Mobile Web Services based Middleware for Context-Aware Applications

Guido Gehlen and Georgios Mavromatis

RWTH Aachen University, Chair of Communication Networks
Kopernikusstr. 16, 52074 Aachen, Germany
guge@comnets.rwth-aachen.de

Abstract: **Mobile devices, communication systems and applications increasingly become more complex. In order to manage mobile distributed applications a middleware is essential which simplifies and speeds up the application development. A mobile environment with its heterogeneity and mobility has different requirements compared to a fixed environment. Thus, new concepts and mechanisms for a middleware have to be developed to cope with the challenge of these heterogeneous distributed systems.**

In this article, we introduce a Web Service based middleware for mobile devices, adapted to a mobile environment with special focus on context-awareness. The context-awareness will be realized by a rule based data monitoring in order to minimize the communication frequency over the mobile links and, thus, to minimize the runtime costs of the mobile application. Within this middleware mobile context providers are able to publish their context-data. This data can be remotely monitored by subscribing to a service of the context-provider and receiving notifications from the context provider.

The Web Service based mobile middleware will be described in general, special focus is given to the rule based data monitoring for context-aware applications. In addition a discussion of the expected application runtime costs regarding the communication is given.

1. Introduction

Mobile systems, composed of mobile devices connected by mobile communication systems, increasingly gain in importance. Wireless bandwidth is continuously increasing[13] as well as the processing power of mobile devices and their capabilities. Today, mobile and handheld devices are often more powerful than personal computers of the ’90s. They provide features like digital photography, speech recognition, personal information management (PIM), and gaming.

The future challenge is to fluently integrate mobile systems into distributed applications. But traditional methods to build a distributed system with mobile devices are not sufficient, since the mobile environment is more heterogeneous and more dynamic than a fixed distributed system. Thus, a middleware, which is the glue in distributed systems, differs in the case of a mobile system compared to a fixed system. For example, a conservative middleware for fixed systems should hide all context information of the distributed environment. In a mobile environment this context information is useful to enable novel and richer applications [11, 3]. In addition, the restrictions of the mobile communication system has to be taken into account to build high-performance and cost reduced mobile applications.

To cope with the requirements of a mobile middleware, the following design concepts and technologies are used. The Service Oriented Architecture (SOA) is a design model for distributed applications simplifying the development of heterogeneous distributed applications. One realization of the SOA are Web Services [1] which are used in our mobile middleware. The big advantage of the Web Service technologies are that they are platform independent and based on standard Internet protocols. Devices supporting a mobile Web Service based middleware have access to a rich number of existing Web Services in the Internet (e.g. see the Web Service database at `www.xmethods.net`). A compact introduction to the SOA and Web Services is given in the following section 2..

The mobile communication link is still the bottleneck of the whole distributed application/system and induces most of the runtime costs. Thus, the reduction of the communication amount is the main optimization criteria for this middleware. Furthermore, protocols and the bindings to the middleware are optimized to the mobile conditions.

The essential use case of context-aware applications will be discussed with the auxiliary condition of minimizing the communication efforts. The realization uses a rule based data monitoring which is smoothly integrated in the SOA. This realization will be the main focus of the paper, see section 3.3.. But first of all the SOA, Web Service technologies, and the overall mobile middleware structure will be introduced.

2. Services Oriented Architecture

Object oriented software architecture is hierarchically structured to build complex and reusable software. On the lowest level, functionalities are encapsulated in an object. A set of interacting software objects is collected into a component. The Service Oriented Architecture (SOA) [1] consequently extends this hierarchical structure to distributed systems. The interaction of services does not take places in one application on one device, but services reside on different heterogeneous systems and collaborate over communication systems. Services have the following characteristics [8]:

- Services are self contained and modular
- Services are discoverable and dynamically bound
- Services stress interoperability
- Services are loosely coupled, reduction of artificial dependencies to a minimum
- Services have a network-addressable interface
- Services have coarse-grained interfaces in comparison to finer-grained interfaces of software components and objects

- Services are composable

These characteristics should make the distributed system as simple as possible, but no simpler. In other words, the SOA is an architectural style to achieve loose coupling among interacting software agents and to minimize their artificial dependencies.

