Communication Primitives for Ubiquitous Systems or RPC Considered Harmful

Umar Saif, David J. Greaves
Computer Laboratory, University of Cambridge.
{us204, djg}@cl.cam.ac.uk

Abstract

RPC is widely used to access and modify remote state. Its procedural call semantics are argued as an efficient unifying paradigm for both local and remote access. Our experience with ubiquitous device control systems has shown otherwise. RPC semantics of a synchronous, blocking invocation on a statically typed interface are overly restrictive, inflexible, and fail to provide an efficient unifying abstraction for accessing and modifying state in ubiquitous systems. This position paper considers other alternatives and proposes the use of *comvets* (conditional, mobility aware events) as the unifying generic communication paradigm for such systems.

Keywords: RPC, RMI, Events, Comvets, CORBA, Jini

1 Introduction

Ubiquitous environments or active spaces are the next generation of device control networks. A user interacts with an active space by using novel interfaces like speech and gesture input [1] to control her environment, and the system interacts with the user using audio/video output. The user can discover and use the environment resources and export the resources she is carrying to the environment [2]. One of best-known application of such systems is Home or Office automation [3]. Our research group has been researching Home Area Networks for the last 5 years [4]. This paper is based on our experience with the communication primitives for such a system.

Although, it is quite well understood in our group that RPC is not a suitable paradigm for such systems [5, 23, 31, 32], we feel that a large portion of the community in both research and industry is still using RPC (or its newer implementation like ROI, RMI) even when it is harmful. For example, UpnP, Easyliving and others have recently devised SOAP [6], based on RPC semantics, for home automation. Likewise, even notable mobile agent systems like Ajanta [7] solely rely on java-RMI for agent interaction. More astonishingly, distributed Event architectures of even the mainstream systems like CORBA [8] and Jini [29] have been implemented on top of RPC (RMI), which, although alleviates the applications from RPC semantics, leads to an inefficient implementation at the system level.

This position paper is motivated by pervasive use of RPC in pervasive systems, which, as we show, is harmful.

2 What is a ubiquitous system?

A ubiquitous system is a sensor/actuator rich environment that provides both mobility-transparent [9] and context aware (or adaptive) access to system resources [2], depending on the requirements. Resources can be mobile; communication channels have varying characteristics, and network partitions are a possibility [10]. Resources can fail, enter or leave the network, and ad-hoc topologies without any backbone connection are possible [12]. Data exchanged between systems resources can vary from short one-way discrete messages to multiparty continuous media streams. The interfaces to a resource might not be known at compile time. Finally, as many sensors/resources/indicators collaborate to define a ubiquitous environment, interactions frequently involve more than two parties.

3 . What is RPC?

Remote Procedure Call (and Remote Object Invocation) was proposed [33] as a simple, efficient and unifying paradigm to transfer data and control to remote resources. RPC promised to achieve this goal by abstracting away the syntactic and semantic gap between local and remote cases, by allowing remote resources to be accessed by a simple procedural invocation just like the local case. To be able to use the RPC facility, server side software needs to be compiled by a special RPC compiler that generates a stub and a skeleton. Traditionally, the stub (or its interface) needs to be in the client address space at compile time, leveraging strong (static) typing. Methods invoked on remote object are intercepted by the client side stub that forwards the call to the server side skeleton to give an illusion of a local invocation to the client application. Remote objects are referenced using interface numbers and version numbers (unique IDs) [12] that provide a (rigid) contract between a client and a server. The RPC runtime system hides the transport details from the client and server stubs, manages the interface IDs, and implements the scheduling policies for server side resources.

4 So what is wrong with RPC?

Some of the semantic inconsistencies in the RPC abstraction of remote access have been discussed in [13]. The authors of that paper conclude that it may be better in many circumstances to use a non-transparent abstraction rather than a transparent one like RPC. While that paper is concerned mainly with traditional distributed computing applications, we focus on the lack of support in RPC for new forms of applications made possible with the advent of ubiquitous systems.

To be able to understand why RPC is harmful, we first need to analyze the RPC syntax and semantics. RPC dictates the semantics of a a) blocking, b) synchronous invocation and has the syntax of a statically bound procedural invocation where the procedure acts on an ordered list of formal parameters and returns a result.

