An infrastructure for developing self-organising services

Kieran Greer*, Matthias Baumgarten, Maurice Mulvenna and Chris Nugent

School of Computing and Mathematics,
Faculty of Computing and Engineering,
University of Ulster, Jordanstown Campus,
Shore Road, Newtownabbey,
Co. Antrim, BT37 0QB, UK
E-mail: krc.greer@ulster.ac.uk
E-mail: kieran.greer@ntlworld.com
E-mail: m.baumgarten@ulster.ac.uk
E-mail: md.mulvenna@ulster.ac.uk
E-mail: cd.nugent@ulster.ac.uk
*Corresponding author

Kevin Curran

School of Computing and Intelligent Systems,
Faculty of Computing and Engineering,
University of Ulster, Magee campus,
Northland Road, Londonderry, BT48 7JL, UK
E-mail: kj.curran@ulster.ac.uk.

Abstract: This paper describes a software package named ‘licas’ (lightweight internet-based communication for autonomic services) that can be used to build networks of service-based components that have the potential to behave autonomously and self-organise through stigmergic links. Discussed will be how these components are able to self-adapt inside of this infrastructure and also show potential for more intelligent behaviour. The links allow for some reasoning possibilities, both as a result of the links themselves and through monitoring the links for optimisation. One research area that appears to be particularly related to this package is that of complex adaptive systems (CASs) and associated features will be described. Comparisons will also be made with the existing established technologies in the area of service-oriented architectures (SOAs).

Keywords: infrastructure; software; self-organise; stigmergic linking; reasoning; autonomic services.

Biographical notes: Kieran Greer carried out this work in the Faculty of Computing and Engineering at the University of Ulster. He completed a DPhil in 1998 in the area of Artificial Intelligence (neural networks and search heuristics) and has worked on several European research projects in the area of distributed information systems. His main research interests are artificial intelligence (especially autonomous/agent-based or neural systems), distributed information systems and scheduling. He has a strong software development background where he has designed and implemented three innovative software systems. He has (co)authored around 40 publications.

Matthias Baumgarten is currently working as a Research Fellow in the Faculty of Computing and Engineering at the University of Ulster. His main duties include project preparation and administration as well as research related tasks within his area of expertise. He is mainly interested in the area of data mining/web mining and has been involved in several European research projects. His main research interests are in the field of web mining, sequential pattern discovery, distributed knowledge systems and personalisation techniques. He has received his PhD in 2005 and has so far (co)authored about 20 publications.

Maurice Mulvenna received his BSc Honours degree in Science (1988) and MPhil Information Systems (1997) at University of Ulster. He is currently a Senior Lecturer in Computing Science at the University. He is an Editorial Review Board Member for the International Journal of Ad Hoc and Ubiquitous Computing, winner of 2001 European 1st Grand Prize and grant holder on 1ST-BONUS, CASCADAS, COGKNOW European projects and Nestling Technologies project. He has over 100 publications in the research area of artificial intelligence and pervasive computing. He is a member of IEEE and ACM, and a chartered member of the BCS.

Chris Nugent received his Bachelor of Engineering in Electronic Systems and DPhil in Biomedical Engineering both from the University of Ulster. His research within biomedical engineering addresses themes of technologies to support independent living, medical decision support systems and the development of internet based healthcare models.

Kevin Curran, BSc (Hons), DPhil, SMIEEE, MBCS CITP, MACM, MIEE, FHEA is a Lecturer in Computer Science at the University of Ulster. He has published in the field of distributed computing especially emerging trends within wireless ad-hoc networks, dynamic protocol stacks and middleware. He is a member of the editorial board of a number of journals including The Journal of Universal Computer Science (J.UCS) and he is also a member of numerous international conference organising committees.

