Abstract - The licas (lightweight Internet-based communication for autonomic services) system is a Java-based open source framework for building service-based networks, similar to what you would do on a Cloud or SOA platform. The framework comes with a server for running the services on, mechanisms for adding services to the server, mechanisms for linking services with each other, and mechanisms for allowing the services to communicate with each other. The general architecture of the system is now fairly well set and so this paper describes recent developments that have focused on making the framework more robust, with additional features for easier programming.

Index Terms - service-based network, autonomous, AI, SOA, Cloud.

1 Introduction

The licas system [5] is a Java-based open source framework for building service-based networks, similar to what you would do on a Cloud or SOA platform. The framework comes with a server for running the services on, mechanisms for adding services to the server, mechanisms for linking services with each other, and mechanisms for allowing the services to communicate with each other. Functionality is provided to allow for XML-based RPC message passing and dynamic linking between services. It is also possible to invoke the server through REST-style messages from a web page. Web services invocation and autonomic management are also provided for. The framework is very lightweight and the architecture and adaptive capabilities through dynamic linking, add something new that is not available in other similar systems. The download packages also come with an All-in-One GUI, that can be used either to test systems, or as a practical platform on which to run your own real system. The system is also peer-to-peer, with the client GUI also acting as a server that can be invoked. Licas can therefore be used in one of two different ways. Because of resource constraints, these have been packaged together into the one GUI application.
After a brief overview of the architecture, the rest of the paper focuses on the main areas of development, that is: web services communication, administration, autonomic components, search and query, and problem solving. Mobile application problems are then discussed and the paper finishes with some conclusions.

2 Architecture
Licas is a distributed system with a lightweight HTTP server running at its base. The server receives all incoming requests, processes them and sends the message to the indicated service. Security is provided by the fact that passwords are required to then invoke the desired service. If the received data does not contain the appropriate password, it will not be passed on to the related service. The licas system is peer-to-peer, but uses this hybrid approach. Any service can also use the communication mechanism to send an HTTP request to any other service running on the system. This means that each service or node can both ask for and receive requests for work. The linking mechanisms have been written about in detail previously [1][2] and have not changed significantly since the original software versions. Basically, they allow for permanent links between nodes that can define a permanent network structure. They then also allow for dynamic links that can change and would reflect more the current system use. There is then also the possibility of nesting services that would have direct links or references to each other and service associations that might not be related to structure at all. It is also relatively easy to extend the base classes to add new services with specific functionality. While resources are limited, the system is relatively stable and could be used in real world scenarios. The system has been written to be J2ME compatible and so porting to a mobile platform looks possible, although, Oracle’s version of Java [7] is not the most mobile-friendly at the moment.

3 Communication
A distributed system is composed of a number of individual components that interact with each other, sometimes over a network. The key to the distributed nature is the fact that processing is not centralised in one place (no central control), but is distributed to the
different components in the system. Because of this, a global picture can only be obtained if these components can then communicate with each other. A communication language is therefore required, where licas uses an XML-RPC style of language. That is: code any communication into an XML-based script and send this to the requesting component. XML provides a standard format and also a tagging system, so that specific parts of the messages can be labelled and understood. This is also what Web Services use, but in a slightly different format. Agent-based systems [10], on the other hand, use a more sophisticated format that includes specifically defined interaction protocols, so that ‘conversations’ can take place.

3.1 Web Services

The licas package also comes with a Web Services WSDL parser and the capabilities to dynamically create either a SOAP or a RESTful-style message, and to invoke a Web Service dynamically from the construction. This essentially involves parsing and conversion into an XML-based object that can be passed through the system using the standard licas classes. Interaction with a user is still required, to enter the selected method name and specific parameter values. The web service invocation however can be made on any WSDL document that is retrieved dynamically at runtime and the specific method call does not need to be hard-coded beforehand. This means that there is little difference if XML-RPC or if Web Services are used, as part of a method invocation. However, if you are using XML-RPC, you are required to provide your own parser for any complex objects, but this means that any reply can be converted back into a Java object again. If invoking Web Services dynamically, conversion of the reply back into a Java object is more difficult as there is no existing definition, and so probably only simple types can be accommodated for, or the result just displayed.

