Distributed Architecture for Mobile Contextual Integrated Field Work Applications

Roberto S. Silva Filho, Anuj Tewari
GE Global Research - Collaboration & Mobility Lab
2623 Camino Ramon
San Ramon, CA, USA 94583
{silva_filho, Anuj.Tewari}@ge.com

Abstract— The current generation of corporate software tools & applications were not designed to support the unique needs of industrial field service work. Business software applications such as project management and time keeping, for example, are typically designed for traditional desktop computing office environments. As such, they assume low user mobility, high network availability and WIMP (Windows, Icons, Menus, and Pointer) user interfaces. These are also agnostic to physical environment context and are loosely integrated with one another, often requiring users to maintain duplicated information records. As a result, Field Services Personnel as Engineers, Superintendents and Craftsmen end up spending significant amount of their work time dealing with the consequences of these inefficiencies. In this paper, we describe a distributed architecture for mobile, contextual and integrated fieldwork software applications (or MCI) designed for mobile and wearable computing platforms. This software architecture defines a contextual and mobility-aware client side API, a flexible integration middleware, and instrumented backend services. We show how MCI can enable the construction of portable, mobile, context-aware and integrated software applications discussing its use in the implementation of SmartOutage, a mobile app used for automating common tasks in Field Engineering work.

Keywords: Distributed Software Architecture, Mobile & Contextual Computing, Application Integration, Field Engineering Automation.

I. INTRODUCTION

Field Engineers (or FEs) are personnel responsible for leading the installation and maintenance of a broad range of industrial equipment such as oil rigs, power generation turbines, drilling equipment, or airplane engines. As such, FE's work has specific requirements that differentiate them from traditional office work. FE's work is largely physical, often performed in inhospitable industrial settings. FEs work is also highly manual and mobile in nature, which requires them to carry extra pieces of equipment such as toolboxes, spare parts, manuals and schematics to the equipment assets being serviced. Finally, FEs activity is highly collaborative. FEs are responsible for working with superintendents and craftsmen (or millwrights) in their repair and maintenance activities, and for interacting with company and site managers, gathering requirements and ensuring the proper execution of planned outages, providing periodic status updates and reports. Different software applications are an integral part of FEs' work routine. These include office suites, for writing reports and producing spreadsheets, Web-based search portals, for obtaining assets (or equipment) schematics and documents, time keeping and task management applications, used to track their work progress, as well as corporate communication tools, including instant messaging, calendar and e-mail. In spite of their importance for the FEs daily activities, these software applications are typically not optimized for the type of environment and the nature of FEs work: they are generally loosely integrated with one another, and are designed for desktop environment, being generally agnostic to the task and physical context. Finally, these traditional applications are not originally designed for mobility: due to environmental conditions as noise, poor network connectivity and lack of power sources, field automation computational systems are usually restricted to specific areas such as field offices and trailers. These offices are typically setup far from the FEs main work place. As a result, this creates the need for multiple trips between workplace and field office during a typical workday, whenever FEs need to search for documents, obtain schematics, communicate with managers and experts, produce and print reports, etc. The time spent in the field office, and not on deck, supervising and instructing craftsmen, quickly adds up, leading to potential delays and increased equipment downtime costs.

Recent developments on mobile and wearable computing technology have enabled novel applications that leverage the potential of the Internet of Things (IoT) [1] and the Industrial Internet [2]. Devices such as mobile smart phones, tablets, wearable computers, and sensors; together with new networking standards including ad-hoc and near field communication protocols [3] have been combined in support of mobile industrial applications.

In this paper, we discuss the MCI software architecture designed to support the development of mobile contextual and integrated software applications for field engineering. In particular, we have applied MCI in the design and implementation of the SmartOutage application, which goal is to better integrate existing corporate tools, already part of FEs daily work, into a single context-aware mobile experience.

In the SmartOutage project, the MCI architecture was utilized to support disconnected operation of client mobile applications and to integrate data from different corporate applications. It also utilizes logical (task-based) and physical (sensor-based) contextual information to pre-fetch data from corporate servers and to provide the right information to FEs during their typical workday. The software architecture is based on the concept of cooperating client and server-side software agents that work together to pre-fetch and synchronize data between client and corporate services.

The benefits of this approach are: a simplified API for client application development with automatic handling of data disconnection & synchronization, improved integration of corporate information systems, and ability to leverage on physical and logical context to provide application that deliver the right information to the right user at the right time.

II. Background

Related work, in both industry and academia, can be used in support of field engineering requirements of mobility, disconnected operation, application integration, and contextual computing. This section discusses some of those approaches with their actual and potential use in field engineering.

A. *Mobility and connectivity in field engineering*

The lack of mobility of traditional computational systems, and their need for connectivity have been generally addressed in field engineering by the use of rugged laptop computers, and the setup of field offices.

