Mobile Work Clearance Management—Challenges and Solutions

Werner Kurschl, Stefan Mitsch, Rene Prokop
Research and Development Competence Center Hagenberg
Upper Austrian University of Applied Sciences
Hauptstraße 117
A-4232 Hagenberg
{werner.kurschl, stefan.mitsch, rene.prokop}@fh-hagenberg.at

Abstract: We present a system called MOSES, which is a mobile work clearance management system for complex and large-area industrial machines. Based on an extensive field study of current work clearance practices, we identified important challenges and developed several solutions. MOSES shows how an enterprise application, like an enterprise resource planning system, can be extended to improve occupational safety for maintenance personnel. The presented solution consists of a work clearance management extension for several existing enterprise resource planning systems and a mobile work clearance management for mobile devices. The paper also describes a framework called Smart Data Off The Spot (SmartDOTS) for building nomadic distributed enterprise applications that supports data replication and synchronization from various enterprise applications. The framework allows the modelling of business data, so that device-, network and task-specific data replication, and context-dependent network selection can be modelled and automatically be executed by the SmartDOTS framework. This paper shows the design and implementation of a prototypical mobile work clearance system, which uses the SmartDOTS approach.

1 Introduction

At present, maintenance of large and complex industrial machines with an extension of more than 300 meters (e.g., a rolling mill in a steel plant) is sometimes a dangerous job. A lot of danger stems from toxic gas and acid, high voltage, or hot and fast moving parts (and this list is by far not exhaustive). In many cases, maintenance personnel have to enter the risky environment to perform the planned tasks. Therefore, on behalf of everyone involved in that business the ambition is to establish and further improve safety standards.
Companies that have to deal with such risky environment are usually enforced by law to establish some form of work clearance process. Content of this work clearance process is the definition of stepwise instructions how to halt a machine or a part of it to ensure a secure working area within the dangerous machine.
Since maintenance work interferes with the production process, it is desirable to perform maintenance tasks simultaneously; thus, the maintenance work is usually planned in advance by a maintenance coordinator. The result of this planning step is a comprehensive document—called work clearance list—that describes the complete shut down process.

Setting up a work clearance list requires a sound understanding of the machine; the number of switches, gas and oil gates, and other so called work clearance elements installed in a machine typically exceeds 10,000. In a common work clearance several hundreds of those work clearance elements need to be cleared (i.e., turned off). The creation of work clearance lists and its execution is tedious and error-prone, as both the work clearance list and the machine may contain many similar work clearance elements that are hard to differentiate. Clearing the wrong work clearance element, or mixing up the order of work clearance elements, may cause immediate damage to man and machine, whereas human damage usually affects the maintenance personnel at a later point in their maintenance work.
Since working accidents are often life threatening, cause production losses and legal consequences against the responsible maintenance coordinator, we have developed a mobile solution called MOSES that improves the occupational safety of the maintenance personnel. MOSES consists of a server extending existing enterprise applications, clients to plan and administrate work clearances, and mobile clients to execute work clearances. MOSES incorporates a novel middleware called Smart Data Off The Spot (SmartDOTS) tailored to the needs of mobile solutions.
Current middleware for distributed systems typically reach for complete transparency (see [CDK01]); their main goal is to hide heterogeneity (e.g., the location and access to networked resources) to make remote resources appear as locally available. This approach is feasible in environments with constant network connectivity and quality of service. Mobile devices, though, face extremely varying network environments with typically sporadic connectivity, low bandwidth, or high costs.
SmartDOTS follows a semi-transparent approach that does not hide all connection issues from the application. Hence, MOSES can adapt to different connection conditions, ask the user for decisions in ambiguous situations, or let the middleware automatically handle connection issues.
The rest of the paper is organized as follows. In the next section we describe current work clearance management practices. The challenges derived from these are presented in Sect. 3. In Sect. 4 we describe related work. Section 5 discusses possible solutions for the presented challenges. Section 6 concludes the paper.

2 Work Clearance Management

In this section, we describe a scenario motivated by our field studies to illustrate work clearance management. We show how the process is handled without MOSES and which problem areas exist.