The server also allows an invocation of a service’s method from a web page. If it does not recognise the standard XML-RPC method object, it will try to parse the request into a RESTful-style form. If that is successful, then it can use the data to create the method object itself. This means that AJAX communications from a web page are also possible, again probably only with simple data types however.
4 Administrative Features

In an attempt to turn the default GUI application into a more useable SOA platform, a number of administrative features have been added to it. In particular, a service factory can be used to load in and save services and external modules, so that they can be automatically re-loaded during start-up, to set up the environment correctly. The details are saved as XML-based configuration scripts. Scripts can also be used to save details about problem solving tasks, again to allow for the automatic execution of the task at a later date.

4.1 All-In-One GUI

A default All-In-One GUI is provided as part of the system download package. It is not open source, but it is free for non-commercial use. The GUI allows you to load and run your own services, and view and interact with them. Each licas server can run services, as would be the case with a SOA (Service-Oriented Architecture) [6] or Cloud [9] system. Details of the running components can then be viewed in a graphical display. The GUI can also link or register with other servers and view those configurations as well. So a group of distributed servers can be registered, linked and viewed, and then each individual server configuration can be viewed in the same way. Figure 1 shows the default GUI running a single service, with the associated GUI interface also being used.

The service panel is showing, where external modules and classes can be loaded in at the top and default services are displayed in the bottom half of the panel. In this case, a message service is selected, with a user id of hercule. The interaction with the service can then be made much easier through its own GUI interface. In this case, the instant messenger service has just received a new message. The GUI therefore provides a visualisation of any network constructed using the licas classes and allows you to test certain functionality as well. The GUI only displays a view of the network, which is based on the metadata descriptions that it retrieves from the server. These metadata objects contain XML-based descriptions of the network components. As the view is XML-based, this means that it should be possible to create your own network of services and still be able to view them, so long as they use the licas base classes. The GUI app runs on J2SE and therefore has access to all of Java’s features. The server itself is written to run on the J2ME subset.
4.2 SOA

Menu options also allow you to save the configuration of the server currently on display. This will parse the services’ descriptions and save them to a file set. The file set can then be read in again, to load the service-based descriptions back onto the server. Because a network is a group of distributed servers, the term SOA is being used to describe the setup at a single server. So this is like a very basic operating system, where instead of loading your initial applications onto your desktop, you load and run services on the server. This is essentially just a parsing exercise and not all of the service details would be saved, but it does allow for the automatic re-creation of the service-based structure. The parsing has recently been updated, to make it easier to extend.
4.3 Service Factory

Another new feature of the GUI is a ‘Service Factory’. This can be opened and used to define what services and base factory classes should be used by the GUI as default. You can also specify or load in external modules, or external Jar files. These will be parsed and you can add services from them to the default GUI set. They are then automatically available for running on the GUI when the configuration file is loaded in again. Figure 2 shows the service factory form.

The service factory can be used, for example, to link a forms-based interface to a service, as shown in Figure 1. You can also select specific factory classes, for processing global operations. You do this by loading in a new module, or jar file. Its classes are parsed and made available for selection (see the top group areas). You can then also select classes from this as new services. There is one external module provided with the default package. It is called ‘licas_services’ and contains the default service implementations, with related GUI interfaces. An instant messenger and a remote file transfer service are available, for example. If you were to add a new jar file as a module and then a new service from that, the
new information could then be saved to the script and loaded back in again during configuration setup.

