Context-Enhanced Web Service Invocations in Mobile Business Processes

Heinz-Josef Eikerling

Fachhochschule Osnabrück – University of Applied Sciences,
Laborbereich Technische Informatik / Institute of Computer Engineering
Barbarastraße 16, D- 49076 Osnabrück, Germany
h.eikerling@fh-osnabrueck.de

Abstract. We present a mechanism which transparently enhances service invocations by contextual data (e.g., location / positioning data gained through other information that is accessible within the invocation chain): the context is applied and extracted in a way such that the called service(s) can be completely agnostic to the intricacies of context encoding, transmission, and processing. This is particularly important when accessing services from within a mobile environment, since mobile processes are sensitive to contextual data like location, user and device status by nature. We intercept the service request / response handling prior to calling the business logic and the delivering of the response to the calling application, respectively. The proposed approach turns out to be rather applicable in Field Force Automation scenarios and efficient which is proved by some experimentation.

Keywords: mobile business processes, context, web services.

1 Introduction

In today's business environment, efficiency and business success increasingly relies on how companies manage to seamlessly integrate employees into the business process, independent from where they are and which device they are using. This is frequently referred to as *user mobility*. In addition to the mobility of employees, within a business process a company has to deal with mobile assets, i.e. moving / movable objects like a specific medical device within a hospital (*asset mobility*). Being able to dynamically involve both as *context data* in business processes offers a huge potential to optimize process execution speed, process cost, and asset utilization.

The above aspects have to be aligned to the tendency to wrap certain functionalities as services in order to foster reusability and gain flexibility to move services to external service providers [10]. Services can perform everything ranging from simple functions to complicated business processes. They allow business entities to deliver their key offerings over the Internet [8]. As an underlying technology, web services (WS) and the underlying service-oriented architecture (SOA) paradigm both have gained attraction. Particularly mobile applications are nowadays built using services [4], i.e., applications consume services and may also offer services to other mobile devices within a certain range.

Y. Cai et al. (Eds.): Mobilware 2010, LNICST 48, pp. 250–262, 2010.

Contextual data may help to tackle the problems linked to the integration of services, similar as for humans where it expands the conversational bandwidth [1] and enables to use implicit, additional data to understand and process transmitted complex information more effectively. Aligning to a widely adopted definition [3], we assume context to be any information suited to characterize the situation of a person, a computing device, or a software agent. Instances of information covered by this definition are distinct locations, capabilities and services offered or sought, or activities and tasks in which services are utilized.

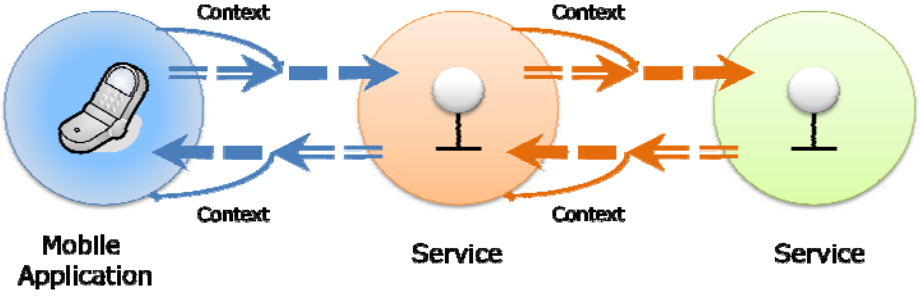

Fig. 1. Context enhancement for accessing services in mobile applications

The target of this work is to devise a mechanism which enhances service calls by context information as shown in Figure 1. The context can be delivered and extracted in a way fully transparent to the involved services. Thus explicit changes to the service interface can be avoided. The context enhancement can be applied to the service request and also to the response returned by the invoked service. A request can be initiated by an application or by an intermediate service in the request processing chain. Due to their importance we focus on web services and describe a mechanism for the automatic, at runtime adaptable and transparent provisioning of context data fulfilling the aforementioned requirement.

2 Sketch of Proposed Solution

Our solution consists of a generic mechanism for *implicit* context enhancement and extraction which makes customized context available to conventional web services. This means that the services do not have to take care of the actions necessary to attach context. Additionally, the target services do not have to process the contextual data themselves; context processing can in principal be delegated to other specific services.