This makes RPC overly restrictive, inflexible and inefficient for the communication requirements of ubiquitous systems in the following ways.

4.1 Logical and physical mobility

One of the key aspects of ubiquitous systems is mobility, both logical (mobile agents or processes) and physical (device mobility)[14].

Logical mobility, or process mobility serves for three purposes in ubiquitous systems. 1) It provides for load balancing, and hence better utilization of system resources, 2) it provides fault tolerance by replication and/or migration and 3) it can be used to dynamically extend the capabilities of the participating resources to enable them to participate in the ubiquitous system. Due to the complexity and overhead of strong migration [15], and relatively limited utility [16], trend in process migration has been towards mobile agents [17] implemented on weakly mobile frameworks [7]. Weak mobility basically reduces the context state to an instruction pointer at a subroutine entry point [17]. Therefore any blocking call would essentially prevent the process (or mobile agent) from migrating (as execution context would need to be saved at an arbitrary point). Although, most of the system calls can be implemented with non-blocking semantics to facilitate weak migration, remote interactions, if modeled after the RPC paradigm, restrict mobility to instances when no remote interactions are outstanding. Even for a strongly mobile system, pure RPC semantics do not allow for the return value to be delivered to a different host or process than the one who made the invocation. This severely limits relocation decisions for load balancing or fault tolerance in a communication oriented system like a device control ubiquitous system [4]. Consequently, applications like follow me audio/video that need to deliver data to a mobile object cannot be implemented efficiently using the RPC semantics. Implementations like M-RPC [18] violate the end-to-end semantics of RPC, and could only be viewed as ad-hoc fixes to a broken paradigm.

Wireless links can be asymmetric in nature [10]. Asymmetry can be in space i.e. bandwidth, throughput, BER or in time i.e. disconnected operation. RPC's bi-directional semantics mean that the properties (QoS or otherwise) of messages in either direction cannot be changed independently. Hence, this feature has not been implemented in any of the RPC packages. For instance, a link might be more error prone in one direction than the other one. Reliability parameters e.g. number of retries, or QoS parameters like priority, can be changed for

such a link with an abstraction that, unlike RPC, does not enforce bi-direction semantics. Likewise, disconnected operation could be supported with an abstraction that, unlike RPC, does not enforce bi-directional blocking semantics. The reply to a request can be delivered when the connection is resumed, without "hanging-up" the application on either side. For instance, wireless devices usually operate in suspend mode to save battery power, waking up periodically to see if any event of interest has happened. With RPC semantics, this would result in orphan processes [13]. Primitives that permit the decoupling of messages in either direction can employ a solution like a docking station [19] to temporarily hold the results for the mobile host and can deliver it to the mobile host when it comes up.

4.2 Device control applications

Our experience with ubiquitous device control systems [4] has shown that most of the remote interactions in such a system are either one way "do this" commands to the device or "this has happened" notifications in the opposite direction. For instance, "switch on the light" or "notify the alarm system if someone breaks into the house".

RPC enforces a tightly coupled bi-directional interaction. The synchronous, blocking semantics mean that the calling thread is suspended even for interactions that do not require a return value to continue with their operation (as mentioned above). Although it might appear that these semantics provide an implicit acknowledgement, the reliability semantics implemented by this implicit acknowledgment mechanism are fixed and are hidden by the RPC runtime system (at least once, or at most once) irrespective of the application requirements. This mechanism actually violates the end-to-end arguments for system design [20]. A better approach would be to make this explicit by notifying the application about any "abnormal" occurrence, and leaving the reliability semantics to it. This is not only more efficient but, at times, is necessary for correct operation. For instance, most of the device control operations are idem-potent e.g. "turn off the toaster" and it is more efficient to just have at-least-once semantics, whereas some other complex computation might require the complexity of at-most-once semantics for correct operation.

As pointed out by [13], the same limitation stops RPC from being used in parallel programming. One possible solution to get around this inherent limitation in RPC semantics is to use multiple threads for a task, but this solution is neither elegant nor efficient, and adds extra burden of thread management even for situations when it is not warranted.

4.3 Multimedia communication

In addition to the one-way device control commands, a large part of the device control architecture is concerned with managing multimedia (A/V) streams [21,22]. After all, ubiquitous environments interact with people, more precisely, with their audio/video sensors.

