

Communication in ad hoc networks Or: CORBA considered harmful

S. Lakin, S. Mount, R. Newman
Cogent Computing, School of Mathematical and Information Sciences
Coventry University, Priory Street, Coventry, CV1 5FB
email : s.lakin@coventry.ac.uk, s.mount@coventry.ac.uk,
r.m.newman@coventry.ac.uk

Abstract

As the cost of basic units of computation (clock cycles, memory, bandwidth, etc.) cheapens; pervasive, wireless, ad hoc networks are increasingly being seen as the next generation in user-owned technology. However, the design of devices in such networks pose significant challenges to systems programmers. Large-scale middleware (such as CORBA) has been hailed as the solution to problems of heterogeneity. Unfortunately, this means that messages between devices (objects) have to be wrapped and unwrapped at either end of a communication as well as between protocol layers. This can cause problems when applied to networks of small sensors, where resources are scarce. We outline our position, and propose an alternative approach for sensor networks, which rejects the use of mediation languages such as CORBA and uses compile-time methods to reduce the likelihood of “misunderstandings” between software components.

1 Introduction

Across the world, the demand for freedom and mobility is growing, and technology must follow suit. It used to be the case that people were happy simply making telephone calls from one fixed location to another fixed location. The set of locations and the connections between them formed a totally static network. The introduction of cell-phones changed all that now people had freedom to talk from wherever they liked, and they were no longer tied down to where the wires took them. The network had become mobile and wireless. However, the base-stations were still in one fixed place and the users moved around the infrastructure created by the various base-stations. Ad hoc networks take this a stage further by removing the infrastructure altogether. Instead, we have a collection of mobile nodes, which all have the potential to communicate with each other, and which can therefore provide for a rapidly-changing environment.

At the same time our technology is getting smaller and smaller. We can have literally millions of very tiny sensors that need to communicate in an ad hoc network. Imagine for example a collection of mobile sensors spread over the surface of a planet, all monitoring the weather communications and terrain in their location, so we can collect all the data to give us an accurate picture

of the surface as a whole. These sensors will have comparatively low power and small resources. Consequently, their system software needs to be optimised for power, time and space. These considerations will influence all aspects of the system software design, including communications protocols, fault management, provisions for security and so on.

Mobile processes (hereafter called *agents*) are one high level abstraction for applications and services running an ad hoc network. These agents need to be able to communicate with each other as they move around, so that they can work together, share the knowledge they gain, and solve any problems. To do this, they need to have some common language in which to communicate. Such a language is known as an Agent Communication Language or ACL.

Unfortunately, the problems of defining such a language are immense. As a minimum, it must be reliable, secure, context-aware, sufficiently expressive for its intended purpose and have a clearly defined semantics. Moreover, all over the world people are working on software systems and developing their own components. For our agents in our ad hoc sensor networks, we need interoperability between the components, so that they can work together, communicate and exchange information correctly. The main hurdle to reaching this goal is heterogeneity, by virtue of the fact they are developed independently, components are written in different languages, different styles, using different hardware and systems, and conflicting with each other.

In this paper we will discuss some of the problems of using large-scale middleware for such low-power ad hoc networks. We present a strategy for simplifying the implementation of system software in ad hoc networks by moving much of the work of middleware into compile time checking and analysis. We believe that this approach allows sensor designers to benefit from the advantages of using mobile processes, whilst minimising the problems of living in a heterogeneous environment.

2 CORBA considered harmful

Our goal is to define a way that very small agents can communicate with each other in an ad hoc network, optimising our design for time, power and space. The agents can be considered as objects, and so one potential approach to deal with the problems with interoperability and heterogeneity would be to use some middleware to allow the objects to communicate. One such example is an Object Request Broker or simply ORB. The ORB takes a request from one object (the client), performs the details of communicating with the other object (the server), retrieves the appropriate response, and communicates this response back to the client. This forms the basis of the Object Management Groups infrastructure and architecture to allow applications to communicate, known as the Common Object Request Broker Architecture, or CORBA. CORBA is widely used for large-scale applications and networks. Because the middleware does all the work, it allows CORBA-based programs written in different languages, running on different platforms and different operating systems, to communicate over the network. This means that it can be useful for large networks, which need to handle a large number of client requests from a variety of different programs. As the cost of basic units of computation cheapens and many devices are permanently networked, CORBA appears to be an elegant solution to

the heterogeneity of modern computing. However, CORBA is resource-hungry, as it adds time and resources to communications which may already be quite complex.

Moreover, one might ask *How correct is CORBA?* The experience of [1] suggests that there are many ambiguities and under-specifications in CORBA. This means that despite their best efforts, implementers can produce incompatible systems based on their differing approach to the under-specifications. Even if systems are not totally incompatible, these problems can lead to errors, misinterpretation and ambiguity. In such resource-poor environments as we encounter in our ad hoc networks, such errors need to be minimised.

Furthermore, how readable and comprehensible is the code that is produced in a CORBA-based system? Everyone has their own approaches to programming, and despite supposed adherence to standard coding practices, code is often almost unreadable to anyone other than the author (and sometimes not even that). We consider systems programming to be a difficult task indeed, and our aim is to simplify it. Adding extra code to marshal and unmarshal data and function calls into a mediation language does not ease readability and may clutter already complex implementations.