We consider that the context-enhanced interaction has to be possible even if the services belong to different infra-structures. Most importantly, the contextualized services will stay usable and functional, no matter if they are capable of processing the context or not. Therefore, the solution will not limit the set of potential target services in a composition by requiring special properties like being consistent with a specific framework and its conventions.

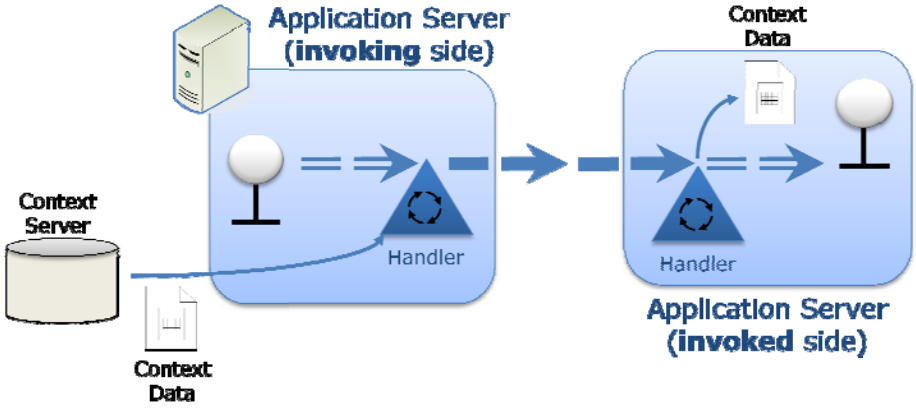

Fig. 2. General principle of enhancing service calls with context data

In the subsequent description we will focus on the invoking side for reasons of conciseness. We assume that components representing the invoking and invoked side are deployed to some kind of container, e.g. an application server or service container. The handlers used for context enhancement have to be installed on these containers as shown in Fig. 2.

3 Related Work

The two research areas, context-aware computing and web service interaction, are rather active. A lot of work on how to apply e.g. spatio-temporal context exists. However, existing approaches are mostly limited to special scenarios and do not use a more capable overall framework for implementing context awareness. They also often consider a fixed set of context elements and thus lack extensibility. Especially, there have been several projects on location-aware systems [2]. [5] describes a system where users can access information depending on their geographical positioning. This idea has also been implemented in [11] where users can attach annotations to geographic positions such that the information is accessible by other users whenever they approach that position. A presence aware system is presented in [7].

In the COWSPOTS [9] project context-awareness is used to provide enriched Web services to health professionals. The system consists of a central server and several mobile devices that run so called SPOTlets. According to this centralized approach, the whole processing is completely done on the server side.

While several approaches to the problem stated above present good mechanisms to context management and/or processing, they fail to constitute a mechanism that considers traditional web services when it comes to context-aware service invocations. Solely [6] presents a framework that facilitates the development and deployment of context-aware adaptable Web services. The framework features context plug-ins that pre- and post-process service requests based on available context information. Context is iteratively appended to the exchanged messages whenever a service sends a message to another service. On the invoked side the appended context

can be extracted, processed, and used either by the invoked service or through the context processing plug-ins. The authors assume one plug-in per type of context, i.e. if a Web service *A* invokes another service *B* with context information, this context is appended in addition to the context transmitted by *B*, if *B* invokes another Web service *C* during that invocation by *A*. The approach supports invocations of services which are not context-aware. However, the solution does not permit the sent context to be customized to an invocation (e.g., the target or the method parameter). Also, the entire context of a service is forwarded along cascaded service calls which might likely cause a significant message overhead. Since context plug-ins are designed to handle one type of context only, several context plug-ins have to be activated simultaneously during each invocation which complicates the use of context combinations.

4 Context-Enhanced Service Calls

4.1 General Principle

We start off by separating the different concerns of the envisaged solution. In general, a framework that separates tasks like context acquisition / retrieval and storage from context queries would be of benefit. Therefore, the general solution consists of two parts. The first part is given by the *context server*. Through defining such server concurrent and remote access to contextual data is granted. The server acts as the central point for context data retrieval requests, context processing and distribution. Such approach ensures the maximum flexibility concerning context handling.

