# AN ARCHITECTURAL APPROACH TO SERVICE ACCESS IN WIRELESS ADHOC NETWORKS [*]

Sebnem Baydere, Mesut Ali Ergin
Department of Computer Engineering
Yeditepe University
Kayisdagi Campus, 81120, Istanbul, Turkey
{sbaydere, ergin}@ics.yeditepe.edu.tr

**ABSTRACT**
This paper introduces an architectural approach to service access in an environment where no established infrastucture at any level exists. Mobile clients invoke services by their descriptive names and a temporary multihop network is dynamically formed between the client and the host offering a service instance. The proposed protocol stack supports dynamically optimized routing to named services. The elements of the architecture for adapting clients and services to a highly varying topology are explained and the algorithms are given in pseudo code like notation using a service model specified in XML.

**KEY WORDS**
Service Access, Adhoc Networks, Routing, XML

## 1 Introduction

In self organizing mobile adhoc networks, nodes with wireless interfaces form a temporary network without the aid of any established infrastructure. The network‘s topology may change rapidly and unpredictably. Mobile nodes can move independently or as group towards a target such as fire, flood, search rescue or a battlefield. Several mobility models are proposed in the literature [1]. There is an increasing trend in offering new services to network users besides classical services, such as those offered by printers, scanners and so on. Following this trend it becomes increasingly important to support mobile users so that they can discover and bind to services “on the fly” where no static configuration at any level or addressing scheme other than wireless link address available. The packet routing problem in adhoc networks has been extensively researched [2-4]. Support for describing, locating and gaining access to services for mobile entities such as sending a job to “a reachable color printer” is also emerging [5,6]. Additionally, architectural approaches are needed to address problems related to the construction of multihop adhoc networks; such as, how they are formed, how long they live, how they are managed throughout their lifetime, what are the characteristics of the applications that can be accommodated and so on.

Accessing a network resource is typically realized as a client-server application, which establishes a logical connection between two networked nodes. We propose an architecture in which each logical connection is mapped to a unique session treated as a temporary network and managed for the duration of the service access. In our target environment, the protocols are designed to accommodate random walk mobility [7] for both clients and servers. In this model, mobile nodes move randomly with a given speed and direction of motion. The speed and direction of the motion in a new time unit is not related to the previous epoch. This model may generate sudden stopping or sharp returns which may not be suitable for some applications. In this paper, the component interaction and the algorithms for non-interactive services are given.

The rest of the paper is organized as follows; Section 2 gives related work and motivation. Section 3 introduces the SeMA architecture, its components and the algorithms for service discovery and binding. Section 4 discusses route optimization and maintenance. Section 5 contains future work and some concluding remarks.

## 2 Related Work

Most related research activities concentrate on automatic service discovery and configuration when a mobile Internet user visits a foreign infrastructured network. The service discovery protocols; Jini [8], Service Location Protocol(SLP) [6], Universal Plug and Play(UPnP), Bluetooth Service Discovery Protocol(SDP) and Salutation [9] provide mechanisms to search for and choose the most appropriate service using an attribute based centralized directory service, complementing DNS. Directory services are maintained by one or more directory agents (DAs) periodically advertising themselves to the network. A mobile device dynamically obtains the directory agent address in the domain and gets the service access information from it. The Berkeley Service Discovery Service (SDS) [5] extends this concept with security and a fixed hierarchical structure for wide area operations. Some research activities concentrate on naming schemes used to describe services; Intentional Naming Scheme (INS) [10] is a simple language based on

[*]This work is partially supported by The Scientific and Technical Research Council of Turkey under grant 101E037/EEEAG-AY-41.

attributes and values for its names. It achieves expressiveness in the service definition and defines name resolution process and late binding in a name resolvers network.

All above mentioned protocols are useful to form an application level overlay network supporting discovery and late binding. The network and internet layer functions in the underlying infrastructured network are left to the existing protocols; DHCP, Mobile IP.

On the other hand, "networks formed on the fly" cannot not rely on an existing infrastructure for service discovery.

We propose a full protocol stack for adapting clients machine to a highly varying topology without any need for configuration changes in the machine. Each node establishes a local service directory by collecting information from its data link neighbors. No intermediary centralized look-up services are needed. The model consists of a set of mobiles acting either as client or server, and a set of valid services on which clients can make requests.