The SOA defines three roles, a Service-Requestor (R), Service-Provider (P), and a Service-Broker (B). A software agent which interacts with other software agents can play one or more roles, see figure 2. They communicate in the way as depicted in figure 1.

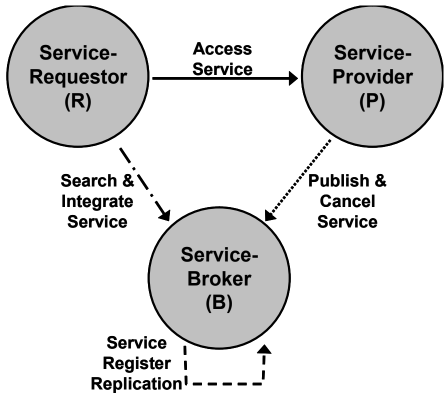

Figure 1: The Service Oriented Architecture (SOA) roles and relations

Providers publish their services to a service registry (service-broker). More than one service-broker within the environment have to replicate their service registries (dashed arrow). Requesters use the Broker to search for services and integrate them by accessing the service description (dashed-dotted arrow). This description includes all information needed to access the service and is used to generate a service-proxy object. The service-proxy represents the remote service, i.e. all published remote service methods are methods of the local proxy object. This architecture bridges the native messaging inside the client environment to the platform independent messaging in the SOA environment, see also section 3.2. and figure 5. The process of proxy object generation maps the platform independent description into a real software object for the client's system environment. In our implementation a WSDL to Java2 Micro Edition (J2ME) generator is used to create Java proxy classes for a J2ME environment, see section 3.2..

To achieve high interoperability, all SOA entities have to use a common language for service description, messaging and service registration. The Extensible Markup Language (XML) is such an appropriate common language. On the basis of XML the World Wide Web Consortium (W3C) has specified a middleware framework, called Web Services, following the SOA. For messaging the Simple Object Access Protocol (SOAP) [12] is used. It is based on standard Internet protocols like HTTP or other arbitrary protocols like the Wireless Application Protocol (WAP) [7] or the Block Extensible Exchange Protocol (BEEP) [14]. The SOAP envelope is structured in XML. Interfaces are described in a XML subset, the so called Web Service Description Language (WSDL) [6]. This description includes all the information needed in order to invoke service methods from other nodes.

In figure 2 a heterogeneous mobile environment is depicted, characterized by nodes with a big range of capabilities and computation power. Some devices are only consumer (requestor) of services, like e.g. remote controls, terminals, tablet PC's. Other devices are only service provider, like e.g. light controller, media server. Some devices are both service provider and requestor, thus, they publish their own services and consume remote services.

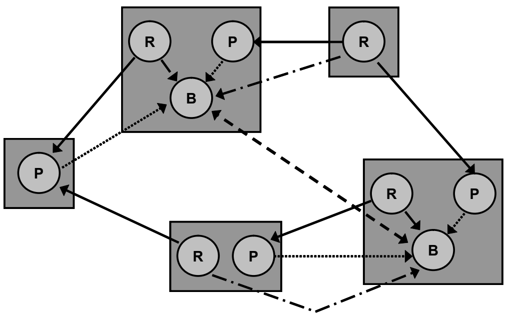

Figure 2: Ad Hoc SOA Realization for mobile heterogeneous environments

We have to differentiate between two cases, networks with one or more broker and networks without any broker. Networks without any broker have to use flooding or index routing algorithms to publish or discover services, as e.g. in Universal Plug and Play (UPnP) [9]. More than one broker within the environment must synchronize the service repository to ensure consistency (dashed arrows). A possible solution to handle multiple redundant brokers is to use on the middleware level a virtual broker. The middleware requests this virtual broker regardless of how many brokers are available. If there is more than one broker, the virtual broker request has to be routed to the best available broker node.

3. Middleware Architecture

Within the SOA, described in the previous section, services play the main part. Each device can publish services which will be provided by the device's application. Each published service (dark-colored squares in figure 4 and 3) will be placed in the middleware and will be used as connection between the application and the middleware. Other devices can use these services by integrating a service-proxy (light-colored squares in figure 4 and 3) inside their middleware. The service-proxy and service pairs, thus, form a loosely coupled distributed application, see figure 3. That means that the applications on each device are independent of each other. The coupling will be done on demand and on runtime.

