Controlling Sensors and Actuators collectively using the COCOS-Framework

Maik Krüger, Reinhardt Karnapke, Jörg Nolte
Distributed Systems/Operating Systems group, BTU Cottbus
Cottbus, Germany
maik-krueger@gmx.de,{karnapke, jon}@informatik.tu-cottbus.de

ABSTRACT

COCOS (COordinated COmmunicating Sensors)[1] is a lean middleware platform for wireless sensor and actuator networks. The major programming abstractions of COCOS are distributed sensor spaces, which enable the collective controlling of sensors or actuators on remote nodes. Thus, COCOS brings high-level data-parallel programming concepts such as global reductions into the world of small networked systems. In this paper we discuss the design rationale of COCOS as well as its in-network processing and aggregation capabilities.

Categories and Subject Descriptors

D [**1**]: Concurrent Programming

General Terms

Design, Performance, Reliability

Keywords

Wireless Sensor Networks, Collective Operations, Overlay Networks

1. INTRODUCTION

Future sensor networks will be typically composed of tiny computers with some sensing as well as wireless communication capabilities. These networks can be fairly large, ranging from a few dozens up to myriads [1] of nodes. Swarms of small robots might patrol chemical plants to detect potential problems and *react* on environmental hazards e.g. by collectively applying suitable chemical agents to neutralize spilled acid.

The nature of sensor networks being collections of computational nodes of the same kind strongly implies a data-parallel programming approach. The sensor network then appears to the programmer as a collection of language level objects that can be grouped according to some application specific criteria such as the location of specific nodes, different actuators attached to them or even dynamic criteria like specific sensor readings. In COCOS, all objects in such a user defined group can be addressed collectively and thus often recurring tasks on the same group of sensors or actuators can easily be expressed and implemented efficiently.

[1]The COCOS Project is supported by the German Research Foundation (DFG) in the SPP 1140.

SANET'07, September 10, 2007, Montréal, Québec, Canada.
ACM 978-1-59593-735-3/07/0009.

2. SENSOR SPACES

All sensor nodes are represented by language level objects that are instances of arbitrary C++-classes. Usually, only nodes that detected a local phenomenon are of interest. Therefore, an effective way to address only these nodes when performing a distributed and parallel analysis of sensor readings is needed. For instance, when the average temperature of all nodes within a certain area should be determined, we would like to group those nodes together, perform parallel aggregations (global reductions) of all temperature readings periodically and transmit the aggregated result to the outside. COCOS provides *sensor spaces* as a basic programming abstraction to support such computational patterns. First, all nodes that detected the requested phenomenon (in this case a temperature above a given threshold) will join a sensor space. Any node may now act as an evaluator and perform parallel method calls on all objects within that distributed space. Provided the detected phenomena are at least stable for a short period of time, we can construct suitable overlay networks that effectively reach all nodes of interest. The same holds for nodes that are grouped together according to criteria such as geographic location (e.g. all nodes in a defined distance around a certain location) or according to functionality (e.g. the group of all nodes carrying a temperature sensor).

However, the situation changes drastically, when either the sensor nodes are mobile, or both the wireless network as well as the detected phenomena are not stable over time. In those cases it is not possible to maintain overlay networks effectively and suitable flooding techniques have to be applied instead. COCOS reflects these differences by providing different categories of sensor spaces. The `AnchoredSpace`, `RobustAnchoredSpace` and `ConnectedSpace` are based on tree topologies, while the `ATBFloodingSpace` was designed for (mobile) networks networks which often change their logical topology. In the latter case all group operations are distributed by (limited) flooding. Consequently, the chance for a node to receive all messages in a dense network is high.

3. IMPLEMENTATION ASPECTS

COCOS is essentially composed of three layers (figure 1). Our implementation for this feasibility study is based on the operating system REFLEX [4], which takes care of scheduling and event handling, but the concepts of COCOS are independent of the underlying operating system.

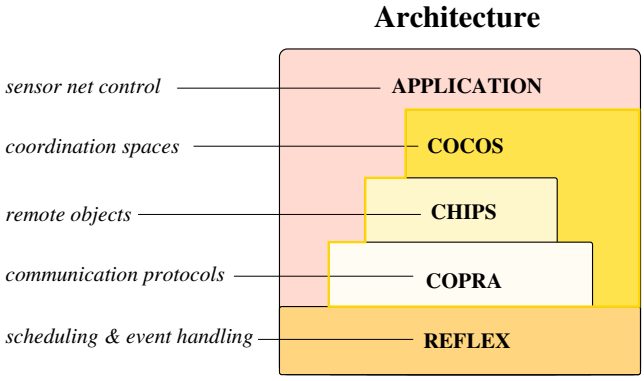

Figure 1: COCOS architecture

The lowest part of COCOS is COPRA (COmmunication PRocessing Architecture) [2], where different communication tasks, e.g. medium access control, are implemented in so called Protocol Processing Stages. These can be combined to form Protocol Processing Engines which represent entire network stacks.