## 3 SeMA Architecture

SeMA architecture is a protocol stack operating at three interacting layers to accommodate access to general purpose network applications such as printing, file transfer and so on in a highly mobile environment. Neither an infrastuctured network underneath nor an overlay look-up network exist. Mobile nodes dynamically establish routes among themselves to form a multihop network when a service request is made. The protocol stack is composed of the following layers:

- **Data link layer:** A standard wireless data link such as IEEE 802.11
- **SeMA layer:** Protocols supporting service access. Three interacting software components are operating together in this layer. *Session Manager* (SM) provides a descriptive naming interface to the application; *Service Agent* (SA) deals with service tracking and delayed binding. *Routing Agent* (RA) sends packets over dynamically optimized routes. All SeMA packets are forwarded to the *Communication Agent* (CA) which passes them to the wireless link interface for frame delivery with the assumption that a SeMA packet can be transmitted in a single frame. No fragmentation is supported at this level.
- **Application layer:** Mobile client and server applications developed with SeMA API.

Two types of services are defined; non-interactive services, such as printing, require best effort delivery from the network. The application is unaware of the session maintained by the network layer for the sequenced delivery of network packets. Duplicate packet control and sequenced delivery is guaranteed without a re-transmission facility. Session Manager assigns a locally unique id to the session and simply fragments the application data into SeMA packets appending each one with a sequence number. A SeMA packet is uniquely identified by the combination of the following fields;

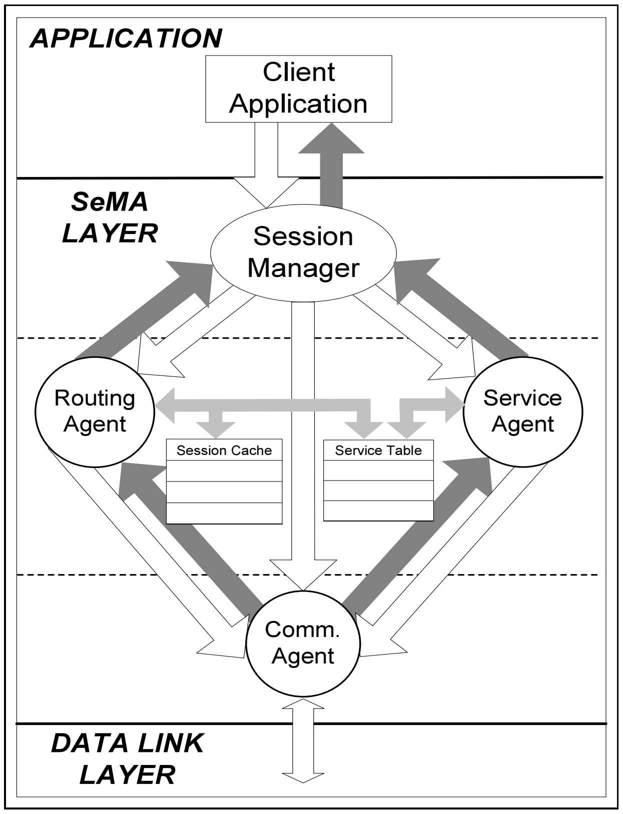


Figure 1. Components of SeMA Layers

**SeMA_ID**= `Client_WirelessLinkID + Session_ID + Seq#`

`Client_WirelesslinkID` is the address of the node where the session is originated. `Session_ID` is a locally unique ID given to the session by the originator and `Seq#` is the packet number flowing over this session.

Interactive services, such as file transfer, require reliable delivery of the packets. Heavy weight sessions with re-transmission facility will not be discussed here. This paper covers algorithms for the light weight sessions.

An adhoc network (*AN*) as defined below is a function of mapping from a set of mobile nodes to a locally unique id;

$AN : (M \rightarrow I)$

$\forall x, y \in M : (x <> y) \vee (\exists z \in M \mid (x <> z) \wedge (z <> y))$

$\forall m \in M : if (\exists (m, i) \in AN) \wedge ((m, j) \in AN)_{then}\ i = j$

where *M* is the set of all SeMA nodes, *I* is the set of identifiers and <> indicates nodes in the same data link transmission range. The network is initiated by the client node when the service is needed and it is terminated by the destination node after the service is used. Interacting components of the architecture is illustrated in Fig. 1.

```
<service name="printer">
        <keyword attribute="location">Engineering B.443</keyword>
        <keyword attribute="color">no</keyword>
        <keyword attribute="papersize">A4</keyword>
        <keyword attribute="papercount">81</keyword>
        <keyword attribute="postscript">yes</keyword>
        <keyword attribute="maxResolution">600*600</keyword>
</service>
```