2.1 A Process Scenario

A common work clearance process, without support from MOSES involves the following steps. In the first step the maintenance coordinator plans the necessary work clearance tasks based on the enterprise resource planning system's information (the ERP system defines through maintenance orders which parts of the machine should be repaired).
The maintenance coordinator then consults the machine documentation to find the work

clearance elements that have to be cleared for the particular maintenance work. These work clearance elements are added with additional information - like the work clearance element's location - to the work clearance list. Based on this list the machine is then cleared (i.e., shut down). The persons involved in clearing the machine sign the work clearance list to confirm the correct clearance. The machine is now secure for the maintenance personnel to start working. When the maintenance personnel finished their work at the machine, the maintenance coordinator can set all work clearance elements back to their initial state, so that the machine is ready to be restarted. This step is called normalizing the machine. The correct normalization of the machine is also confirmed by signing the work clearance list. The work clearance is finished by the machine commander, who restarts the machine.

2.2 Problem Areas

The process described above contains several potential problem areas. Enterprise resource planning systems provide information about maintenance orders. They do not contain information how to clear or repair a machine; this information is usually contained in the machine's documentation. The maintenance coordinator, who plans the work clearance list, consults this documentation to find the necessary work clearance elements. We observed that faults in the planning stage are very unlikely to be discovered.
Switching cabinets usually contain a large number of similar work clearance elements. Though they are identifiable with a unique code, it is often hard to distinguish them and to find the element listed on the work clearance list. The worker could also mistakenly omit to clear or normalize single elements, or confuse the sequence in which elements need to be worked on. When such faults occur during work clearance they might lead to machine damage, or worse, to maintenance personnel being hurt; during normalization they usually manifest in production losses—e.g., the machine does not produce the needed quality, or it does not perform at the desired speed.

3 Challenges with Mobile Work Clearance Management

Mobile work clearance management, as described in Sect. 2, is a collaborative working process. The maintenance personnel stay in contact with a control centre—which administrates the work clearance—to exchange information on the working progress. The challenges for a software solution facing the described environment are briefly discussed in this section.

3.1 Mobile Data

While executing a work clearance, the maintenance personnel needs information on the machine being worked on, the tasks to be executed, and the overall working progress. A mobile work clearance management solution cannot assume a constantly available network connection, as network connections in mobile environments—and especially in industrial environments—are subject to unpredictable interruptions. The maintenance personnel must be able to work under these circumstances (so-called disconnected operation) without being hindered. Thus, a mobile work clearance management solution needs to

support data replication to buffer essential data on the mobile device, and to synchronize changes between local data and the control centre.

3.2 Energy Consumption

An extensive work clearance often lasts up to one shift (i.e., eight hours) or beyond. During this time period, maintenance personnel are working in the machine using their mobile devices. Thus, a mobile device's battery can only be charged after the end of a shift. Extensive CPU load, memory consumption, or network access imposes an additional energy drain on the batteries (see [Sta03]). The measures for mobile data described above also help to limit the energy consumption needed for transmitting data.

3.3 Connectivity

The collaborative nature of a work clearance process demands for data transmission between the maintenance personnel's mobile devices and the control centre. The communication networks available in industrial environments are often of different types; especially high-bandwidth communication networks, like WLAN, might not be available everywhere. Thus, a mobile work clearance management solution needs to support different communication protocols (e.g., WLAN, GSM, GPRS, or UMTS) and also adapt to the restrictions a particular network involves.

3.4 Notification of Working Progress and Data Changes

For members of the maintenance personnel it is always interesting to have the latest information from the control centre. Data changes in the control centre need to be distributed to the mobile devices. We distinguish data changes by their importance; while some data changes are non-relevant during work clearance (e.g., the name of a work clearance element might change), others are vital for the maintenance personnel (e.g., working progress describing a normalization work). The importance of a data change describes whether it is necessary to immediately disseminate the change to the maintenance personnel. Depending on a particular mobile device's current connection conditions, a communication network needs to be selected. Non-relevant information can be cached until the mobile device has access to a communication network free of charge, while vital information needs to be transmitted regardless of costs. For vital information, it might even be necessary to switch to a backup communication network, when the primary one is not available.