The service-proxies and the services will map the device's platform dependent method calls and data objects into platform independent SOAP-calls and vis versa. This architecture bridges the native messaging inside the device environment to the platform independent messaging in the SOA environment, see subsection 3.2..

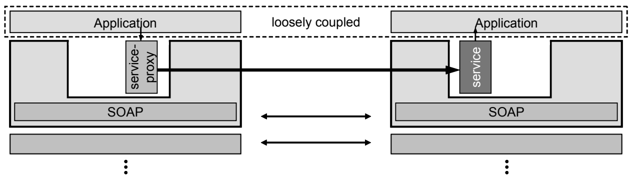

Figure 3: Coupling independent applications over the network

Our Web Service based middleware architecture is shown in 4. The middleware is bordered upwards by the application and downwards by communication layers. The middleware is capable of coupling either to a session layer protocol, like HTTP, BEEP, or WSP, or to a transport layer protocol, like TCP or UDP.

The middleware itself is structured in a protocol part and a service part. The protocol part is based on SOAP, including the bindings to the underlying protocols and security mechanisms. The service part of the middleware is once again divided into a static part within, the U-shaped block, and a dynamic part. The static part acts for base middleware functions, like Service Publishing/Discovery and Object Monitoring and Eventing. The Monitor-Service, Notification Service and the Rule Engine are described in section 3.3.. The dynamic part depends on the application and adapts on compile or runtime.

In addition, all elements in the service part of the middleware can be distinguished in services and service proxies. As mentioned before, the service-proxy is a representative of a remote service and offers the application an interface to this remote service.

The realization of the middleware has been done for Java enabled mobile devices, compliant to the J2ME standardization. The SOAP part is based on the Open Source libraries kSOAP and kXML and has been extended by additional SOAP-Bindings to alternative underlying protocols and by server capabilities, see 3.1..

The following subsections will present each of this middleware parts in detail. First, SOAP and our extensions in respect of a mobile environment will be explained. The next two subsections describe the static middleware parts, the Publishing/Discovery and the Rule based data monitor services. The last subsection will introduce the use of these middleware functionalities to build context-aware applications.

3.1. SOAP and Bindings

Within the Web Service framework the services and service-proxies communicate via the Simple Object Access Protocol (SOAP) [12]. It is based upon messages encoded in XML, called SOAP-Envelopes, transmitted by arbitrary transport protocols. The SOAP-Envelope is divided in an optional SOAP-Header and the mandatory SOAP-Body. The body contains the message to be transmitted, the header contains additional information regarding this message, as e.g. message ID's for the session management or security related information.

As mentioned, SOAP can be coupled to an arbitrary protocol by using a protocol specific SOAP-Binding. The most common protocol used by SOAP is Hypertext Transport Protocol (HTTP), since it is mostly used in the Internet and easily allows Remote Procedure Calls (RPCs). But HTTP on top of TCP has a bad performance in mobile communication systems [7, 10]. Thus, this middleware provides a set of alternative SOAP-Bindings, like a binding to the Wireless Session Protocol (WSP), User Datagram Protocol (UDP) or BEEP. The WSP binding for SOAP has the advantage, that the protocols are adapted to the mobile communication system [7]. The WSP header is more lightweight compared to HTTP. In addition, the XML content can be encoded binary [16]. Existing Internet Web Services which are bound to HTTP can be accessed via WAP using a WAP gateway.

The middleware realization is based on kSOAP, which solely provides a HTTP client binding. Thus, kSOAP enables by default only Web Service access, but no Web Service provision. We extend the middleware by a lightweight Web- and SOAP-Server. By using this lightweight server, the middleware can publish arbitrary service methods and make these services available via HTTP. In addition, static web content, like HTML pages could be accessed on the mobile device. The HTTP-Server binding for SOAP runs on MIDP1.0/2.0, Personal Java, and on any arbitrary standard Java platform. Service classes which should be published by the server have to implement one interface and have to declare their exposed methods and objects to the server.

An additional extension which is not included in kSOAP is the SOAP-Security. The package provides functions to encrypt or sign SOAP messages or only parts of these messages. SOAP-Security is a building block of the Web Service Security specification [15] and is based on XML-Encryption and XML-Signature. The security implementation will be presented in a further paper.