As stated above, RPC has tightly coupled bi-directional semantics i.e. a discrete request in one direction followed by a discrete reply in the opposite one, delivered to the same process which sent out the request. With multimedia communications, the reply is no longer a discrete message, but a continuous media stream, and it might need to be sent to a different destination than the one requesting the media flow to be set up [23]. Therefore, RPC semantics are not a general solution for multimedia communications. Instead an abstraction outside of RPC is employed to augment the distributed system to accommodate media streams. Usually explicit bindings are used to represent the media flows. These bindings provide a number of stream management functions in addition to the facility to add new end points [24] or processing modules [25] to the stream.

An abstraction that does not enforce synchronous bi-directional interaction can more easily be used to model multimedia communications.

4.4 Multi-party interaction

Ubiquitous environments are sensor rich where many sensors/resources/actuators collaborate to provide a service [26]. For instance, "when I enter the room, please notify the Hifi (so that it can play music), dim the lights, and draw the curtains". In this case, the notification from the door read-switch sensor needs to be delivered to three (and possibly) more sensors. For more applications, refer to [26]. As pointed out by [13], RPC is not well suited for group communication, and hence is not a good choice for collaborative sensor rich ubiquitous environments.

4.5 Interface Definition

RPC models all remote interactions as procedural invocations on strongly typed interfaces defined in a (quasi-) formal description language such as IDL [27]. This interface is then used to provide the language level mappings and to consequently generate the client and server side stubs. Therefore the client must know the exact method signature of the remote service at compile time to be able to use it.

Ubiquitous environments are loosely coupled, with different components designed and implemented separately at different points of times. Hence, it not possible to have the remote interfaces of all the services at compile-time. Further, one of the key aspects of ubiquitous systems is agility or context aware adaptation. Services must be able to discover and bind to new interfaces as they move in the ubiquitous environment [2]. Our experience has shown that RPC's static typing is overly restrictive, and does not allow the desired interoperability. In such environments, it is often very useful to be able to use a "nearest-match" service that fulfills some but perhaps not all of the criteria for a remote interaction. For example, "make the ambient light very dim" could be carried out as "switch off the light" in an environment where lights do not have the facility to be dimmed. Or "display this colored video on the nearest screen" can be carried out as "display this video on the nearest screen" where only a mono-colored screen is available in the environment. This facility can be useful for graceful degradation of services, as well. Clearly, strongly typed static interfaces are not the right paradigm for such a system. This has, indeed, been recognized by recent middleware systems that now have facilities for dynamic invocations [28] and use reflection [29], essentially changing the semantics to message passing and pushing the problem to dynamic typing.

5 So what is the alternative?

To recap, what is needed is a set of primitives that allow the decoupling of messages in either direction, both in space i.e. to a different host or process with independent properties, and in time, alleviating from the synchronous blocking semantics. The primitives should be generic, allowing interoperability, and type checking should be done at runtime. All in all, it should provide a unifying, efficient abstraction for all the ubiquitous device control functions mentioned above.

This criterion leads to an already well-understood paradigm of asynchronous events [5] implemented on top of message passing. Quite intuitively, ubiquitous device control is all about handling events; "this has happened", "please do this", "start your video stream on a particular channel", "there was an error in executing your command" etc.

An event architecture models a system as a set of producers, consumers and moderators (channel) of events. Producers of events advertise their events, and producers of events can register interest in them. The architecture can include a general-purpose event channel or moderator that serves as an event bus [30]. Registration and notification of events are decoupled operations, and do not restrict the interaction to one discrete message in either direction (there could be more than one notifications for one registration), which could be used to accommodate continuous streams. The clients only need to know about one interface REGISTER, and the servers only need to know about the NOTIFY interface. Most of the type checking is done dynamically. This lends itself naturally as a unifying paradigm for federating heterogeneous systems [31] like ubiquitous systems.

Unfortunately the traditional event architectures are not rich enough [30,29] to support all of the above requirements. What we need is an event architecture that supports the following features.

- Generic interfaces for event registration, generation and/or notification by dynamic type checking of the event streams, instead of static type checking of remote interfaces, to support federation of heterogeneous systems.
- Conditional notification of events expressed in extended event algebraic expressions (e.g. "sound the alarm if someone enters the main door and it is later than 10:00pm")
- QoS support for event delivery, to allow QoS sensitive data, e.g. multimedia streams, to be modeled as continuous delivery of events.
- Mobility support for event notification by allowing events to be delivered to a different host/process than the one registering interest in it i.e. by allowing the notification object to be set explicitly (and changed if need be).