The use of rugged laptops with WAN/LAN network adapters allows FEs to interact with corporate systems using cellular or WiFi networks. Those laptops, however, are relatively heavy and large to carry along, and the desktop class software applications they run have complex UIs and generally require reliable WAN connections, which is not generally available in many worksites.

Hence, a popular solution is the set up of field offices. Field offices typically come in the form of acclimatized and network-enabled trailers where a fully equipped workspace, with printers, connectivity and desktop computing is provided. Those interim office spaces, however, are expensive to maintain, and are typically located relatively far from the actual pieces of equipment being serviced.

This physical distance requires FEs to go back and forth to the field office to interact with computational systems to obtain information and handle different ad-hoc matters during a typical workday. For example, to search for and print the schematics and maintenance procedure for a part that was found broken, to order new parts for an equipment, to contact experts with maintenance questions, to generate hand-over reports, and to report to distant supervisors. In one of GE worksites, for example, the field office was located a 15 min walk away from the equipment being serviced.

Constant FE trips to field office are translated into less time on deck supervising craftsmen, which may lead to quality issues, tasks delays, and ultimately, in higher costs due to longer equipment downtimes.

B. *Application integration*

Another source of inefficiencies in field engineering is the lack of application integration. FEs have to interact with different systems, including timekeeping, project management, e-mail, document management and office automation tools. Those are loosely integrated with one another, often-requiring repeated data inputs, which may also lead to information error and duplicated work.

The need for application integration has long been recognized by the software industry, and has approaches that leverage on both client and server-side strategies. On the server-side, different industrial initiatives exist as the Oracle Application Integration Architecture (AIA) (http://www.oracle.com/us/products/applications/communications/application-integration) and the work of [4] and of OPC-UA (www.opcfoundation.org).

Oracle AIA provides a common middleware for workflow-drive integration of Oracle products supporting common data exchange formats and complex integration procedures involving multiple systems. Approaches such as AIA, however, are vertical and limited to vendor-specific systems.

OPC-UA is an Industry-driven approach, based on Open Web standards for application integration. It defines a set of standard data exchange formats and utilizes Service-Oriented Architectures based on Web protocols to define exchange APIs. By adopting open protocols, they can be made compatible with a broader set of applications. Another example is the work of [4], where a plug-in oriented architecture is proposed to integrate, or exchange data and control signals between server processes.

Most approaches tend to be application-agnostic and focus on the use of standardized protocols for horizontal integration of applications. They do not address the problem of semantics integration, for example, the combination of information from different related sources, which need to be consistently combined and abstracted. As a consequence, this functionality is typically handled by the business logic of the application.

On the client-side, different application integration approaches such as Web 2.0 mash-ups have been used [5]. Through the use of Java Script libraries and Web standards, they support the combination of information from different sources in the development of Web applications. This approach, however, has the potential to create complex applications and lacks appropriate support for disconnected operation & contextual application development.

C. *Disconnected operation*

A common solution to the lack or intermittent connectivity problem in field engineering has been the adoption of applications that support disconnected operation.

Both research and industry work has been done in the areas of disconnected operation for mobile software applications. The work on CODA [6] and the work of [7] are examples of approaches for file-based caching and offline mode operation. The work of [8] formalizes those approaches into a general language. Automatic algorithms for pre-fetching data in support of disconnected operation have also been proposed [9]. Those seminal works are based on application-agnostic operating-system abstractions as files and protocol blocks. As such, it is not possible, for example, to synchronize individual data records or to replay transactional operations on databases.

The work of [10] and [11] employ agent-based protocols in support of disconnected operation in the telecommunication industry. In this computational model, a client agent cooperates with a server agent to ensure the correct delivery of network streams and data packets on highly unreliable networks. Similar to network file system approaches, these approaches operate over application agnostic network abstractions as packets and streams, which limits the optimizations that can be accomplished in those models.

Finally, a popular way to support application-level disconnected operation has been the use of mobile software agents, i.e. the transfer of data and process from the server side to the client as mobile code. The most popular example has been the use of Java Applets in web browsers, and the use of JavaScript embedded on Web pages [12]. Another approach for disconnected operation of thin-client applications has been proposed by [13]. By downloading and executing part of server code to the client, applications preserve their original functions during disconnected operation. These approaches, however, are not always fit for deployment on mobile devices, which have limited power, memory and processing capabilities. The transmission of agents through unreliable networks may not be always possible, and different security measures must be taken in order to safeguard client applications form malicious software.

D. Contextual mobile applications

Field Engineering is a highly regulated domain, where people must abide by strict safety rules, and must perform planned tasks according to schedule and well-known procedures. The highly structured nature of their work opens opportunity for context-driven applications, that leverage the knowledge about user's tasks and information needs to anticipate user's needs.

The use of contextual information in mobile computing has gained increased attention in the literature [14]. In particular, contextual information such as knowledge about tasks, user profile and physical location has been used to optimize the amount of data mobile devices should cache during disconnected operation, and to customize the user experience and information to the task at hand.

