JSalsa: Jini Service Agglomeration in Large Scale Assemblages

Burkhard Ehleringer, Steffen Rothkugel, Peter Sturm

System Software and Distributed Systems, University of Trier,
D-54286 Trier, Germany
{ehlering,sroth,sturm}@uni-trier.de

Abstract. Jini Connection Technology provides simple mechanisms enabling devices to form impromptu systems. JSalsa investigates this promotional statement, taking it to and even beyond its limits. We modeled a setting where thousands of entities are building ad-hoc Jini communities in a radio network. Those entities are products tagged with smart labels, stored in a wireless, smart warehouse. The large number of entities together with their severely limited processing power as well as the characteristics of wireless communication technology demand for suitable implementation strategies. In JSalsa, scalability is attained through distributed self-organization in combination with agglomeration. Resource limitations are tackled via a delegation approach. We describe the concept behind JSalsa, its implementation in the form of a simulation environment and present results of simulation runs.

1 Introduction

The traditional way of labelling products with a barcode probably will be replaced by the revolutionary technology of smart labels within a few years [1]. Labels in conventional applications are only recognized when they are scanned, e.g. at conveyor belts or cash desks. This limits a warehousing solution to only registering incoming and outgoing items.

An automated solution covering the entire stock at any time, enabling a permanently consistent view at the inventory, obviously is more desirable. Because smart labels use radio signals for communication purposes, they allow for a fast and fully automated registration of products without the need for manually scanning barcodes. Through a permanent coverage by radio stations, smart labels become participants in a wireless network. Hence, warehousing gets spontaneous and dynamic. The information that smart labels provide leads to the enhanced view as services. The result is a decentralized database of services, based on spontaneously networking items.

The distinct characteristics of wireless network technology strongly influence the design of the underlying algorithms. For example, communication patterns allow to broadcast information from a base station to all participating smart labels. In the reverse direction, depending on the wireless communication technology employed, at most a subset of labels may reply concurrently. However, not only message flooding has to be avoided. Limited bandwidth and relatively high bit error rates also demand for suitable implementation strategies.

1.1 Smart labels

Smart labels are highly integrated circuits embedded for example in tags, tickets or documents. Their size is about a few millimeters and they are even thinner than a sheet of paper [2]. One advantage of smart labels over barcodes is the use of radio signals for communication purposes. Therefore, smart labels are equipped with an antenna. In contrast to bar codes, they do not require a direct line of sight and can be read from a longer distance. Furthermore, multiple labels can communicate simultaneously with a single base station.

Current smart labels contain persistent storage only. Their capacity ranges between a few bits up to some kilobytes. The memory content is modifiable and due to their further miniaturization, they are even getting "smarter". The integration of processing capabilities as known from smart cards is in progress [3]. Smart labels can contain various information, e.g. a product description or an expiry date. Due to their small size they rely on external power supply. The electromagnetic field of the antennas induces enough power to activate the smart labels.

1.2 Wireless smart warehousing

Employing smart labels in a warehouse enables a multitude of new areas of application for both the company management as well as for the customers. A permanently consistent view on the inventory allows for common tasks to be automated. For instance, not only if the inventory falls below a predefined minimum, new items can be

