Tuplespace-Based Collaboration for Bluetooth-Enabled Devices in Smart Environments

Frank Siegemund and Pascal Keller

Institute for Pervasive Computing
Department of Computer Science
ETH Zurich, Switzerland

Abstract: Smart environments are often populated by resource-restricted devices that need to cooperate with each other in order to access remote sensors and benefit from other devices' resources. This paper presents and evaluates a software platform that facilitates dynamic cooperation between resource-restricted Bluetooth-enabled devices. In order to cooperate, nodes actively participate in a distributed tuplespace that serves as a shared data structure and a medium to access remote resources. The paper motivates the communication related design decisions that led to a concrete implementation on an embedded device platform, the BTnodes. We evaluate this implementation on single- as well as multi-hop topologies and show how efficient inter-device cooperation must consider Bluetooth's communication properties as well as the resource restrictions of participating devices.

1 Introduction

Recently, Bluetooth has received considerable attention in building resource-restricted device platforms designed for smart environments and wireless sensor networks. Among others, the Intel Motes [Int04], the BTnodes [Beu04], and the sensor node platform built at the University of Karlsruhe [Bla03] all use Bluetooth for communication. However, because of the severe resource-restrictions of such embedded platforms and the consequential restricted functionality provided by isolated nodes, it is difficult to build sophisticated applications on single devices. Especially in smart environments where smart objects need to interact with each other to fuse sensory data and share resources, cooperation between multiple devices becomes increasingly important.

This paper addresses the problem of dynamic cooperation among Bluetooth-enabled devices in smart environments. Thereby, we enable nodes to form dynamic groups of cooperating entities and support collaboration by means of a distributed tuplespace spanning cooperating devices. We also present a concrete implementation on the BTnodes, evaluate its performance, and report on the major design decisions that influence the underlying communication protocols. Our experiences with the platform on single- as well as multi-hop topologies show that efficient collaboration among multiple nodes can only be achieved when the specific properties of the underlying communication technology (i.e.,

Bluetooth) are taken into consideration.

The paper is structured as follows: Section 2 presents a platform facilitating tight cooperation between devices in smart environments, which is based on a distributed tuplespace. Section 3 reports on the design decisions influencing the underlying communication protocols and evaluates their performance. Section 4 concludes the paper.

2 Tuplespace-Based Collaboration

Because of their resource restrictions, smart objects often need to cooperate with other devices in order to benefit from remote resources such as sensors, memory, or other devices' processing capabilities. In our approach, resource-restricted nodes dynamically form groups of cooperating entities by participating in a distributed tuplespace. The latter serves as a shared data medium for participating entities and hides low level communication issues from an application. In the following, we describe how a distributed tuplespace can facilitate close cooperation in smart environments and present a concrete implementation on the BTnode device platform[1].

A tuplespace [CG89] is an associative, content-addressable memory where data is stored in form of tuples. Because the underlying concept focusses on content, tuples are accessed by their form or type rather than by memory addresses. As a content-addressable memory, a tuplespace simplifies ad hoc cooperation since access to shared data is possible without knowledge of specific memory locations or service interfaces of particular nodes. A distributed tuplespace also hides low-level communication issues because the corresponding API encapsulates the distribution of tuples among different nodes.

In our implementation, each node has its own local tuplespace, i.e., a subset of its local memory where local processes can store and retrieve tuples. When a set of nodes want to cooperate with each other, they establish a distributed tuplespace. Such a shared data medium then consists of the interconnected local spaces of all participating entities, and operations that consider tuples on all nodes. For example, a distributed $read$ operation does not only search for a tuple on a single node but on all entities cooperating with each other.

The distributed tuplespace for the BTnodes does not only support the standard tuplespace operations $read$, $write$ and $take$, but does also provide more sophisticated functions that operate on sets of tuples. $countN(templ)$ returns the number of tuples that match the given template; $scan(templ)$ returns all matching tuples; $consumingScan(templ)$ behaves like $scan$ but removes all tuples found. There are always three versions of these functions: one that operates on the local tuplespace, one that operates on a distinct remote tuplespace, and one that operates on a whole set of cooperating objects.