For example, the work of [15] has studied the use of context information in support of offline operation in mobile devices. The user profile information and the knowledge of user calendar appointments are used to cache data to the users PDA. A rule-based contextual engine for performing application data reconciliation when the device is reconnected to the network is also discussed. This work, however, the user context is mostly defined by logical information including appointments, business contacts and assigned tasks.

The work of [16] describes a contextual engine for mobile devices that account for both physical and logical context, allowing the combination of both physical location and logical application data for the determination of user context. A context engine is described, based on RDF-defined domain models compatible with offline mode in mobile devices.

Research has also been done on the application of mobile contextual applications in physical activities and procedures. For example, the work of [17] describes the results of a case study where medial records are provided to nurses according to their proximity to patients in hospitals. This works demonstrates the potential of context to procedure-oriented environments.

III. MCI Approach

The MCI architecture advances on existing research and industrial approaches by integrating: client and server-side agents to handle disconnected operation, reconciliation and application integration; together with logical and physical context determined by integrating data from: corporate systems such as task and project models, and the user environment such as iBeacons, RFIds, bar codes, and measurements from smart tools. It also supports multi-modal user interaction in the form of mobile and wearable devices as illustrated in Figure 1.

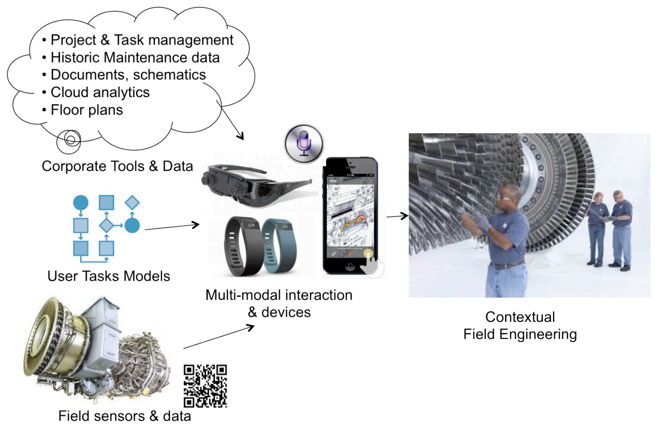

• Project & Task management
• Historic Maintenance data
• Documents, schematics
• Cloud analytics
• Floor plans
Corporate Tools & Data
User Tasks Models
Multi-modal interaction & devices
Contextual Field Engineering
Field sensors & data

Figure 1 High-level description of the approach

Applications developed according to MCI architecture execute in mobile phones, tablets and wearable devices that can be easily carried by Field Engineers. These applications provide ready access to information anywhere in the field. In a typical scenario, users start their day by logging into the mobile app; the app then determines the user's current task assignment based on the corporate task & project management tool, and the current timesheet of the user. Immediately after logging in, data is pre-fetched for the day: documents and schematics that might be required in the context of current work assignments are downloaded from corporate document management system; project and timesheet info are also downloaded and stored in a local cache of the mobile device. The user then leaves for the day journey in an environment that may or may not have reliable connectivity. The mobile devices sense the environment via near field communication such as Bluetooth Low Energy (BLE), or through auxiliary devices such as wearable cameras and virtual reality glasses. Those can quickly scan

QR and bar codes attached to equipment or even recognize pieces of equipment by their picture. The application responds to the context by suggesting task information to the user, including complex task steps, and by allowing the user to update the status and progress of current tasks; when the user approaches equipment being serviced, documents about that machine are made available. Field annotations and observations can also be performed and published to a news feed shared by all FE's working in the same project. Feeds may include pictures and procedure observations; or EHS (Environmental Health and Safety) hazards spotted on the field. At the end of the day, events recorded in the feed are used for automatic generation of handover reports. Individual timekeeping can also be performed and submitted. All operations are available in both connected or offline modes. Offline data is synchronized opportunistically, once network connectivity is reestablished, using application-specific synchronization strategies for conflict resolution.

Compared to existing approaches, the MCI architecture supports disconnected operation and synchronization at the API level, i.e. it leverages application semantics to resolve conflicts. For example, status updates a combined into a single update to the server; multiple timesheets, submitted during the day, are collated and compared with server-side submission. It also performs pre-fetching of data based on application and physical context. MCI notion of context combines both user profile information as defined by different corporate applications, as well as physical location and proximity to assets (equipment in the field), and existing connectivity. In particular, we rely on environmental beacons and markers, associated to assets, to fetch related information for the user task with that specific asset.

IV. Software Architecture

As briefly discussed above, and illustrated in of Figure 2, the MCI architecture defines 3 software layers: the client-side layer, that executes in the mobile & wearable devices, the middleware layer, that executes in cloud services, and the corporate applications layer, that provides adapters to existing corporate applications distributed through different sites in the organization. Layers communicate with one another through RESTful Web Service [18] utilizing existing network technology when connection is available.

