A Middleware Service for Coordinated Adaptation of Communication Services in Groups of Devices

Jordan Janeiro, Thomas Springer
Technische Universität Dresden
Faculty of Computer Science, Computer Networks Group
Dresden, Germany
{jordan.janeiro, thomas.springer}@tu-dresden.de

Markus Endler
Departamento de Informática
PUC-Rio
Rio de Janeiro, Brazil
endler@inf.puc-rio.br

Abstract—Recent research in pervasive computing has shown that context-awareness and dynamic adaptation are fundamental requirements of mobile distributed applications. However, most approaches that focus on context-aware dynamic adaptation use only the context information available at the mobile device to trigger a local adaptation. However, for distributed collaborative applications this is clearly insufficient, since a same adaptation has to be done, in synch, at all mobile devices of the group, and hence should also be based on a commonly agreed context. Therefore, for such kinds of applications one requires mechanisms and protocols to exchange the context information among the devices and to coordinate the adaptation operations at a group of mobile device. In this paper we present a middleware service for coordinated adaptation of communication services in groups of devices. At each device this adaptation is achieved with minimal disruption for the application's remote interactions. This middleware service is based on the notion of global context and a generic protocol for global context election and synchronization of the adaptation steps, which we called Moratus. Our middleware service was implemented using JGroups and evaluated for groups of up to 30 devices, showing acceptable latency for groups of such size.

I. INTRODUCTION

Recent research has shown that context-awareness and dynamic adaptation are fundamental requirements of pervasive distributed applications. Such applications generally monitor their device's system context in order to trigger local dynamic adaptations whenever they detect significant variations of the network conditions or the local resources' states. Typical examples are the reduction of the communication frequency to save energy, or the exchange of the message encoding procedures in order to improve the efficiency of the wireless communication.

Most work adopt Deys’ context definition \cite{1} and use only local context information available at the device about the system's state (i.e., the device's resources and the quality of connectivity) and/or the user (i.e. her preferences and hints of her activity) to perform corresponding local adaptations at the application’s client, to maximize overall performance and usability.

However, for mobile collaborative applications, which employ N-to-N communication patterns, it is not sufficient to use only context information from a single device, since this may not reflect the adaptation needs and capabilities of the other devices of the group. Hence, there is need of a new notion of context - a global context computed for all devices of the group - that can be used for triggering a global adaptation in a coordinated way. For instance, consider a group of mobile devices executing a distributed collaborative application such as iph \cite{2} DreamTeam \cite{3} or \cite{4} in a common open collaboration session, and with the capability of enabling different message compression algorithms in their communication layer. Of course, all devices should adapt their compression algorithms in synch, since otherwise some messages from one peer would not be delivered accordingly at the other peers. In this case, the global context could be, for example, the minimum amount of free memory or the minimum throughput level (1Mbps, 5Mbps or 11Mbps) being used by the IEEE 802.11 wireless card at each of the peers.

Thus, to make sure that all peers of the group switch to the same compression algorithm for correct communication (i.e. correct encoding and decoding of group messages), they have to use a common context value in order to guarantee that the distributed adaptation choice leads to the same compression/de-compression components. The other alternative would be that a single peer decides for an adaptation, based on its local context only, and then forces the other peers to perform the same adaptation. In this case, however, adaptations may be concurrent, and since each peer may be at a different stage of the adaptation-specific decision process an inappropriate, or outdated, global adaptation may occur. Thus, the best way to guarantee that the group of devices executes the same adaptation in synch is to use consensus to reach a global context.

However, as the mobile devices are typically connected through wireless links, disconnections may occur. Therefore, the entire coordinated adaptation protocol must be resilient to temporary disconnections of some peers of the group.

Moreover, at each peer the aforementioned adaptation should be achieved with minimal disruption of the application, i.e. it should not interrupt its ongoing remote interactions (message exchange) during the course of the coordinated adaptation.

There exist some approaches for coordinated dynamic adaptation of groups of devices, but each of these works has some limitations. For instance: Ensemble \cite{5}, does not execute a
transparent system component adaptation (the devices interrupt their communication in order to execute the adaptation); NeCoMan [6] and Graceful Adaptation Protocol [7] do not handle device disconnection; and NeCoMan is limited to executing adaptations only for pairs of devices, but not for larger groups of devices.