Figure 2. A valid *printer service* instance

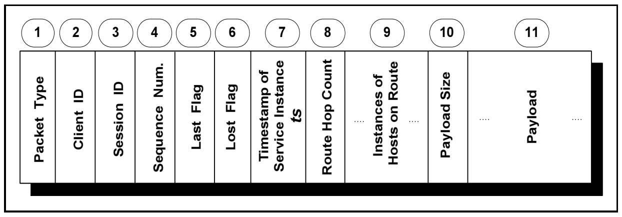


Figure 3. SeMA Packet Structure

## 3.1 Modeling Services

In the architecture, services are defined and represented as eXtensible Markup Language *(XML)* [11] instances that conform to an *XML Schema* [12], designed to specify structure and content of the instances. XML provides a flexible, easy to parse, structured text-only format to access data efficiently.

Upon deciding to offer a service, the provider creates a service object instance conforming to the *ServiceSchema*, in which a service is defined via its attribute-value pairs as suggested in SLP by IETF [6]. A well formed and valid XML instance of ServiceSchema holds all the necessary information for a host to correctly determine whether its need for a service can be satisfied by this instance. This service definition instance travels the network encapsulated in an appropriate broadcast message together with other information added by hosts on route and stored in service table for future references. A *service object* instance may be bound to more than one *host object* instance indicating the existence of replicated services in the network. A client who wishes to use the service, binds itself to an instance of it.

An example printer service definition is given in Fig.2. Attribute-value pairs of this printer service in the definition help the service agent to successfully select the appropriate service, currently available to the host in the need of the printer service. For the case given, service agent may try to find another printer instance to be returned to session manager, if application has requested a printer to print a colored document of a hundred pages.

## 3.2 Packet Structure

The packet structure is illustrated in Fig.3. and the description of the fields is given below:

1. **Packet Type (ptype):** Five packet types are defined. These are given in Table 1.

2. **ClientID:** Wireless Link ID of the node on which the session is initiated. Each packet transmitted in a session carries out the initiator node's ID.

3. **SessionID:** A locally unique ID generated by the initiator node. SessionID is zero in announcement and look-up packets.

4. **Sequence No:** Sequence number of the packet that is unique for this session.

5. **Last Flag:** End of data packets.

6. **Lost Flag:** Enforces flooding due to route loss.

7. **Timestamp (*ts*):** Announcement time of the named service instance. At each hop, routing agent checks its local service table to see if there is a fresher announcement entry for the same service or not. This information is used for dynamic route optimization.

8. **Hop Count:** The number of intermediate nodes the packet has passed through since it has originated.

9. **Route:** Source route of host instances.

10. **Payload Size:** Size of the data field.

Upon receiving a packet, one of the cooperating agents on the node, process it and take appropriate actions. SeMA_Announcement and SeMA_Lookup packets are processed by the service agent (SA). SeMA_Data and SeMA_Terminate packets are processed by the routing agent(RA). If the destination SeMA_ID matches the ID of the agent, the packet is passed to the session manager for further processing. If the packet has not reached its final destination yet, RA makes a routing decision based on the algorithm given in Section 4.

## 3.3 Service Announcement and Discovery

A set of valid named service instances that a client can ask for is defined. Each node maintains a local table of service announcements seen at the node‘s network interface. Information kept in the table for each service is given below and illustrated in Fig. 4.

| *Packet Type* | *Explanation* |
|---|---|
| SeMA_Data | Carrying user data |
| SeMA_Lookup | Broadcast packet for binding request |
| SeMA_Announcement | Broadcast service advertisement |
| SeMA_Lookup_Reply | Sent to the originator of the lookup |
| SeMA_Terminate | Terminate indication |

Table 1. Packet Types

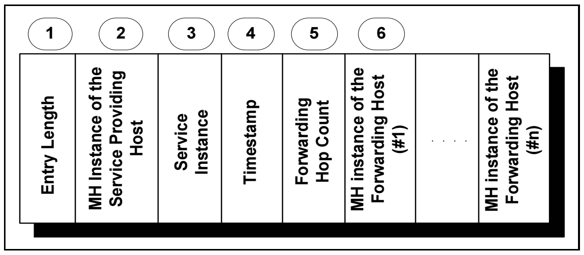
1. Entry length
2. Host XML instance of service providing node
3. Service XML instance
4. Timestamp of original announcement
5. Number of forwarding hosts
6. Host instances of the forwarding nodes.