3 Layered protocols

The standard OSI-type model [2] is well-known and separates communication into seven layers, ranging from the application layer (which identifies the communication partners) down to the physical layer (which sends the actual bit-stream of data). Many networks operate using something like the OSI model, although few if any implement it faithfully. In fact, the popularity of the OSI structure seems to have had a profound influence on systems designers. Most new communications systems have a protocol stack and it seems unfashionable to do otherwise.

In the OSI model, to send a message, we move from layer to layer, at each stage wrapping up our intended message in order that it can be used at the layer below. CORBA acts as a layer below the application layer to perform the communications. Above it, we have our applications (objects), and below it, all the services that perform actions on the objects. This means that in order to send a message successfully, our original message needs to be wrapped up to the ORB layer, before being communicated to the target, which then has to unwrap it before they finally obtain the question that was being asked. Then to send their response, we wrap and unwrap all over again. Essentially, we are communicating by playing pass-the-parcel!

This is not a particularly efficient way of doing things. In a large network with vast resources and immense power (such as the Internet, or the GRID), lack of efficiency is not a primary concern. In the scenario we are considering, with low power and low resources, efficiency and correctness are both essential.

4 Our position

We believe that middleware and complex protocol stacks are inefficient and unnecessary in resource-scarce ad hoc networks. Although research has been

done on such options, for example with TinyOS (see [5]) this does not approach the problem in the way that we desire. We are advocating, above all, clarity. We propose an approach to creating system software which places an emphasis on compile-time analysis and checking. We do not believe that high-level abstractions are inherently bad, or that all systems programming should be very low-level. We simply advocate an approach that is as clear and straightforward as possible.

The function of CORBA and protocol stacks is to ensure that data emerging from one object (or other run time entity) is in an appropriate format to become the input to another object. In an environment such as the Internet, where software may be new or legacy and can come from almost any networked source, middleware appears to have a meaningful place. However, in a relatively closed scenario, such as our sensor networks, APIs can be planned and designed to interoperate. One way of achieving this is to use a common markup language such as XML to encode data and utilise a common semantic “ontology”.

Beyond a common strategy for data marshalling, we have three specific recommendations for the design of system software in ad hoc sensor networks. Firstly, we believe that agent communication should have a sound theoretical basis. Agent communication languages should have a formally defined semantics which is fully abstract and therefore does not make unnecessary distinctions between program elements. Time, spatial context and run-time environments must be addressed by the semantics, since they are crucial to the correctness of applications in ad hoc networks.

Secondly, we believe that application code should be developed with correctness in mind. Ideally, important algorithms would be proved correct. As a minimum Meyer’s design by contract [3] can ensure that pre- and post-conditions are met and well documented.

Thirdly, we advocate the use of static checking for coding standards, Exstatic is an extensible static checker, written by the first author. Exstatic is a framework for building system or project specific static checkers. It provides a language independent intermediate language (ICODE), a domain specific language for writing checkers (although any programming language with an XML library may be used) and a user interface. We have used Exstatic to check for various violations of coding standards in little languages, documentation [4] and we are currently applying the system to a general purpose programming language (Python).

Whilst static checking cannot ensure freedom from errors, there are specific “rules” that may apply in our scenario that could not be caught by a compiler and may be overlooked during code reviews. For example, we probably want to avoid busy waiting, as it uses unnecessary resources.

By using a combination of approaches, we believe we have a strategy to provide simple, efficient communication in low-power, ad hoc networks. Our code is more likely to be correct because we use formal methods. Our programs are more likely to be readable and easy to maintain, because we have avoided using complex broker systems and layered protocol stacks. Static checking gives us confidence in the efficiency and standards compliance of our code. And finally, our agents will communicate via a language that is understandable and reliable.

5 Summary

In this position paper, we have outlined our view with regard to communication in low-power ad hoc networks. We have argued that the use of bloated, large-scale middleware is inefficient, and comes with many additional problems and errors. Instead, we have proposed a solution involving formal languages, static checking, and agent communication languages, that removes some of the problems caused by the middleware. There is no need for our messages to be wrapped up, passed through layers, and then unwrapped - our agents communicate directly. We have established a framework which, at the same time as increasing efficiency, also reduces the risks of software error, both in the code and in the implementation of the code.

We hope to be able to extend this approach in the future. Of course, our method is not the most suitable for every situation and we need to examine larger-scale applications to see how our approach applies there, and establish under what circumstances our approach is beneficial, and identify and deal with the disadvantages in those situations when it is not. But our approach adds efficiency, correctness, and helpfulness, and we believe it is highly appropriate for the low-power ad hoc networks under consideration.

6 Acknowledgements

The authors would like to thank Elena Gaura, Rob Low, Leon Smalov and Peter Every for their helpful comments on drafts of this paper.

References

- [1] Bastide, R., Palanque, P., Sy, O., Navarre, D.: Formal specification of corba services: experience and lessons learned. In: Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ACM Press (2000) 105117
- [2] ISO 7498: Open systems interconnection, basic reference model. (ISO)
- [3] Meyer, B.: Design by contract. Technical Report TR-EI-12/CO, Interactive Software Engineering Inc. (1986)
- [4] Mount, S., Newman, R., Low, R., Mycroft, A.: Exstatic: extensible static checking applied to documentation, ACM (2004) Submitted to ACM SIGDOC.
- [5] Levis, P., Madden, S., Gay, D., Polastre, J., Szewczyk, R., Woo, A., Brewer, E., Culler, D. : The Emergence of Networking Abstractions and Techniques in TinyOS. In : Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (2004).