3.2. Service Publishing, Discovery and Integration

We have to differentiate two kind of services. Unique services and well defined plural services. The first kind of services are specialized services provided by one unique service provider, like e.g. the *amazon* web or *google* service. The second ones are specified services, which can be implemented by any arbitrary service provider, like e.g. the Domain Name Service (DNS) or UPnP services.

The unique services and service-proxies are published and integrated on compile time. It makes in general no sense to do these tasks at runtime, since the application on top of the services and service-proxies is coupled with them. Thus, the application developer has to do the publishing and integration process for unique services only, see figure 5. From the service source code, written in an arbitrary programming language, a WSDL description is generated and published. A second application developer, which wants to use this service, automatically generates source code for his target platform from the WSDL description.

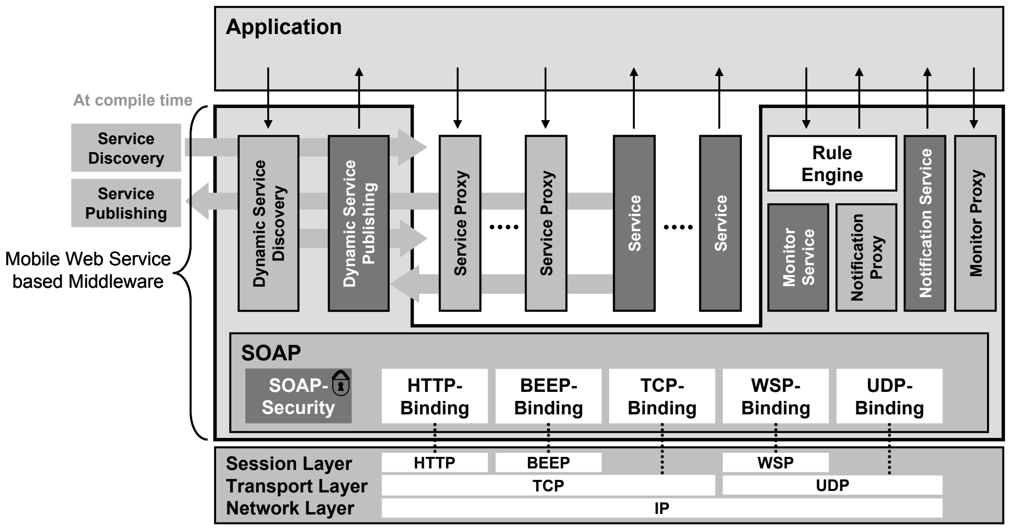

Figure 4: Mobile Web Service based Middleware Architecture

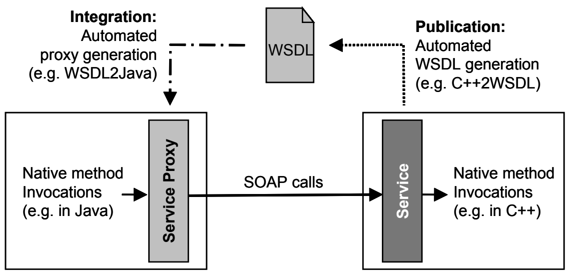
Integration: Automated proxy generation (e.g. WSDL2Java)
WSDL
Publication: Automated WSDL generation (e.g. C++2WSDL)
Native method Invocations (e.g. in Java)
Service Proxy
SOAP calls
Service
Native method Invocations (e.g. in C++)

Figure 5: Publication and Integration of Web Services using the WSDL description

In contrast, well defined services can be published and integrated on runtime, assumed that the middleware supports these services and that the application can use or provide them. Upon entering a new service environment with a device, its services are published and remote services, which are of interest, will be integrated in the middleware. This dynamic service publishing and discovery is e.g. applied in UPnP. Currently, it will be discussed to align UPnP version 2 with the web service technologies. Thus, our middleware will support in future both, Web Services and UPnP services.

3.3. Rule Based Data Monitoring

Context-aware applications are often distributed, since the context information provider are distributed themselves. We will consider without loss of generality one mobile context provider device and one back-end system with the main application logic as depicted in figure 6. The context provider will be generalized as a set of data, updated by persons or sensors. Thus, not only context-aware applications are supported by this application class, but also applications which provide data sets on the the mobile device in general.