In the architecture of Figure 2, the left-hand-side boxes represent the software components and sub-systems of our approach; whereas the right-hand side clouds and devices represent the physical hardware components where those software sub-systems execute.

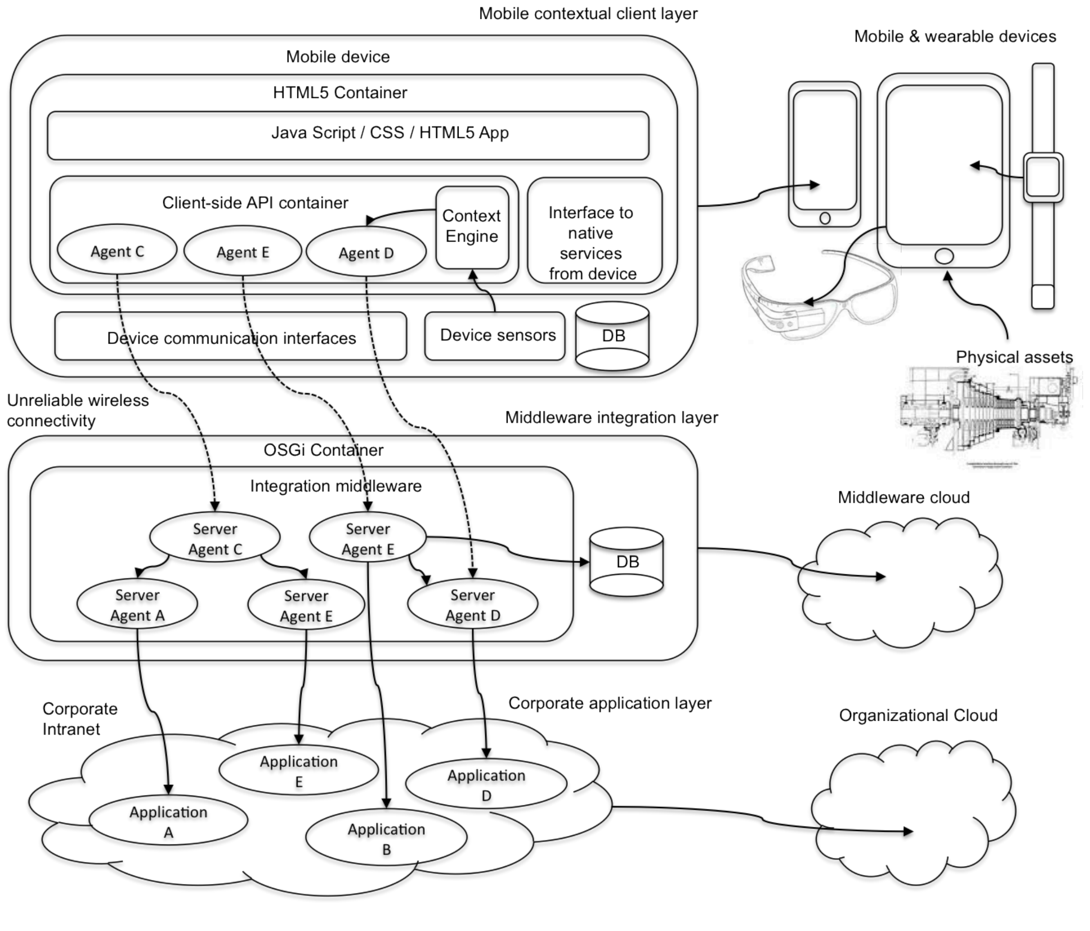


Figure 2 Main layers & components of the architecture

A. *Mobile devices layer*

The mobile devices layer (shown in Figure 2) supports the development of applications for different mobile devices including tablets, phones and wearables. Mobile applications are developed using open Web standards as JavaScript, CSS and HTML5, and execute within a HTML5 container such as PhoneGap (http://phonegap.com/), or Apache Cordova (http://cordova.apache.org/). The use of Web standards on an OS independent container provides portability for the application, supporting its migration to different devices. These applications interact with a local representation of the middleware services within the client-side API container. This container executes different agents, one for each required middleware service for the application. Interaction with environmental sensors is also important. Bridges to device-specific sensors and capabilities are utilized for this purpose. Existing bridges provide interfaces to local storage and to common sensors such as GPS, Cameras, Bluetooth and others.

From the application developer perspective, the interaction with middleware services and device sensors is mediated by local client-side agents. The term agent is employed here to highlight the collaborate aspect of these services. Agents initiate collaborate sessions with their middleware counterparts to synchronize and pre-fetch data and application-specific messages. In other words, agents are smart components that provide mobility transparency by detecting when the mobile device network connection is on or off. They automatically switching from online to offline operation, and perform opportunistic data synchronization using application-specific policies. The communication between client and middleware agents is performed using HTTP RESTful method calls over the available network.

An important aspect of the system is its support for disconnected operation. This support involves three main modes of operation as illustrated in Figure 3.