As the distributed tuplespace implementation is tailored towards the needs of embedded devices, it provides its own *memory management* for tuples because some embedded plat-

[1] Please refer to www.inf.ethz.ch/~siegemun/software/ClusterTuplespace.pdf for a more detailed description of our implementation.

forms do not provide a reliable dynamic memory allocation mechanism. To enable processing of continuously generated data (e.g., from sensors), the implementation maintains an *age structure* which allows automatic deletion of the oldest tuples when the space is full and new tuples are generated. The concept of protected tuples, however, prevents the implementation from automatically deleting explicitly marked tuples. In addition to the basic tuplespace operations, *callback*s are supported as an event mechanism. Thereby, a callback function can be registered that is executed when a tuple of a specified form is written into the shared data space.

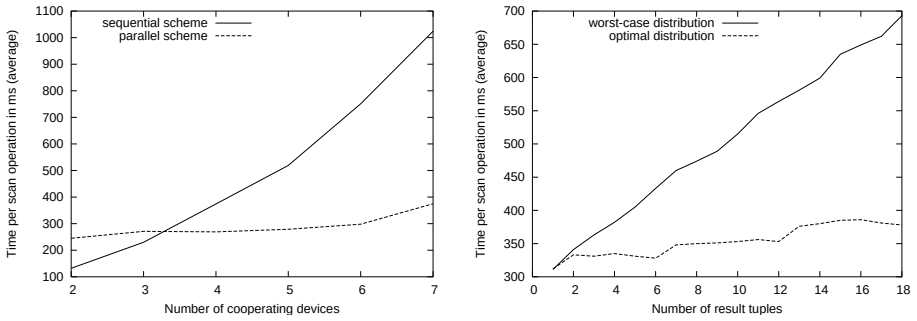
3 Communication Protocols for Effective Inter-Device Cooperation

This section focuses on the design of communication protocols that facilitate effective collaboration between Bluetooth-enabled devices. We argue that a platform for inter-device cooperation must consider the properties of the underlying communication technology, and show how the properties of Bluetooth effect the performance of such a platform.

When a distributed tuplespace is used to share data and access remote resources, the performance overhead introduced by cooperation largely depends on an efficient means to multicast data. This is because every tuplespace function that operates on a set of cooperating devices must be able to efficiently send a request to all nodes it cooperates with. To enable efficient multicasts, we suggest to adapt the network topology of the underlying ad hoc network such that cooperating nodes share a single broadcast channel. A request can then be sent to other nodes by broadcast. Thereby, it is advantageous that Bluetooth is a multi-channel communication technology. A broadcast channel in Bluetooth corresponds to a single Piconet, and multiple Piconets can operate in range of each other with only little interference. When there are many nodes in range of each other, we therefore try to group cooperating nodes into a single Piconet in order to enable efficient collaboration [Sie04].

A tuplespace function operating on a set of cooperating objects first tries to respond to a request by querying its local tuplespace. A $read$ operation for example searches for a single matching tuple. When such a tuple is found in the local space, no communication with other nodes is necessary. Otherwise, when the issuing node is not a master, it forwards the tuplespace request to its master node, which in turn evaluates the query and transmits the request to all of its slaves. Then, the slaves send their results back to the master which then forwards them to the slave that issued the request.

For evaluation purposes, we have implemented several variants of distributed tuplespace operations on the BTnode device platform. Some of the corresponding results are depicted in Fig. 1, considering a $scan$ operation as an example. Fig. 1 a) compares different means for sending requests to cooperating nodes. Thereby, we distinguish between a broadcast-based, *parallel* strategy and a unicast-based, *sequential* strategy. In the parallel strategy, the master sends a tuplespace request to all slaves concurrently via broadcast. The slaves execute the corresponding method on their local tuplespaces and immediately send back the result tuples via unicast. The master receives the results from the slaves in a parallel manner, reassembling the unicast packets that arrive from different slaves in no specific



(a) (b)

Figure 1: (a) The effect of the sequential and parallel scheme on the performance of a $scan$ operation returning 10 result tuples considering all possible distributions; (b) the effect of different tuple distributions on the performance of a $scan$ operation using the parallel scheme.

order. In the sequential strategy, the master queries one slave at a time, sending the tuplespace command and the template to each slave via unicast. The next slave is queried only after the answer of the current slave has been received.