5 Distributed Autonomic Processing

A distributed system would really benefit from autonomic computing, particularly large-scale systems that need to monitor themselves. Autonomic can mean possibly one of two things. The first meaning would be simply that a component can perform its own operations independently. It has its own internal control system and understanding of what it is required to do. The second meaning is based more on a reactive system with sets of self-features, based loosely on the human nervous system [3]. These features operate automatically and in the background, without prompting and monitor the system, to ensure that it is able to deal with potential faults or threats. There are great technical problems with building a system that would be 100% autonomous as this would also require the system to be very intelligent. This can be helped by allowing a system administrator to define rules for the system through admin scripts, or by simply limiting the level of intelligence that the system is allowed to use. Figure 3 is a schematic of the general framework and architecture of an autonomic service provided by licas.

Figure 3. Schematic of the licas service architecture
The base class is a ‘Service’ class. For autonomic components, the extended ‘Auto’ class is required. You then extend this yourself with your own service behaviour. The Auto class includes an Autonomic Manager framework, with the 4 modules – Monitor, Analyze, Plan and Execute. There is also a Message Interface (MI). On executing a behaviour, the Auto class will automatically send the result to the Autonomic Manager using the message interface. If autonomic modules have been implemented, they can monitor and evaluate the messages. If there is a fault, both the service itself and the base server can be informed. There is default metadata relating to the service and also a Contract Manager, to store and reason over proposed contracts for the service work. Nested or utility services can be added to any other service, but should be used more as an aid, rather than fully independent services themselves.

6 Search and Organisation

For large-scale distributed systems, there is a need to organise them efficiently, to optimise for time and resource usage. The individual components also need to be found and ideally, be able to link or combine with each other, to allow for more complex tasks and interactions. This is known as service Deployment, Discovery and Composition. Using licas, individual services can be loaded at run time and coded to perform any sort of action. They can be configured through scripts and can communicate and link with any other service. There is also a query language and service-specific metadata, to allow for search and find operations.

The query language and engine are provided as part of a text package. It is not open source, but is part of a set of utility packages that licas also uses. It does allow for processing of XML-based, or simply text-based data and uses a new type of query description. The language is not a standard or anything, but it can define the fields and operators required to perform relatively complex search operations. So the licas system already provides for descriptive metadata and query operations that would allow for relatively sophisticated query processing over distributed networks.
7 Problem Solving

The licas system is both distributed and provides a framework for adding autonomic monitoring components. The communication language allows individual nodes to talk to each other, allowing them to perform evaluations based on local information, typically between two services only. The other classical setup is to bring all of the information together into a centralised place. In that case a more global and complete picture can be obtained, but the processing requirements are much more heavyweight. Licas allows for a centralised problem solver to be run. This can request or receive information from the services running on its network, process them, including AI tree searches for solutions and then send the results of this back to the individual nodes. So both the lightweight localised processing and the more heavyweight centralised processing can be accommodated for. The individual servers can also link up, making scalability to larger systems possible. One intention is for the system to still be useful as a research tool and so the problem solving aspect is particularly important there.

The problem solver can be configured with its own script, where the different variable values can be entered through the GUI. This includes the type of problem-solving service that is run. The default system provides a behaviour service and an information service, and a mediator to try to combine the results. The behaviour service can be extended by programming only one or two methods and will then execute the behaviour of those methods periodically and return the results for processing. The default problem-solving mechanism is genetic algorithms, where text-based information can be parsed and processed through evolving genetic solutions. The download package contains user manuals that describe each of the system applications. The problem solver can also be run in a Command Prompt box, if a network structure is not required.

8 Mobile Applications

One central goal of the system is that it is J2ME, or Java Mobile, compatible. The platform however is built on the CDC platform and not the simpler CLDC version. CLDC can still be
used on mobile devices relatively easily, but CDC appears to be more difficult to run. It is a larger subset of Java and Android or Apple, for example, will not run it by default. There are still technical problems, so the hope is that the technology will develop and allow for more substantial programs in the near future. There are probably ways to run a CDC program, but it might involve installing a DOS or Java emulator, or similar first, which would be awkward for a general user. If using just CDC, then there might also be problems with Reflection and dynamic class loading, if considering dynamically loading in services from external jar files. For a self-contained installer however, an early version of a J2SE JRE [8] could be included and would keep the download size to a minimum. There is also the original intention of using the mobile device just as a client interface, to access a server running on a PC or laptop remotely. Then the All-In-One GUI, or something similar, can be used to run the services and the mobile device only needs to invoke methods remotely. This still allows for some level of privacy, as the server is on a personal machine and would be password protected. Porting to Linux also looks possible, but external installations and variables need to be set again. So a complete or easy installation package is still problematic.