The second part, CHIPS (Convenient High-level Invocation Protocol Suite), enables the usage of remote method calls and uses the communication protocols provided by COPRA. This way, the remote calls can be routed through the whole network.

COCOS (COordination and COoperation Spaces) supplies a high level group abstraction for application programmers in wireless sensor networks. Using COCOS, the sensor network can be programmed as a whole, or some part of it. This is realized by extending the communication layers from COPRA and the remote calls of CHIPS to commands for logically connected groups of nodes that reside within a distributed object space. An object space summarizes chosen sensor nodes, so that it is easy to join nodes which have special sensors or actuators or which measured values of interest. Once an object space is created group operations on all nodes in this space can be executed. These operations can include the reading of sensor values and their aggregation as well as controlling actuators, e.g. starting fire extinguishers collectively in fire areas.

At the moment there are three group operations implemented. The method `apply`() is an asynchronous one way group operation, which invokes a method on all group members. The `aggregate`()-method is a synchronous group operation which returns a value that is received by aggregating all values with a specified aggregating operation. The participating nodes block until they return their aggregated values. The algorithm which is used in the aggregation can be specified, by defining a class which is inherited from the abstract class `Aggregator`. The third method is a a deferred synchronous version of `aggregate`(), which can be used for longer operations, when blocking is not wanted.

4. RELATED WORK

Abstract regions [5] offer a tuple-space like communication abstraction. An abstract region consists of neighboring sensor nodes, which are a certain number of hops or meters distant from an anchor node around which this region is centered. Like COCOS, abstract regions are designed as parallel programs. There are three major differences between abstract regions and COCOS. First, the membership of a node in one of COCOS's spaces can change at any time, while the membership in an abstract region is static. Second, COCOS offers control over e.g. actuators additionally to the in-network processing capabilities. Finally COCOS is based on C++ templates resulting in a high-level programming abstraction without the need for a special purpose programming language.

The major difference between Regiment [3] and COCOS is that the former is based on a new functional programming language while COCOS is completely implemented in C++ which enables the usage of existing tools and saves the application programmers the need to learn a new language. Furthermore, as mentioned above, COCOS offers possibilities to control actuators, which Regiment does not.

5. CONCLUSION

The different spaces of COCOS enable convenient collective control over groups of sensors and actuators. They also offer in-network processing of values by aggregating them with user specified aggregation functions. We have shown that the usage of remote method invocation mechanisms and sensor spaces is feasible and useful by evaluation in a real sensor network, using enhanced RCX robots (figure 2).

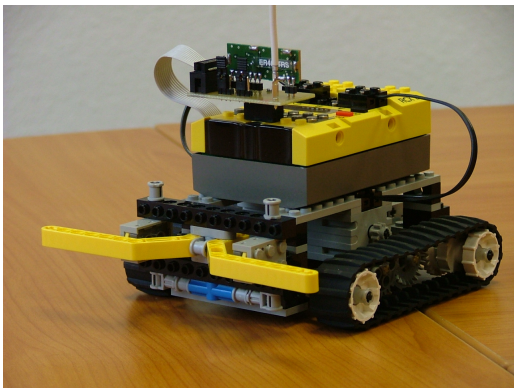

Figure 2: A modified RCX robot

6. REFERENCES

[1] I. Chatzigiannakis, S. Nikoletseas, and P. G. Spirakis. Efficient and robust protocols for local detection and propagation in smart dust networks. *Mob. Netw. Appl.*, 10(1-2):133–149, 2005.

[2] R. Karnapke and J. Nolte. Copra - a communication processing architecture for wireless sensor networks. In *Euro-Par 2006 Parallel Processing*, pages 951–960. Springer, 2006.

[3] R. Newton and M. Welsh. Region streams: Functional macroprogramming for sensor networks. In *Proceedings of the First Workshop on Data Management for Sensor Networks(DMSN 2004), Toronto, Canada*, Aug 2004.

[4] K. Walther and J. Nolte. A flexible scheduling framework for deeply embedded systems. In *In Proc. of 4th IEEE International Symposium on Embedded Computing*, 2007.

[5] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *NSDI*, pages 29–42, 2004.