There are three different high level communication relations between the mobile device and the back-end system, see figure 6 a). The first possibility is to periodically forward the relevant data to the back-end system. This is simple to realize, but data will be transmitted even if the data is not of interest for the back-end application.

The second possibility is to access the data on demand from the back-end system, see figure 6 b). The advantage is that the back-end system can decide at which time the mobile data is needed, but it can only be realized if the mobile device has server capabilities and is addressable from the Internet. As mentioned in section 3.1., our middleware supports HTTP server capabilities and consequently supports also this second possibility. However, this solution is only advantageous, if the back-end application knows the point of time the mobile data is needed. In case that the decision, whether the data is relevant or not, depends on the mobile data itself, this solution as well as the first one are not sufficient.

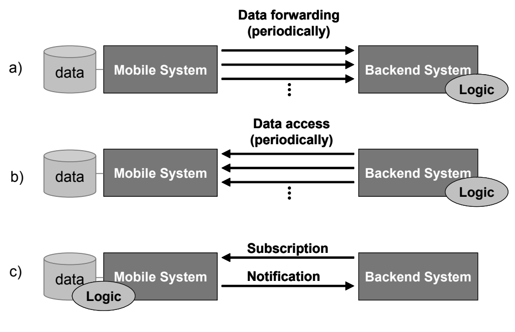

Figure 6: High level communication possibilities for context-awareness:
a) Periodically forwarding of the context data
b) Periodically access of the context data
c) Rule based object monitoring

The third possibility, in figure 6 c), breaks this disadvantage by shifting the logic. Thus, the mobile device can filter the data before forwarding it to the back-end system (notification). Before, a self-defined rule composed by the backend-system has to set up the data filter. This is done by submitting a subscription to the mobile device containing this rule.

The high level communications are divided into two phases separated in time, see 7. In the first phase the back-end system will subscribe with a rule to the mobile device. The rule is related to the data published by the mobile device and defines which data changes cause a notification of the back-end system. The second phase is the notification itself. A notification, specified in the rule, is send from the mobile device to the back-end if the rule is fulfilled.

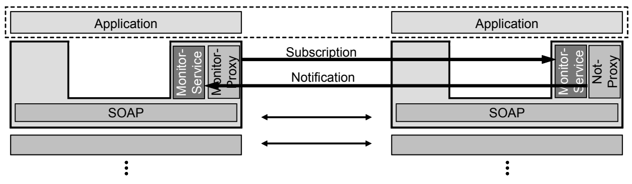

Figure 7: Subscription to the rule based monitoring and notification

To realize this concept, a rule parser, a rule evaluator, the subscribe and un-subscribe methods, and a notifier are necessary. The subscribe and un-subscribe methods will add or delete a new rule on the mobile device application. The rule parser will transform the serialized rule, in our case a specialization of the XML, into a processable data object, which can be evaluated.

The implementation of this functionality is based on the Service Oriented Architecture (SOA). The mobile node offers and publishes a rule based object monitor service with two public methods *Subscribe(Rule)* and *Un-subscribe(RuleID)*. The subscription method contains the Rule and starts at the mobile node the evaluation of this rule. The un-subscription method stops the evaluation of the rule with the corresponding *RuleID*, see figure 8. The *RuleID* is assigned by the mobile node after verifying the rule and is returned back to the backend system.