The paper [4] proposes a new Cloud-based architecture for dynamic recognition-based tasks. They discuss image or speech recognition tasks that would include location-based information. The system is a parallel architecture of smaller Intel Atom Core processors, for low power and high throughput. If there are worries about the workload on the server, processing more intelligently on the mobile device first and then sending a simpler problem to a central server, would also help with a solution to that.

9 Discussion
Recent work has focussed on making the system more robust, so that it might be used in real-world scenarios and not just as a test platform. However, the research-side has also been kept, as is it a flexible system that is ideal for certain distributed or dynamic network tests. There have been real improvements on the framework side, with regard to how a user might add autonomic processes. He/she now only needs to implement one or two additional methods, to allow for the autonomic monitoring or processing of information, for
example. Security has also been improved with passwords, contracts and a wrapper to protect direct service references. The web service parsers and now quite robust and the web service invocation has been integrated into the system, as part of the whole communication mechanism. Finally, system configuration has been tackled, with a service factory and configuration scripts, allowing for SOA configurations to be saved. These can then be re-used with minimal effort to the user.

A lightweight framework, such as that provided by licas, offers a nice solution to running your own server. Your mobile device is usually on and connected to a network. You would not typically host static web pages on your phone, but process location-based or information-based data, in a much more dynamic way. The system therefore also needs to be helpful, when autonomic processing is useful. The licas system can therefore be used to build more localised networks between small numbers of people that do not flow through the major vendors’ systems. This might even help with network congestion problems. The problem at the moment is the compatibility issue with the main mobile operating systems. Development for mobile devices would be easier if the CDC platform became easier to use. This is not the main goal of the project however, only a very useful addition.

10 References

[1] Greer, K. (2010). Recent Developments with the licas System, Distributed Computing Systems. Distributed Computing Systems, published on Scribd at http://www.scribd.com/doc/28144576/Recent-Developments-With-the-Licas-System. Also available from the DCS web site

[2] Greer, K. Thinking Networks – the Large and Small of it: Autonomic and Reasoning Processes for Information Networks, published with Lulu.com, 2008, ISBN: 1440433275, EAN-13: 9781440433276.

[3] IBM (2003). An Architectural Blueprint for Autonomic Computing, IBM and Autonomic Computing.

[4] Iyer, R., Srinivasan, S., Tickoo, O., Fang, Z., Illikkal, R., Zhang, S., Chadha, V., Stillwell Jr., P.M. and Lee, S.E. (2011). Cogniserve: Heterogeneous Server Architecture for Large-Scale Recognition, Systems for Very Large-Scale Computing, IEEE, pp. 20 - 31.
[5] Licas. http://licas.sourceforge.net.

[6] MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R. and Hamilton, B.A. (2006). Reference Model for Service Oriented Architecture 1.0, OASIS, Standard. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.

[7] Oracle Java, http://www.oracle.com/us/technologies/java/overview/index.html.

[8] Oracle Java Archive Repository, http://www.oracle.com/technetwork/java/archive-139210.html.

[9] Ranjan, R., Buyya, R. and Benatallah, B. (2012). Special section: software architectures and application development environments for Cloud computing, Softw. Pract. Exper. Vol. 42, pp. 391–394. Published online 9 November 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.1144.

[10] Wooldridge, M. and Jennings, N.R. (1995). Intelligent Agents: Theory and Practice, Knowledge Engineering Review.