1) *Connected with contextual pre-fetch*

When the mobile device is connected to a wireless network and the communication between client and middleware can take place, the client-side API switches to connected mode. During this mode, API calls are forwarded to the corresponding middleware agents, and information flows from clients to servers using the network. Information produced and consumed by the application is also cached in the local client-side database.

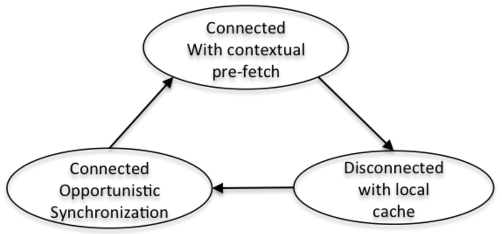


Figure 3 Offline to online mode transaction

In this mode, client agents also pre-fetches application data, based on the application context. This prefetching can be done either automatically, with the help of contextual information, or manually, explicitly directed by the user. Automatic pre-fetch is performed by the system using application-level and location-based knowledge provided by the context engine. In the SmartOutage application (discussed in section D), for example, users indicate the tasks they will be performing during the day. Tasks have information about assets being serviced, including the serial number of the asset. That information is used to automatically download documents and schematics to the mobile device, and to pre-populate the local project management database and timesheet.

2) *Disconnected operation*

When the mobile device connectivity becomes unavailable or unreliable, and the middleware agents cannot be reached through normal network connection, the client-side agents automatically switch to disconnected operation. During disconnected operation, all read and write operations on services data is served from the local cache. In particular, write calls are timestamped and moved to a local queue where they stay until the system gets reconnected, and local data state is updated to reflect the change. This work is performed by software agents associated to each middleware service.

3) *Opportunistic Synchronization*

When the system switches from disconnected to connected operation, synchronization takes place. During this process, the inconsistencies between client and server data are detected and appropriate read/writes are issued in order to deliver unsent items or to update local cache. Data obtained from the middleware agents may have precedence over data stored locally, and appropriate merging algorithms are implemented in an application-specific basis.

In order to save bandwidth and maximize the use of high latency and low bandwidth networks, synchronization is performed in an opportunistic fashion. In this process, data is synchronized at each client API method call instead of reading/writing all inconsistent information to the middleware at once, when network connection becomes available. That's how it works: the first time, after a period of disconnection, the application invokes an API method in the client API, all locally stored, not submitted information is synchronized. The synchronization process compares timestamps and data ids with those of the middleware services, sending data for which the local timestamp is higher than the last known good local call. Note that this process can be overwritten according to each application semantics.

In order to prevent data from non-invoked methods to remain in the client side indefinitely, a push timeout is defined, after which all queued data is flushed from the client to the corresponding sever agent.

B. *Contextual computing*

Another important component of the client-side API is its contextual engine. The contextual engine continuously gathers information from the device surroundings and from corporate information systems. Surrounding information include beacons and tags ids (such as QR codes) attached to the environment and machinery. Data from corporate systems include current tasks, job assignments, assets, and documents. Based on that information, the context model, defined by a set of content-based subscriptions, is updated continuously, producing context change trigger events that are consumed by the client-side agents.

Agents subscribe to specific types of events produced by the context server, e.g. proximity to assets, in-door location coordinates, task progress, and receive notifications when those values change. For example, the application can respond to the proximity to an asset to be serviced by producing an audible or visual notification, or it can offer to open a schematic associated to that asset; tasks can be automatically marked as in progress, and feeds can be automatically posted to the project activity feed timeline.

C. *Middleware layer*

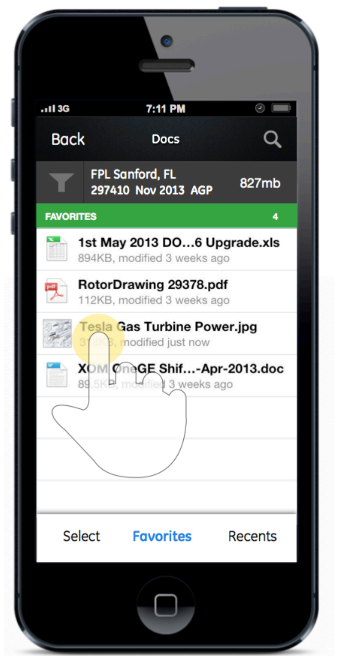
The middleware layer hosts agents that support the disconnected/connected operation of the client as well as the integration, summarization and filtering of information from different sources, performing large part of the computation required by the application, and keeping data consistent across heterogeneous applications.

Each agent, therefore, provides a service abstraction layer, representing general back-end services e.g. document service, timekeeping service, project management, etc. This allows the client app to be ported to different backend applications without change. Middleware agents can also provide novel features based on existing agents, e.g. news feeds and reporting services.

