Context Adaptation of the Communication Stack[*][†]

José Mocito Liliana Rosa Nuno Almeida Hugo Miranda Luís Rodrigues Antónia Lopes
Faculdade de Ciências da Universidade de Lisboa
{jmocito,lrosa,nalmeida}@lasige.di.fc.ul.pt {hmiranda,ler,mal}@di.fc.ul.pt

Abstract

This paper presents a middleware framework to support the development of context-aware adaptive communication protocols, that can be reconfigured according not only to the local context, but also to the context of the remaining remote participants. The advantages of the framework are illustrated by using the concrete example of an adaptive group communication protocol. The protocol supports a distributed chat application that can be executed in both fixed PCs and mobile devices.

1. Introduction

Today's applications need to be designed to operate in a wide range of heterogeneous devices, including servers, PCs, laptop computers, PDAs, or even mobile phones. When the application is distributed, each participant may execute it in a different device. Consider, for instance, distributed cooperative applications such as network games or messaging applications ("chat", "messenger", etc). In these applications it is perfectly possible to find scenarios where some participants execute in the fixed network and others execute in mobile devices (it is worth notice that many PDAs and mobile consoles also support multi-user interaction in *ad hoc* mode).

Given this diversity, it is fundamental to be able to design and deploy adaptive communication protocols that can be reconfigured according not only to the local context, but also to the context of the remaining remote participants. We illustrate this idea by using a multicast protocol as an example (the need to multicast messages to a group of participants is common in many multi-user cooperative applications, such as on-line games). Typically, the most straightforward design of a multicast protocol consists of implementing the multicast as a sequence of point-to-point messages (one for each participant in the system). This implementation is quite generic, as it does not depend of any specific network support (such as native multicast) but is also very inefficient. When the participants are located on the same local-area network, or inside a single autonomous system with IP-multicast support, the usage of native multicast offers a much better performance. When the participants are in large numbers and distributed geographically over a large-scale network, it can be preferable to rely on epidemic protocols to implement the multicast [17]. If some participants execute in the fixed network and others in mobile devices, fixed nodes can play a bigger role in the data dissemination to save the resources of mobile nodes. When all participants execute in mobile devices, one can use information about the available battery at each device to increase the lifetime of the network [19]. Note that similar trade-offs appear when designing other protocol layers (transport, group membership, etc). As it should be obvious from the previous example, the context adaptation of the communication protocols is a topic of major relevance.

This paper presents a middleware framework that supports the development of communication protocols that are adaptive to the distributed context. The framework includes several components, namely: a sub-system to collect and disseminate context information; a protocol composition and execution environment; a sub-system to control the adaptation and reconfiguration of the communication protocols. The paper motivates the need for these components and describes their first prototype implementation. The advantages of the framework are illustrated by using a concrete example of an adaptive group communication protocol that supports a distributed chat application, executed in both fixed PCs and mobile devices.

The paper is organized as follows: Section 2 motivates our work and Section 3 describes the *Morpheus* framework. The advantages of the architecture are illustrated by a concrete application, which is described and evaluated in Section 4. Related work is referred in Section 5. Section 6 concludes the paper and overviews future work.

[*]This work was partially funded by LaSIGE and by FCT project MICAS, POSI/EIA/60692/2004 through POSI and FEDER.

[†]A short version of this paper has been published in the Proceedings of the 25th International Conference on Distributed Computing Systems Workshops, Columbus, Ohio, USA, June 2004

2. Motivation

This paper addresses the problem of designing and implementing a middleware framework to support the development of communication systems that have the ability to adapt to the distributed execution context. Before enumerating the components of the architecture, we capture the requirements imposed by our target applications.

As we have noted before, the characteristics of the devices where the application components are running represent an extremely relevant portion of the context information required to properly configure the communication protocols. In the scope of this paper, the word context is used to describe *system context*, i.e., information that can be directly inferred from network interface cards or operating system calls. Examples are route lengths or the available network bandwidth. We do not exclude however that subsequent research in *Morpheus* also considers more high-level context information.