4 Related Work

Siren (see [JCH+04]) is a peer-to-peer network for sharing knowledge between fire-fighters and supports tacit communication between them. Siren focuses on gathering, integrating and distributing contextual data. Its peer-structure allows Siren to share knowledge between peers. Single peers, however, might work on outdated data, as Siren does not provide any infrastructure besides peer-to-peer communication. In the environment Siren is targeted at, this behaviour is valuable. But in our environment it is essential to provide an

exhaustive up-to-date view on the machine state and working progress to all clients in the system.
Efstratiou et al. (see [ECDF01]) mention some architectural requirements for adaptive mobile applications in their work, but they do not provide a detailed design of their ideas and they do not show how the ideas can be transformed in an effective implementation.
XMIDDLE is a mobile middleware for transparent sharing of XML documents in a peer-to-peer network (see [MCE01]). XMIDDLE supports sharing of tree-structured data between peers; therefore, each peer offers access points for other peers to manipulate its data online or to replicate the data for working offline.
SodaSync, described in [CGK+04], is a programming framework that provides a generic synchronization model for mobile enterprise applications. It can integrate multiple heterogeneous backend data stores through a unified high-level data model. SodaSync focuses on the interface between an idealized generic data model and real data sources. Data is modeled using a Service Data Object (SDO)-based representation of application data; this approach limits SodaSync to Java applications based on the SDO framework. Furthermore, it leaves important aspects for mobile applications like device-, network and task-specific data replication, and context-dependent network selection unsolved.
The SyncML standard (see [HMPT02]) is a specification for an interoperable data synchronization framework using an XML-based format. SyncML can be used as an underlying protocol for data exchange in SmartDOTS.
In contrast to the described systems that primarily synchronize single data stores, SmartDOTS focuses on synchronization of data for complete applications that integrate multiple data stores. Additionally, SmartDOTS allows application designers to specify data packages for synchronization. These data packages can be tailored to fit tasks, devices, and networks; they usually describe a set of data needed to perform a specific task on a specific device and are designed for "take-away". SmartDOTS focuses on providing synchronization capabilities over real world (often synchronous) communication networks and models, and over multiple data stores. SmartDOTS aims to provide this functionality as a service to application developers.

5 Solutions

The following section describes suitable solutions for the identified challenges in detail. The architecture of MOSES follows the client-server model described in [CDK01]. It consists of a server integrating existing ERP systems, a mobile device client, a planning and administration client, and the SmartDOTS middleware.

5.1 Mobile Data

We observed that mobile industrial workers typically need a known set of data to perform a specific task. The devices used in an industrial environment often are of diverse nature; notebooks, cell phones, and PDAs are used interchangeably. Mobile workers often start their work at a location with constant network connection (e.g., the office), but in the course of their task they stay in areas with only sporadic connectivity. Thus, data can be packaged to fit the task at hand and loaded to the device while a network connection is

available. Then this data can be worked on even without a network connection. The tasks of mobile industrial workers are often collaborative. Hence, synchronization of data is performed whenever a network connection is available.

The SmartDOTS Architecture

This section describes the design of SmartDOTS. SmartDOTS, depicted in Fig. 1, consists of a server-side SmartDOTS-Service and a client-side SmartDOTS-Engine.