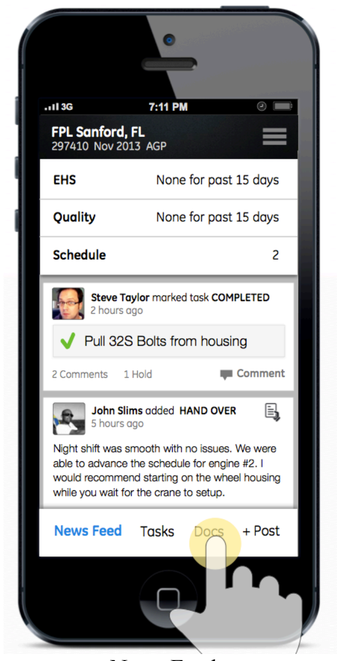
As middleware agents are assumed to have better network connectivity with corporate services, as well as higher bandwidth than client counterparts, they perform most of the computationally expensive processing required by the client applications, including the validation, submission, reading, summarization, filtering and duplication of data from different sources. Middleware agents cooperate with client agents during synchronization, comparing timestamps and data ids, finding differences between data stored on the server and on the clients.

The middleware layer is dynamic and modular. Agents are implemented as bundles in an OSGi container such as Apache Karaf (karaf.apache.org) that provides automatic activation/deactivation of agents, hot deployment of new or existing agents, and dependence management. Agents can also be replicated in different servers. Agent update is supported by OSGi hot deployment.

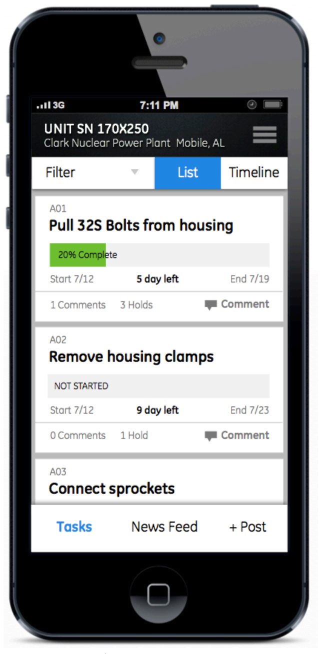
Security is another requirement addressed by the middleware services. Enterprise services must be protected from external attacks. The use of a middleware agents that mediate the access of enterprise information provide an additional protection layer where data can be checked, users can be authenticated and access can be authorized. The agents in our architecture require previous authentication of clients. In a corporate setting, the authentication mechanism is provided by a single-sign-on entity whereas the authorization can be implemented in the agent-level based on those credentials. The middleware container provides basic authorization and authentication mechanisms for the development of new middleware agents, and the communication between agents is performed via HTTPS.


3G
7:11 PM
Back
Docs
FPL Sanford, FL
297410 Nov 2013 AGP
827mb
FAVORITES
4
1st May 2013 DO...6 Upgrade.xls
894KB, modified 3 weeks ago
RotorDrawing 29378.pdf
112KB, modified 3 weeks ago
Tesla Gas Turbine Power.jpg
modified just now
XOM OneGE Shif...-Apr-2013.doc
3 weeks ago
Select
Favorites
Recents

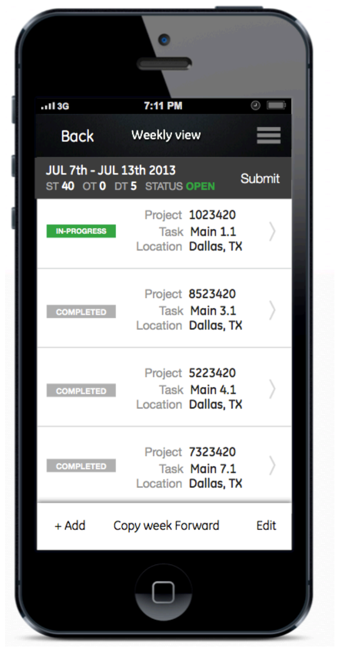
Document management


3G
7:11 PM
FPL Sanford, FL
297410 Nov 2013 AGP
EHS
None for past 15 days
Quality
None for past 15 days
Schedule
2
Steve Taylor marked task COMPLETED
2 hours ago
Pull 32S Bolts from housing
2 Comments
1 Hold
Comment
John Slims added HAND OVER
5 hours ago
Night shift was smooth with no issues. We were able to advance the schedule for engine #2. I would recommend starting on the wheel housing while you wait for the crane to setup.
News Feed
Tasks
Docs
+ Post

News Feed



Task Management



Task-based Timekeeping

Figure 4 Sample SmartOutage app screenshots

D. *Corporate applications layer*

The corporate application layer provides points of access and adapters to existing applications and services. This layer provides raw data by means of read/write methods to corporate services and databases, exposed through RESTful services.

In the current implementation of the architecture, for example, we utilize RESTful interfaces to corporate databases behind key applications such as timekeeping, project management, document management and directory services.