In this paper we introduce the notion of a global context as a trigger to execute coordinated adaptations. We also propose a general protocol for global context election and coordination of adaptation operations, called Moratus. Based on it, we implemented a middleware service to support the execution of coordinated adaptation transparently and without disruption for the distributed application.

The paper is organized as follows: in section II, we introduce the basic concepts used by our approach. In section III, we describe and discuss the Moratus protocol and its basic steps, i.e. for computing the global context and for performing the coordinated adaptation. In section VI and V respectively, we report on our prototype implementation of the middleware service and its performance evaluation. The paper concludes with a summary and outlook to future work.

II. BASIC CONCEPTS

A. Global Context

The variation of the local system context is an important information that determines the necessary local adaptation at a device. However, for coordinated adaptations in a group, all the members have to agree on a single value for a context variable (e.g. freeMemory, throughputLevel, RSSI, etc.) in order to determine which adaptation - corresponding to this context variation - should be executed by all group members.

For example, in most group communication systems, the communication protocol modules for reliable/un-reliable delivery, FIFO/Causal/Total ordering, and message encoding/decoding, are the most common targets of protocol adaptation, as each one provides specific communication guarantees at the cost of differentiated performance overheads. Conceptually, each such protocol element has two basic functions: the encode/send of a message at the sender, and the decode/accept of a message, at the recipient.

To reach the consensus about the adaptation to be performed, the group has to decide about the global state of its context, called global context and then, based on this global state, each group member can deterministically select the specific adaptation to execute at its communication layer. The approach used in this paper is that each group member distributes its local context to the remaining group members, and each of them individually applies a deterministic function to determine the global context. Moratus uses the fact that the function is deterministic and its implementation is the same for all instances [8], to guarantee that the group members determine the same global context and therefore select the same collective adaptation.

For example, a high-efficiency data compression algorithm A can be associated with a certain RSSI interval \(i1\) \((-51\text{ dbm} \text{ to } -100\text{ dbm})\) and a low-efficiency data compression algorithm B associated with a certain RSSI interval \(i2\) \((-20\text{ dbm} \text{ to } -50\text{ dbm})\). If the communications among some of the group members experience some interference then the value of global context value RSSI will probably fall into the interval \(i1\), and then the group members would all together choose compression algorithm A.

B. Message Transformation Services

The adaptation process defines a sequence of operations that a device has to execute to dynamically adapt its protocol components coherently, consistently and without interfering with the application’s execution. In the case of this work, the adaptation relates only to communication protocol components, which we generically call a Message Transformation Service (MTS).

A MTS is a distributed service which runs at both communication endpoints (sender/receiver) and which modifies the data format (transcoding), the size and/or the content (data summarization) of the messages exchanged between the group members. Such services are, for example: encryption, compression or message fragmentation. The idea is that applications use MTS at the sender to encode messages for transmission and at the receiver, to decode the received messages. Therefore, we assume that each MTS is composed of two modules: the encoder module and the decoder module.

III. MORATUS

Moratus is a distributed protocol which coordinates the collective MTS adaptation process within a group of devices that interact with each other. In a mobile collaboration scenario, like m-learning or cooperative design, Moratus starts the group adaptation process as soon as it detects a significant variation of the local context at a device.

The protocol extends the context-awareness concept, using the local context variation as an event which triggers the adaptive process execution, but the decision about which adaptation to execute depends on the resulting context consensus among the group members. Therefore, Moratus is a protocol for coordinated adaptation that uses group consensus to determine the global context based on local context of all participating devices.

We identified three main requirements for Moratus. One of the requirements is that Moratus should execute on mobile devices connected through (IP-based) wireless networks. In such a network, it is natural that the mobile devices may eventually become disconnected unintentionally due to radiofrequency interference or sudden drop of the RF signal intensity [9]. Considering this common wireless network problem, Moratus has been designed to be tolerant to disconnections and network partitions.