Different types of devices have different characteristics in terms of battery, available memory, processing capacity, etc. These characteristics bias the operation of protocols that run at each device. There is a broad range of literature covering this type of adaptation (for instance, [3, 11]).

The adaptation of the protocols to the characteristics of each device can be performed using off-line configuration (in a process similar to the one used to build versions of most current operation systems). However, there are multiple aspects of the context information that can not be foreseen. For instance, the network error rate may influence the type of error recovery: for small error rates it is preferable to detect and recover (using retransmissions) while for larger error rates it is preferable to mask the errors (using forward error recovery techniques [13]). To react to this sort of context changes, run-time adaptation is mandatory.

Our work encompasses also another form of adaptation that consists in adapting the protocols not only as a function of the local context, but also as a function of the context of the other participants in the distributed application. For example, when some participants execute in the fixed network and others in mobile devices, we intend to promote the use of protocols that make use of the better resources of fixed nodes (such as network bandwidth and energy). To support this type of adaptation one needs a service to capture and disseminate the context information. At this point, it is useful to distinguish two orthogonal aspects in the adaptation of the protocol stacks: the granularity of adaptation and the location of the adaptation code.

Regarding the granularity of the adaptation, one can perform adaptation by reconfiguring the entire communication stack or by reconfiguring just individual components of the communication stack. The first approach is typically simpler and can be applied to monolithic implementations of the communication stack. The second approach permits an accurate tuning of each individual protocol in the stack. However it requires the protocol stack to be implemented in a modular manner. Furthermore, it imposes an additional burden on the reconfiguration code, as the reconfiguration policies must balance the local optimization criteria of each individual component with global optimization criteria.

Regarding the location of the adaptation code, one may embed the reconfiguration code within the protocol code or let all reconfiguration code be executed in an independent component. When the first approach is used, each protocol includes one or more operational modes and the code to commute between modes. A typical example of this sort of protocol is the TCP implementation that automatically adjusts several operational parameters in response to the observed network behavior. However, there are disadvantages of entangling the reconfiguration code with the protocol code: it becomes harder to re-use the reconfiguration policies in another context and it becomes impossible to apply global optimization policies (that do not consider each protocol individually, but the whole system). For instance, using this approach it may be impossible to exclude a protocol from the communication stack. An alternative approach consists in having the reconfiguration be controlled by an independent component that applies global configuration policies.

Many of the ongoing research projects in this field experiment with different combinations of the strategies referred above [12, 7, 9, 18, 1]. In this paper we are interested in building a framework that supports fine-grain reconfiguration of protocol compositions and where the reconfiguration is controlled by a dedicated component that is able to apply global optimization policies. Therefore we need to use a protocol composition framework that simplifies the task of combining and adapting individual protocol components and to develop mechanisms that allow the control component to obtain information about the system context in order to apply the reconfiguration policies.

3. The *Morpheus* framework

To address the requirements above we have designed a middleware framework called *Morpheus*, which uses the following components: a protocol composition and execution environment, that eases the task of building modular protocol compositions; a system to collect and disseminate context information; a control and reconfiguration subsystem, to perform the run-time adaptation of the communication protocols; a set of modular protocol components that can be combined in different manners, according to the context.

In the following paragraphs, we describe the first prototype of the *Morpheus* architecture, depicted in Figure 1. The

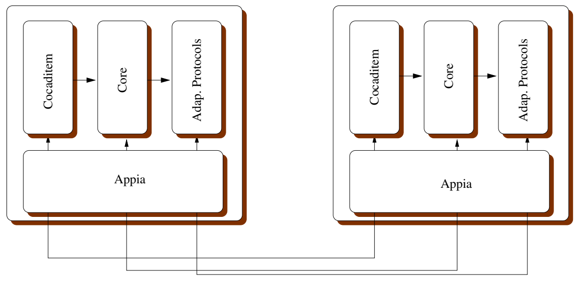


Figure 1. *Morpheus* **prototype**

goal of this prototype is to provide the means for performing an early evaluation of the architecture, illustrate the operation of each of its components and validating the interaction among those components. Therefore, this first prototype implements a simplified version of each component; the design and implementation of more sophisticated versions of each of *Morpheus*'s architecture is current work. It should be noted, however, that the prototype is fully functional and, as described in Section 4, it allowed us to perform an experimental evaluation of the architecture and demonstrate its practical advantages.