1 Introduction

This paper describes a lightweight software package that can be used to build distributed service-oriented systems. The lightweight architecture is particularly useful in the emerging pervasive or sensor-based environment. The architecture is open, allowing services of any type to be loaded into the system. The individual components thus need to be autonomous and provide their own security checks when deciding to communicate with other components, or act as hosts for other services. It is also possible for the
components to implement agent-based features, such as low-level communication tracing. An autonomous service would also benefit from the capability of running on a separate thread and so this is also provided. The communication protocol is an XML-based remote procedure calling (RPC) mechanism, roughly similar to that provided by the Apache XML-RPC framework (Apache). This system has been developed however to add some additional features to the Apache system. Self-optimisation through stigmergic linking is the main adaptive area provided by the system. Stigmergy (for example, Dorigo et al., 2000), could be defined as a mechanism where agents perform actions based on reactions to their environment rather than through any internal knowledge. In the context of this work, links are formed through the experience or use of the network rather than through a knowledge-based algorithm. A self-organising mechanism is provided that uses this linking structure to link associated components with each other. While this has been tested with respect to linking through the results of queries, the linking mechanism is flexible and could be applied to many different scenarios. Thus, depending on the functionality built on top of the framework, different behaviours or patterns could emerge.

Central to the intentions of this framework are autonomic computing, bio-inspired techniques such as stigmergy and service-oriented architectures (SOAs). Combining SOA with autonomic computing can lead to autonomic services, which are services that can autonomically manage themselves. Autonomic computing (IBM, 2003) is a philosophy that tries to add a kind of artificial nervous system to a computing system, so that it can monitor and manage itself. With the increasing size and complexity of networks, centralised or human-based management is becoming impossible and so components in the network will need to be able to manage themselves. This will be done through adding intelligence to the component so that it can understand its internal state and situation in its environment. Typical autonomic features include self-configuration, self-optimisation, self-healing and self-protection. The ‘licas’ system (lightweight internet-based communication for autonomic services) provides in-built capabilities for self-optimisation. There are also reasoning capabilities that can be included as part of the linking process, derived from the linking process, or as part of an autonomic supervision system. This system has comparisons with existing web-based technologies and also with complex adaptive systems (CASs), which will be made in the following sections.

The rest of the paper is structured as follows. Section 2 describes the licas software package that is the main contribution of this paper. Section 3 describes CASs. Section 4 describes SOAs and related technologies. Section 5 compares the licas system with other systems. Section 6 gives one example of where the system has been used. Section 7 considers self-adaptation and reasoning possibilities using the infrastructure and Section 8 gives some conclusions on the work.

2 The licas system

A software package called licas has been written and is available as open source code on sourceforge.net (licas). It is written in Java (Java) and can be used as a framework to build peer-to-peer (p2p) networks of autonomic services. The network is internet-based in the sense that you can start an HTTP server running on a computer that can communicate with other servers on other computers through the internet. The lightweight framework would allow mobile devices to run the software and thus also connect to the network.
Currently, technical issues such as the reflection package being missing from Java ME make dynamic method invocation impossible for mobile devices. However, a client-side device only needs to be able to parse to/from a string to send/receive a message from the server. The method invocation on the server side is done through reflection, which could be a more heavyweight component. Thus, lightweight mobile devices could use the network as clients, even in today’s conditions. Each server can store any number of user-provided components (nodes) that can be organised in any manner required. As the network is service-based, each node can extend a base ‘Service’ class provided by the package. This class can then store other services, so that a nested or hierarchical structure of services operating on other services can be built up. Figure 1 shows the basic network architecture, with distributed servers storing nodes organised in a hierarchical manner. A client would access one server that would have access to other servers running in the network. Thus, through one entry point multiple servers can be accessed. In a pure definition of a p2p network, the nodes function as both ‘clients’ and ‘servers’ to the other nodes on the network. In licas, there is one server for several services that receives the remote calls and passes them to the referenced nodes. These can then act as clients to other servers, but do not act as servers themselves. So there is thus, just one entry point for many services and each node is not strictly both a server and client. However, the p2p definition is flexible and the architecture that has been implemented is not new.

