Automatic Form Filling on Mobile Devices

Enrico Rukzio[1], Chie Noda[2], Alexander De Luca[3], John Hamard[4], Fatih Coskun[3]

[1] Computing Department, Lancaster University, InfoLab21, LA1 4WA Lancaster, UK
Tel.: +44 1524 510358 / Fax.: +44 1524 510492
rukzio@comp.lancs.ac.uk

[2] NTT DoCoMo, Inc., 3-5 Hikarinooka, Yokosuka, Kanagawa, 239-8536, Japan
Tel.: +81 46 840 3842 / Fax.: +81 46 840 3725
noda@nttdocomo.co.jp

[3] Media Informatics Group, University of Munich, Amalienstrasse 17, 80333 Munich, Germany
Tel.: +49 89 2180 4688 / Fax.: +49 89 2180 4652
{alexander.de.luca, coskun}@ifi.lmu.de

[4] NTT DoCoMo Euro-Labs, Landsbergerstrasse 312, 80796 Munich, Germany
Tel.: +49 89 56824 216 / Fax.: +49 89 56824 300
hamard@docomolab-euro.com

Abstract
Filling out forms of web based services on mobile devices is a very time consuming and frustrating task for users because of the limited text input capabilities. This is a critical point to get a wide acceptance of such services, especially mobile commerce that often requires filling user data. We developed an architecture based on a local proxy on a mobile device and a lightweight algorithm for a comprehensive analysis of forms, which leads to the highest probable user data to be filled in, driven by an initial rule set [1]. We further discuss our implementation and the evaluation results of the algorithm as well as the usability of the prototype.

1. Introduction

The convergence of mobile communications and web-based services has emerged in the last decade. Most of current mobile devices support internet browsers (e.g. *Opera mobile* [2]) which are often preinstalled. However, mobile commerce services (e.g. hotel reservation) requiring user data to be filled in are still not as widely used as download services (e.g. ring tones, games). In fact, the main issues regarding mobile commerce usage are the complexity of user interactions and mobile payment systems. To solve this last issue, NTT DoCoMo introduced in 2004 the Mobile Wallet service [3], based on the Sony's FeliCa IC chip [4], in order to support electronic payments with mobile phones.

The lack of simplicity is one of the most important issues for users who face more and more functionalities, applications, services and networks when using mobile devices. One approach to overcome this issue is to provide context aware mobile services. This is achieved by considering user context information (e.g. user location or history of selected services) to adapt applications and services accordingly.

In this paper we present an approach to support the automatic form filling on mobile devices. Indeed, filling forms manually on mobile devices through limited user interfaces is a time consuming and stressful task. At the beginning of our research we analysed which mobile services are currently offered and which are the most popular ones. One result was that mobile entertainment applications which consist in downloading ring tones, games or logos are widely used [5]. We found most of mobile services are accessible through simply navigating hyperlinks. However mobile commerce services (e.g. hotel reservation, auctions) which often require filling personal data, have not been yet widely used. Hence, by improving the user experience with mobile services, automatic form filling could then also support mobile commerce markets growing.

Our approach is based on the knowledge about existing web pages in form of rules and user data. We analysed the most required user data when using mobile commerce services. More precisely, we identified their types, their sequence of appearance on web pages, as well as the surrounding form elements (e.g. labels or input fields) of the input field in which they are filled. Based on this analysis, we developed a local proxy architecture and a lightweight algorithm for dynamic rules generation based on a set of initial rules. The proxy acts as a mediator between the browser on the mobile phone and the web server hosting the requested page. It fills input fields with user data stored on the mobile device by analyzing the input fields, the nearby from elements and checking initial rules, and generates dynamic rules, which leads to the highest probable user data. The proposed architecture has the following advantages:

- Support for existing mobile services and devices based on a proxy architecture,
- Optimization for the memory and processing constraints of mobile devices based on a initial rules set and dynamic rules generation,
- Privacy protection by only storing user data locally on the mobile device
- Support for user control by enabling the editing of filled data before sending the form.

The paper is organized as follows. The next section analyses text input capabilities and presents popular mobile services. Furthermore we show the results of an initial user evaluation of the potential and acceptance of such a form filling functionality on mobile devices. In addition to that an analysis of existing forms is discussed. Section 3 discusses in details our architecture and algorithm for automatic form filling on mobile devices. Afterwards an implementation of our system is presented. In section 5 we present the performance evaluation of our algorithm and the results of further user studies. Section 6 relates our work to existing approaches. Finally we summarize our research and discuss further steps.

2. Analysis

In this section which is based on [5] we analyze first text input capabilities on mobile devices. Afterwards we present an initial user test which goal was to evaluate the concept of automatic form filling on mobile devices. Next we present a compact analysis of existing forms.

2.1 Text Inputs Capabilities

There are significant differences in text input speed on a personal computer and a mobile device. We can distinguish three different text entry techniques for mobile devices: key-based, stylus-based and predictive input techniques [6]. The text entry speed is usually expressed in words per minute (wpm). A skilled touch typist using a conventional keyboard can enter an average of 72 wpm [7]. As shown in Table 1, text entry on mobile devices is much slower.

Device type	Input technique	Input type	WPM	User skills	Ref.
PDA	Graffiti	stylus-based	21,5	average user	[7]
	QWERTY keyboard	Key-based	20,2	novice user	[7]
Mobile phone	Multi-press method	Key-based	25-27	expert user	[8]
	T9	predictive	41-46	expert user	[8]

Table 1. Text entry speed on mobile devices

Three to four alphabetical letters are assigned to one button on mobile phones. When using the traditional multi-press method the user has to select the intended letter through pressing a key multiple times until reaching the desired one. The T9 system is based on a predictive algorithm which takes into account the occurrence frequency of words stored in a database. Thus once the user selects buttons, which represent the intended letters to write a word, probable words are shown up. The T9 system is the fastest input mean for expert users. However it does not often support data such as first name required for form filling.

2.2 Initial User Test and Definition of User Requirements

Furthermore we studied the experience of users using mobile commerce services on a PDA. A HTML-based mock-up of a hotel reservation service was built on a Sony Ericsson P800 Smartphone. The mock-up included a form with 10 input fields[1] whereas the Smartphone provided a virtual keyboard, a stylus and integrated the Opera browser. Two test cases were considered:

- The users fill out forms manually,
- The users need to identify and correct pre-filled forms including two errors.