Another problem of wireless networks is, a direct consequence of the frequent signal intensity variations, is the high message loss rate. To handle this problem, the second requirement was to guarantee reliable exchange of messages among the group members, as the coordinated service adaptation within the group heavily depends on the correct exchange
of protocol control messages. Consequently, Moratus employs a reliable group communication mechanism, called JGroups [10] in order to guarantee reliable multicast communication.

Finally, the third basic requirement is that the MTS adaptation process should be transparent for the applications which use such services. Consequently, in order to achieve this, Moratus establishes a uniform interface to access any message transformation service, and a specific order of deployment of its decoder and encoder modules, as will be explained at the end of section III.

A. Protocol Steps

The message transformation services’ coordinated adaptation process comprises two steps: global context election and message transformation service adaptation. The first step is responsible for the determination of its global context, and in the second step the group executes the collective MTS adaptation. Both steps will be clarified along this section.

1) Global Context Election: This protocol step allows the group members to reach a consensus in order to determine a unique global context. Like every consensus protocol, one of the requirements is that the group participants exchange their local information (local contexts) with the other group members. Figure 1 presents a state chart diagram that describes the whole process for global context election within a group.

The adaptation process starts when a device detects a meaningful local context variation. The device which detects the variation locally starts immediately a global context election. It spreads a global context calling message with its local context through the whole group in order to notify the group about the beginning of a consensus. All other group members receive this calling message and prepare a receiving global context election answer message which is sent to all other group members.

After a group member has initiated the global context election, or answered a global context election call, it should wait until it receives answer messages from all other group members. In this way, it will collect the local context of whole group members. As soon as answer messages from all group members have arrived a global context can be determined. Whenever a group member receives a message with the local context attached to it from another group member (global context election convocation message or global context election answer message), the receiver must verify whether it has all the remaining local contexts belonging to every device within a group, in order to determine when the global context election ends.

As the group members are mobile devices, some of them can disconnect unintentionally due to factors like: wireless signal mitigation or battery emptiness. In case a device disconnects at this step, the unbounded wait problem may arise. In such situations, the whole group waits indefinitely for the context of a disconnected device, since to reach a consensus every participant must contribute with its local context. Therefore, as soon as the group detects a device disconnection, Moratus will handle the disconnection and assist the group to reach the consensus instead of waiting for a disconnected device. Therefore, the global context election step can treat device disconnection before the device sends its local context. However, when a device sends its context to the group and suddenly disconnects, another problem arises, because this protocol step will take also into account such local context to determine the global context. Despite this approach is not appropriate for some applications, due to the consideration of the local context of the disconnected device, it makes sure the end of the global context election process.

In order to clarify the last situation, consider the situation where two members of a group, m1 and m2, participate in the global context election. m1 is the responsible to call the global context election and it sends its local context to the group members, in this case only m2. Then, m1 disconnects itself from the group. Just after m2 receives m1’s local context, m2 finishes the global context election and determine locally the group global context. However, both operations (m1’s local context transmission and m2’s global context processing) may have been executed so fast that m2 could not be aware of m1’s disconnection. Hence, in this case, the problem is that m2 consider m1’s context in the determination of the global context.

2) Message Transformation Services’ Adaptation: This step is responsible for the installation and activation of the message transformation services in each group member. The step encompasses four phases to allow a transparent MTS adaptation to the application which uses Moratus. The existing phases and its execution order are as follows:

1) Service Installation. If the modules for message encoding and decoding are not available locally, they have to be obtained remotely by all group members.

2) New Decoder Activation. Moratus loads the new decoder module of the MTS. At this moment, both decoder modules, the old one and the new one, are simultaneously active. Hence, during a certain time interval, the application using Moratus is able to decode two different message types.

3) New Encoder Activation. Moratus changes the old en-
4) *Old Decoding Deactivation*. Moratus removes the old decoder module.

In the beginning of this step, before the service installation phase, the group determines which device acts in the role of the coordinator. Such role is important in this step because the coordinator is the entity that makes sure all group members are synchronized through the phases.

After accomplishing the phases 2 and 3, a group member must send a message to the coordinator, in order to notify it about the correctly and successfully phase execution. It is unnecessary to send a notification message to the coordinator after the phase 1, since the execution of this phase does not require synchronization with the coordinator. After receiving all *phase conclusion confirmation messages* from each group member, the coordinator requests the group members to start the next phase, or determines the adaptation process is over, in case of phase 4.