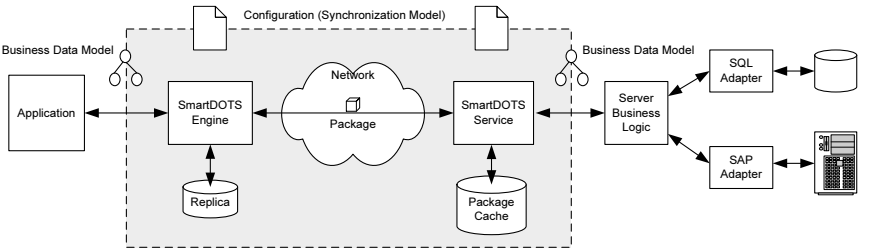

Figure 1: The SmartDOTS architecture

The SmartDOTS-Service integrates the enterprise application's server business logic. The server business logic typically accesses data in different data sources (often legacy systems) and combines it in a single object-oriented business data model. The SmartDOTS-Service continuously builds packages containing data from this business model and stores them for immediate retrieval in a package cache. Thus, sensitivity to the legacy systems' latency is eliminated. The content of the packages can be configured on a device, network, cost, etc. basis.
The SmartDOTS-Engine is the primary data storage for an application. The application can query the SmartDOTS-Engine for data; SmartDOTS provides these data in the application's business model, similar to directly accessing the server business logic. The SmartDOTS-Engine retrieves data in packages from the SmartDOTS-Service and stores it in a local replica. The communication between the mobile client application and the application's server business logic is completely asynchronous (even over a synchronous connection). Yet, through the local replica the SmartDOTS-Engine is able to offer a synchronous programming model to its clients, which is more familiar and convenient to most programmers than asynchronous or message oriented programming. The SmartDOTS-Engine handles data replication, tracks changes and updates to local data, and synchronizes the local replica with the enterprise applications.
Communication between the SmartDOTS-Engine and the SmartDOTS-Service is not limited to a particular data representation or communication channel, though, for performance reasons, binary protocols are preferred.

The SmartDOTS Engine

The SmartDOTS-Engine offers features to persist and synchronize a business model with the SmartDOTS-Service. Therefore it has to track accesses to the SmartDOTS-Service, handle data synchronization and perform automatic service-relocation when the service is not reachable.
To enable an unhindered working process, it is necessary to decouple the availability of data from the network conditions. This requires a local data management on the mobile device, which fetches data from the server, holds it locally, and keeps it up to date on both sides.
Because of the different requirements of the mobile devices SmartDOTS' data management offers an abstraction of storage media - named DataStore - and is able to work with several DataStores concurrently, independent of their location. SmartDOTS uses a configuration to arrange these DataStores. A mobile device, which is prepared for working disconnected, typically uses (i) a local DataStore that holds the data in a local replica, (ii) a remote DataStore that is connected to the SmartDOTS-Service, and (iii) a logging DataStore that tracks the changes on the local data. The usage of these DataStores depends on the current network state and can be influenced by the automatic service-relocation: as long as the mobile device is online, the changes are done on the local data (local DataStore) and submitted immediately to the server (remote DataStore). Therefore the changes are performed on the local DataStore and the remote DataStore. When the mobile device becomes disconnected, the remote DataStore is not reachable any more and therefore all changes are performed on the local DataStore and on the logging DataStore. This logging DataStore tracks the changes and builds up a description of the delta between the local data and the data on the server.
Synchronization is performed upon service-relocation from offline access of the local replica to online access of the server. By replaying the delta, which was tracked by the logging DataStore, a synchronous state can be re-established on the remote DataStore.
The management of replicated and distributed databases requires algorithms for guaranteeing data consistency and for resolving conflicting updates. Several architectures, such as Bayou [DPS+94] and SodaSync [CGK+04], have been proposed to address these important problems. SmartDOTS was in the first stage designed for an environment where conflicting updates cannot occur. We consider resolving conflicting updates as part of our future work.

Automatic Service-Relocation

When no network connection is available, the SmartDOTS-Engine performs an automatic service-relocation to provide data either using a different connection (e.g., GPRS instead of WLAN) or from the local replica. We use a combination of the Bridge and State pattern, both described in [GHJV95], to enable the service-relocation (see Fig. 2).