6 Practical work

Although a few existing event architectures have some of the attributes listed above [29,30,5], none of them addresses all the issues. Among those CORBA Notification architecture comes closer to the above requirements, but its conditional notification framework has not been fully specified or implemented.

Our research group [4] has implemented an evolving framework to use events in home networking device control. We are currently working to extend the Cambridge Event Language [5] to make it a general-purpose event algebraic language to be used for ubiquitous device control environments. As these events include support for Mobility, and Conditional notification, we call them *comvets*. The events language provides event filtering, aggregation, federation, archiving and query services [23]. We have already prototyped a system using GENA architecture [32] and we are working on an implementation to use XML to encode the meta-data typing information of events to allow flexibility needed for interoperation.

The basic API offered by Comvets is the same as CEA [5], with additional support for QoS and mobility. Comvets API includes two methods, SUBSCIBE and NOTIFY. SUBSCRIBE is used by the clients to register interest in the specified events types, whereas NOTIFY is used by the server to notify the clients about the happening of an event matching a specified condition. Conditional expressions are specified in an XML encoding of CEL. Mobility is supported by two additions to the basic framework. First, clients explicitly specify the host to be notified when subscribing to events; the host notified is not necessarily the host subscribing to events. Second, DEST method is added to the basic API that can be used to change the destination of a notification subscribed for another host. QoS is supported by allowing additional QoS specific parameters to be specified with SUBSCRIBE requests. These QoS parameters are read by a QoS agent that negotiates and maintains reservation of resources to deliver future notifications.

We also note that implementing RPC semantics on top of our events architecture is a simple matter of modeling event generators and handlers as procedure calls, allowing applications written with RPC semantics to be accommodated without any modification. We have found that implementing RPC semantics on top of the events paradigm is both more efficient and offers the above stated flexibility.

7. Conclusion

We have shown in this paper that RPC is a harmful paradigm for ubiquitous device control systems. Its tightly coupled bi-directional semantics are overly restrictive, inflexible and fail to provide a unifying paradigm that satisfies the needs of such a system. We argue that instead of trying to fit these semantics in inefficient ways, such as use of multithreading, another, more generic paradigm should be used that satisfies all the requirements of such a system. We propose the use of asynchronous events paradigm, with support for conditional notification and mobility awareness – what we call *comvets*.

Acknowledgements

We would like to thank Jean Beacon for her insightful comments that helped us improve the paper. We would also like to acknowledge contribution from Daniel Gordon in implementation of the GENA prototype and his comments on an earlier draft of the paper. This paper also benefited from comments of the IWSAWC reviewers.

References

[1] Weiser, M. 1993b. "*Ubiquitous Computing*". IEEE Computer, 26(10):71--72.

[2] T. Hodes, R. H. Katz, E. Servan-Schreiber, L. A. Rowe, "*Composable Ad-Hoc Mobile Services for Universal Interaction*," Third ACM Mobicom Conference, Budapest, Hungary, (September 1997)

[3] Easy living project, MSR. http://www.research.microsoft.com/easyliving/

[4] AutoHan Project, University of Cambridge, Computer Laboratory.
http://www.cl.cam.ac.uk/Research/SRG/netos/han/

[5] J. Bacon et al. "*Generic Support for Distributed Applications*", IEEE Computer, March 2000:68-76.

[6] D. Box et al. "*Simple Object Access Protocol 1.1*", http://www.w3.org/TR/SOAP

[7] A. Tripathi, N. Karnik, M. Vora, T. Ahmed and R. Singh. "*Mobile Agent Programming in Ajanta*", In Proceedings of the 19th International Confernce on Distributed Computing Systems (ICDCS '99)

[8]OrbixEvents Programmers guide, http://www.iona.com/docs/manuals/orbix/33/html/orbixevents33_pguide/index.html

[9] P. Sewell, P. Wojciechowski and B. Pierce, "*Location-Independent Communication for Mobile Agents: a Two-Level Architecture*", Internet Programming Languages, LNCS 1686, April 1999

[10] B. Raman, R. H. Katz, A. Joseph, "*Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network*," 3rd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA2000), Monterey, CA, (December 2000).