Service table maps a service XML instance to a valid source route through which the announcement was received. There may be more than one entry for the same service instance as the announcement may arrive via different routes. Each node maintains a *session cache*; packets seen before, to detect duplicates. Duplicates are discarded. A Client establishes a light-weight session with the resource using the handle returned to its service request. Initial route for a session is obtained as a part of the distributed service discovery algorithm. Every node collects information from the network and make their decision using the locally available information in the service table. Every node is assumed to be capable of resolving descriptive names to network addresses, namely; specific name resolvers or directory services are not needed. The modeling environment can be characterized as follows:

- network connectivity is highly varying,
- routes for active sessions dynamically change,
- sessions are not preemptive; once a session is established for a resource, service handoff with a closer resource for the named session is not supported.

As given in service definition, upon deciding to offer a service, the node announces its service XML schema which is encapsulated in an announcement packet. Service announcements are initially caught by all the nodes in the same transmission range of each other. Upon receiving a service announcement, the node first updates its service table, increments the hop count field by one, appends its own host instance in the route and forward the announcement. Circular flooding for the announcements are prevented by keeping a session cache as explained above. The hop count field is also used to prevent circular flooding. The same announcement arriving with a hop count greater than the previously arrived announcement is discarded. Eventually all reachable nodes in the network get the announcement with a full route record. The algorithm for the *Service_Enable()* function is given below. A number of general set definitions are used in the algorithms given in pseudo code like notation. These are:

| | |
|---|---|
| M : | set of SeMA node instances |
| S : | set of SeMA service instances |
| T : | list of service table entries on a node |
| C : | session cache |
| I : | set of unique id's |
| <> : | nodes in the same transmission range |



1
2
3
4
5
6
Entry Length
MH Instance of the Service Providing Host
Service Instance
Timestamp
Forwarding Hop Count
MH instance of the Forwarding Host (#1)
. . . .
MH instance of the Forwarding Host (#n)

Figure 4. Service Table Entry

```
$\mathbf{Service\_Enable}(\&\mathbf{packet})$
$SeMA\_Packet\ packet;$
$packet.ptype = SeMA\_Anouncement;$
$m \in M, s \in S$
{
    $M' \supset M : m' \in M' \wedge m <> m'$
    $\forall m' \in M' : SeMA\_Send(packet, broadcast)$
}
```

## 3.4 Maintaining Service Table

Service announcement packets received are processed by service agent(SA). It checks if it has seen this announcement packet before or not. New announcements are appended to the service table. Valid old entries for the same service are kept as alternative routes. The service agent appends its link id to the source route list, increments hopcount and broadcasts the announcement packet. The algorithm for table update is given below.

```
$\mathbf{Service\_Table\_Update}(\&\mathbf{packet})$
$SeMA\_Packet\ packet;$
$packet.ptype = SeMA\_Anouncement;$
{/* if packet seen before, then drop it, otherwise if service
    is not seen before or is fresher than the cached entry,
    then update the table.*/
    if$(\exists s \in C : s.SeMA\_ID = packet.SeMA\_ID)$
        $[DropPacket]$
    if $\forall t \in T : (t.service \neq packet.service) \vee$
        $(t.ts < packet.ts)\{$
        $T \equiv T \cup packet.service$
    }
    $packet.Route = packet.Route \cup local\_link\_ID$
    $packet.Hopcount++$
    $SeMA\_Send(packet, broadcast)$
}
```

## 3.5 Service Binding

Service binding is initiated in session manager(SM) when client makes a service request. A handle for the service instance is obtained. First, service agent searches the local service table for a valid service entry. A valid service entry corresponds to a service announced less than a threshold time earlier than the current time (i.e. *current.time - ser-*