The advancement of this work over the state of the art is given by the second part, a *handler mechanism* deployed to the application server that post- and pre-processes exchanged messages to acquire and adjust context from the context server during the invocation of business methods and provide it to the invoked web service.

4.2 Context Storage and Retrieval

The context server supports multiple remote data sources by a standardized interface and concurrent access. This is important in order to enable different kinds of sensors to send context information to the server. Part of this solution is an adapter interface to attach different types of *context producers* (sensors). The adapter hides the details of low-level data acquisition and retrieval. It receives context data from the proprietary sensor interface and uses the server interface to publish context to the connected *context consumers*. For the general use of the context server in different scenarios, the ability to define new context types is offered.

For implementing this functionality the context server comprises a powerful rule engine as shown in Fig. 3. In this regard context pre-processing rules can for instance specify how to aggregate and transform context data delivered by the producers. Clearly, the variety and precision of changes to the context data prior to its persistent storage are directly linked to the supported complexity of these rules. Though the data flow is always directed from the context producers to the context consumers, for both – producers and consumers – *push* and *pull mode* have to be supported. This is handled by the management component inside the context server which permits to define rules needed to query for context (pull mode consumer).

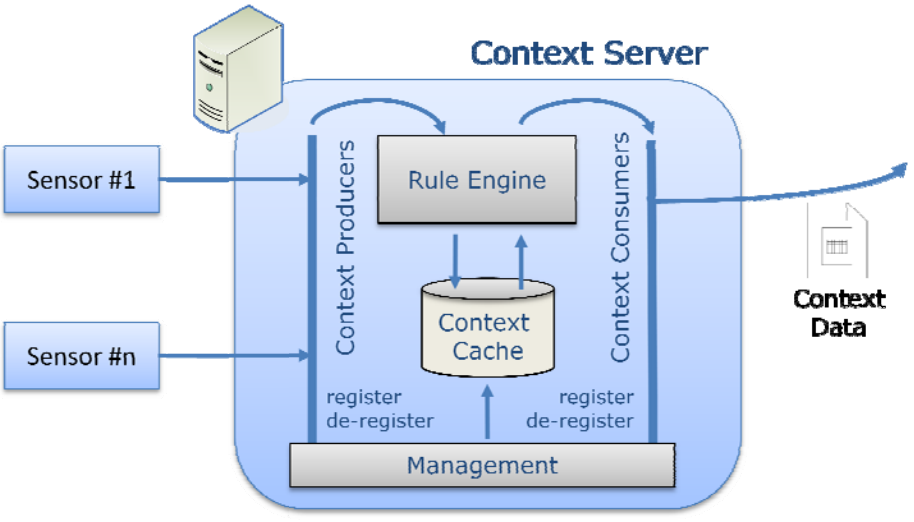

Fig. 3. Context storage and retrieval through the context server

An important task of the context server is to store the collected context information and make it available for subsequent use (*historic context data*). Hence, if no active context is available and requesting the respective context source / sensor is not possible, the most recent value can be obtained directly from the context cache.

It is worth mentioning that the conceptually centralized context server can be implemented using several physical servers for increased scalability or for providing the service to different application domains. This is easily achieved by making one context server a *context producer* to another one.

4.3 Context Transmission, Encoding and Provision

The transmission of context has to comply with web service standards. In addition, the mechanism has to be *implicit* such that context does not have to be transmitted *explicitly*, e.g., by adjusting WSDL files. Usually redirection, port and message filtering techniques are in place to secure the company network; only special point-to-point-connections are maintained for the communication with other (external) businesses. Setting up the network devices to allow a further communication channel can therefore be very expensive and ineffective, thus impossible. Hence, the context should be transmitted along with the SOAP message.

Different standards exist to populate the SOAP message header with meta-data in order to accomplish certain tasks. Our solution makes use of a header block which is inserted into the message sent by the invoking service. The header block contains the actual XML-encoded context information. This is in line with a W3C recommendation [12] stating that all information which could be of importance to nodes other than the actual communication endpoint (the target Web service) should be moved into the header, so that these nodes can provide value-added functions.

Instead of attaching context data on a dedicated middleware layer underneath the level of the service, a *handler* inside the application / service container is used to attach and detach the context. The handler extends the core functionalities of an application server by implementing functions that are invoked during the processing of incoming and outgoing messages. The handler used in the solution basically queries the context server for contextual information and creates the according header block in the SOAP message.