[1] First name, last name, address, city, ZIP, phone number, e-mail address, payment method, credit card number, and expiration date.

Table 2 shows the results of average time comparison between the two configurations for three runs.

	(i) Empty forms	(ii) Pre-filled forms
1. run	240 seconds	60 seconds
2. run	170 seconds	37 seconds
3. run	115 seconds	33 seconds

Table 2. Average input times over all users

The most noteworthy result is that it takes four times longer for users to fill empty forms manually compared to pre-filled forms. Another interesting point is that users learn quite fast to use a virtual keyboard or a stylus. Beside these quantitative results we recognize that most users are frustrated when using the stylus of the Smartphone for inserting texts. We conclude that automatic form filling on mobile devices is extremely usable since it just requires limited user input.

With regards to privacy issues, we learnt from this user study that many users would not give their personal information away (e.g. as in the Microsoft .NET Passport). Furthermore this study underlined that users need to keep control, e.g. by seeing which data are filled in and by possibly deleting or modifying automatically inserted data. We thus defined the requirements to only store user data on the mobile phone and to enable users to edit these automatically inserted data.

2.3 Analysis of Existing Forms of Web Based Services

Afterwards 20 mobile commerce services were analyzed in order to find out which user data are required (e.g. for ordering or reserving a product). Similar data were usually required so that a basic data set was clarified. Furthermore, we noticed that labels were mostly fixed groups and that input fields could be named differently in the source code. From this analysis we concluded that web forms were rather similar and therefore, an automatic form filling feature could be implemented. Table 3 shows specific sets of variable names for input fields. It shows some of the concept names used in our algorithm (first column) in relation to the variable names used in three arbitrary selected web based services. In the context of this paper, a concept name stands for the internal naming of personal information in our algorithm. Specific naming variations must be handled since, for example, each variable name on *Amazon Anywhere* is prefixed with the word *shippingAddress*. A system for automatic form filling must therefore also support sub-string analysis of variable names.

Concept names	Variable names		
	mobile.quelle.de	Amazon Anywhere	Hilton.com
FirstName	FirstName	shippingAddress.name	firstName
LastName	LastName	shippingAddress.name	lastName
AddressStreet1	Street	shippingAddress.address1	adress1
AddressStreet2	Street2	shippingAddress.address2	adress2
Email			Email
AddressTown	City	shippingAddress.city	City
AddressCode	PostalCode	shippingAddress.zip	postalCode
AddressCountry		shippingAddress.countryCode	Country
TelephoneHome	MobilePhone1	shippingAddress.voice	phoneNumber

Table 3. Variable names in three different forms

Furthermore, the form elements (e.g. labels and input fields) before and after an input field are also an important parameter. We noticed many correlations between requested concept names and the placement/labelling of input fields in the forms. For example, the probability is extremely high that an input field labelled *First Name* should be filled with the first name of the user.

3. Architecture and Algorithm

We developed a local proxy architecture and applied a lightweight algorithm for dynamic rules generation based on a set of initial rules. This section presents details on the architecture, the form filling rules, and the algorithm.

3.1 Architecture

Figure 1 shows the architecture of our approach for automatic form filling on mobile devices. The proxy of the mobile device acts as a mediator between the web browser and the web server hosting a requested web page. The form filler in the proxy fills input fields of forms with user data locally stored by analyzing the the nearby form elements (e.g. labels and input fields) of the input fields. Hereby initial rules (locally stored or downloaded) are used to generate dynamic rules. This leads to the highest probable user data to be filled in a form. The rule server is an external component, which stores and provides the form filling rules. The rule repository of the rule server enables updating the rules, for example monitoring users' behaviours and adding new concept names. The proxy of the mobile device uses it to keep the local rule set up-to-date.

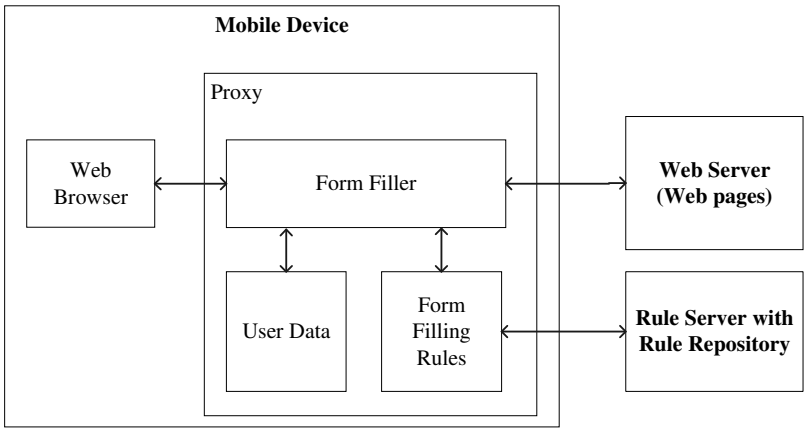


Figure 1. Architecture for automatic form filling on mobile devices

Figure 2 shows internal components of the form filler. The parser parses a received web page (e.g. HTML, XHTML, cHTML (i-Mode), WML/WAP) and creates an object structure containing objects for each input field as well as the surrounding form elements. The rules inspector retrieves rules available locally or from the rule server, which match to a given object structure, and creates dynamic rules. Finally the user data filler fills out input fields with user data corresponding to a concept name with the highest probability.

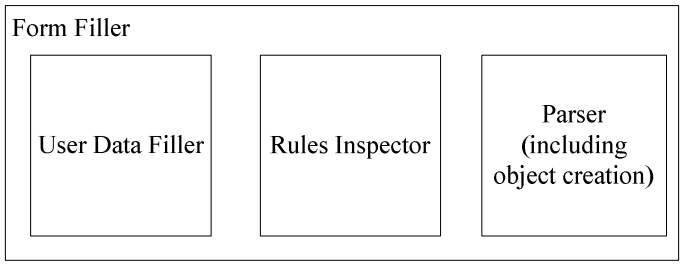


Figure 2. Elements of the form filler

The proxy can be pre-installed on the mobile device or downloaded as a 3rd party application. User data can be specified by the user through the user interface of the mobile device, retrieved from the user profile on the device or on the SIM/USIM card, or stored by monitoring and gathering which data are input on forms by the user.