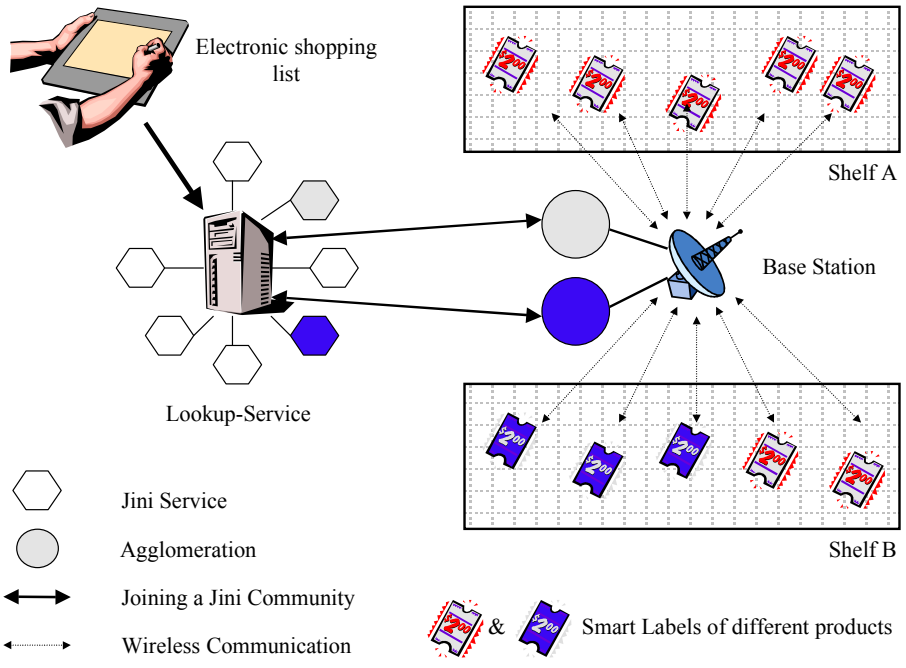


Fig. 1. Integration of smart labels into a Jini community. The scenario shows two agglomerations of different product types. Base stations act as network proxies and take over the task of providing the Jini services.

ordered autonomously. When they are delivered, they join the stock without manual intervention. Moreover, expired goods can call for attention by themselves.

The features offered to customers emphasize the advantages of smart labels represented as network services (cf. figure 1). A pocket computer containing an electronic shopping list may connect spontaneously to the warehouse's network. Based on a list of products, the warehouse server could suggest an appropriate path along all items requested. On its way, the customer may be provided with additional information, e.g. about special offers or related products. For instance, the service of a cornflakes box could suggest to buy milk.

1.3 Wireless communication

The characteristics of wireless data communication substantially affect an implementation of such a smart warehouse. The transmission of information from several participants via radio signals assumes a so-called multiple access method. Code Division Multiple Access (CDMA) [4], based on the direct spread spectrum technology [5], has proven to be a viable approach in very large mobile communication environments. In contrast to frequency/time division multiple access methods, CDMA allows several senders to transmit their data concurrently on the same carrier. The concept of CDMA distinguishes between a forward channel (base station to mobile object) and a reverse channel (mobile sender to base station).

Communication over radio signals is error-prone. In case of CDMA, the error rate mainly depends on the noise strength, the individual signal power and the number of objects sending simultaneously. The bit error rate rises exponentially with the number of interfering signals. For instance, in the case of identically distributed power levels and a number of 31 bits to represent one single data bit, the data bit error rate reaches 1% at a number of 10 interfering signals [6]. Because the power of smart labels is induced by the surrounding field, the signal strength and therefore the error rate depends on the distance to the antennas.

2 JSalsa: The Concept

Jini is a technology enabling the establishment of an infrastructure of networked services [7]. However, in application scenarios like ours where the number of items is very large, Jini cannot be applied out-of-the-box. Representing each item as a full-featured Jini service is too resource intensive. The main reason behind this is that Jini services rely on Java Remote Method Invocation (RMI) [8]. In the context of the SanFrancisco Component Framework [9], a case study clearly showed the limitations of RMI when applied in large-scale Java applications [10], mostly because of the number of threads and their memory consumption.

One of the basic ideas behind JSalsa is the concept of agglomeration. Single items form dynamic groups with membership validation. These groups establish hierarchically organized agglomerations, creating single Jini services as representatives. Agglomerations may also be decomposed on demand, e.g. when items are removed. This happens for instance when customers put products into their shopping carts. JSalsa has

been designed to work decentrally in a wireless environment. It takes advantage of the broadcast ability in the forward channel and expects the number of reliable, concurrent unicasts in the reverse channel to be limited. Furthermore, JSalsa takes the restricted capabilities of smart labels into account.