A handler is not only used on the sending side, but also on the receiving side, inside the container hosting the invoked service where it pre-processes the SOAP message. Before the web service is invoked with the according parameters contained in the SOAP body, the handler extracts the context information. If the web service is context-aware, it can intercept the context during the operation invocation and does not have to parse the SOAP message itself. In case the web service is not context-aware, it will simply ignore the context and will be not affected.

An important requirement constraining the context provisioning mechanism inside the service container is that the employed data structure has to be unique for every invocation. If for instance an operation of one service is invoked simultaneously by two different services *A* and *B*, inside every invocation access to the according context has to be provided. Otherwise a race condition could occur (see Fig. 4).

4.4 Context Selection and Handler Configuration

For deciding which context data should be appended for a specific invocation, *general* and *invocation-related* properties have to be distinguished. While the *invocation-related* properties (i.e., the invoked target service, the invoked target operation of the service, the current operation parameter values) are configured and detected on the handler side, the *general* properties are set up and evaluated in the context server.

Thus,

1. the *invocation-related* properties are set up in a handler configuration file
2. the *general properties* can be configured directly in the context server via according rules

To decide for the general properties which context information to include, different rule sets are defined directly inside the server. Depending on the set of rules used, different contextual information is collected and returned upon a query.

The decision is made in two steps.

1. First, the handler of the invoking service takes the SOAP message to determine the targeted service, the invoked operation and the parameter values. A *scenario identifier* (*SCID*) in the handler configuration file is provided for each service operation. This identifier along with the parameter values is used to query the context server for the required data.
2. In the second step, the context server uses the transmitted scenario identifier to select one of the rule sets. The rules define how the parameters have to be taken into account and which context should be collected in general. Rules are also used to transform the contextual information into an XML structure such that the handler can directly insert the response of the context server into the header block for the SOAP message.

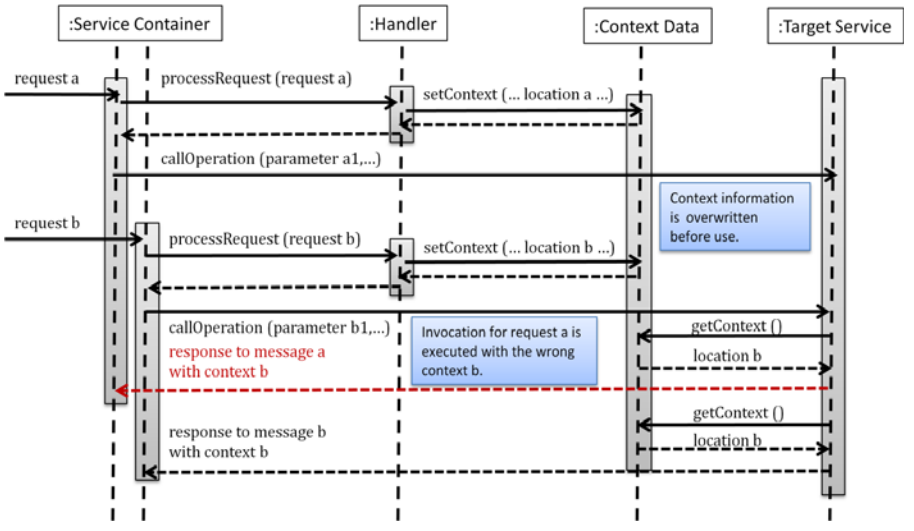

Fig. 4. Invocation of a service by two entities A and B causing race condition if the context data (here location of an object) is not handled per invocation

In general, the parameters of a service invocation can be complex, e.g., multi-dimensional types, for example composed objects or arrays. For use with the context server, scalar values are most suitable because usually one might not need all entries of a complex type for the context query, but only certain ones. In order to offer a general way to describe which scalar value inside a possibly multidimensional parameter should be chosen, XPath expressions are used.

With this approach, it is possible to keep the handler as simple (and thus effective) as possible and leave all context-related decisions to the context server.

4.5 Implementation Notes

The previously specified system was built on top of the context server described in [13] which is not specifically addressing contextualized services as context consumers and producers. The handler mechanism was initially developed for a stationary environment [14] and makes use of the according features in the Apache Axis2 as the service container.