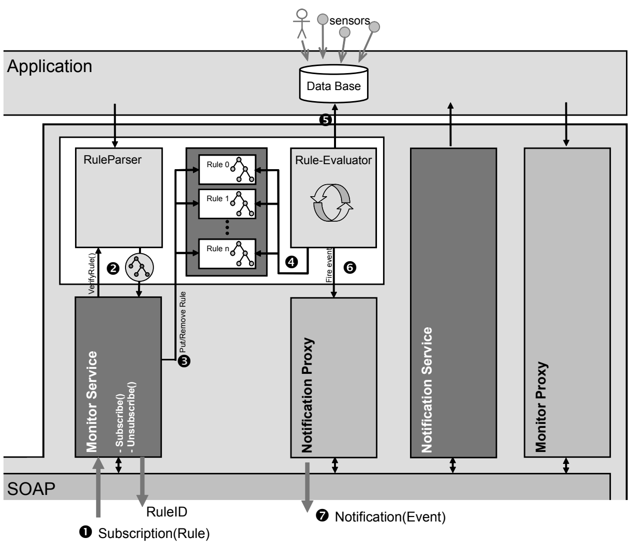

Figure 8: Architecture of the rule based monitoring middleware part

After the subscription, see (1) in figure 8, each incoming rule is parsed and verified, (2). If the rule syntax is valid, the subscribe method returns an acknowledgment message to the back-end containing an unique Rule ID, else an error message. A valid rule will be saved into an list of rules, (3). In a parallel process this list is sequently evaluated by a *RuleEvaluator*, (4). Each time the rule applies, a notification service of the back-end system will be invoked with the event as parameter, (7), by invoking the *Notification Proxy*, (6).

The rules are defined in an platform independent and XML based rule language, named RuleML (see subsection 3.3.1.). The RuleParser generates in the first step a Document Object Model (DOM) of the XML rule, as illustrated in figure 9. This DOM tree can be recursively processed, thus, arbitrary hierarchically structured rules can be evaluated. The RuleEvaluator accesses each DOM rule tree, beginning with the root element. Within the condition of the rule, two different nodes occur. Relation nodes (AND, OR) with two or more child nodes and the leave nodes without child nodes, containing the atom conditions ($x > y$, ...)

3.3.1. The Rule Meta Language (RuleML)

RuleML is a XML based meta language, which defines rules in a platform independent syntax. The RuleML hierarchy of general rules branches into the two direct categories of reaction rules and transformation rules. On the next level, transformation rules are specialized to the subcategory of derivation rules. Then, derivation rules have further subcategories, namely facts and queries. Finally, queries are specialized to integrity constraints [2].

Within this article we solely consider derivation rules. They state a sort of if-then-statements building a filter for the logic-component on the mobile device.

The root element of each rule is the `<rulebase>`-element. It can contain attributes such as Namespaces. Among others, child elements like `<Query>`, `<Fact>` or `<Imp>` are possible. As mentioned before, the main focus in this work lies on implication rules (`<Imp>`) and their structure in order to come up with the demands. The `<Imp>` element consists of a `<head>` and a `<body>` element. Inside the `<head>` element the implication is stated e.g. send an event to this URL. The condition is nested in the `<body>` element, which is processed recursively. Many conditions are possible, where each condition is declared in an `<Atom>` element. These are coupled by `<And>` or `<Or>` relations, resulting in a complex tree structure. See in the following an example rule description in Rule ML and the corresponding document object model tree in figure 9.

```
<Imp>
  <head>
     <Atom>
      <opr><Rel>send</Rel></opr>
      <Ind>Event</Ind>
     </Atom>
     <Atom>
      <opr><Rel>URL</Rel></opr>
      <Ind>137.226.4.133:8080</Ind>
     </Atom>
  </head>
  <body>
    <And>
```

```
      <Atom>
        <opr><Rel>lower</Rel></opr>
        <Var>Lat</Var>
        <Ind>5040.0000</Ind>
      </Atom>
         ......
      <Atom>
        <opr><Rel>greater</Rel></opr>
        <Var>Lon</Var>
        <Ind>10020.0000</Ind>
      </Atom>
    </And>
  </body>
</Imp>
```

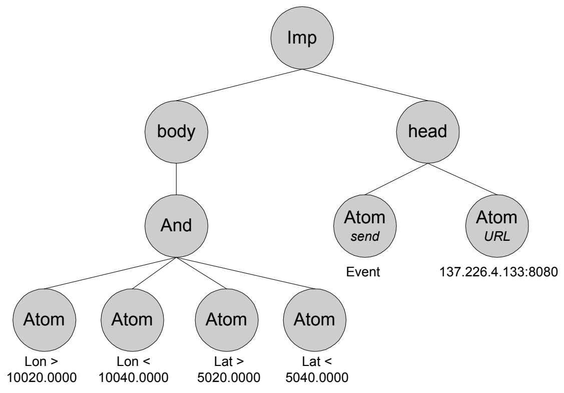

Figure 9: RuleML Document Object Model tree

3.4. Application costs and reaction time

In this section the performance and the costs of applications using this Rule Based Monitoring middleware instead of periodical data polling is estimated.