Figure 1  Basic network architecture

The service class extends the Java ‘Thread’ class, so that each service can run independently. An ‘Auto’ class is also provided that extends the service class but provides more agent-like functionality, such as the storing of communication IDs. The system is open and security is the responsibility of the programmer, but some level of protection in the form of passwords is provided. Each service component is initialised with two passwords. One password should be known by the administrator of the service only. This is used to load, start and remove the service. The other password needs to be known by any other component that wishes to use the service. This gives some other component permission to access the service and thus invoke one of its methods. If using...
threads, the ‘run’ method needs to be implemented and thread synchronisation also needs to be managed, although a monitor class to synchronise on is provided.

The service class also requires two other abstract methods to be implemented to realise full functionality. The first of these is a method that some other component can call to ask for the service’s password. This method is passed a description of the calling component and it must then decide if it wants to return its password. The service’s decision to return its’ password is application dependent and must be implemented by the programmer. However, handling of the passwords can be complex, especially when tracing through nested services and so routing a request to the correct service and comparing the entered password with the service’s password is automatically done by the system. The second method is more agent-based. A component can pass a communication ID and an object representing some result to another service. The other service can then store its current communications with their states and process the result as required. This provides a simple call back mechanism, thus allowing for some degree of asynchronous processing and tracing of communications between services.

2.1 Communication protocol

The communication protocol itself is an XML-based RPC mechanism. It is very similar to the Apache, but is built on the Apache Jakarta HttpCore package. The new system has been developed so that some extra features can be added to the XML-based RPC mechanism. For example, objects can exist permanently and allow for nested or hierarchical services. Implementing the communication mechanism requires a parser for each complex object that is used. Parsers are provided for both DOM (Java) and JDOM (JDOM) XML elements, Vectors and Hashtables. The lists can store nested objects of simple types, or types provided by the default parsers or user provided parsers. The user must write the parser for his own complex object and it must implement a parser interface with methods to serialise the object into XML or to parse the XML back into the object. For remote communication, the message passing mechanism then uses these to generate XML that is then converted into a string and passed to the called server. That server then converts the message back into the appropriate objects and passes it to the specified service. Alternatively, the object to be passed can extend the ‘serialisable’ interface, when it will be serialised into Base64 (Harder) and passed in that format. Parsing lists of complex objects or JDOM would be extensions to the Apache default parsers. Local calls do not parse, but pass existing objects by reference, through the same message passing procedure. This is also different to XML-RPC, although that is probably only intended to be used for remote communication. The services would also not be permanently running, but re-created for each call, which could also cause problems.

2.2 A description of a method call

There are a number of classes that are used to construct a method call. A ‘CallObject’ class is used to actually make the call. It determines if the call is local or remote by being passed a full path description of the component to call. If it is a remote call, it creates an RPC client, parses the method information and passes it to the appropriate server. If the call is local, then the object can be referenced directly and passed the information in the form of the existing objects. If the method call indicates to call an object through a path description rather than through a direct object reference, the path through the local
objects can be traced and the specified service invoked again with the referenced parameter objects. If the path is remote however, then the objects are parsed and passed through the RPC mechanism. The method description is put into a ‘MethodInfo’ class. This class stores the name of the object to call and the name of the method on that object. The method description also records the return parameter type and stores any number of input parameters in ‘ParamInfo’ objects. The passwords used to access the remote server and called object also need to be included. For agent-like communication you can also pass a communication ID. The CalObject class then accesses the server (remote call) or object (local call) indicated and passes to it the appropriate information. If the object reference is a path, the string description includes the URL of the server to call and then a path of nested/hierarchical objects on that server to access the desired service. For example, the reference for the object to call might look something like:

Object reference path: http://127.0.0.1:8888;service1.service12

If the server that the calling object is on, recognises the URL as it’s own, then it can identify the call as a local one, otherwise it can consider it to be a remote call. The MethodInfo class would then specify the service on the final object to call and the method name. For example the method description might be:

Method name: service121.getData

Method return type: String

Method password: anon

Method parameter: ‘Big data’

A call is then made through the CalObject class in the form:

CalObject.call (object reference, method info)

The package allows for remote class loading through a remote JAR class loader. This allows a user to load any object into the system. The user needs to specify the URL of the JAR file that can be remotely accessed and contains the classes to load. If it extends the service class, it can be stored as it is. If not, then it will be wrapped into a service wrapper and stored in that format. Java reflection can then be used to access its methods through the wrapper class. While communication or loading of a service require a password, once a service is loaded, there are no restrictions on what it can do.

2.3 The linking package

The licas package also includes a package that allows a component to store links to other components. The linking structure itself and how it works are described in more detail in Greer et al. (2008). A link is defined as a reference from one component to another that has a weight value attached to it. Linking structures are stored in nested Hashtables that are keyed by the descriptions of the concepts that led to the reference being created. The keywords, for example, could relate one component to the other in a query. The link update method is flexible with the number of keywords to include and so the whole key can be as complex as desired. The link itself is then stored in an object that records the full path reference to the linked service and the related weight value. It also provides a timestamp of when the link was last updated. The weight value defines at what level the
link exists in a linking structure. It is only the references in the top level that are returned as links when the component is queried. Thus, the referenced source must be consistently associated with the linking source before the link is considered as reliable. The current implementation limits the linking structure to three levels, where each new level has a threshold value that must be met before a link reference is moved there. The three level structure is more flexible than a two level one, because it can help with resource allocation, or may allow for more flexible monitoring and link updating. The following theory has not been proven, but explains why a three level structure could be useful. The bottom layer can update links as a result of a whole process, for example, a whole query answer. A middle layer could then update based on associations between just the two components concerned. If an association is found to be possible, then it may be better to monitor the individual relation between the two components after that. From the second level onwards, the links would then be updated based on parts of a process and not the whole process. A three level structure however should not make it more difficult for links to reach the top level, compared to a two level structure.

Methods exist to increment or decrement the weight values of the links so that they move up or down the levels in the linking structure. The linking mechanism also allows for a maximum number of references at each level to be specified, when this number of references cannot be exceeded. If a level becomes full, it must first remove a reference before another can be added, but this is managed automatically. Limiting memory means that the structure can remain lightweight. There are also options to allow sources to borrow memory from each other. This actually means borrowing a certain number of entries with which to store extra references. The current implementation limits this to borrowing only between sibling services of a parent service. One component can ask for memory from any others that are its siblings. If the sibling services have a number of unused entries, they can lend it to the requesting component. They reduce their maximum allowed number of entries by the borrowed amount, while the borrowing service can increase its allowed allocation by the same amount. However, if a component then runs short of memory it can ask for its borrowed entries back, when the component that borrowed the entries must then give them back. This is also automatically implemented and can be included as a feature through a keyword description of the linking method. The idea of allowing components to borrow memory is to produce a better distribution of the memory allocation, or resource allocation, so that more heavily resourced components can acquire more memory while less resourced ones can release memory that they do not use. While the linking structure arose out of looking at stigmergic principles, it could also be called Hebbian.

3 Complex adaptive systems

The term CAS (or complexity science), is often used to describe the loosely organised academic field that has grown up around the study of such systems. Complexity science encompasses more than one theoretical framework and is highly interdisciplinary, seeking the answers to some fundamental questions about living, adaptable and changeable systems. A CAS is a collection of self-similar agents interacting with each other. They are complex in that they are diverse and made up of multiple interconnected elements and adaptive in that they have the capacity to change and learn from experience.
One definition of CAS by John Holland, one of the founders of this science, is given in Waldrop (1993).