I. SMARTOUTAGE APPLICATION

We have applied the described mobile contextual integrated architecture in the development of an integrated mobile application called SmartOutage, with some screenshots shown in Figure 4. SmartOutage provides a single point of access for timekeeping, project management, document management, reporting and other corporate systems that are integral part of FEs work.

An outage is a planned set of maintenance tasks performed in critical equipment, such as power generation turbines, airplane engines, oil rigs, etc. Even though planned outages are necessary for the long-term operation of heavy machinery, they are costly and time critical, due to the need for equipment shutdown. Therefore, optimizations of time spend during those operations and the removal of workflow inefficiencies can result in millions of dollars in savings a year. SmartOutage helps outage workers achieve this goal by bringing the information closer to the end user, the FE, by supporting field annotations and documentation, and by optimizing tasks as report generation and progress reporting.

Before the introduction of SmartOutage application in mid 2014, FEs had to input their timecards, update their project statuses, print document and schematics and generate handover reports manually, using field notes that were later translated into printed reports, and by using different corporate applications, one for each task. Work was many times duplicated, utilizing corporate systems from different vendors e.g. Oracle T&L for time keeping and Primavera for project management, office automation tools for reporting, and Web-based document management systems for asset documentation management. There were many document repositories, one for each type of equipment, parts and historical data. Heavy stacks of paper, with documents and schematics were often carried along and produced each day. Moreover, in order to produce this material, FEs were often required to move back and forth from the asset deck to the field office.

SmartOutage was developed according to the MCI architecture. As illustrated in Figure 5, the SmartOutage client application provides a simplified and integrated interface to corporate services such as time keeping, project management, and documents. An activity feed is also provided allowing users to post occurrences and observations during the day.

The middleware layer agents implement an abstract API that combines, filters and caches data from different organizational services. The time keeping agent communicates with Oracle T&L time keeping; whereas the project management agent interacts with Primavera Project Management. The document management agent is responsible for filtering, indexing and caching documents from different proprietary GE document repositories, creating a unified point of access to this type of information. Finally, the activity feed agent stores user-provided message feeds, as well as application-generated feeds (such as project progress notifications), and use that information to the generation of handover reports that can be printed or e-mailed. Handover are reports used by FEs to communicate health and safety occurrences, activity and project status reports to other FEs on different work shifts.

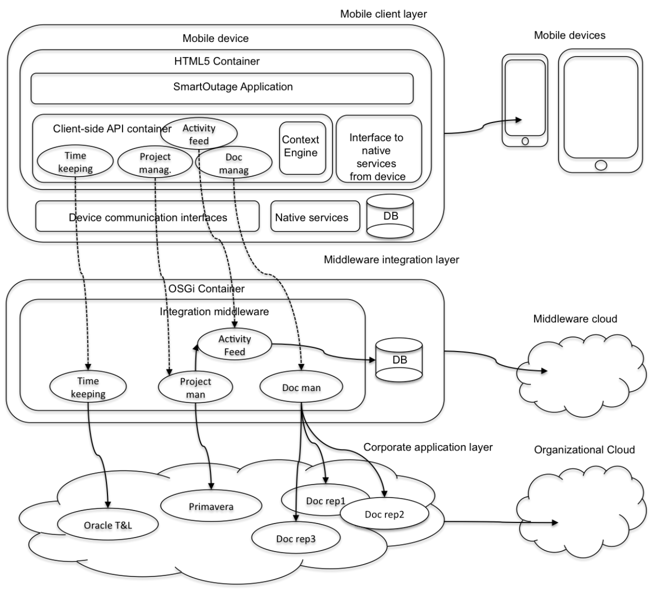


Figure 5 Smart Outage architecture

Before entering in offline mode, the SmartOutage client app allows FEs to download required documents and cache project data. Upon login, based on the user location and the task management system information, the system knows which projects and assets the user must service, and proactively downloads required data and documents to the mobile device. SmartOutage also allows FEs to update their timesheets and task status on the field, and downloads all required project and task management info to support that task.

All these operations are accomplished utilizing cached data, without the need for an active network connection. At the end of the day, using a reliable wireless network connection, the system automatically synchronizes its data with the SmartOutage middleware agents, which update corporate systems accordingly.

A common issue faced by FEs was the generation of daily outage reports, which summarized the activities performed during the day. This report includes maintenance tasks as well as any health and safety issues found during the day. Reporting could take hours and required copying of hand notes taking during the day to standardized forms. With the app, FEs can now report the issues as they see them in the filed. They can take pictures, and post observations to activity feeds. At the end of the day, a report is automatically generated using that information.

The benefits of SmartOutage use are many. As the software gets adopted by FEs in the field, we expect to eliminate 17% of the FE's non-value added time, e.g. commuting time between deck and field office, transcribing notes into reports, and searching for documents in different repositories. We also expect quality gains in outage work by increasing FE's deck time in 40%. This time will be better used supervising craftsmen work, solving problems on the deck and transferring knowledge to new workforce. Overall, we expect a 10% decrease in outage time, which translates in lower downtime and maintenance costs.