5 Experiments

5.1 Purpose

The above approach can be judged concerning two criteria:

1. *Performance* is a key demand for applying the above described mechanism in business environments. This translates into the requirement that the benefits of context-enhanced services come with an as small as possible overhead in terms of message latency, throughput and processing time.

2. The overhead can be also measured in terms of the *message size* which has to relate the net size of a service invocation (non context-enhanced) with the gross size of it (context-enhanced).

The latter is very scenario specific which is why we have focused on evaluating the performance. One way to approach this is to compare the context handling mechanism by *explicitly* extending the service interface through the *implicit* mechanism using the proposed handler. If the latter is not slower and does not produce significantly larger messages than using the service interface to transfer context information, then the performance requirements are met. Hence we conducted an experiment test on a service invocation in which a certain amount of information (and context) is transferred using the explicit and the implicit transmission of context. First, the runtimes for executing the invocations are compared.

5.2 Environment

In order to factor out effects like packet loss or latency that occur in network environments, all applications are executed on the same computer using the local loopback device for communication. In such environment, the measuring of execution times can be simplified since the local system time can be used to determine execution time intervals between different invocations. The time between the end of the invocation of operation *A* and the start of the invocation of operation *B* is simply determined by measuring the interval between the time for the last instruction in *A* and respectively first instruction of *B*.

5.3 Results

Aside from tests measuring times spent at the different stages of the processing pipeline (request creation, context retrieval, context appending, request processing, sending response) we focused on the direct comparison of the execution times for the explicit and implicit method for transmitting contextual information.

Fig. 5 shows the results of executing these tests. We conducted a series of 6 scenario runs (*A*, *B*, *C, D, E, F*) each comprising 500 invocations; each invocation is handled by the explicit and by the implicit method. The execution times are recorded and the truncated averages (max / min values are dropped to factor out to temporal phenomena) are computed and shown for all three runs.

The tests indicate that the deviation concerning runtime between the handler-based and explicit interface-based mechanisms for context delivery is below 2% on average over the examined scenarios. This is because the additional software components (handlers for context attaching / detaching, context server) on the processing chain perform quite efficiently. As will be explained later, the benefits (separating the static business logic from the rather instance-specific dynamics of context management) more than compensate this overhead.

Similar observations were gained when evaluating the message size overhead: assuming the same encoding of the context to be transmitted in either approach, the message sizes are the same. The explanation is straight-forward: whereas in the explicit, interface-based approach the context data is contained in the SOAP body, the handler-based, implicit approach moves this data to the SOAP header. Thus the overall message size is the same.

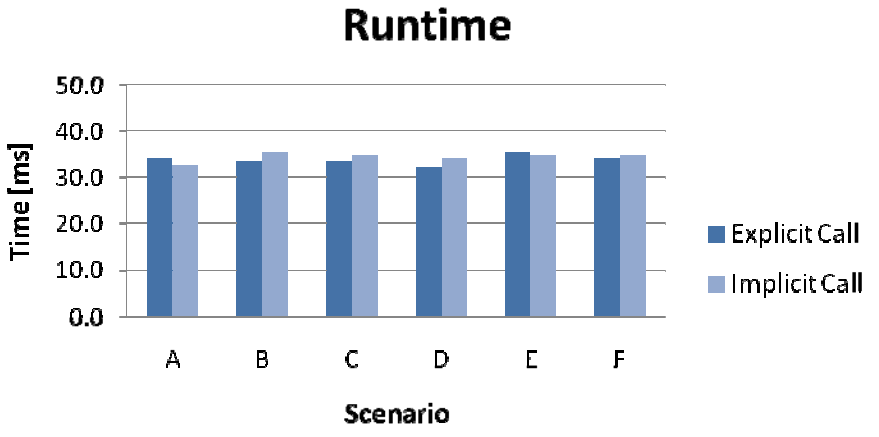

Fig. 5. Comparison explicit / interface and implicit / handler based approach for testing scenarios

6 Case Study: Field Service Automation

6.1 Field Service Automation in the Automotive Sector

Advances in technologies have lead to an increasing diversity in the automotive sector. The consequences are bigger differences between car manufacturers and their models, so that the collaboration between car manufacturers and business partners (e.g., field workers) handling repairs is gaining importance. In our scenario, we look at the interaction of the entities in case a client car is broken during the warranty period.