We are considering a mobile cellular and packed switched network, like the General Packet Radio Service (GPRS) or the Universal Mobile Telecommunications System (UMTS) with data volume oriented charging. Thus, the costs (C) are proportional to the transmitted data volume (V). In the case that the rule evaluation logic is running on the back-end side, the poll frequency is f_{poll}, the overall processing time is τ_{proc}, and the network delay is τ_{net}, the maximum delay of the application is

$$\tau_{poll,max} = 1/f_{poll} + \tau_{net} + \tau_{proc} \qquad (1)$$

The runtime costs are proportional to f_{poll} and independent of the probability density that a rule is true (p_{event}). Thus, the costs C_{poll} can be calculated from the data volume V_{poll}, the communication costs per data volume r_V, and the application running time T_{use} to

$$C_{poll} = V_{poll} \cdot r_V \cdot f_{poll} \cdot T_{use} \qquad (2)$$

In the case that the logic is running at the mobile side, as in the Rule based Data Monitoring, the application delay depends only on the network delay (τ_{net}), the processing time of each rule, and the number of rules running on the mobile node

$$\tau_{rule,max} = N_{rule} \cdot \tau_{proc} + \tau_{net} \qquad (3)$$

.

The costs are now depending on the event probability density p_{event} referring to time ($[p_{event}] = 1/s$).

$$C_{rule} = V_{Not} \cdot r_V \cdot p_{event} \cdot T_{use} + \left\lceil \frac{T_{use}}{T_{sub}} \right\rceil \cdot V_{Sub} \cdot r_V \quad (4)$$

V_{not} and V_{sub} are the data volumes of one notification resp. subscription message and $n_{sub} = \left\lceil \frac{T_{use}}{T_{sub}} \right\rceil$ is the number of subscriptions within the runtime T_{use}.

Comparing the application delay of option a), b) with option c) by disregarding the processing time and network delay (τ_{proc}, $\tau_{net} \rightarrow 0$), the application delay of option c) is 0 and $\tau_{poll} = 1/f_{poll}$. Thus, the application delay depends only on the polling frequency.

The cost ratio of the runtime application costs in case c) compared to a), b) is

$$\frac{C_{rule}}{C_{poll}} = \frac{V_{not}}{V_{poll}} \cdot \frac{p_{event}}{f_{poll}} + \frac{n_{sub} V_{sub}}{V_{poll} f_{poll} T_{use}} \qquad (5)$$

Assuming that $V_{poll} \approx V_{not}$, the cost ratio is

$$\frac{C_{rule}}{C_{poll}} \approx \frac{p_{event}}{f_{poll}} + \frac{n_{sub} V_{sub}}{V_{poll} f_{poll} T_{use}} \qquad (6)$$

The main and time independent portion of the cost ratio is the factor p_{event}/f_{poll}. Thus, the decision which solution should be implemented in a concrete application is dependent on the application demands concerning the application delay ($T_{react,poll} \propto 1/f_{poll}$) and the estimated probability density of an event. For example, if an application demands a reaction time of 1 minute and the estimated event probability density is 1 per hour, the Rule based Monitoring solution is approximately by the factor 60 cheaper.

3.5. Context-Aware Applications

Context awareness enhances existing applications and even enables a new class of applications in pervasive computing. These applications provide clients with a customized and personalized behavior to better suit the needs of the user and their tasks [5]. Especially applications running on mobile devices have to deal with dynamically changing execution environments. Thus, the ability to exploit efficiently context information is challenging in consideration of the limited resources of mobile devices and mobile communication systems.

Context information is everything that could change the behavior of an application, like the location, user preferences, environment, or properties of connectivity [4]. In our architecture, see figure 8 the context is modelled as a data base, which will be continuously updated by sensors or persons. The middleware provide the base functionalities to remotely access these context data or to subscribe to specific context changes. The application developers are simply able to integrate these context services in their own application in every arbitrary programming language on every device.

The work has been motivated by logistics and vehicular application, where the location is the most essential context property, but also health-care applications are qualified to build up on this middleware, where in addition to the user's location also the user's mental health is important. For example, the patient's mobile device collects medical data, like heart rate or blood sugar level. A health-care center application can subscribe to his mobile with rules which are adjusted to the patient's clinical picture. Thus, the health-care center will be notified, if a medical value or a combination of values are critical.