The Bridge hides the concrete State (online or offline) currently in use from its clients. Incoming requests are forwarded to the current State, which in turn forwards it to its DataStore (e.g., the SmartDOTS-Service or the local replica). When the DataStore is not able to process the request (e.g., because the network connection fails), the State switches offline

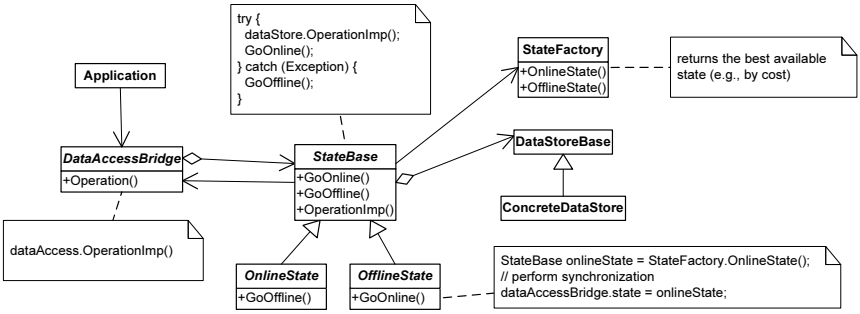

Figure 2: Combined Bridge and State pattern

(for details see Sect. 5.2). The StateFactory provides the State to switch to (e.g., depending on the costs involved with using the State). The Bridge is then modified to access the OfflineState in subsequent requests; the current request is satisfied by the OfflineState as well.

5.2 Energy Consumption

SmartDOTS can be adjusted between information timeliness—which demands more frequent network access and more energy—and resource consumption, as shown in Fig. 3.

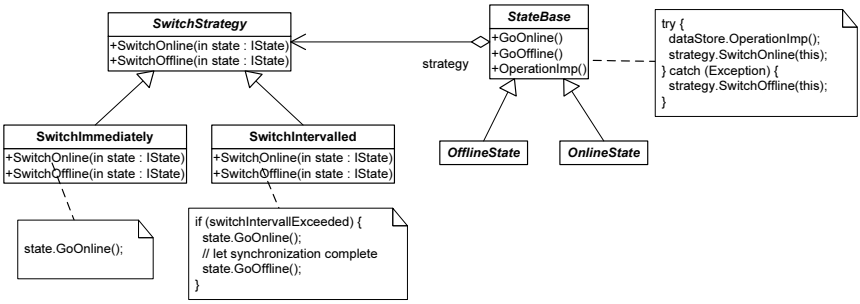

Figure 3: Strategies for switching between states

The Strategy pattern (see [GHJV95]) provides configurable alternatives for switching between states. In shorter work clearances, or when the work clearance demands close collaboration between members of the maintenance personnel, more frequent updates are needed or possible. In longer work clearances, the mobile middleware is typically configured to provide data mostly from the local data store. Only in critical situations, or in longer time intervals, data is synchronized with the control centre.

5.3 Connectivity

A wireless connection manager observes the network adapters available on a device. Depending on the available connections, the wireless connection manager automatically configures the middleware's StateFactory and SwitchStrategy. For example, when the device has access to a WLAN connection, the StateFactory is configured to provide a WLAN OnlineState that always accesses remote data. As SwitchStrategy, switching immediately might be used. In contrast, when only a GSM connection is available, the GSM OnlineState could access remote data only in urgent cases, and the SwitchStrategy synchronizes critical data in intervals.

5.4 Notification of Working Progress and Data Changes

A distributed notification mechanism demands for the server actively transmitting events. But especially service-oriented servers act passively. Therefore, the notification mechanism is outsourced from the server and based on .NET Remoting. The core of the notification mechanism is the ClientManager (see Fig. 4) that holds information about each registered client and its communication options, receives new events from the MOSES server and is responsible for distributing these events to each client.

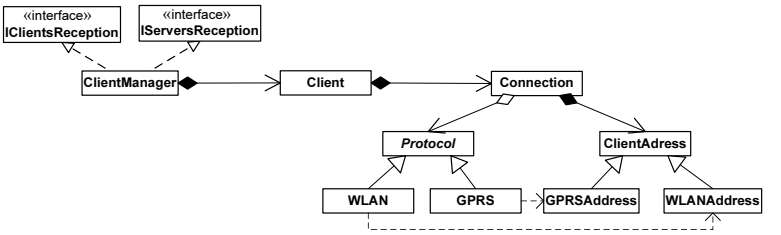

Figure 4: Clients and their supported connections

The functionality of the ClientManager is split into two interfaces: the interface IClientsReception for communication with the clients and the interface IServersReception for communication with the MOSES server.