“A CAS is a dynamic network of many agents (which may represent cells, species, individuals, firms, nations) acting in parallel, constantly acting and reacting to what the other agents are doing. The control of a CAS tends to be highly dispersed and decentralised. If there is to be any coherent behaviour in the system, it has to arise from competition and cooperation among the agents themselves. The overall behaviour of the system is the result of a huge number of decisions made every moment by many individual agents.”

Al-Obasiat and Braun (2007) describe that CAS share common properties and methodologies which assist them to survive, evolve and adapt to changes in the dynamic environment. Based on the work of Holland, some of these can be summarised as:

1. an agent is the main work component in the system
2. the agents have simple, primitive constructs, communication language or protocols
3. the agents operate individually according to stimuli-action rule instead of event-condition-action rule
4. the environment consists of a diverse set of randomly distributed devices and services interacting together through communication protocols
5. an existing adaptation mechanism whereby agents can vary their responses to changes.

With complex systems, the interactions between individual components in the system give rise to the emergent behaviour. Emergence is the process of complex pattern formation from simpler rules. An emergent behaviour arises at the global or system level and cannot be predicted or deduced from observing the behaviour of the individual components in the lower level entities. These sorts of emergent behaviours can be realised through bio-inspired techniques such as stigmergy, or swarm optimisation processes. The linking mechanism used in this paper has been developed along the lines of an ant colony optimisation (ACO) algorithm (Dorigo et al., 2000). ACO works by copying the actions of ants as they try to find the optimal route from one position to another. They randomly select a number of routes and leave a pheromone trail behind indicating the route that they took. The shortest route will build up the strongest pheromone amounts and so the other ants will be attracted to this route. The ants do not know what the optimal route is, but rather discover it through the experience of all the routes that all ants take. This does not require any knowledge of the environment, but only to be able to read the pheromone trail. In a network environment, the pheromone trails are represented by links between nodes that can be strengthened or weakened to define optimisation routes. Because this weight updating is calculated through changes in the environment (nodes visited) and not by any cognitive process, it can be called stigmergic.

Adaptive infrastructures should also be able to behave independently, where they make their own decisions on what adaptive measures to take. IBM proposed the ‘autonomic computing architecture’ (IBM, 2003). This architecture consists of a collection of autonomic elements. IBM defined autonomic systems as ‘systems that have the ability to manage themselves and dynamically adapt to changes according to administrative policies and objectives.’ They have independent control systems that can
self-configure, self-heal, self-optimise and self-protect. Self-optimisation is the main area of interest in this paper.

4 Service-oriented architectures

SOAs are an IT approach in which applications rely on services available on a network such as the internet to facilitate business processes. There is no universally agreed upon definition of what a SOA is, but MacKenzie et al. (2006) describes SOAs as requiring the following functionality, which is particularly relevant to the focus of this paper.

“It is natural to think of one person’s needs being met by capabilities offered by someone else. In the world of distributed computing, this could be one computer program’s requirements being met by another computer’s service. This sets up a client-server or p2p situation that can have many-to-one and/or complex relations between the components involved. The perceived value of SOA is that it provides a powerful framework for matching needs and capabilities and for combining capabilities to address those needs. Visibility, interaction and effect are key concepts for describing the SOA paradigm.”