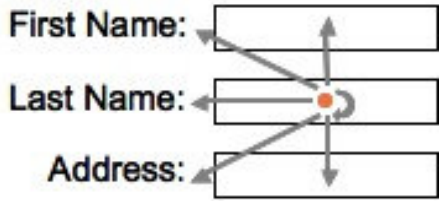
3.2 Form Filling Rules

3.2.1 Rules Format

This subsection describes the syntax of the form filling rules which were developed based on the results of the analysis of existing forms as discussed in the subsection 2.3. Rules present which information of the web document (e.g. HTML, XHTML, cHTML (i-Mode), WML/WAP) is used to find the right input data and additionally provides a certain probability. There are 6 different values, so called positions that can be used to assume a required data for the input field in the web document as shown in Figure 3. Not only the current input field but also the upper and the lower input fields are analyzed. There is for

instance a specific probability that the last name is requested after the first name. The mentioned positions are:

- *Upper LABEL*: the last text before of the upper input field
- *Upper NAME_ATTRIBUTE_VALUE*: the value of the name attribute of the upper input field
- *Current LABEL*: the last text before of the current input field
- *Current NAME_ATTRIBUTE_VALUE*: the value of the name attribute of the current input field
- *Lower LABEL*: the last text before of the lower input field
- *Lower NAME_ATTRIBUTE_VALUE*: the value of the name attribute of the lower input field




```
<label for="firstname">First Name:</label>
<input id="firstname" name="firstname" type="text"/>
<label for="lastname">Last Name:</label>
<input id="lastname" name="lastname" type="text"/>
<label for="address1">Address:</label>
<input id="address1" name="address1" type="text"/>
```

Figure 3. Example of a user data form and its HTML representation

Figure 3 shows some input fields on a form and presents the corresponding HTML code (e.g. for reserving a hotel room). The arrows indicate the 6 positions: the label and the name attribute value (e.g. *name=" firstname"*) of each field (*current, upper*, and *lower*). The 6 different values respective positions of the example shown in Figure 3 are:

Upper LABEL: First Name
Upper NAME_ATTRIBUTE_VALUE: firstname
Current LABEL: Last Name
Current NAME_ATTRIBUTE_VALUE: lastname
Lower LABEL: Address
Lower NAME_ATTRIBUTE_VALUE: address1

Based on these positions we defined the following syntax for our form filling rules:

Position **|** ***Condition*** **|** ***Value*** **|** ***Concept Name*** **|** ***Probability***, whereby

- ***Position*** = {*UPPER_LABEL, UPPER_NAME_ATTRIBUTE_VALUE, CURRENT_LABEL, CURRENT__NAME_ATTRIBUTE_VALUE, LOWER_LABEL, LOWER__NAME_ATTRIBUTE_VALUE*}
- ***Condition*** = {*CONTAINS, EQUALS*}
- ***Value***= arbitrary string of labels and name attributes, e.g. a label tag, the string just before the form, the name attribute of the form
- ***Concept Name*** = element of the user data (i.e. be used to fill in the form), e.g. *FirstName*
- ***Probability*** = number between 0 and 100

Every rule can be interpreted in the following way:

If the *position* has the *condition* of the *value*, then the *probability* is x% that the *concept name* y has to be filled in.

Examples for rules are:

- *CURRENT_NAME_ATTRIBUTE_VALUE |CONTAINS|firstname|FirstName|100*
  If the name attribute *(CONCEPT_VALUE)* of the input field *CONTAINS firstname* then the probability is 100% that the input field should be filled out with the first name *(FirstName)* of the user.
- *UPPER_ NAME_ATTRIBUTE_VALUE |CONTAINS|firstname|LastName|81*
  If the name attribute *(UPPER_CONCEPT_VALUE)* of the input field which is above the current input field *CONTAINS firstname* then the probability is 81% that the input field should be filled out with the last name *(LastName)* of the user.
- *CURRENT_LABEL|CONTAINS|address|AddressStreet1|46*
  If the string left to the input field *(LEFT) CONTAINS address* then the probability is 46% that the input field should be filled out with the *address street 1 (AddressStreet1)* of the user.

3.2.2 Creation of the Basic Rules Set

In a first step of this work, we needed to define a rules set that could be used for the automatic form filling. We semi-automatically analyzed about 200 arbitrary selected web pages including forms. In a first manual step, the correct concept name was applied to all to the input fields of these websites. The rest could be done almost completely automatically. Rules are generated and added to a rule repository by applying conditions of labels and name attributes. That is, if an input field is found, its concept name is used to create the rules based on all the labels that belong to this field. If the same rule is found, the number of appearance is increased. Otherwise a new rule is added in the rule repository. The resulting rules set contained a huge amount of rules. We set a threshold to eliminate meaningless rules caused by the following reasons:

- Rules appearing only once are considered meaningless since the probability of their appearance is very low.
- Some website authors use meaningless HTML name attributes like “field1”, “field2” and so on. These names cannot be used to find out the meaning of an input field.
- Due to bad web design, some rules may contain information that is not near the input field. This may be the case, if some table layouts as well as positioning in CSS are used.

At the end, the probability of rules is added by calculating the ratio of the same labels and name attributes, but different concept name. More generally speaking, for the definition of the probability of a specific rule to be the right choice for the input field, all occurrences of a specific label (in a unified form as explained below) are compared and used for the

calculation of the probability. For instance, if the label “firstname” appears in 3 different rules, whereas the first rule has 5 occurrences, the second has 3 and the last has 2 occurrences, the probabilities for these three rules are 50%, 30% and 20%.

This way, we were able to define 142 rules as an initial rules set for our prototype.

It is important to mention that the analysis as well as the automatic form filling algorithm contains a step to unify labels. That is, unnecessary information as well as captions is removed to create more general rules. For example the label “First Name” and “FirstName” will both be changed to “firstname” and thus, the appearance of that rule will be two instead of creating two single rules.

3.3 Form Filling Algorithm

The form filler uses the initial rules set and the locally stored user data to fill in the requested webpage. Hereby the following algorithm is used:

(1) The user data and the form filling rules are loaded from the storage of the mobile phone and on demand from the rule server.

(2) The browser requests a web page through the proxy.

(3) The parser of the proxy extracts the downloaded webpage and creates an object structure while checking each input field. This process is depicted in Figure 4. The created object structure contains an object for every form and every input field. For example, if the following input field is included in a parsed form *First Name <input type="text" name="firstname" value=""/>*, the generated object includes the attributes label, type (the type of the input field), name, and value. It also keeps information on locations of input fields.