Among the actors is the car *dealer (A)* contracted to a *manufacturer* (*B*). *Field workers* (*C* and *D*) are associated with the manufacturer. In the scenario the customer contacts his dealer because an issue (e.g., malfunctioning of the car electronics) with his car occurred.

For fixing the problem the dealer contacts the manufacturer by sending a notification to the manufacturer's (step 1.) service via a straight-forward interface. The sent message (2.) contains the incident id (*I-ID*), a number the dealer uses to refer to the case, and a problem class id (*P-ID*), a number which describes the class of the problem (electronic problem, tank leakage, …), to *B*. Since the incident happens at the site of the dealer, the dealer's location might be of importance to field worker for deciding whether to take over the job or not. The manufacturer maintains a directory from which the contact data for *A* can be retrieved (3.).

With the *I-ID* and *P-ID* information plus the contact information of the dealer, the car manufacturer *B* tries to assign the problem to one of the field workers. He checks with one field work after the other, until the issue is successfully assigned which is indicated by an according response of the field worker.

After the issue was successfully assigned, the field worker accepting the issue (here *D*) contacts the dealer with the *I-ID* (the number the dealer assigned to refer to the case) which was obtained from the manufacturer and they negotiate the next steps to cooper up the car.

6.2 Service-Oriented Implementation of Scenario

Figure 6 shows the service-oriented implementation of the scenario. The web service of manufacturer *B* provides an operation *assignIssue()* which requires two arguments, an integer value for the *I-ID* and another integer *P-ID*. The interface to this operation is known by the service of dealer *A*. Similarly, the services of *C* and *D* published their operation *assignIssue()* that requires an additional parameter besides the *I-ID* and the *P-ID* for the contact information. A list of the associated field worker services is maintained in the service for *B*. Every time *A*'s service invokes the method *assignIssue()* of the manufacturer *B*, *A*'s application server generates and sends the according SOAP message.

At the target URI of *B* the SOAP engine in the service container of *B* accepts the message and invokes *assignIssue()* of the according service with the transmitted parameters *I-ID* and *P-ID*. The service *B* looks up the contact information of *A* by the IP address of the message. With the parameters *I-ID*, *P-ID* and the contact information of *A* (contact) the service *B* now loops through all known field worker's services. For one after another, it invokes their *assignIssue()* method with the three parameters. The application server of *B* therefore creates the according SOAP messages and delivers them to the according end-points.

This straight forward implementation reflects a very low usage of context information. Context information relevant to specific actors (dealer's location for field workers) is resolved through explicit service interfaces which complicates the business logic of the manufacturer unnecessarily.