3.1. Protocol Composition and Execution Support: *Appia*

We start by describing the protocol composition and execution support environment used in our prototype. The literature is rich in systems that satisfy the modularity and reconfigurability requirements imposed by *Morpheus*, including Coyote/Cactus [4], Ensemble [8], and *Appia* [14]. We have selected the *Appia* system mainly because it has been developed "in house", has a number of features particularly well suited for our goals, and it is available in the Java programming language, which helps rapid prototyping.

Appia [14] is a framework that support the implementation and execution of modular protocol compositions. Each *Appia* module is a layer, i.e., a micro-protocol responsible for providing a particular communication service. These layers are independent and can be combined. A combination of layers constitutes a protocol stack that offers a given quality of service, QoS for short (in the broad sense of QoS, encompassing reliability, security, etc).

Once a QoS has been defined, by composing the appropriate layers, it is possible to create one or more communication channels. To each channel is associated a stack of *sessions*: for each protocols layer there is a session responsible for maintaining the state required for the execution of the correspondent protocol. Two channels that share a given layer may share the same session. In this case, the protocol may correlate events exchanged in different channels with the help of the state maintained by the shared session. For instance, if two different channels share a session of a causal order protocol, messages exchanged by these channels are ordered among each other.

Layers interact through the exchange of events. Events are typed and each layer is responsible for declaring which types of events it needs to processes and which type of events the layer creates. Using this information, *Appia* system automatically optimizes the flow of events in the stack.

A recent extension to the *Appia* system, developed in the context of this work, allows for the run-time to dynamically instantiate a channel based on its XML description [15]. This functionality is particularly useful as a means for the control and reconfiguration sub-system to load the correct configuration in each node.

The *Appia* distribution also includes a protocol suite implementing reliable group communication. This protocol suite has been used and enhanced in our prototype. Available group communication services include membership services, reliable multicast services, view-synchrony, ordering services (causal and total order), among others.

3.2. Context Capture and Dissemination System: *Cocaditem*

The *Co*ntext *Ca*pture and *Di*ssemination Sys*tem* (*Cocaditem*) is, as the name implies, the subsystem for capturing and disseminating context information in the prototype. This is a distributed component, composed of: *i)* a set of context retrievers, located in all nodes of the system, and; *ii)* a publish-subscribe component responsible for disseminating the collected information to the interested parties.

The current prototype of *Cocaditem* implements a topic-based publish-subscribe interface. The components interested in this information (namely the control component described below) subscribe the topics required for their operation.

The *Cocaditem* is a distributed component executed in each node of the distributed system. It is responsible for collecting the local context information and for coordinating with others to select the most effective manner to make this information available to subscribers. The actual *Cocaditem* implementation is quite simple. A group communication control channel is used to coordinate all instances. Each multicasts in the control channel the locally collected context information. This is a clearly simplified and non-scalable version of the publish-subscribe system under development, but enough to support the example used in the paper to demonstrate the *Morpheus* architecture.

3.3. Control and Reconfiguration: *Core*

Similarly to the *Cocaditem* sub-system, the control and reconfiguration is also a distributed sub-system, composed of two main components: *i)* a control component, responsible for monitoring the state of the distributed application and for coordinating the reconfiguration and; *ii)* a set of local modules, responsible for locally deploying a new configuration of the communication protocols when needed. *Core* components also coordinate using a group communication control channel (in fact, for performance reasons, *Core* and *Cocaditem* share the same control channel even though they are logically independent).