In the end of the parsing process, there is an object for every form which knows every input field that belongs to it. The input fields are also objects, which are augmented with, for example their name attribute values and their neighbours.

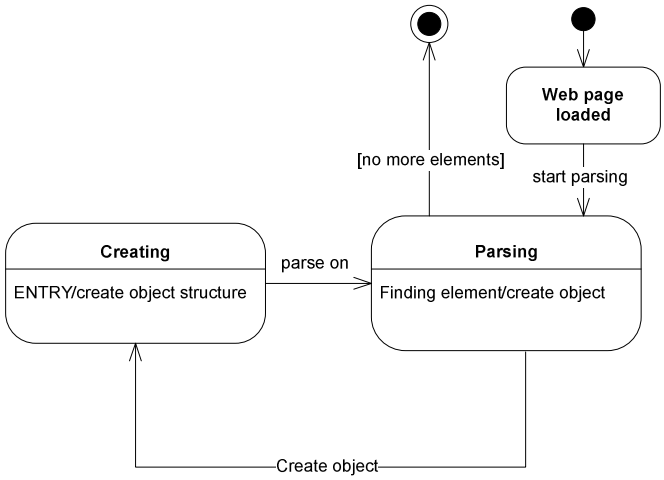


Figure 4. Parsing of a web page

(4) For every input field of a form the proxy selects rules and generates a dynamic rule as follows:

The rules inspector retrieves all rules that fit to the field. There is mostly more than one rule which fits to the input field. This means that for one input field there can be rules for some or all 6 positions around it. Every rule includes a probability that describes how often a value for a specific concept name is found in a specific position. An example is, that *left contains address* fits to different concept names like *AddressStreet* or *EmailAddress* but with different probabilities.

When a rule is found for an input field, this rule's probability is converted to what we call *concept points*. The conversion takes some aspects into account, i.e. the probability value and the type of the rule such as normal or superior rule.

During checking the input field, all concept points found for a specific concept name, like this field fits to the concept name *AddressStreet*, are summed up. At the end of the checking, our algorithm compares the concept points between the different concept names found. The concept name with the highest amount of concept points is selected.

There are two special cases, so called *superior rules*, which are used to increase the speed of the algorithm and to utilize the limited capability of mobile devices, especially memory space:

1. Rules for CURRENT_LABEL or CURRENT_NAME_ATTRIBUTE that have a probability of 100% are instantly chosen if they are found for the current input field.

2. Rules for CURRENT_LABEL or CURRENT_NAME_ATTRIBUTE that have a probability of less than 100% are rated higher than the other rules found. This means, we rate higher concept points compared to the other rules.

Figure 5 illustrates that every input field is analyzed by the form filling algorithm. For each input field found, the procedures depicted in Figure 6 are applied, to check corresponding rules, and sum up concept points.

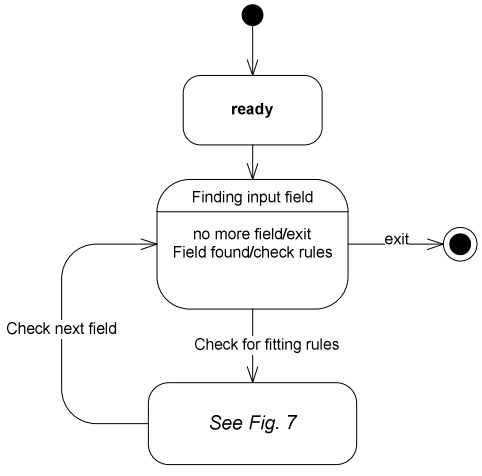


Figure 5. Checking the input fields

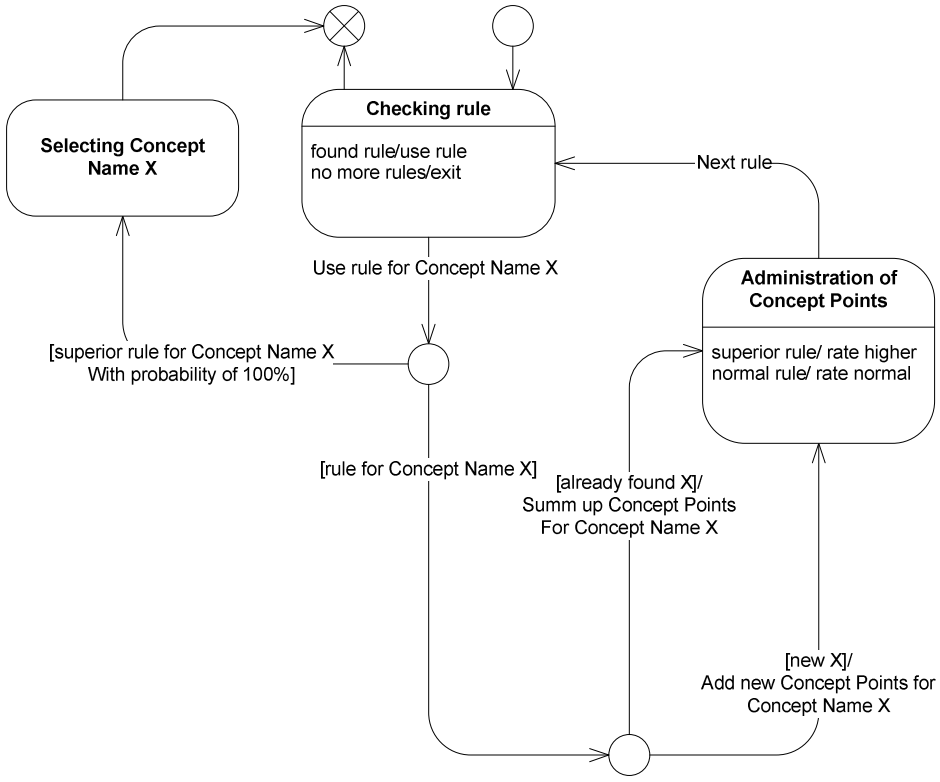


Figure 6. Checking an input field on rules

Superior rules are to increase the speed of the process. If a fitting rule for concept name X has been found, there are two possibilities:

1. It is a superior rule with a probability of 100%, then the rule for the concept name X is instantly chosen for the field.
2. Else go on proceeding the rule.
   The Administration of Concept Points checks if 'X' is an existing or a new concept name:
   (i) when the concept name exists, the concept points are summed up, or
   (ii) when the concept name is new, the concept points are stored for the new concept name X.
   If all rules are checked or no superior rule with a 100% probability has been found, then the concept name with the highest concept points is chosen.