2.1 Introduction to Jini

The Jini technology establishes an infrastructure for spontaneous networking, based on the Java technology, and its concepts enhance the view of distributed systems. Any object in the network, software or hardware, is abstracted as a service that is offered by joining a community of Jini services. Among the main concepts of Jini are Discovery, Lookup and Leasing [11]. Remote Events and Transactions are not considered here.

Plugging a device into the network starts a discovery process. The device searches for suitable Jini communities and joins them by offering a service proxy via the lookup service. The lookup service basically is a directory service, capable of searching for Java object types implementing a specific interface. As a result of the lookup process, the proxies of matching Jini services are returned. Selected proxies are dynamically downloaded into the client. The client makes use of a service by invoking methods of its proxy. Calls to methods of a service proxy can be handled either locally by the proxy itself or result in a method of the service being invoked remotely, using an arbitrary protocol.

Leases are used in the Jini technology to handle failures. Clients need to apply for a lease before being allowed to use a service. Services grant the lease only for a limited time. Leases have to be renewed before they expire. The concept of leases is central to the self-healing nature of Jini. Resources remain locked for at most the duration of one lease.

2.2 Smart Labels as Jini services

The requirements placed on a full Jini-capable device are a network connection and a Java Virtual Machine that is capable of running the Jini classes and a full RMI system to discover and join Jini communities [12]. As an alternative for devices with insufficient capabilities, but with an own network connection, the Jini specification provides special network proxies [13]. The devices can discover and register with these Jini-capable proxies using a separate protocol. The network proxy acts as a representative for the device, implementing all responsibilities of a well-behaved Jini service.

This is the appropriate way to integrate smart labels into the Jini technology, where a base station takes over the tasks of the network proxy. It acts like a gateway between the radio network with the smart labels and the Jini lookup services.

However, even with delegation, the resource consumption of Jini services inhibits a naive implementation creating one service for each particular smart label. Fortunately, it is sufficient to be able to retrieve a subset, or summary, of all information. For example: How many products of the category “Cornflakes” are currently in stock? Does the amount of butter fall below a lower bound? Which products have reached their expiry date?

Two basic concepts are fundamental to JSalsa in tackling this scalability issues: distributed self-organization and agglomeration. Jini services are only created to represent an agglomeration of a set of items. On the one hand, this "meta service" provides a summarized view on all members of the agglomeration. On the other hand, it is still possible to contact each particular smart label.

2.3 Agglomerations

Managing and maintaining an agglomeration requires some kind of glue. We apply a technique similar to leasing in Jini for this task. However, since smart labels do not really lease services from one another, but confirm periodically their existence as members of groups, we use the term *ticketing*. Groups of smart labels are called *federations* and their hierarchical organization is called *agglomeration*. Federations are limited in size and contain single product types only. Because the number of levels in a single hierarchy is also limited, multiple agglomerations can exist for one product type.

One label takes over the task of managing a single federation, respectively. *Federation managers* at the root of single hierarchy keep a complete agglomeration together and know about all contained items, possibly via multiple stages of indirection. Those federation managers are responsible for creating the Jini service representing the whole agglomeration.

2.4 Distributed self-organization and agglomeration

The requirements for an algorithm managing agglomerations are fault-tolerance, reliability, quick self-stabilization and economic resource usage in terms of memory consumption and bandwidth usage. Moreover, the characteristics of a radio network, such as broadcast abilities, bandwidth, noise-ratio and crosstalk have to be to considered.

The following description of the algorithm starts in a worst case situation: a complete new re-organization without previously saved states. Initially, all smart labels are mutually unknown. It is neither necessary nor intended that each label gets to know each other, as is the case in standard election algorithms [14]. In particular, message flooding has to be avoided. An appropriate solution is that only a small, random number of smart labels initially create and announce empty federations. The announcements, too, are delayed randomly. All other items are simply listening, in a kind of discovery phase, for announcements of federations. Thereafter, smart labels request a ticket from one federation manager, again chosen randomly.