[11] J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu, and J. Jetcheva. *"A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols"*. In Proc. of the ACM/IEEE MobiCom, October 1998.

[12] Sun Microsystem Inc. *"Remote procedure call; protocol specifications*", RFC1050.

[13] A. S. Tanenbaum and R. van Renesse, "*A Critique of the Remote Procedure Call Paradigm*," in *EUTECO '88 Proceedings, Particpants Edition*, (Amsterdam, Netherlands), pp. 775-783, North-Holland, 1988.

[14] G. P. Picco et al."*Lime: A Middleware for Physical and Logical Mobility*". Technical Report WUCS-00-05, February 2000, Washington University in St. Louis, MO, USA.

[15] Tanenbaum, A., van Renesse, R., van Staveren, H., and Sharp, G. 1990. *"Experiences with the Amoeba distributed operating system"*. Communications of the ACM , 336--346.

[16] A. S. Tenenbaum. *"Distributed Operating Systems*", Prentice Hall, 1995.

[17] Wong D., Paciorek N., Walsh T, "*Concordia: An Infrastructure for Collaborating Mobile Agents"*, in Rothermel K., Popescu-Zeletin R. (eds), Mobile Agents (Proc. 1st Int. Workshop), Springer-Verlag, LNCS 1219, 1997, pp 86-97

[18] Bakre, A., Badrinath, B.R.: *"M-RPC: A Remote Procedure Call Service for Mobile Clients"*, Proceedings of the 1 st ACM Mobicom Conference, 1995, pp. 2-11

[19] Baumann J., Hohl F., Rothermel K., Straßer M., *"Mole - Concepts of a Mobile Agent System"*, to appear in: WWW Journal, Special issue on Applications and Techniques of Web Agents, 1998

[20] J. H. Saltzer, D. P. Reed, and D. D. Clark. *"End-to-end arguments in system design"*. ACM Transactions on Computer Systems, pages 277288, 1984.

[21] Home Audio Video Interoperability, "White paper",
http://www.havi.org/techinfo/whitepaper.html

[22] D. J. Greaves, R. J. Bradbury. *"Warren: A low-cost Home Area Network*", IEEE Network, 12(1):44-56, January 1998.

[23] M. Spiteri and J. Bates, "*An Architecture for the Storage and Retrieval of Events*". Proceedings of Middleware'98, September 1998

[24] S. Mungee, N. Surendran and D.C. Schmid*t. "The Design and Performance of a CORBA Audio/Video Streaming Service"*, Hawaii International Conference on System Sciences (HICSS), Minitrack on Multimedia DBMS and the WWW, Hawaii, January 1999

[25] Andy Hopper Stuart Wray, Tim Glauert. *"The medusa applications environmen"t*. Technical Report 94-3, Olivetti Research Ltd, 1994.

[26] H. Balakrishnan, S. Seshan, P. Bhagwat, and F.Kaashoek *"Self-Organizing Collaborative Environments*", NSF/DARPA/NIST Workshop on Smart Environments, Atlanta, GA, July 1999.

[27] Object Management Group. *"CORBA Language mapping specification*", http://www.omg.org/technology/documents/formal/corba_language_mapping_specifica.html

[28] Object Management Group. *"CORBA/IIOP Specifications, Dynamic Invocation Interface*", http://www.omg.org/technology/documents/new_formal/corba.htm

[29] Edwards, W. K. , 1999. *Core Jini*. Prentice Hall

[30]Object Management Group. *"Notification Service Specification*", http://www.omg.org/technology/documents/new_formal/notification_service.htm

[31] . Bates, J. Bacon, K. Moody and M. Spiteri, "*Using Events for the Scalable Federation of Heterogeneous Components*". Proceedings of ACM SIGOPS European Workshop, September 1998.

[32] U. Saif, D. J. Gordon, D. J. Greaves, *"Internet Access to the Home Area Network*", *IEEE Internet Computing*, 54-63: Vol. 5, No. 1, January/February 2001

[33] .D. Birrell and B.J. Nelsen. *"Implementing remote procedure call"*. ACM Transactions on Computer Systems, 2(1), 1984.