Registration and Management of Clients

Each client that wants to be notified about changes of the central data, must register itself through the interface IClientsReception at the ClientManager. By that, the ClientManager receives a remote reference to the client. As discussed in Sect. 5.3, different communication protocols can be in use concurrently and at a later point of time the client might be accessible via a completely different protocol. Thus, the address information can not be implicitly determined from the connection during the registration process. Instead, the client itself has to provide explicit information about which types of protocols are supported and how it can be addressed over these protocols. Fig. 4 shows the ClientManager holding a number of clients. Each Client has a set of supported connections (e.g., WLAN

or GPRS) including address information about how the client can be reached using these protocols (e.g., IP-address or phone number).

Event Management

The MOSES server can send events to the clients over the interface IServersReception. These events are not transmitted immediately, but managed in an EventQueue until it is their turn to be sent. Fig. 5 shows the relationship between the clients and their QueuedEvent objects.

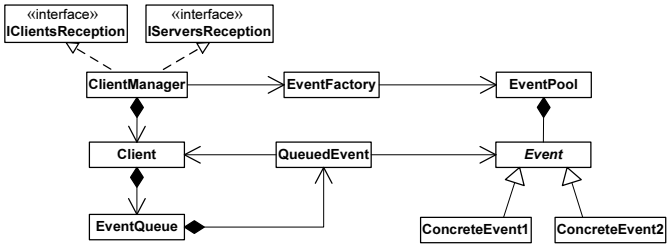

Figure 5: Buffering and queuing of events

To keep the number of instantiated Event objects small, the server delivers only the event type. The ClientManager uses an EventFactory to acquire a matching Event object. The EventFactory holds exactly one Event instance per type in an EventPool. By that the number of Event objects is constant.
The event type implicitly defines the priority, with which the event will be queued, and the maximum cost its transmission is allowed to cause. Based on these values, send order and the transmission protocols will be chosen later.
The ClientManager adds the new event to all registered clients; the clients evaluate the event individually and add it to their EventQueue. The evaluation checks if the event has influence on already queued events. For example, it may be useless to send an event ”list changed” twice, because the client has to update the whole list anyway. So each event type has to know its influence on other event types.

Transmission of Events

The EventDispatcher performs a continuous process, which sorts queued events of all clients according to their priority and queue time and invokes the transmission mechanism. Fig. 6 shows the involved components.

The transmission of single events is performed by the TransmissionStrategy, which depends on the event type and is created using the TransmissionStrategyFactory (factory pattern, see [GHJV95]). The concrete strategy is determined just in time because most of the events will expire or will be suspended by subsequent events before reaching this point of process. The TransmissionStrategyFactory works analogous to the EventFactory with its EventPool described before.