The manager of a federation can accept the request for a ticket, granting it by sending an acknowledgment. If the request is rejected, e.g. when the limit of members of this level has been reached, *no* response is sent back. After a time-out, when no acknowledgment has been received, the requesting label itself creates and announces an empty federation. This leads to a larger number of federations in the start-up phase. However, the following section describes how federations optimize themselves in terms of their size. The advantages of this method are avoiding unnecessary retries and long waits as well as a fast achievement of a stable state. All smart labels are organized in federations within a bounded time. The time-out values depend mostly on the total number of smart

labels. Simulation runs have shown that a few seconds for a discovery time-out is an appropriate value.

After all items are organized in federations, they have to renew their tickets with their manager periodically. While renewing their ticket, multiple error conditions can arise. For example, items may not reach their manager due to communication failures. Furthermore, federation managers possibly may have been removed, e.g. when a customer takes the associated item out of the shelf. Items not receiving an acknowledgment from their manager can proceed in two ways. In case they hold information about other federations, they try to join one of them. Otherwise, abandoned items create and announce new, empty federations.

2.5 Self-optimization

It is very likely that after the initial start-up phase many federations are, compared to others, rather empty. Some of them may be comprised of only a single item, the federation manager itself. These federations should merge with other ones. The strategy of JSalsa is that all smart labels try to move to federations that are larger than their own one. But this simple idea raises some problems: Where do the labels take their information from? What happens if many items break with their federations to join a few larger ones? The latter case can lead to an unwanted "ping-pong" effect. Most labels will be rejected, leading to the creation of new, empty federations, trying the same procedure again and again.

We tackle this problem by utilizing the broadcast property of radio networks. Smart labels renewing their ticket receive the current size of their federation in the acknowledgment. Because the acknowledgment—like any other message sent by the base station—is a broadcast, smart labels are able to snoop communications, maintaining a list of known federations together with their size.

In order to prevent the ping-pong effect mentioned above, federation members switch to larger federations only with a low probability. Completely empty federations are broken up by their managers only if there are any known free federations to join. Additionally, base stations recognize the amount of network traffic and remotely ask all smart labels to adjust their time-out values. This is done by periodically inserting a particular control parameter into messages sent by the base station.

2.6 Implementation Considerations

The design of JSalsa was driven by the characteristics of a radio environment. Additionally, the limited storage capacity and low computational power of smart labels have been considered. JSalsa places only relatively simple requirements on smart labels. They have to provide a memory capacity of a few kilobytes for storing a limited quantity of known federations. Simulation runs have shown that a number of 50 entries for a total number of 10.000 items is sufficient. Furthermore, a random number generator must be included in the label's circuit. The processing capabilities of the smart label has to be sufficient for managing some state variables, adjusting the time-out values and requesting or renewing tickets by sending appropriate messages.

However, not all functionality can be realized inside the smart label. The tasks of managing federations like storing federation members and checking for ticket validity remains to be implemented. In JSalsa, we use a delegation approach. Proxy objects for all smart labels are created in the base station.

3 JSalsa: The Simulation

The feasibility of the concept behind JSalsa has been investigated in a simulation environment. In the design we had to consider the large number of smart labels to be simulated. Moreover, the characteristics of radio networks have been modeled in a separate module, allowing different wireless communication technologies to be investigated. The prototype has been implemented as an event-driven simulation.

3.1 The simulation core

The simulation core includes abstractions for a realistic environment, e.g. `SmartLabels`, `BaseStations` and `Messages`. Lookup services have been omitted because they do not affect the process of self-organization and the Jini technology relies on real time for lease management. Thus, Jini cannot be incorporated directly in an event-driven environment. As an additional simplification, `BaseStations` are modeled using a single antenna only.

Due to the large number of `SmartLabels` it was not possible to implement particular `SmartLabels` in a separate thread. We realized `SmartLabels` as stateful objects controlled by the central `Simulation` loop. A `SmartLabel` becomes active by receiving a `Message` or by callbacks resulting from previously generated future wake-up `Events`. All processing is done sequentially by the single `Simulation` thread.