Of course, superior rules are seldom and may lead to wrong filled fields but their advantage lies in the increased speed, especially for mobile phones. Nevertheless, in our tests, no wrong filled fields due to superior rules occurred. Additionally, the automatic update of rules as explained in more detail in the rules server section, will remove superior rules that have proven to be not 100% sure. For example, if a superior rule for the label "firstname"

exists but a user of the system has a wrong filled form (e.g. for the label "ChildFirstName"), then the probability of the rule will be decreased and thus the problem solves itself.

(5) The *user data filler* fills user data according to selected *concept names* with the highest *concept points*.

(6) The proxy delivers the filled out web page to the browser.

As mentioned before, one of our goals was to develop an algorithm that can not only compete with existing commercial products but also works on resource scarce mobile devices. When taking a deeper look at the algorithm, it becomes clear that the memory usage will grow linear with the number of rules. Fortunately, we managed to create an algorithm that works very well with a small amount of rules. In our prototype, 142 were enough to fill out all required Concept Names. Furthermore, when looking at usage statistics of the rules, we believe that with an improved rules generation algorithm, we can minimize the amount even more. Since the rules format has been chosen really small (e.g. no XML has been used), this is an amount that can be easily handled by most mobile phones.

Regarding the execution speed, we tested our algorithm (with the 142 rules) on standard mobile phones with web sites including up to 10 forms to see whether the users recognize a waiting time for the algorithm execution. There was no recognizable delay at all. Even if the rules set would be increased, the time will only go up slightly, mostly due to the superior rules that are the most common rules used.

Another important point regarding performance is that the algorithm can be performed in an automatically distributed manner. Even if the number of web forms in a web page is increased, the algorithm can perform in the order of the web forms. Once the forms fits to the size of the mobile phone's display are filled, the rest can be performed in the background. We may further assume that the algorithm performs and fills user data only when the user interface focuses on an input field on the display.

3.4 Rules Server

The rules server stores a basic set of rules generated by analyzing existing web pages. The proxy of the mobile device downloads rules from it and stores them locally. The rules server can further support keeping up-to-date rules, for example adding new concept names, or keeping track of different users' actions.

If users change values of input fields filled by the automatic form filling function, these changes can be send to the server which creates rules for new concept names or change existing rules depending on the submitted data. It enables optimization of rules by users' actual usage. Rules optimization processes are followings:

1. The proxy of the mobile device monitors if the user change user data filled out automatically.
2. It translates every changed field, and extracts a concept name and its location.
3. It sends these data to the rules server.

4. The rule server either changes the probability of existing rules or creates a new rule if it is for a new concept name.

The advantage of such a server is that all users optimize the rules to archive higher probable results of automatic form filling.

4. Implementation

To prove our concept we implemented a prototype which architecture is depicted by Figure 7. For that purpose, we used the Java Micro Edition (Java ME), MIDP 2.0 and CLDC 1.1 that is supported by most current mobile phones. Furthermore we used the "Opera for Mobile" Browser since it could easily be connected to our proxy. We configured the proxy settings of the browser as depicted by Figure 8a. Hence, each page request from the browser is sent via the port 8110 to the proxy which forwards the request to the corresponding web server. The proxy is realized as a Java ME Midlet running in parallel to the web browser. The user data as well as the form filling rules are stored in a Java ME record store. We found out that our prototype worked successfully on a Nokia 6600, Nokia 6630 and on a Nokia 6680.

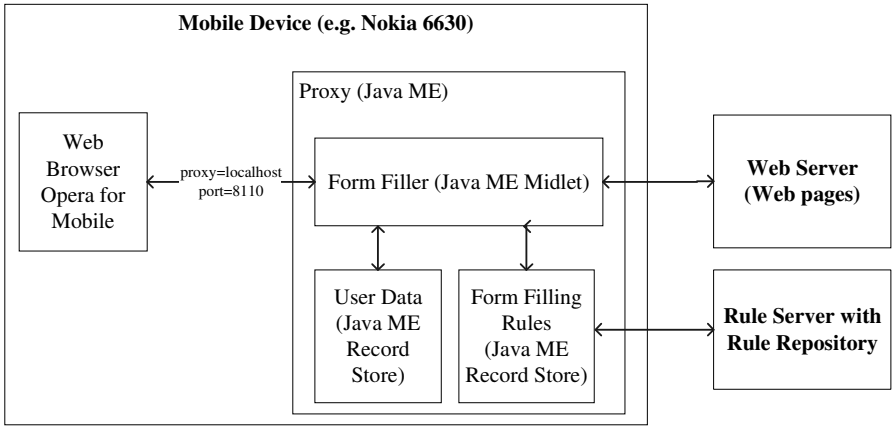


Figure 7. Architecture of the prototype

For the implementation of the prototype we used the following user data: *FullName, FirstName, MiddleName, LastName, AddressStreet1, AddressStreet2, AddressTown, AddressCode, AddressShire, AddressCountry, Email, TelephoneHome, TelephoneFax, TelephoneMobile, TelephoneWork, CardOwner, CardNumber, CardType, CardExpirationMonth, CardExpirationYear, CardExpirationComplete, Homepage* and *Email* which are based on the tags defined in [9] and are stored in a Java ME record store. We implemented a corresponding user interface which is depicted by Figure 8b for the management of these data by the user. As previously mentioned we analyzed about 200 web pages and generated 142 rules. Figure 8c shows the proxy Midlet after loading the user data and the form filling rules.