*vice.ts* < *threshold*). If a handle can be obtained locally, binding is done without generating any additional network traffic. If there is no local entry for the service then binding is delayed until a reply to the service lookup packet returns. *SeMA_Lookup* packet is broadcast to the network. This packet is first seen by the nodes in the same transmission range. Each node checks its own table to see if there is a valid entry for the requested service. If so, returns a reply to the originator and stop flooding. Otherwise, the node waits for a random amount of time to see if any node will reply to the request. If not, then broadcasts the request in its own transmission range by appending its own id to the packet. Each service entry in the table has a timestamp indicating when the service announcement was initiated. The service agent may choose the most recently advertised service as the "best" instead of the shortest hop path with the assumption that the probability of the recently advertised route being up-to-date is greater than an older advertisement. If the lookup message arrives at the service providing host, it waits for a random period of time for the data packets to arrive for the service assuming that at least one node in the network would have a service entry in its table and would have replied to the request. The random delay may prevent unnecessary message overflow. The session is initiated with the initial route returned in the handle. The route may be updated by an intermediate node if a better route is known. The algorithm below describes the actions taken by service agent for binding.

```
$\mathbf{Bind\_Service}(\&\mathbf{service})$
$ServiceXML\ service;$
$SeMA\_Packet\ packet;$
{ /* Find all table entries for service and return the one
      with the greatest timestamp.*/
if $(\exists t \in T_{local} : t.service = service) \Rightarrow$
  $\forall t' \in T_{local} : (t.service = t'.service) \wedge (t.ts > t'.ts)\{$
        $return(t)$
}else{ /* Generate a lookup packet */
        $packet.service = service$
        $packet.ptype = SeMA\_Lookup$
        $packet.SessionID = 0$ /* control packet */
        $packet.ClientID = local\_link\_id$
        $SeMA\_Send(packet, broadcast)$
            /* collect all replies until timeout */
        $return()$
    }
}
```

### 3.6 Session Establishment

After obtaining a binding for a service request, SM assigns a session ID, divides the application data into SeMA_Data packets with a unique sequence number and passes them to the communication agent for transmission.

```
$\mathbf{Create\_Session}(\&\mathbf{service}, \mathbf{data})$
$ServiceXML\ service;$
$SeMA\_Packet\ packet;$
{
    $Route = Bind\_Service(\&service)$
    $Session\_ID \in I : \forall S \in I : Session\_ID \neq SID$
    $packet.Route = Route$
    $packet.ClientID = Local\_node\_link\_id$
    $packet.ptype = SeMA\_Data$
    $packet.LostFlag = False$
    $packet.seq\# = 0$
    /* Fragment the data field and generate
        SeMA_Data packets with unique seq# */
    while$(data)\{$
        $SeMA\_Send(packet, packet.Nexthop)$
        $data = data - packet.data$
        $packet.seq\# + +$
    }
}
```

## 4 Route Optimization and Maintenance

An intermediate node on the source route sends SeMA packets to an appropriate next node. The algorithm tries to optimize the route for every packet processed. Possible states for a new arrival are;

- The node might have seen the packet before due to circular flooding or route loss.
- The packet may have an attached valid route or may have lost its route (LostFlag set).
- Packet may belong to a terminated session.

RA maintains a cache of seen packets. The session cache also has an indication for the terminated sessions to prevent unnecessary packet processing. RA tries to find a fresher route in its own service table. If there is such an entry then the route field in the packet is updated and the packet is forwarded to the next hop in the new route. Otherwise, the packet is forwarded to the next hop in the arriving source route. If the lost flag is set in the packet and there is no entry for this service instance in the service table then the algorithm switches to flooding until an intermediate node finds a service entry in its table. Route_Packet() algorithm is given below:

```
$\mathbf{Route\_Packet}(\&\mathbf{packet})$
$SeMA\_Packet\ packet;$
$packet.ptype = SeMA\_Data;$
{/* For a packet seen before, if it is not lost or the session
    is terminated then drop it. Otherwise check if there is
    a fresher route in the service table */
if$(\exists s \in C : s.SeMA\_ID = packet.SeMA\_ID) \wedge$
    $((\neg packet.LostFlag) \vee (s.Terminate))$
        $[DropPacket]$
/* For a new packet if there is a fresher entry then update
    the route in the packet and send it. */
else if$(\exists t \in C : t.Service = packet.service) \wedge$
    $(t.ts > packet.ts)\{$
    $packet.LostFlag = false$
    /* update route in the packet */
    $packet.Route = t.Route$
    $SeMA\_Send(packet, packet.Nexthop)$
}/* if there is no fresher entry and packet has no route
     then switch to flooding. Otherwise. if the packet has
     a route then use it. If the new route is fresher than
     the local table entry then update the service table.*/
else if($LostFlag$){
    $SeMA\_Send(packet, broadcast)$
}else{
    $SeMA\_Send(packet, packet.Nexthop)$
    if($\exists t \in C : (t.Service = packet.service) \wedge$
        $(packet.ts > ts)$){
        $t.Route = packet.Route$
    }
}
```