The characteristics of the `CommunicationSystem` affects the message transmission to a `BaseStation`. Parameters can be supplied to adjust for example assumed bit error rates and the number of communication frames per second. As a first communication technology, we implemented CDMA. In CDMA, mobile items can concurrently send messages in the reverse channel during a communication frame. Real base stations receive a single signal and decode the particular messages. In our implementation, we assume that a `BaseStation` already decoded received `Messages`. The error probability of concurrent messages depends on their number and signal strength. The `BaseStation` determines the error rate for each `Message` based on mathematical formulae for CDMA as described in [6].

For our purposes, it was sufficient to simulate only the main concepts of a CDMA-like communication [15], such as concurrent message transmission and error rates. Because of the limitations of smart labels in terms of processing capabilities, error correction in the reverse channel has been omitted. A `BaseStation` simply drops incorrectly received `Messages`.

3.2 Simulation runs

Simulation parameters are either set in a graphical interface or in configuration files. Adjustable values are several time-outs, the initial ticket duration, the communication

frames per second, sizes and levels of federations, the number of `SmartLabels`, the probability to create empty federations and the duration of a simulation run. The simulated environment resembles a part of a warehouse. A couple of shelves are covered by two `BaseStations`. The shelves contain an adjustable amount of virtual products, divided into several product groups. Each product is tagged with a `SmartLabel`.

At start-up time, all `SmartLabels` are completely unorganized. During a simulation run the labels first agglomerate, subsequently optimizing the federations. Concurrently, a random amount of `SmartLabels` is removed periodically, emulating the dynamics of a warehouse. The simulation logs several values, e.g. the total number of federations, agglomerations, unorganized labels and messages statistics.

First simulations have shown that the algorithm used in JSalsa in fact is self-stabilizing. The number of unorganized labels converges against zero. We found out that a number of about 10 concurrent messages, without error correction, forms an upper bound for the stability of the algorithm.

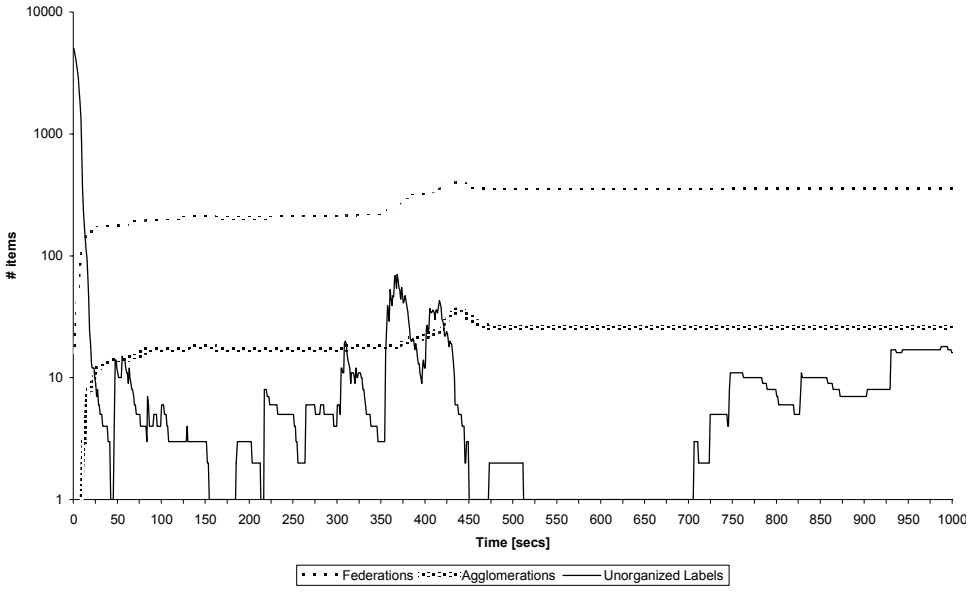


Fig. 2. Simulation run with 5000 items showing the number of federations, agglomerations and items currently unorganized.