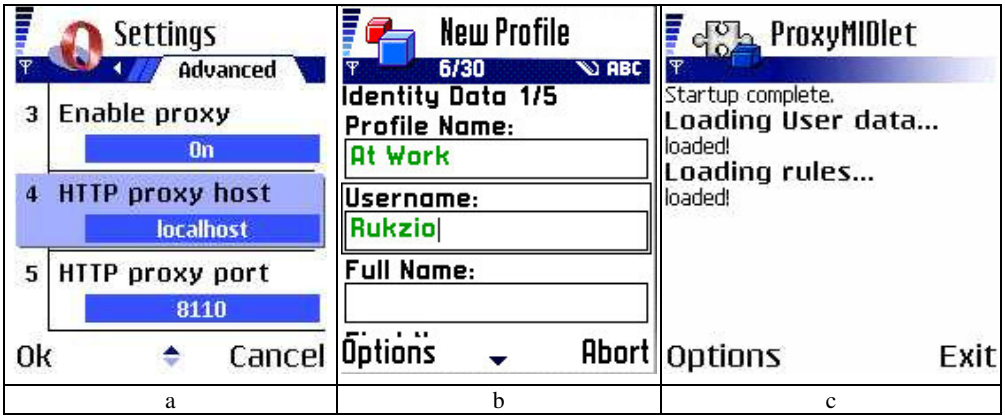


Figure 8. Screenshots of the prototype

5. Evaluation

In this section we discuss several evaluations. The first one is the accuracy evaluation of the form filling algorithm and the second one is the usability evaluation of our prototype through a user test.

5.1 Evaluation of the algorithm accuracy

First we evaluated how important the consideration of the surrounding form elements (e.g. labels and input fields) of an input field for its correctly filling is and in our second test we compared our algorithm with existing tools.

In the first test we evaluated the accuracy of our automatic form filling algorithm and compared it with a simplified algorithm only considering the name attribute of an input field; without considering the co-located from elements and without considering rules relationships between different rules.

We tested 37 arbitrary selected web sites which were different from the ones we used for rules creation. We checked whether the following 7 concept names*: FirstName, LastName, AddressStreet1, TelephoneHome, AddressTown, AddressCode,* and *Email* were correctly filled in. We counted how many input fields were filled out correctly and how many were filled out wrong by both algorithms. Some of the forms in the selected web pages did not include all of theses concepts names. Therefore we had to check just 242 fields instead of 259 (37*7). Other fields than the above-mentioned 7 concept names, like *organization* or *country*, which were sometimes filled in, were ignored during this evaluation. As shown in Table 4, considering the co-located form elements and the combination of the knowledge represented by several rules increases the recognition rate.

	Presented algorithm	Simplified algorithm
Correct filled fields	224 (92,6%)	207 (85,5%)
Wrong filled fields	18 (7,4%)	35 (14,5%)

Table 4. Comparison of the algorithms

Afterwards we evaluated the accuracy of our automatic form filling algorithm in comparison to the *AutoFill* function of the Google Toolbar [10] for Firefox 1.0 and the *Form Fill* function of the MSN Search Toolbar [11]. We used again the 37 arbitrary selected web pages as well as the same test protocol. As shown in Table 5, our algorithm provides similar accuracy as solutions primary designed for Laptops or Desktop PCs.

	Presented algorithm	MSN Search Toolbar	Google Toolbar for Firefox 1.0
Right fields	224 (92,6 %)	219 (90,5 %)	211 (87,2 %)
Wrong fields	18 (7,4 %)	23 (9,5 %)	31 (12,8 %)

Table 5. Comparison of the algorithms accuracy

The following figure 9 shows which algorithm filled how many of the 37 forms with a given number of wrongly filled fields. For instance filled the presented algorithm 27 of the 37 forms correctly (0 wrongly or not filled fields). The corresponding results show that most forms are filled in correctly or with just 1 to 2 errors. Furthermore can be seen, that every of the three algorithms had a serious problem with at least one form. The presented algorithm was for instance not able to fill any field of one form at all because of a conflict between the rules which occurred when they were applied for this form.

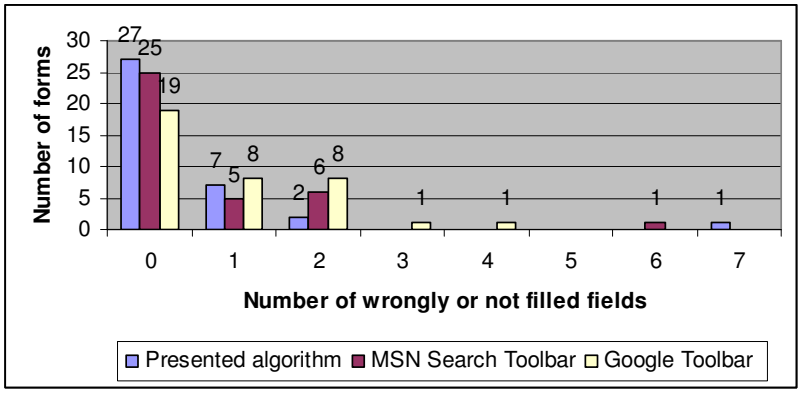


Figure 9. Comparison of number of wrongly filled in forms

The following figure 10 shows how often a given field was wrongly filled in by which algorithm. For instance was *FirstName* only two time wrongly or not filled in by the presented algorithm. The results show that the algorithm of the Google Toolbar had especially problems with filling in the *FirstName, Last Name* and *AdressCode*. The MSN Search Toolbar algorithm had problems with the filed *Last Name* and *AdressCode*.

Furthermore can again be seen that the presented algorithm performs better than the other two.

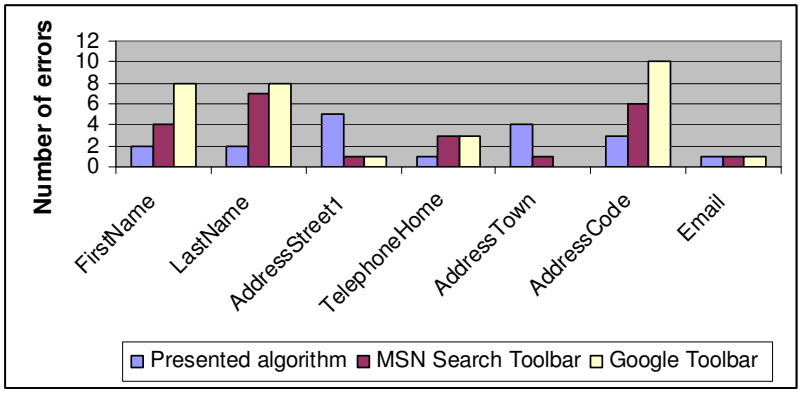


Figure 10. Comparison of the wrongly filled in fields

5.2 User Test

The goal of this user study based on the prototype described in section 4 was to figure out whether the automatic form filling feature would be usable for end users [12].

We had 18 volunteers that participated in our study, 9 women and 9 men, aged from 20 to 26. They were all students in computer science, communication science, politics, ethnology, linguistic or literature.

