | 600 | 120 | 5 |
| 4 | $\{A, B, C, D\}$ | 600 | 120 | 5 |
| 5 | $\{A, B, C, D, E\}$ | 600 | 120 | 5 |
| 6 | $\{A, B, C, D, E, F\}$ | 600 | 120 | 5 |

Table 2: A variety of different sets of service types for different *demand variety* settings. Note that 600 consumers each requiring one service can, in principle, be satisfied exactly by 120 providers each with a capacity 5.

| $V$ | Set of service types ($Capability$) | Number of Consumers | Number of Providers | Service Capacity $C_{type}$ |
|---|---|---|---|---|
| 2 | $\{A, B\}$ | 120 | 60 | 2 |

Table 3: Parameters used for the scenario when service demand is dynamic.