A sample simulation run is shown in figure 2. We start in a worst-case scenario with 5000 items belonging to 8 different product types being completely unorganized. After 25 seconds, only 10 items remain unorganized. The other 4990 items have formed 165 federations, hierarchically organized into 11 agglomerations. After 45 seconds, the complete set of items has been organized into 174 federations and 14 agglomerations. The number of unorganized items again raises above zero for one of the following reasons. Either the ticket renewal message was lost or the federation manager has been removed. For example, figure 2 clearly demonstrates the removal of a top-level federation manager by the simulation after 400 seconds. The reorganization first results in an in-

creased number of federations and agglomerations. However, the numbers will lower again after a period of time, depending on a parameter specifying the probability of triggering the creation of new federations. In the example shown, the value of this parameter was chosen very low.

Further simulation runs were performed, varying e.g. the total number of items, the maximum size of federations, the number of levels in a single agglomeration, and the duration of the discovery phases. Those simulations clearly showed the effect those parameters have on the algorithm's effectiveness. For example, if the number of items is very high and the discovery phase is too short, the number of bit errors will rise significantly, resulting in longer stabilization times. However, if parameter values are chosen in a reasonable fashion, self-stabilization remained a stable property of our algorithm.

4 Related Work

One prominent example of an RF/ID system is TIRIS [16]. TIRIS is being used in a broad variety of applications, including automotive anti-theft systems, logistics management, and automated payment programs at gas stations. Over 30 million TIRIS tags are currently in use worldwide. The newest family of TIRIS products is Tag-it, a smart label technology that offers cost-effective solutions for high volume baggage tracking, parcel delivery, electronic ticketing and anti-counterfeiting applications. After all, TIRIS clearly shows that smart labels are an effective approach for developing sophisticated, individualized services.

Smart warehousing, as considered in JSalsa, has already has become reality nowadays. SmartShopper [17] is one current example. Customers are equipped with a device including a scanner and a display. Products are scanned before putting them into the shopping cart. The device then contacts a central server in the market, retrieving and displaying information about the product. Examples are hints on related products, special offers or even recipes. Additionally, the returned data contains the current price used for charging purposes. Customers know at every time the total sum of all products inside their trolley. However, SmartShopper is a centralized approach based on traditional barcodes. Products remain passive entities, being incapable of setting up agglomerations by themselves.

A similar scenario employing smart labels is investigated by Fraunhofer IZM [1]. However, their research focuses more on the hardware aspects of smart label technology, while our work can be interpreted to be complementary at the software layer.

Hive is a project aimed at providing an application infrastructure for Things That Think [18]. Hive introduces the metaphor of an ecology of distributed agents. Applications are created out of the interaction of multiple agents across a network of so-called cells. Local resources are made available to agents via so-called shadows, encapsulating e.g. physical devices. The combination of Hive's agents and shadows is roughly analogous to Jini services. Remarkably, the implementation of Hive is reported to suffer from "several implementation limitations [of RMI] that make it difficult to scale to systems with many concurrent agents" [18]. This observation is in parallel with our own experiences, thereby underpinning the usefulness of agglomerations in terms of scalability.

Two different technologies, MCoM [19] and Salutation [20], can be considered as parallelizing the role of Jini as applied in JSalsa. In the context of the MCoM research project, small devices are equipped with wireless network adapters. MCoM permits automatic device registration and lookup [21]. The Salutation Consortium promotes an open architecture, enabling interoperability between networked devices. Salutation-compliant devices joining a network register and provide information about themselves via a standardized API. However, like in Jini, service agglomeration is no integral part of those technologies.

5 Summary and Conclusions

In this paper, we presented a scenario of a wireless, smart warehouse storing products tagged with smart labels. We employ the Jini technology to represent those networked items as services, spontaneously forming dynamic communities. JSalsa is our approach at managing several thousand entities using wireless communication technology.

The most basic concepts behind JSalsa are distributed self-organization and agglomeration. In its design we explicitly considered the limited capabilities of smart labels and the characteristics of wireless networks. Among those characteristics are limited bandwidth, high bit error rates, and the phenomenon of crosstalk. Messages from the base station are sent as broadcasts, allowing the implementation of JSalsa to benefit from. In the reverse direction, explicit care is taken to avoid message flooding.