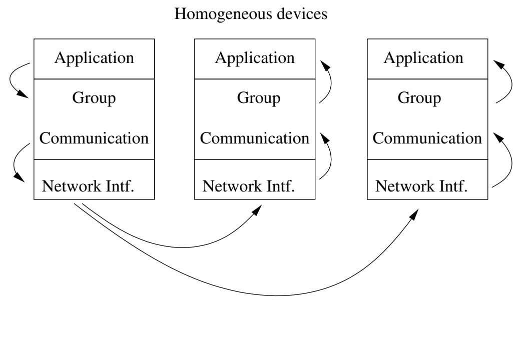
The current version of the control component is based on a coordinator, deterministically elected in run-time among all the members of the control group (the election and the replacement of a failed coordinator can be trivially derived from the properties of the underlying group membership service). The coordinator is responsible for deciding when adaptation is required, in response to changes in the context information.

The reconfiguration procedure executes as follows. The coordinator first instructs all participants to trigger a group view change in the data channels. The view-synchronous properties of the group communication protocol suite ensure that those channels become in a quiescent state. In parallel, the coordinator sends to each participant the configuration that should be deployed at that node. For this purpose, a XML description of the relevant channels is used. Then, each local *Core* module deploys the new protocol stack (using the *Appia* features) and the data flow is reinitiated.

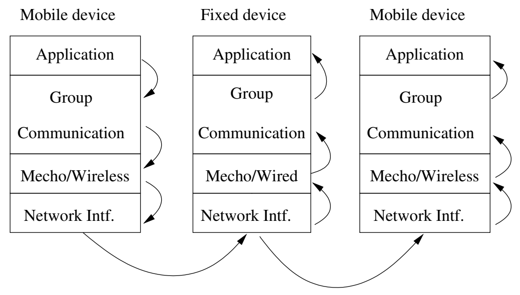
3.4. Adaptive Protocols

One of the responsibilities of the *Core* sub-system will be to evaluate context information in order to select the more adequate configuration. As the number of protocols made available to *Core* increases, so will the number of alternative compositions and, therefore, the adequacy of the compositions to the running environment. To illustrate the adaptation to the context in group communication protocols, we have designed and implemented a protocol stack that includes an adaptive multicast algorithm that we have named *M*ulticast *Echo* (*Mecho*).

Multicast Algorithm: *Mecho* *Mecho* implements an algorithm for a best-effort multicast service and is executed at the base of the group communication protocol suite used in our prototype. We recall that, on top of this best-effort service, one offers a broad range of services including reliable view-synchronous multicast.



(a) homogeneous



(b) hybrid

Figure 2. Protocol stacks

The original (non-adaptive) best-effort multicast implementation of the *Appia* group communication protocol suite implements multicast as a sequence of point-to-point messages (one for each member of the group). When available, it may also use native multicast. This approach is adequate when the system is homogeneous (i.e., when all nodes are mobile nodes or connected to the fixed infra-structure). However, in hybrid scenarios where the mobile nodes are in range of the base station and one or more hosts are connected to the fixed infra-structure, the protocol stack is extended with *Mecho*, whose behavior is quite different. In this case, mobile nodes only send a single point-to-point message to a selected fixed node which, in turn, is responsible for relaying the message to the remaining participants.

The *Mecho* protocol was designed in a modular manner and, according to its operational mode (wired or wireless node), it is implemented by a different algorithm. Figure 2(a) presents a configuration with homogeneous devices, where *Mecho* is not used while, Figure 2(b) outlines a configuration using *Mecho*.

4. Application and Validation

With the aim of validating the prototype described in the previous section, we have implemented a simple multi-user chat application. Each group of users, defined from their interests, is supported by a different multicast group. The application relies on the *Appia* group communication protocol suite to exchange data among the users of the chat application. A non-adaptive implementation of the application would require each mobile node to send one data message for each participant at each user interaction. The adaptive version that we have implemented using the *Morpheus* architecture reconfigures the underlying best-effort service that supports the group communication protocol suite, as described in Section 3.4.