This step is also able to handle disconnections, like the previous step. It uses the same mechanism described for global context election step, in order to avoid the unbounded wait problem. However, the difference is that in this step the protocol must handle the disconnection of a device which assumes the coordination role. In this case, the procedure is to select the next group member of the view to replace the disconnected one.

**IV. SERVICE FOR COORDINATED SERVICE ADAPTATION**

The distributed middleware service is called SACS (Service for Coordinated Services Adaptation) and it implements the Moratus protocol. Such middleware service allows applications based on group communication to use it in order to modify the communication of group members, through the message transformation services, as presented in Figure 2. Such applications are called SACS user applications along this paper.

![Fig. 2. Architecture for using SACS middleware service.](image)

As Moratus protocol is responsible to coordinate the collective adaptation of the message transformation services, all SACS user applications, executing in each group member, communicate using the same message transformation service agreed among the group members. Besides, as Moratus executes the MTS adaptation in a transparent way, none of SACS user applications realize that an adaptation process is executing. A common uniform interface makes sure the adaptation process is transparent, as well as the substitution of the encoding and decoding modules in a synchronized way.

As SACS is designed to execute dynamic and transparent adaptation of software components, which are important requirements for middleware based on group communication, it was designed as a middleware service to be easily incorporated in other middleware of the same type that needs such requirements.

**V. IMPLEMENTATION**

SACS architecture has the Monitor component, which is responsible to obtain the computational context information of a device. In this work, we use the Context Information Service (CIS), a service offered by the middleware MoCA, to retrieve device context information.

CIS is a client/server service, where the server receives the device system context information periodically. The client, called *monitor*, is a daemon responsible to collect the device context information and send them to the CIS server. Such context information is for example: processor percentual usage, battery power status and total available RAM memory [11].

There are two approaches to retrieve device context information within CIS: synchronous query and asynchronous query. The first approach follows a request/response communication protocol, where the applications request the context information of interest and wait for the CIS to respond containing the requested context information. In the asynchronous approach, the application subscribes with CIS to receive notifications whenever occurs a context variation associated to, at least, one context variable [11]. For this work, we choose the synchronous approach, since CIS does not allow the asynchronous access to the wireless signal strength context variable, which, for SACS, represents an indicator to the wireless link quality.

Moratus requires reliable group communication because no message related to the coordination of the adaptive process should be lost. Otherwise group inconsistencies may be the result. We used JGroups [10], a framework for reliable group communication. Through JGroups, processes can connect to a group and send reliable messages to the group members. It also monitors group members and notify them about new group views (members list). JGroups notifies each new group view whenever a group member joins the group or leaves it, spontaneously or due to a fault.

**VI. EVALUATION**

The performance tests aim to evaluate the protocol scalability relating to the variation of the group elements. These tests measure the message transformation service adaptation times as a function of variation in the size of the group. The coordinated group adaptation performance tests were executed
with groups of: 5, 10, 15, 20, 25 and 30 devices and with a 2 kilobytes message transformation services.

We executed the performance tests in one of the cluster in PUC-Rio University, due to the ease to use many devices with the same configurations and capabilities. Such cluster is composed by 32 machines configured as: CPU Intel Pentium II 400 MHz and 280 Mb of RAM. The network connections between the machines are about 100 Mbps. As Moratus is implemented through simple algorithms, which do not require much processing time, we believe that the results of the executed tests in mobile devices are similar to the executed tests in the cluster machines. However, the major difference is the network bandwidth, which is faster and more reliable in the cluster than in 802.11 wireless networks.

The tests consist of a group member realizing a meaningful local context variation and calling the global context election. Then, the group members select the adaptation process coordinator, as stated in the section Message Transformation Services’ Adaptation, and this one coordinates the message transformation service substitution.

![Fig. 3. Elapsed time for the Moratus adaptation process.](image)

Figure 3 presents a graph which shows that, the increasing of the adaptation process time as the number of group members grow is approximately linear. This represents that Moratus is a scalable protocol and executes the adaptation process in a reasonable time even for groups of 30 elements. Considering this graph as a basis, we believe the function follows a linear growth, even for greater groups, which maintains proper group adaptation execution times.