We implemented a simulation investigating the effectiveness of distributed self-organization in a wireless environment. The simulation models a section of a warehouse made up of some shelves, two base stations and a large number of smart labels. As wireless networking technology we employed CDMA, implementing relevant parts of its mathematical model.

Simulation runs showed that the distributed algorithm used in JSalsa is self-stabilizing. Starting in a worst case scenario with a set of completely unorganized items, a hierarchy of groups can be setup in very short time. For example, in one simulation run with well-chosen communication parameters, 5000 items of 8 different types have been organized in about 180 groups within approximately 40 seconds. Those groups were combined hierarchically into about 25 agglomerations. Therefore, 5000 items can be represented through only 25 Jini services, resulting in a savings factor of 200. Reorganizations due to removed items or influences of the wireless network are handled in similar short time.

References

1. Reichl, H., Pelka, J., Lang, K.-D., Wolf, J., Hahn, R.: *Visionäre Produkte für den mobilen Menschen*. http://www.izm.fhg.de/izm_public/paper/vision.pdf (1999)
2. Miller, Franz: *Das elektronische Etikett*. Fraunhofer Magazin 2/1999, http://www.fhg.de/german/publications/df/df1999/299-06.htm (1999)

3. Fraunhofer-Gesellschaft: *Research News 10/1999 Topic 4: The electronic label*. http://www.fhg.de/english/press/md-e/md1999/index.htm
4. Yang, Samuel: *CDMA RF system engineering*. Artech House (1998)
5. Stüber, Gordon: *Principles of Mobile Communication*. Kluwer Academic Publishers (1996)
6. Rappaport, Theodore: *Wireless Communications, Principles & Practice*. Prentice Hall (1996)
7. Sun Microsystems, Inc.: *Jini Connection Technology*. http://www.sun.com/jini (2000)
8. Edwards, Keith W.: *Core Jini*. Prentice Hall (1999)
9. IBM Corp.: *SanFrancisco Component Framework.* http://www.ibm.com/software/ad/sanfrancisco (2000)
10. Christ, R. Halter, S. L., Lynne, K., Meizer, S., Munroe, S. J., Pasch, M.: *SanFrancisco performance: A case study in performance of large-scale Java applications.* In: IBM System Journal, Vol. 39, No. 1, http://www.research.ibm.com/journal/sj/391/christ.html (2000)
11. Sun Microsystems, Inc.: *Jini Architecture Specification*. http://www.sun.com/jini/specs (2000)
12. Waldo, Jim et al.: *The Jini Specification*. Addison-Wesley (1999)
13. Sun Microsystems, Inc.: *Jini Device Architecture Specification*. http://www.sun.com/jini/specs (2000)
14. Lynch, Nancy A.: *Distributed algorithms*. Kaufmann (1996)
15. Lenz, Alfons: *Simulation eines CDMA-Systems unter Berücksichtigung verschiedener Daten- und Sprach-Modelle*. University of Trier (1999)
16. Texas Instruments, Inc.: *TIRIS.* http://www.ti.com/mc/docs/tiris/docs (1999)
17. Siemens AG: *Der SmartShopper—Einkaufen mit elektronischem Berater und ohne Warten an den Kassen*. In: NewsDesk 9930, http://www.siemens.de/de/press_service/newsdesk/c_9930_d.html (1999)
18. Minar, N., Gray, M., Roup, O., Krikorian, R., Maes, P.: *Hive: Distributed Agents for Networking Things*. Proceedings of ASA/MA'99 (1999)
19. Microsoft Corp.: *Mobile Communications Research.* http://research.microsoft.com/~bahl/MS_Projects/projects.html (1999)
20. The Salutation Consortium: *Find-and-Bind™ for Pervasive Computing.* http://www.salutation.org (2000)
21. Callahan, J.: *Moving Towards Effortless Networking.* In: IEEE Computer, Vol. 31, No. 11, November 1998, pp. 12-14 (1998)