Implementing an SOA can involve developing applications that use services, or are available as services, so that other applications can use them. Typical technologies related to this architecture include web services (Curbera et al., 2002) and the grid architecture (Buyya and Venugopal, 2005; Foster et al., 2002; Talia, 2002). Web services are an attempt to standardise communication across the internet, to allow for computer to computer communication. Through a group of XML-based specifications, different programs can call each other’s services in a machine readable and platform independent way. The grid is a service-based architecture that tries to use the power of several distributed computers to collectively provide services to a user. Grids can integrate heterogeneous machines and provide common resource-access technology and services across widely distributed virtual organisations. Collectively, the services in a grid can thus solve more complex problems. Grid computing differs from conventional distributed computing in that it focuses on large-scale resource sharing, offers innovative applications and, in some cases, is geared toward high-performance systems. The increased power from several computers makes the processing of larger problems possible. However, Foster et al. (2002) state that the emphasis for grids is now on resource sharing in virtual organisations rather than on computing power. If web services can then provide a standard communication protocol for the grid services to communicate with, then computer to computer communication is possible. This integration of the two technologies has already been developed as part of the open grid services architecture (Talia, 2002). Unfortunately, the problem is that while the interface may be standardised, it is still not understood. Thus, the semantic web (Berners-Lee et al., 2001) tries to add understanding to the internet contents so that programs can also understand what they are reading, through the use of global ontologies. While the construction of these ontologies should eventually be fully automatic, at the moment, it is still only semi-automatic and so the semantic web is not yet fully realised. The semantic web has also been combined with the grid to produce the semantic grid. De Roure et al. (2005) describe the concept of the semantic grid and its current state. They describe many of the issues considered in this paper.
Also of importance in this context are agent-based systems (Wooldridge, 2002), from which autonomous systems developed. They have been shown to be particularly useful in a network environment, as search agents for example. With relation to the technologies just mentioned, software agents bring the dynamic decision-making, decentralisation, coordination and autonomous behaviour needed to realise virtual organisations (De Roure et al., 2005). Agent-based systems use standard protocols to communicate with each other, allowing different agents to understand and interact with each other. They also have an internal control mechanism so that they can make decisions by themselves and self-adapt to a changing situation. The licas system provides a framework in which components that will exhibit such features can be built. It also provides a framework to allow the system to adapt. Most of the functionality must be added by the programmer, but the framework is in place to allow this.

5 Comparing licas with existing technologies

The licas architecture has obvious comparisons with existing SOA technologies. Loose comparisons can be made with the grid. The XML-based communication protocol however can be compared more strongly to the web service technologies. If semantic methods can be developed to allow programs to automatically call web services, then the same principles could be used to build method descriptions that a technology like Java reflection could use to make automatic procedure calls in a p2p environment as well. One example of using semantics and web services in the area of web media content can be found in Sakkopoulos et al. (2006). Web services, for example, use SOAP as their message passing protocol, which is essentially also an XML-based description of the procedure call. This must then also be automatically re-constructed and invoked on the web service when it arrives at the server site. SOAP is a more sophisticated mechanism, but it also carries extra overheads. Direct comparisons however have not been made and further tests are required before any sort of conclusions in that area could be made. Cameron et al. (2004) have used record linking as part of an optimisation process to link health records in distributed databases. Another potential area that the querying mechanism would relate to is the semantic web. Future queries will not try to retrieve information from a single web page, but will require knowledge to retrieve information by matching several web pages. This will require a linking of the web pages through certain concepts in the query.

There are a number of related systems, of which the following are some examples. Cai et al. (2006) have designed a lightweight approach to autonomic service-oriented computing systems development. The SOA is defined by a three level pyramid. At the bottom is the basic services level, providing service provider, service consumer and service broker roles. The middle level is for composite or aggregated services and the top level is for managed services. They state that autonomic computing and SOA do not currently provide adequate supports to combine both technologies. One possibility is to use cybernetics and agent-based technologies as the underpinning control and intelligent management theory. Cybernetics is a general theory of control. Wiener (1948) defined cybernetics as the science of communication and control in mechanisms, organisms and society. One law of cybernetics is that if you seek to control a (self) system you must have a model of that system. The intention with licas, however, is that each service should be self-contained and should be independent with respect to other services. This is
a more distributed solution, along the lines of the tightly coupled architecture described in McCann and Huebscher (2004). Services can be nested, but their interaction must still be implemented. Centralised management components are not part of the system but could easily be added as additional services. For example, in an actual application described in Section 6, a query engine feeds the results back through the whole network from a centralised position. One very similar system has been implemented by Liu et al. (2002). It is called flow-based infrastructure for composing autonomous services (FICAS) and is intended to enable the distribution of data-flows within megaservices. They separate the control flow (concerned with service state) from the data flow (actions on data elements) and also have a service composition infrastructure capable of composing and executing megaservices. A megaservice is a service composed of several smaller services. Megaservices are specified through a specification language called composition language for autonomic services (CLAS). Individual applications can also be added to the network inside of wrapper classes. This seems to be a much more sophisticated infrastructure than licas, but the linking possibilities in licas would add something new.