As can be seen in Fig. 1 a) the parallel strategy clearly outperforms the sequential strategy with an increasing number of cooperating nodes. This is mainly because of two reasons. Firstly, broadcasting data to all nodes concurrently is faster than sending a request to every node separately. Secondly, Bluetooth's Time Division Duplex (TDD) scheme for handling master-slave-transmissions significantly reduces the efficiency of the sequential approach. This can be seen in Fig. 1 b) where we measured the time needed for a $scan$ operation on seven cooperating nodes with respect to the distribution of matching tuples. When matching tuples are equally distributed, a master almost never polls a slave without receiving a tuple as result. Consequently, it makes almost no difference receiving 1-6, 7-12, or 13-18 tuples when they are equally distributed. Because the Bluetooth modules on the BTnodes seem to poll nodes in a simple round-robin fashion, a master node accesses sequentially one node after the other and therefore needs i rounds to receive $6 * (i - 1) + 1$ to $6 * i$ results when the results are equally distributed. In contrast, when all matching tuples are on a single node, i rounds are required to retrieve i results.

Unfortunately, grouping nodes on a single broadcast channel in Bluetooth is only possible when nodes can be assigned to a single Piconet. When nodes are too far away from each other, they must be grouped into a scatternet. Fig. 2 shows the performance of a $read$ operation on a distributed tuple space with respect to the number of devices cooperating with each other. The results have been measured using our tuplespace implementation on the BTnodes. Fig. 2 also shows how the BTnodes where organized into scatternets; the $read$ operation was always issued on the black node in the illustration. It can be seen that as long as a $read$ operation finds a result on a node participating in its piconet, the $read$ operation scales well. But as soon as it must access nodes over multiple hops in different piconets, the performance decreases significantly.

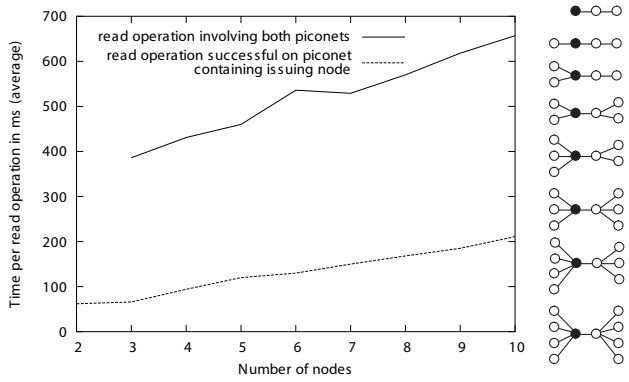


Figure 2: The effect of multihop topologies on the efficiency of distributed tuplespace operations.

4 Conclusions

In this paper, we presented a tuplespace-based approach facilitating tight collaboration among Bluetooth-enabled devices in smart environments. Thereby, a concrete implementation of a distributed tuplespace on an embedded device platform, the BTnodes, enabled us to evaluate the performance of the underlying communication protocols on single as well as multihop topologies. The main lessons learnt can be summarized as follows. (1) A distributed tuplespace supports efficient cooperation among devices when they can be grouped on a single broadcast channel. (2) The underlying communication protocols must consider Bluetooth's mechanism for scheduling master-slave communications. A parallel scheme for receiving tuples from multiple slaves is necessary to take advantage of this Bluetooth feature. (3) In contrast to implementations on fixed networks, communication is by far the most time-consuming factor while executing tuplespace operations. This is a result of the severe memory restrictions of embedded device platforms, which makes the time needed for local operations almost negligible.

References

[Beu04] J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund, and L. Thiele. Prototyping Sensor Network Applications with BTnodes. In: *IEEE European Workshop on Wireless Sensor Networks (EWSN)*, Berlin, Germany, January 2004.

[Bla03] E.-O. Blass, H.-J. Hof, B. Hurler, and M. Zitterbart. Erste Erfahrungen mit der Karlsruher Sensornetz-Plattform, *GI/ITG KuVS Fachgespräch Sensornetze*, 2003.

[CG89] N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(4), April 1989.

[Int04] Intel motes, http://www.intel.com/research/exploratory/motes.htm, 2004.

[Sie04] F. Siegemund. A Context-Aware Communication Platform for Smart Objects. In: *2nd Intl. Conference on Pervasive Computing (PERVASIVE 2004)*, pp. 69-86, Linz/Vienna, Austria, April 2004.