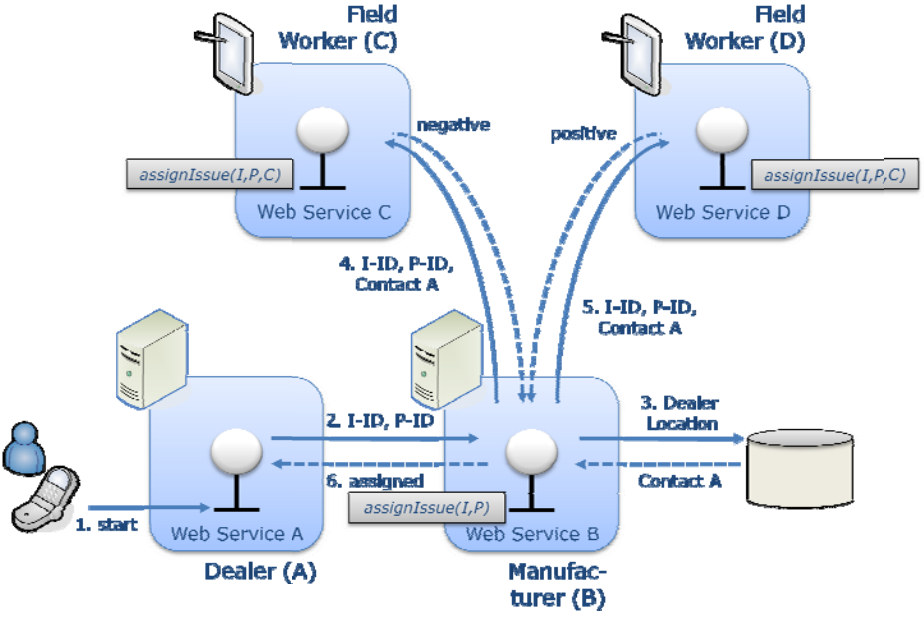

Fig. 6. Assigning an issue in a car maintenance scenario

6.3 Context-Enhanced Implementation of Scenario

For the context-enhancement of the service calls, we now consider the simple case of identifying the contact information (location) of the dealer and propagating it to the field workers. Figure 7 depicts the implementation.

Before the implicit context enhancement mechanism can be used, the sending handler (*ContextAppender*) and the context server have to be set up. For the context-enhanced invocations by the manufacturer, the handler configuration file of the manufacturer *B* contains entries for the field worker service operations *assignIssue()*. Attached to this entry are the value of a scenario identifier *SCID* and the XPath expression to determine the *I-ID* parameter of the SOAP message that is to be sent to *C*'s and D's web services. The context server of the manufacturer is configured with a rule set that returns the needed context when the server is queried by providing an *I-ID* and the *SCID*.

The *ContextAppender* handler determines the target of the SOAP message and looks up the according *SCID*. Afterwards it executes all XPath expressions that were set up to gather the parameters for the context server invocation.

There are different ways to identify the targeted location context of the dealer. A rather simple and fully implicit approach consists of analyzing the IP address of the originating service end-point (i.e., the dealer's service): the context server of *B* compares the IP address of the message to a database table of IP addresses of all dealers and retrieves the correct contact information. The handler inserts the context server response into the header block of the SOAP message. This message is handed back to the service container. *B*'s service container now sends the SOAP message to

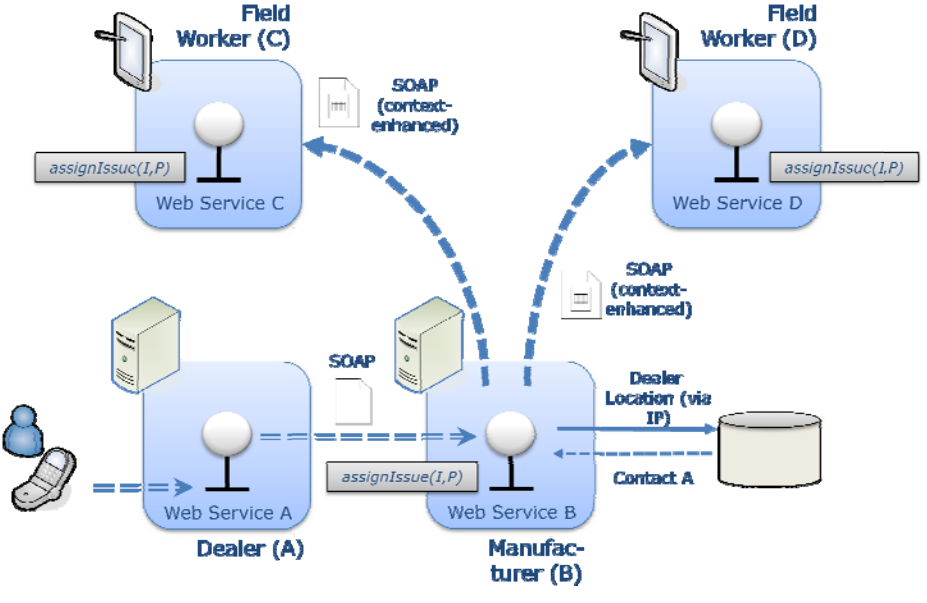

Fig. 7. Context-enhanced service calls in car maintenance scenario. The IP address of the dealer is used to determine the location of the dealer which is attached to the SOAP requests delivered to the repair shops.

the URIs of C and D. Here the *ContextExtractor* handlers preprocess the messages, before the actual operation is invoked. The *ContextExtractor* looks for the header block with the context data. In this case, the message was sent with the context enhancement mechanism, so the handler finds the according header block and extracts the context information.

To make the context accessible to the field worker service, it sets up the according data structure using the API of the application server. Afterwards it passes the control back to the application server.