We have measured the advantages of adapting the communication stack by comparing the number of messages sent by mobile devices using the non-adapted against the adapted version of the best-effort multicast protocol during the use of the *chat* application. Fixed participants executed in PCs running either *Windows* or *Linux*. Mobile participants executed in HP Ipaq 5550 PDAs using a 802.11b wireless network and the *Jeode* Java run-time. For each version we have experimented scenarios with 2, 3, 6 and 9 nodes. Each run consisted of the exchange of 40.000 messages at the pace of 10 *msg/s*. We have counted all the messages transmitted by the mobile device, including data and control messages.

The results, depicted in Figure 3, have shown that significant advantages can be obtained by using the Mecho protocol in hybrid scenarios. While for two nodes the number of messages sent is approximately the same for both configurations (in fact, all interactions are point-to-point), for a larger number of nodes the adaptive protocol is able to prevent the increase in load of the mobile node (naturally, at the expense of an increase in the number of messages of the fixed node)[1].

5. Related Work

In the context of wireless applications, there have been many proposals of adaptive systems that automatically reconfigure according to the characteristics of the devices or the quality of the communication link. However, most approaches do not rely on a middleware framework able to integrate adaptation and protocol composition. A reference architecture has been proposed in the MobileMan [6] project. MobileMan assumes that each protocol must adapt independently of the remaining protocols and does not provide any support to reconfigure, in run-time, the stack of

[1]Note that even in the adaptive version there is a small increase in the traffic due to the need of exchanging more control information.

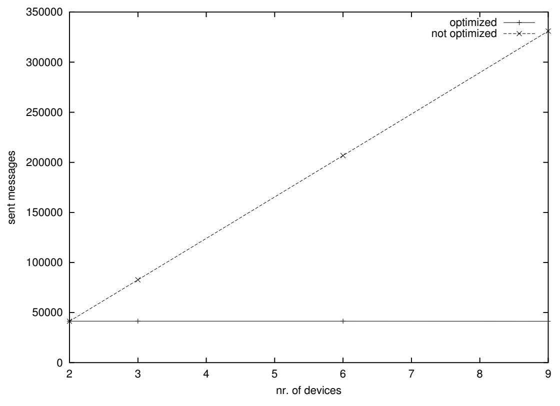


Figure 3. Messages sent by mobile nodes

protocols to be used by the application. Chisel [10] supports adaptive protocols using reflexive programming language constructs, such as Iguana/J. However, it also does not allow the reconfiguration of the entire communication. In both MobileMan and Chisel it becomes very difficult to use protocols other than those built specifically for those architectures, as the adaptation logic is entangled inside the protocol implementations.

Odyssey [16] is a framework for data access in mobile environments that is used for context-aware adaptation. The system notifies the application when a relevant change in the available resources occurs and the application is responsible by reconfiguring itself accordingly. In this paper we are interested in performing the adaptation of the communication protocols in a manner that is transparent to the application.

The WebPads [5] system exploits adaptation in Web access services. Context information is collected by the individual components of the communication stack, that reconfigure autonomously. This makes very hard to perform a global reconfiguration that takes into account the context of all participants.

The modularity of the *Morpheus* architecture is particularly suited to encompass a myriad of adaptation algorithms, possibly developed in independent projects. As an example, we cite [2], which proposes to relocate services to the nodes near the barycenter of an ad hoc network and that could be easily adapted to complement *Mecho*'s functionalities.

6. Conclusions and Future Work

Morpheus is a *middleware* framework that supports the development of adaptive communication systems. This paper motivates and describes the main components of the architecture. A first prototype of the architecture, built to validate the interaction among the components, and to illustrate the benefits of the resulting system has also been described. Early experimental results illustrate the advantages of building adaptive communication systems that are able to reconfigure according to the context. We are currently implementing more sophisticated versions of each component of the *Morpheus* architecture.