II. Conclusions and Future Work

In this paper, we described the MCI software architecture for mobile contextual integrated field engineering applications. This architecture integrates contextual information from the environment and other devices and information from different corporate applications in the development of mobile applications that are more portable and contextual and integrated than their desktop counterparts.

The benefits of this approach are: a simplified API for client application development, and the automatic handling of data disconnection and synchronization, with the benefits of improved integration of corporate information systems and support for both logical and physical contextual information. We discussed the use of the architecture in the development of SmartOutage, a mobile application utilized by Field Engineers, and its benefits to the business.

We are currently extending the contextual client container to better leverage on information generated by other mobile devices in the field, and to respond to a broader range of to events generated by beacons and markers attached to assets. For such, we utilize additional devices such as augmented reality glasses, smart tools and smart watches paired with existing mobile phones. We are also studying the use of mesh networks to improve the communication between devices in the field.

III. Acknowledgements

We would like to thank all the members of the SmartOutage project for their hard work, in particular: Prasad, Amit, Venki, Chandra, Adeeth, Huawei, Casey, Soumia, Lana, Supriya and Uday.

References

[1] T. L. Koreshoff, T. Robertson, and T. W. Leong, “Internet of Things: A Review of Literature and Products,” in *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, New York, NY, USA, 2013, pp. 335–344.

[2] “Introducing the Industrial Internet.” [Online]. Available: http://www.ge.com/stories/industrial-internet. [Accessed: 10-Jan-2014].

[3] J. Nieminen, C. Gomez, M. Isomaki, T. Savolainen, B. Patil, Z. Shelby, M. Xi, and J. Oller, “Networking solutions for connecting bluetooth low energy enabled machines to the internet of things,” *IEEE Netw.*, vol. 28, no. 6, pp. 83–90, Nov. 2014.

[4] D. C. Fernando, A. Candappa, K. Pawar, and N. Maheshwari, “Server side application integration framework,” US8584081 B2, 12-Nov-2013.

[5] S. S. Minhas, P. Sampaio, and N. Mehandjiev, “A Framework for the Evaluation of Mashup Tools,” in *2012 IEEE Ninth International Conference on Services Computing (SCC)*, 2012, pp. 431–438.

[6] J. J. Kistler and M. Satyanarayanan, “Disconnected Operation in the Coda File System,” *ACM Trans Comput Syst*, vol. 10, no. 1, pp. 3–25, Feb. 1992.

[7] T. G. Cantrell, S. Jaji, A. A. Shaheen, and R. B. Ward, “System and method for efficient caching in a distributed file system,” CA2142797 C, 12-Apr-2005.

[8] M. E. Fiuczynski and D. Grave, “A Programming Methodology for Disconnected Operation,” University of Washington, Technical Report, Mar. 1994.

[9] G. H. Kuenning and G. J. Popek, “Automated Hoarding for Mobile Computers,” in *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, New York, NY, USA, 1997, pp. 264–275.

[10] D. Ochi and K. Yamazaki, “Twin Agents: network-assisted disconnected operation and distributed processing for mobile communication,” in *2004 Intl. Symposium on Applications and the Internet, 2004. Proceedings*, 2004, pp. 31–39.

[11] J. C. Navas and Y. A. Shu, “System and method for facilitating asynchronous disconnected operations for data access over a network,” WO2005038614 A2, 28-Apr-2005.

[12] J. R. Erenkrantz, M. Gorlick, G. Suryanarayana, and R. N. Taylor, “From Representations to Computations: The Evolution of Web Architectures,” in *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, New York, NY, USA, 2007, pp. 255–264.

[13] R. Valia, “System and method for disconnected operation of thin-client applications,” US7685253 B1, 23-Mar-2010.

[14] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context Aware Computing for The Internet of Things: A Survey,” *IEEE Commun. Surv. Tutor.*, vol. 16, no. 1, pp. 414–454, First 2014.

[15] M. Rangan, E. Swierk, and D. B. Terry, “Contextual replication for mobile users,” in *International Conference on Mobile Business, 2005. ICMB 2005*, 2005, pp. 457–463.

[16] S. Zander and B. Schandl, “A Framework for Context-driven RDF Data Replication on Mobile Devices,” in *Proceedings of the 6th International Conference on Semantic Systems*, New York, NY, USA, 2010, pp. 22:1–22:5.

[17] B. Skov and T. Hoegh, “Supporting Information Access in a Hospital Ward by a Context-aware Mobile Electronic Patient Record,” *Pers. Ubiquitous Comput*, vol. 10, no. 4, pp. 205–214, Mar. 2006.

[18] R. T. Fielding and R. N. Taylor, “Principled Design of the Modern Web Architecture,” in *Proceedings of the 22Nd International Conference on Software Engineering*, New York, NY, USA, 2000, pp. 407–416.