On the field worker side, a *ContextExtractor* in the according application servers makes this additional context information available to the invoked services. The service of the field worker can then use the location context of the dealership to decide to either accept or reject the job.

6.4 Discussion

We demonstrated the use of context-enhanced service calls in a field force automation scenario. By nature, such scenarios are rather location-sensitive. The power of the approach presented here stems from the fact that the use of other types of context can be easily supplied, e.g.

- additional information concerning the issue to be fixed (beyond the problem class ID) like for instance a problem description could be delivered as context;
- for smaller businesses like field workers, business data on the status of the client (e.g., solvency) could influence the result of the *assignIssue()* method;
- information on the status of the field workers (e.g., work load) constitutes context information useful for the manufacturer during dispatching.

The advantage is that this information can be delivered without changing the core logic of the business process and the involved services.

7 Conclusions

Context information is to be perceived as an essential part of mobile processes. We have described a mechanism which implicitly enhances service calls with context information. This means that the context can be applied and extracted in a way which is transparent for the involved services. The focus is on automatic, customized and transparent provisioning of context and not on how the context is stored or acquired.

The proposed mechanism addresses particularly web services which nowadays are used in all sorts of business processes. While this is evident for services being delivered by stationary hosts, mobile hosts featured in mobile business processes like for instance field force automation scenarios become more appealing. Web service containers offer elegant and powerful mechanisms to deploy the according handlers. Future work will focus on generalising the approach to low foot-print containers running on more limited mobile devices.

References

1. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a Better Understanding of Context and Context-Awareness. In: Gellersen, H.-W. (ed.) HUC 1999. LNCS, vol. 1707, pp. 304–307. Springer, Heidelberg (1999)
2. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context aware systems. IJAHUC 2(4), 263–277 (2007)
3. Chen, H.: An ontology for context-aware pervasive computing environments. The knowledge engineering review 18(3),197–208 (2003)
4. Dustdar, S., Schreiner, W.: A survey on web services composition. IJWGS 1(1), 1–30 (2005)
5. Espinoza, F., Persson, P., Sandin, A., Nyström, H., Cacciatore, E., Bylund, M.: GeoNotes: social and navigational aspects of location-based information systems. In: Proceedings of the 3rd International Conference on Ubiquitous Computing, Atlanta, Georgia, USA, pp. 2–17
6. Keidl, M., Kemper, A.: Towards context-aware adaptable web services. In: The Thirteenth International World Wide Web Conference. Alternate track papers & posters, pp. 55–65. Association for Computing Machinery, New York (2004)
7. Kerer, C., Schahram, D., Jazayeri, M., Szego, A., Gomes, D., Caja, J.A.B.: Presence-Aware Infrastructure using Web services and RFID technologies. In: Proceedings of the 2nd European Workshop on Object Orientation and Web Services, Oslo, Norway
8. Papazoglou, M.P.: Service-Oriented Computing: Concepts, Characteristics and Directions. In: Catarci, T. (ed.) Proceedings of the 4th International Conference On Web Information Systems Engineering, pp. 3–12. IEEE Computer Society, Los Alamitos (2003)
9. Pauty, J., Preuveneers, D., Rigole, P., Berbers, Y.: Research Challenges in Mobile and Context-Aware Service Development (2006)
10. Turner, M., Budgen, D., Brereton, P.: Turning Software into a Service. Computer 36(10), 38–44 (2003)
11. Ryan, N., Pascoe, J., Morse, D.: Enhanced Reality Fieldwork: the Context Aware Archaeological Assistant. In: Bar International Series, vol. 750, pp. 269–274 (1999)
12. W3C. (27.04.2007). SOAP Version 1.2 Part 0: Primer (2nd edn.), `http://www.w3.org/TR/soap12-part0/` (retrieved September 18, 2007)
13. Eikerling, H.-J., Benesch, M., Berger, F.: Integrating Analysis of User / Asset Spatiotemporal Relationships for Mobile Field Processes (Demo). In: 4th International Conference Networked Sensing Systems. Braunschweig, Germany (2007)
14. Fazal-Baqaie, M.: Design and Implementation of a Handler Mechanism for Context-Enhanced Service Calls, B.Sc. Thesis, Paderborn University (2008)