At the beginning we explained to each participant the concept of automatic form filling on mobile devices. We discussed that filling out forms, e.g. for ordering a product or making a reservation, takes a long time when using a mobile device such as a mobile phone. Then we briefly explained how such a feature as the auto form filling could support the mobile user. We said that the forms were automatically filled and the user could check and reedit them before submitted the form. Furthermore we insisted on the fact that personal data are only stored on the mobile device and thus, are not transmitted to any other party.

As the next step, users were asked to provide their first name, last name, address, postal code, city, phone number and email-address. Later on this information was used by the form filling feature as content for the form filling. Each test user was asked to check 20 forms with 0, 1 or 2 wrong automatically filled fields.

Each test run was executed according to the following scheme: After selecting the current setting, we waited until the form was completely loaded and put the mobile phone onto the table in front of the user with its display faced down. As shown in Figure 11, the user task was to turn the mobile phone around, check the pre-filled form, find errors and turn the mobile phone around again when ready. Please note there was no need for scrolling since the whole form fitted on the screen.

Figure 11. Method used to measure the time.

We used these gestures to exactly measure the time between the first look on the display and the moment when the user turned the display up. Afterwards the test user was asked to tell if the form was correctly filled out or if there were any error. Through this experiment we tried to prevent measuring the time users need for explaining errors. In fact, we measured the time the user needed to only recognize the errors only. In the cases where the user found errors we asked in which field a wrong content was filled in.

As already mentioned each of the 18 participants was asked to do 20 test runs. Thus we measured 360 runs altogether. The results of the user study are depicted in table 6. In general there were no significant differences in the time the participants needed for the different combinations. One result is obviously that the more errors were included the more time the participants needed for completing a run. One reason for that is that the participants had to mention after every run how many errors they found and where they errors have been. The participants needed 23% more time when 2 errors were included when compared to the test cases without any errors. Furthermore, not surprisingly, the frequency of non recognized errors and false positives was higher than 0% when 1 or 2 errors were included in the pre-filled form. A false positive is when the user wrongly mentioned that a field was filled with the incorrect content.

Errors	0	1	2
Runs	90	108	162
Average time needed in seconds	5,34	5,75	6,55
Frequency of errors and false positives: percent (sum of all errors)	0%	2% (2)	12% (19)

Table 6. Needed time and frequency of errors related to the included errors

Afterward the runs we asked the participants about their opining about the prototype. 83% (15 of 18) of the tester would use such an automatic form filling function if available. More detailed information about this study, the test setting and the results can be found in [12].

6. Related Work

This section relates our work to existing solutions. Chusho et al. [13] presented a system where an agent supports the automatic filling of forms in web applications. For this a corresponding architecture - similar to modern architectures in the field of artificial intelligence - was developed. This architecture includes an inference engine, a learning facility and a knowledge-base. Barton et al. presented their XForms approach [14] that supports adaptive services through clients that fill forms with sensor data. Furthermore existing commercial applications like Google Toolbar [10], MSN Search Toolbar [11], RoboForm [15] or iOpus Internet Macros [16] also provide automatic form filling. In

contrast to these solutions we decided to focus on mobile phones and mobile services. This is a challenge due to the very limited working memory and processing speed of mobile phones. All these publications and products address the same problem that each programmer can define her or his own forms using arbitrary labels, different data types and variable names. Theoretically it would be better if each form would follow a standard ontology that would make automatic form filling a much easier task. The W3C working draft Client Side Automated Form Entry [9] is an example for this and includes an ontology for the description of identity, contact, postal, billing and organisational information. But this is not a practical approach because it assumes that everybody has to use a standardized and well-accepted ontology. Another and certainly more practical solution is the usage of semantic web technology to described ontologies and the relationship between them [17]. Through this it would be possible that an ontology A is used to describe the elements in a form, that an ontology B is used to describe the user data and that the form filling would be a straight-forward reasoning task.

The investigation and development of context-aware services is currently considered by many researchers and within several scientific projects. This context information[2] is used to initiate services and contents adaptation. In the application area of this paper particularly personalized web applications for mobile devices have to be concerned that adapt web applications according the user and according the used device [18-21].

The usage of rule- or policy-based systems based on artificial intelligence concepts are one standard approach when designing systems for context-aware services. Suryanarayana and Hjelm presented an architecture [22] that takes different profiles such as user profile, application profile and transport profile into account. Regarding the processing of this data they discuss possibilities that are based on rules languages such as RuleML and policies. They also considered the usage of XSL Transformations (XSLT) to adapt services according to context information. Platforms supporting coordinated adaptation in mobile systems that are based on policies are presented in [23] and [24]. They strictly distinguish monitored context information, policies and adaptation mechanisms. It is possible to use policies for different adaptations and the adaptation mechanisms are independent from the policies. Through this the mobile services can be adapted in a system-wide manner. *Rei*, a policy language for pervasive computing application was presented by Kagal et al. [25]. This language enables expressing rules for rights, obligations, dispensations, and prohibitions. We restrict our approach to the domain of form filling for mobile devices.

One other problem when using services on mobile phones is that the most web based services are developed for desktop PCs or laptops. A popular solution for that is the usage of proxies which are located either on the mobile phone or on a server to adapt existing services. Examples for this is are the Opera Mini and the Opera Mobile Accelerator which use a remote proxy that eliminates unnecessary content and compresses web pages for sending them to the mobile phone. Our system can be seen as a proxy as well because our form filling application is located between the web browser and the web server.

---

[2] User data, device, location, surrounding devices, profiles, time, activity etc.

7. Conclusion and Future Work

We discussed that form filling on mobile devices is a time consuming and error prone task because of the limited input capabilities of mobile devices. We underlined through our initial user test that filling in forms automatically reduces the input time by the factor 3.

In this paper we presented a solution for automatic form filling on mobile devices. The basic idea is that a proxy running on the mobile phone uses the locally available user data to fill out forms on mobile services. This proxy acts as a mediator between the browser on the mobile phone and the mobile service on the network. Through this approach our solution can be used with already existing mobile phones, mobile browsers and mobile services. Another advantage is the optimization of the algorithm memory and processing constraints of mobile devices based on an initial rules set and dynamic rules generation. Furthermore we showed through our prototype that it is feasible to implement and use such a feature with currently available mobile phones.