References

[1] Fault-tolerance in audio / video communications via best effort networks. http://www.informatik.uni-hamburg.de/TKRN/world/abro/ongore.htm.

[2] M. Avvenuti, D. Pedroni, and A. Vecchio. Core services in a middleware for mobile ad-hoc networks. In *Proc. of the 9th Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, pages 152–158, 2003.

[3] N. M. Belaramani, C.-L. Wang, and F. Lau. Dynamic component composition for functionality adaptation in pervasive environments. In *Proc. of 9th Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, 2003.

[4] N. Bhatti, M. Hiltunen, R. Schlichting, and W. Chiu. Coyote: A system for constructing fine-grain configurable communication services. *Trans. on Computer Systems*, 16(4):321–366, 1998.

[5] S. N. Chuang, A. T. S. Chan, J. Cao, and R. Cheung. Actively deployable mobile services for adaptive web access. *Internet Computing*, 8(2):26–33, 2004.

[6] M. Conti, G. Maselli, G. Turi, and S. Giordano. Cross-layering in mobile ad hoc network design. *Computer*, 37(2):48–51, 2004.

[7] R. Cunningham. Architecture for location independent corba environments. Master's thesis, University of Dublin, Trinity College, Sept. 1998.

[8] M. Hayden. *The Ensemble System*. PhD thesis, Cornell University, Computer Science Department, 1998.

[9] J. Keeney and V. Cahill. Chisel: A policy-driven, context-aware, dynamic adaptation framework. In *POLICY '03: Proc. of the 4th IEEE Int'l Workshop on Policies for Distributed Systems and Networks*, page 3, 2003.

[10] J. Keeney and V. Cahill. Chisel: A policy-driven, context-aware, dynamic adaptation framework. In *Proc. of the 4th Int'l W. on Policies for Distributed Systems and Networks*, pages 3–14, 2003.

[11] S. Lalis. Supporting adaptive operation in a dynamically composable personal system. In *Proc. of the Workshop on European Research on Middleware and Architectures for Complex and Embedded Cooperative Systems*, 2003.

[12] S. Lalis, A. Karypidis, A. Savidis, and C. Stephanidis. Runtime support for a dynamically composable and adaptive wearable system. In *ISWC '03: Proc. of the 7th IEEE Int'l Symposium on Wearable Computers*, page 18, 2003.

[13] M. Luby, L. Vicisano, J. Gemmel, L. Rizzo, M. Handley, and J. Crowcroft. Forward error correction (FEC) building block. RFC 3452, 2002.

[14] H. Miranda, A. Pinto, and L. Rodrigues. Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Proc. of The 21st Int'l Conf. on Distributed Computing Systems (ICDCS-21)*, pages 707–710, Phoenix, Arizona, USA, Apr. 16–19 2001.

[15] J. Mocito, L. Rosa, N. Almeida, and L. Rodrigues. AppiaXML - brief tutorial. http://appia.di.fc.ul.pt/, Sept. 2004.

[16] B. D. Noble and M. Satyanarayanan. Experience with adaptive mobile applications in odyssey. *Mobile Networks and Applications*, 4(4):245–254, 1999.

[17] J. Pereira, L. Rodrigues, M. J. Monteiro, R. Oliveira, and A.-M. Kermarrec. Neem: Network-friendly epidemic multicast. In *Proc. 22th Symp. on Reliable Distributed Systems (SRDS'03)*, pages 15–24, 2003.

[18] P. Veríssimo, V. Cahill, A. Casimiro, K. Cheverst, A. Friday, and J. Kaiser. Cortex: Towards supporting autonomous and cooperating sentient entities. In *Proc. of European Wireless 2002*, pages 595–601, Florence, Italy, Feb. 2002.

[19] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. Energy-aware wireless networking with directional antennas: The case of session-based broadcasting and multicasting. *Transactions on Mobile Computing*, 1(3), 2002.