4. Conclusion

The paper has presented a mobile middleware based on Web Services with special focus on conext-awareness. The overall architecture and processes have been described as well as the rule based data monitoring in detail. The middleware provides basic functionalities to integrate and provide Web Services and to reduce the communication frequency by applying a subscription mechanism.

In the future, context-aware mobile applications increasingly gain in importance. Then, such a middleware is essential to simply build distributed applications in a high heterogeneous mobile environment. It has been proved in recent years, that a middleware based on Web Services become accepted also for mobile devices, even though the technologies seem to waste quite a lot of network and device resources. But the plainness and flexibility of the technologies enable an easy adaptation of the middleware to the mobile environment. Application developer with existing Web Service knowledge will be able to migrate easily to a mobile development environment.

Acknowledgment

The work presented has been motivated by a co-operation with Ericsson Research Germany within the German BMBF project INVENT-VMTL[1]. The authors would like to thank Prof. B. Walke and L. Pham of ComNets for their help and support, as well as T. Dinsing and M. Gerdes of Ericsson Research.

[1] Information available at http://www.invent-online.de

REFERENCES

[1] Web service architecture. Published on the internet. Available at `http://www.w3c.org`, 2004. 1., 2.

[2] H. Boley, S. Tabet, and G. Wagner. Design rationale of RuleML: A markup language for semantic web rules, 2001. 3.3.1.

[3] L. Capra, W. Emmerich, and C. Mascolo. CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, October 2003. 1.

[4] Kimmo Raatikainen et al. Service adaptability. Published: `http://www.wireless-world-research.org`, December 2003. Whitepaper. 3.5.

[5] Olaf Droegehorn et al. Service personalisation. Published: `http://www.wireless-world-research.org`, December 2003. Whitepaper. 3.5.

[6] Roberto Chinnici et al. Web services description language (wsdl) version 1.2. Published on the internet. Available at URL `http://www.w3.org/TR/wsdl12`, June 2003. 2.

[7] Guido Gehlen and Ralf Bergs. Performance of mobile web service access using the wireless application protocol (wap). In *Proceedings of World Wireless Congress 2004*, pages 427–432, San Francisco, USA, 05 2004. University Aachen, Communication Networks. 2., 3.1.

[8] J. Mc Govern, S. Tyagi, M. Stevens, and S. Mathew. *Java Web Service Architecture*. Morgan Kaufmann, 2003. 2.

[9] Michael Jeronimo and Jack Weast. Upnp design by example. Intel Press, Intel Corporation, USA, ISBN 0-9717861-1-9, April 2003. 2.

[10] Jaakko Kangasharju, Sasu Tarkoma, and Kimmo Raatikainen. Comparing SOAP performance for various encodings, protocols, and connections. In Marco Conti, Silvia Giordano, Enrico Gregori, and Stephan Olariu, editors, *Personal Wireless Communications*, volume 2775 of *Lecture Notes in Computer Science*, pages 397–406, Venice, Italy, September 2003. Springer-Verlag. 3.1.

[11] Cecilia Mascolo, Licia Capra, and Wolfgang Emmerich. Mobile computing middleware. pages 20–58, 2002. 1.

[12] Nilo Mitra. Soap version 1.2 part 0: Primer. Published on the internet. Available at URL `http://www.w3.org/TR/soap12-part0/`, June 2003. 2., 3.1.

[13] Werner Mohr. Trends in mobile communications towards systems beyond imt-2000. ITU Seminar, Ottawa, 2002. 1.

[14] Marshall Rose. Rfc-3081: Mapping the beep core onto tcp. Published on the internet. Available at `http://www.ietf.org/rfc/rfc3081.txt`, Mar 2001. 2.

[15] Bob Atkinson und Giovani Della-Libera und Satoshi Hada und Maryann Hondo und Chris Kaler und Johannes Klein und Brian LaMacchia und Paul Leach und John Manferdelli Hiroshi Manuyama und Anthony Nadalin und Nataraj Nagaratnam und Hemma Prafullchandra und John Shewchuk und Dan Simon. Web services security (ws-security). Published: `http://msdn.microsoft.com/library/`, April 2002. 3.1.

[16] WAPForum. Binary xml content format specification. version 1.3, wap-192-wbxml-20010725-a. Published on the internet. Available at URL `http://www.wapforum.org`, July 2001. 3.1.