From the results of our initial user study we intensively considered privacy aspects during the development of our architecture. Therefore the user data is just stored on the mobile device and is not distributed to any server. Furthermore the user can keep control over the automatically filled data by viewing and reediting them before submitting the form. In addition to that, we showed through the accuracy evaluation of the algorithm that considering the co-location of form elements (e.g. labels and input fields) is necessary for improving the correctness of the automatic form filling. In addition we noticed that the accuracy of our form filling algorithm was similar to corresponding solutions such like the Google Toolbar and the MSN Search Toolbar which are not primary designed for use on mobile devices. Through a user test we found out that people understood and well-perceived the concept of automatic form filling on mobile devices. Furthermore we presented in our related work section similar approaches but also underlined that no comparable concept or tools for automatic form filling on mobile devices was available.

To show the efficiency of our system from a different perspective, we plan to compare it with some of input prediction engines such as T9 that have been implemented on mobile devices. In a further user study we will count the time and the number of keystrokes which are needed to delete or reedit user data filled by the automatic form filling tool. Furthermore we will analyse how more complex forms which require some scrolling interactions influence our current results. Furthermore we plan to introduce and evaluate a technique to delete all data filled in a form field at once.

The present research focuses on user data that is usually managed by typical personal information management applications, in particular the address book. But novel and future mobile services may provide form elements that ask for location or activity information. The previously discussed algorithm can probably be used for filling in trivial location information (e.g. current location) but is certainly not the ideal solution for more sophisticated services. As discussed in the related work section would the usage of semantic web technologies be the most obvious solution for this problem. The question here is whether and when this will be used by most of the developers when creating new forms.

8. Acknowledgements

This work was performed in the context of the framework of IST Project Simple Mobile Services (SMS) funded by the EU. The authors wish to express their gratitude to the other members of the SMS Consortium [26] for valuable discussions.

9. REFERENCES

[1] Chie Noda, John Hamard, Enrico Rukzio, Alexander De Luca. Method and Apparatus for Automatic Form Filling on Mobile Devices. Patent. Publication number EP1777629, Publication date 2007-04-25.

[2] Opera Mobile, http://www.opera.com/products/mobile/

[3] NTT DoCoMo Osaifu-Keitai, http://www.nttdocomo.co.jp/english/service/osaifu/

[4] FeliCa, Sony, http://www.sony.net/Products/felica/

[5] E. Rukzio, A. Schmidt, H. Hussmann. Privacy-enhanced Intelligent Automatic Form Filling for Context-aware Services on Mobile Devices. Workshop Artificial Intelligence in Mobile Systems 2004 (AIMS 2004) in conjunction with UbiComp 2004, Nottingham, UK, September 7 2004.

[6] I. MacKenzie and R. Soukoreff, „Text entry for mobile computing: Models and methods, theory and practice", Human-Computer Interaction, 17, 147-198. 2002.

[7] J. Pierce and H. Mahaney, "Opportunistic Annexing for Handheld Devices: Opportunities and Challenges", Human-Computer Interface Consortium, 2004.

[8] M. Silfverberg, I. MacKenzie and P. Korhonen, „Predicting Text Entry Speed on Mobile Phones", Proceedings of the SIGCHI conference on Human factors in computing systems, The Hague, The Netherlands, ISBN 1-58113-216-6, pp. 9-16, 2000.

[9] P. Hallam-Baker, "Client Side Automated Form Entry", W3C Working Draft WD-form-filling-960416, http://www.w3.org/TR/WD-form-filling.html

[10] Google Toolbar, http://toolbar.google.com

[11] MSN Search Toolbar, http://toolbar.msn.com/

[12] Enrico Rukzio, John Hamard, Chie Noda, Alexander De Luca. Visualization of Uncertainty in Context Aware Mobile Applications. 8th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI 2006). Espoo, Finland, 12.-15.09.2006.

[13] T. Chusho, K. Fujiwara and K. Minamitani, "Automatic Filling in a Form by an Agent for Web Applications", Asia-Pacific Software Engineering Conference 2002, IEEE Computer Society, pp.239-247, 2002.

[14] J. Barton, T. Kindberg, H. Dai, N. Priyantha and F. Al-bin-ali, „Sensor-enhanced Mobile Web Clients: an XForms Approach", Proceedings of the twelfth international conference on World Wide Web, ISBN 1-58113-680-3, Budapest, Hungary, pp. 80-89, 2003.

[15] RoboForm, http://www.roboform.com/

[16] iOpus Internet Macros, http://www.iopus.com

[17] S. McIlraith, T. Son and H. Zeng, “Semantic Web Services”, IEEE Intelligent Systems, 16(2):46-53. 2001.

[18] G. Abowd and A. Dey, “Towards a Better Understanding of Context and Context-Awareness”, in: Technical Report GIT-GVU-99-22, College of Computing, Georgia Institute of Technology, pp. 12, 1999.

[19] P. Korpipää, J. Mäntyjärvi, J. Kela, H. Keränen and E-J Malm, “Managing context information in mobile devices”, IEEE Pervasive Computing 2(3):42-51. 2003.

[20] G. Rossi, D. Schwabe and R. Guimar, “Designing Personalized Web Applications”, Proceedings of the tenth international conference on World Wide Web, Hong Kong, ISBN 1-58113-348-0, pp. 275-284, 2001.

[21] D. Billsus, C. Brunk, C. Evans, B. Gladish and M. Pazzani, “Adaptive interfaces for ubiquitous web access”, Communications of the ACM 45/5, pp. 34-38, 2002.

[22] L. Suryanarayana and J. Hjelm, “Profiles for the situated web”, Proceedings of the eleventh international conference on World Wide Web, Honolulu, Hawaii, USA ISBN 1-58113-449-5, pp. 200-209, 2002.

[23] C. Efstratiou, A. Friday, N. Davies and K. Cheverst, “A Platform Supporting Coordinated Adaptation in Mobile Systems”, Proceedings of the 4th {IEEE} Workshop on Mobile Computing Systems and Applications (WMCSA) 2002, pp 128-137, 2002.

[24] C. Efstratiou, A. Friday, N. Davies and K. Cheverst, “Utilising the Event Calculus for Policy Driven Adaptation in Mobile Systems”, Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), 2002.

[25] L. Kagal, T. Finin and A. Joshi, “A Policy Language for a Pervasive Computing Environment”, IEEE 4th International Workshop on Policies for Distributed Systems and Networks, Lake Como, Italy, pp. 63. 2003.

[26] Simple Mobile Services (SMS) project, http://www.ist-sms.org/