## 4.1 Session Termination

The packet transmission protocol for non-interactive services simply offers best effort delivery. Destination node on which the service is running detects end of packet transmission by checking the *LastFlag* and *Seq#* information in the received packet and generates a termination indication for the session after the service is completed. Termination indication message received by intermediate nodes are recorded and any packet which has a greater timestamp than the termination packet is discarded. When the termination message arrives at the client, it is accepted as an overall acknowledgment for the packets and the client code returns successfully. SM clears its table and the network is terminated.

## 5 Future Work and Concluding Remarks

Multiple session management and service migration are to be considered for a complete discussion of the topic in the sense that the creation, maintenance and termination of an adhoc network should be clearly defined.

Different scenarios of mobility must be used in order to evaluate the performance of the proposed model. (i.e mobile hosts acting for common goals or moving based on daily needs). For this purpose, the simulation environment is being designed carefully to apply different mobility models without affecting the service model itself. There are some important issues to be covered before introducing the proposed model for wide area usage. First is the need for a domain restriction (i.e. a domain service resolver) in order to have a well scaling protocol. The second obvious need is the integration of a security model to define hierarchical access structure for services.

In this paper, we have addressed that applications on hosts of mobile adhoc networks will need a complete architecture to access network resources by simply describing what they are looking for in a highly dynamic topology where no infrastructure underneath exists. A simulation is currently being implemented and the results will be presented somewhere else.

## 6 Acknowledgment

The authors would like to thank Nurcan Tezcan who has implemented the simulation trace generator as her senior project.

## References

[1] X. Hong, M. Gerla, G. Pei, and C.C. Chiang, A group mobility model for adhoc wireless networks, *Proc. ACM/IEEE MSWiM 1999*, Seattle, WA, 1999, 53-60.
[2] J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva, A performance comparison of multi hop wireless adhoc network routing protocols, *Proc. ACM/IEEE Mobicom 1998*, Dallas, TX, 1998, 85-97.
[3] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek, and M. Degermark, Scenario based performance analysis of routing protocols for mobile adhoc networks, *Proc. ACM/IEEE Mobicom 1999*, Seattle, WA, 1999, 195-206.
[4] S.R. Das, C.E. Perkins, and E.M. Royer, Performance comparison of two on demand routing protocols for adhoc networks, *Proc. IEEE Infocom 2000*, Tel Aviv, Israel, 2000, 3-12.
[5] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz, An architecture for a secure service discovery service, *Proc. ACM/IEEE Mobicom 1999*, New Orleans, 1999, 24-35.
[6] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, Service Location Protocol, *IETF, RFC 2165, http://www.ietf.org/*, 1997.
[7] M. Zonoozi and P. Dassanayake, User mobility modeling and characterization of mobility patterns, *IEEE Journal on Selected Areas in Communications, 15*(7), 1997, 1239-1252.
[8] Jini, *http://www.sun.com/jini/*.
[9] C. Bettstetter and C. Renner, A comparison of service discovery protocols and implementation of the service location protocol, *Proc. $6^{th}$ EUNICE Open European Summer School*, Twente, Netherlands, 2000.
[10] W. Winoto, E. Schwartz, H. Balakrishan, and J. Lilley, The design and implementation of an intentional naming system, *ACM Operating Systems Review, 34*(5), 1999, 186-201.
[11] T. Bray, J. Paoli, and C.M.S. McQueen, Extensible Markup Language (XML), *W3C Recommendation, http://www.w3c.org/*, 2000.
[12] D.C. Fallside, XML Schema Part 0: Primer, *W3C Proposed Recommendation, http://www.w3c.org/*, 2001.