6 Case study example: linking through querying

The licas system has been used as part of a larger project concerned with the creation of a network of knowledge. This network organises information sources into a structure that allows them to be efficiently retrieved. Services of different types are loaded into the network to perform functions such as query engines, metadata parsers, evaluation functions, linking services, etc. to provide any kind of functionality required over the network. The network is hierarchical in nature, but the stigmergic linking features of licas can provide for temporary overlay views. These views reflect the current use of the system and are created through the querying process. If the use of the system changes, then so will the views. By feeding the results of queries back through the network, it can effectively self-optimise with respect to querying. The intention is that these temporary overlay views should complement any permanent organisation based on something like semantics. This is because the views can also link nodes or concepts that are not semantically related. Tests have considered queries of the ‘select-from-where’ type. Nodes that are typically used in the same types of query can be clustered together through the links. Then when the query type is executed again, these nodes will know about each other and can indicate related nodes to look at. This will mean that the search process does not need to search the whole network and so some level of optimisation will have been achieved. A query might look like:

Select clothes.what_to_wear, From clothes, weather Where (clothes.season equals weather.season) and (weather.temperature equals hot)

Thus, by retrieving the sources that satisfy this query, clothes sources can be linked with weather sources that contain the same weather information and also with the specified conditions. The link can be reinforced by feeding the query result back through the network to inform the appropriate nodes to link to each other. Once this link is established, if a user again asks for clothes suitable for hot weather conditions, the linked clothes source can be immediately returned.

The performance is measured with regard to the reduction in the node count and the related deprecation in quality of answer when links are used. The node count and quality
of answer are compared to those produced by a full search that is still guided by the hierarchy, but then considers all possible sources. A full search would return the best answer. A larger search would be expected to produce a better quality of answer as it would have access to more sources and so a key problem is balancing the search reduction with an acceptable reduction in quality of answer. Full details will not be given as they are described in other papers (for example, Greer et al., 2008), but you could expect an 80–90% reduction in search with a related 5–10% reduction in quality of answer, depending on the network configuration. The quality of answer measurement is compared to the optimal or best possible answer. If the network is returning numerical information then this could be important, but for textual information this level of depreciation in accuracy could be acceptable. When we browse for web pages, we probably do not find the web page with the best textual description every time, but settle for relevant information. Thus, there is some margin of error here in finding an acceptable information source.

The logic behind the proof that the linking mechanism works is as follows. The optimal solution is found and the sources that are used to obtain this are noted. The linking mechanism then tries to link and return these sources when the same query is executed again. If the linking mechanism is correct, then it will link these sources and they will be returned when the query is executed again. If the linking mechanism is less accurate, then it will return other sources and these will provide a less optimal answer. The tests that have been carried out used numerical values and so the differences in the two types of answer (optimal or linked) could be measured. This thus gives some indication of its accuracy. The linking mechanism itself however does not use any metric and only tries to link the sources it is told to. Thus, this level of accuracy could apply equally to numerical sources or textual ones and so could be considered as a general kind of conclusion. The key factor is the variation in the query type that is executed and this is considered in detail in Greer et al. (2008).

7 Self-adaptation

The tests results show that a system built on licas can self-adapt with regard to self-organisation/self-optimisation through the querying process. Links can be created stigmergically, are dynamic in nature and are temporary. They are created or destroyed depending on system use and so they can adapt to the system use and change over time. For example, if the type of queries being executed change, then the links between nodes will change through time to reflect this. This process is adaptive because it depends solely on the use of the network and will only link the concepts that are queried for.