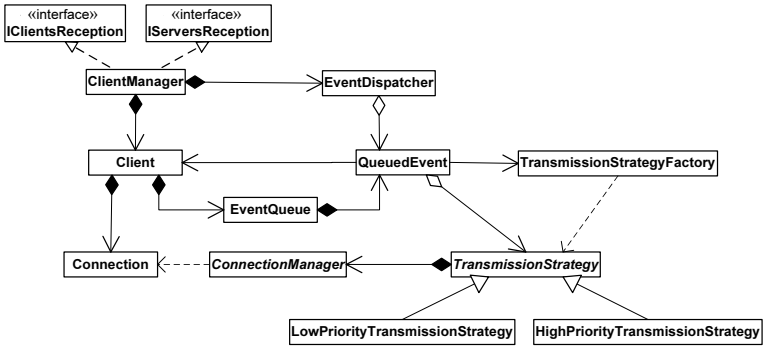

Figure 6: Transmission of events

The concrete TransmissionStrategy is responsible for sending the event to the client in the cheapest and fastest possible way. A possible implementation of the strategy could be to start trying to send over the ”cheapest” connection and continue with the next ”expensive” connection if the cheaper one is not available. The strategy pattern (see [GHJV95]) is used here, because depending on the event type a different approach of finding the best connection can be used. The TransmissionStrategy uses the ConnectionManager to identify a concrete Connection, which is supported by the Client and matches a connection category (e.g., cheap connection or high-bandwidth connection). The concept behind this step of abstraction is to keep the TransmissionStrategy simple while offering a point to introduce performance enhancements. Enhancements could be achieved by monitoring the client's communication behaviour (i.e., which protocol was used lately) and use this information for a better choice of the protocol.
If the transmission succeeds, the event is deleted and an internal ”keep alive” message is sent to the ClientManager to prevent the expiration of the client. If the transmission fails, it is assumed that the client is offline. Therefore, the event is left in the QueuedEvents collection and the queue time of the event is modified to ensure, that the event is realigned correctly by the EventDispatcher and sent at a later point of time.

6 Conclusion

The proposed solution makes a work clearance process both more secure and more agile, but it also preserves the collaborative nature of the process. Such a process demands for information flow in both directions—the control centre needs information on the actions in the field, and the mobile workers need the latest data. We concentrated on identifying the important challenges and providing a software solution that does not overwhelm the maintenance personnel with technical gadgets; instead it provides reasonable assistance in currently error-prone situations.
Mobile applications usually face bad network connections and energy constraints, where an always on-line situation simply is not suitable. Thus, we have optimized wireless data exchange with a middleware called SmartDOTS that incorporates data replication,

synchronization, event and caching mechanism. SmartDOTS supports different data exchange protocols (e.g., SOAP) and it can be extended to support several different wireless networks like WLAN IEEE802.11(b/g) and GPRS.
We developed a prototype called MOSES, which shows the feasibility of our design. Future development will comprise alternative transport mechanisms, as well as the optimization and evaluation of the data replication and synchronization capabilities. In a pilot installation we have seen, that our approach provides non-intrusive support for maintenance personnel carrying out their day-to-day work at a rolling mill. Besides the direct benefits—i.e., the assistance and security enhancements—additional positive side-effects (e.g., comprehensive documentation of a mission critical process which so far based on experience values of single maintenance coordinators) arose while testing the prototype.

References

[CDK01] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems—Concepts and Design*. Addison-Wesley, Boston, 3 edition, 2001.

[CGK+04] P. Castro, F. Giraud, R. Konuru, A. Purakayastha, and D. Yeh. A Programming Framework for Mobilizing Enterprise Applications. In *Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'04)*, pages 196–205, Washington DC, USA, 2004. IEEE Computer Society.

[DPS+94] A. J. Demers, K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and B. B. Welch. The Bayou Architecture: Support for Data Sharing Among Mobile Users. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, USA, 1994. IEEE.

[ECDF01] C. Efstratiou, K. Cheverst, N. Davies, and A. Friday. Architectural Requirements For The Effective Support Of Adaptive Mobile Applications. In *Proceedings of the Second International Conference in Mobile Data Management*, pages 15–26, Hong Kong, 2001.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, 1995.

[HMPT02] U. Hansmann, R. Mettala, A. Purakayastha, and P. Thompson. *SyncML: Synchronizing and Managing Your Mobile Data*. Prentice Hall, 2002.

[JCH+04] X. Jiang, N. Y. Chen, J. I. Hong, K. Wang, L. Takayama, and J. A. Landay. Siren: Context-Aware Computing for Firefighting. In *Proceedings of the Second International Conference on Pervasive Computing*, pages 87–105, Linz/Vienna, Austria, 2004.

[MCE01] C. Mascolo, L. Capra, and W. Emmerich. An XML Based Middleware for Peer-to-Peer Computing. In *Proceedings of the International Conference on Peer-to-Peer Computing*, pages 69–74. IEEE Computer Society, 2001.

[Sta03] T. Starner. Powerful Change Part 1: Batteries and Possible Alternatives for the Mobile Market. *IEEE Pervasive Computing*, 2(4):86–88, 2003.