Through this graph it is also possible to realize that the elapsed time for the global context election step is shorter than the elapsed time for the service adaptation step. This is due to the implementation of each step. On the first step, each device sends a message to the group with its local context. On the second step, for each of the five phases, each device sends a message to the group, confirming the conclusion of the phases, resulting in a greater convergence time for the step, due to the processing time for each message.

The Figure 4 shows a graph presenting the elapsed time for each of the five phases related to the message transformation service adaptation step (second step).

![Fig. 4. Elapsed time for each phases involved in the Message Transformation Service adaptation step.](image)

This graph highlights that the installation phase requires more time than the other phases. This is due to the requirement of the message transformation service by SACS and the time it waits until the service arrival from the network. As the beginning of this section states, this time is related to 2 kilobytes service size. We expect that the time of the phase increases for a service size greater than the one used for this tests.

**VII. RELATED WORK**

This work raises four requirements in order to compare coordinated software adaptation protocols: transparent adaptation, disconnection tolerance, protocol stack composition and coordination of distributed adaptation. The first requirement refers to how the user percepts the substitution of a distributed service. The second requirement refers whether the device handles a group member disconnection. The third requirement refers to the existence of a mechanism to allow the coordinated adaptation of a service in groups of devices. The last requirement refers to whether an approach allows the connection of a set of protocols in order to establish a protocol stack.

We compare Moratus with three other approaches, described in the introduction section: NeCoMan, Graceful Adaptation Process and Ensemble.

Regarding transparent adaptation, Moratus, NeCoMan and Graceful Adaptation Process cope with the requirement. However, Ensemble cannot do transparent adaptation because it interrupts the application execution to substitute a current protocol stack by a new one. After accomplish the substitution in the whole group, Ensemble resumes the application execution.

Regarding disconnection tolerance, just Moratus and Ensemble are able to handle disconnection terms or faults associated to devices. NeCoMan and Graceful Adaptation Process do not have any disconnection mechanisms; they recognize this limitation and point it out as a future work.

Regarding coordinated adaptation, Moratus, Graceful Adaptation Process and Ensemble are able to execute coordinated service adaptation in groups of devices. NeCoMan is also able to execute coordinated service adaptation, however just in pairs and not in groups like the other protocols presented.
Regarding protocol stack composition, Ensemble is the only protocol which accomplishes with such requirement. Moratus and NeCoMan do not consider the whole protocol stack substitution, just atomic services. The Graceful Adaptation Protocol also does not allow protocol stack composition, however, there are four layers to accommodate different purposes services.

VIII. Conclusion

This work presents Moratus, a protocol to coordinate the installation and activation of message transformation services in groups. Such protocol is the core element of a middleware service called Service for Coordinated Services Adaptation (SACS), which allows the definition of triggers to execute collective adaptation based on the context of the members of the whole group, called global context.

The protocol consists of two steps: global context election and message transformation service adaptation.

On the first step the protocol considers the local context of the group members to determine the global context. The group members exchange among them local contexts and each member determines locally the global context. Due to the fact that the global context computation algorithm is the same in all group members, and all of them receive the local context of all members, they reach the same result (global context).

On the second step, having the global context, the group executes a message transformation service adaptation. This step is divided in four phases to make sure a transparent adaptation of a service and to avoid a service substitution interruption during the substitution period between one service to another.

Due to the fact that Moratus is designed for mobile devices, the protocol is able to handle disconnections of group of devices. Hence, on each protocol step, there are mechanisms to handle such disconnections, making sure they do not interrupt the adaptive execution process.

This work introduces two main contributions, a global context determination mechanism and the software adaptation in groups. The first contribution is about the design and implementation of a mechanism to determine the global context within a group. There are many dynamic software adaptation mechanism proposals based on one device context information. However we did not find proposals about considering global context information of a group of devices as the main trigger to execute dynamic software adaptation. The second contribution is the development of mechanism to allow a collective software adaptation within a group, which also is able to handle the disconnections of devices and to execute transparent adaptations.

IX. Acknowledgement

This work was partially supported by the Brazilian National Research Council (CNPq) grants 474188/ 2007-8 and 550895/2007-8.