7.1 Reasoning possibilities

Tests with regard to querying have shown that the linking process needs to be monitored, as it can provide too few or too many links (Greer et al., 2007a). If there are too few links, then the quality of answer will be too poor because the correct links are missing. If there are too many links, then the search will be increased without a comparative improvement in quality of answer. Thus, a supervision system needs to supervise the optimisation performance and adjust the linking method based on current performance. This evaluation, which would be a combination of search time and quality of answer,
would require some sort of decision making component to determine what levels were acceptable and what levels were not. Lewis et al. (2006) describe a number of benchmarks or metrics that can be used to measure the performance of autonomic systems. They state that QoS is often highly application specific, depending on a service level agreement (SLA) between the interested services. Similar metrics are also described in McCann and Huebscher (2004). This suggests that the QoS measurement needs to be very flexible and may in fact be determined between individual components. Other reasoning capabilities would include adjusting memory allocations to provide more optimal link numbers, or encouraging links to change if the use of the system changes. Alternatively, reasoning as part of the linking mechanism can take the form of algorithms that learn optimal configuration parameters. For example, if it is found that too many links are being moved up the levels then the weight increment value can be reduced, or if too many are being moved down the levels then the weight decrement value can be reduced. This can all be measured as part of the linking process.

There are also possibilities that the links themselves could provide reasoning capabilities. This has currently been considered as part of a query process, as discussed in Greer et al. (2007b). As they represent what the users query they also represent the knowledge of the users. If there are several links relating to a single concept, then these could be averaged to provide some sort of general estimate for a value relating to that concept. If the links represent the use of the system, then this estimation can be considered to be reliable and initial tests show that using links improves performance over a full search (Greer et al., 2008). It could answer questions like ‘what is the best value of one concept based on the values of other concepts?’ or ‘is a value or concept possible based on other values or concepts?’ For example, consider the network in Figure 2.

If you can imagine that users have queried a network asking about buying t-shirts and the network has also recorded the weather at the times that the network was queried, then links similar to those shown may have been created. A user could then query this structure and ask what the best temperature for buying t-shirts was. Through the use of a simple averaging mathematical operator, together with the created links, the network could reason that 20°C was the best temperature for buying t-shirts.

**Figure 2** Example of a reasoning possibility through links

![Diagram of reasoning through links](image)

Note: The average temperature for buying t-shirts can be calculated as 20°C.
8 Conclusions

The licas system can be compared to a very lightweight SOA infrastructure with an XML-based communication protocol, such as that provided by web services. Of course, many systems are like this and can make a similar comparison. The main advantage of licas is that because specific functionality is not currently implemented, it could be used to create many different kinds of network. It essentially provides the framework on which to build such things. When automatic semantic abilities are added to the internet, then other XML-based message passing protocols will be able to benefit from those techniques as well. Thus, automatic machine-to-machine communication will also take place in lightweight sensor-based or p2p environments as well.

It is evident that licas has been built on the principles suggested for CASs. It provides for agent-like properties, it has a basic communication protocol and it adapts through stimuli-action (stigmergic) rather than event-condition-action rules. Intelligence can be introduced in the form of a supervision component that must monitor and adapt the linking mechanism, or learning algorithms as part of the linking mechanism itself; that adjust configuration parameters to optimise performance. Reasoning as part of a querying mechanism is also possible. Through the linking process, adaptive organisation or behaviours can emerge, depending on the complexity of the services involved in the interactions. The collective actions of these components could even begin to show more cognitive or intelligent behaviours.

Acknowledgements

Work supported by the project CASCADAS (IST–027807) funded by the FET Program of the European Commission.

References


Harder, http://iharder.net/base64/ (accessed on 29 August 2008).


An infrastructure for developing self-organising services