MobiDSL - a Domain Specific Language for Mobile Web Applications

: developing applications for mobile platform without web programming

Ankita Arvind Kejriwal
Birla Institute of Technology and Science, Pilani.
Goa Campus, India
kejriwal.aa@gmail.com

Mangesh Bedekar
Birla Institute of Technology and Science, Pilani,
Goa Campus, India
bedekar@bits-goa.ac.in

Abstract

The enormous potential of mobile web as an information appliance presents all organizations an urgent need and a compelling reason to not only create mobile specific versions of certain parts of their current systems, but also develop new mobile web applications to derive maximum benefit from this medium. However, as mobile web applications are generally being built using the same web engineering methodologies and tools which are used for building desktop web applications, organizations around the world require significant resources, making it difficult for many to quickly build these applications. In this paper, we describe our aim to mitigate this problem by using a Domain Specific Modeling (DSM) based approach. We explain MobiDSL - a Domain Specific Language (DSL) for modeling Mobile Web Applications and show how it can enable system designers and analysts to easily define an application's specification at a very high level of abstraction without any web programming. We also explain how a Virtual Machine (VM) is used to execute MobiDSL models, which helps to radically simplify the testing, deployment and life cycle management of such mobile web applications.

1. Introduction

Mobile Web refers to web content (static or dynamic) which is specifically designed to be accessed on mobile devices via a browser [2]. The mobile web is emerging as an information medium whose reach is projected to surpass all other mass media (including the print, cinema, radio, television and desktop internet) as mobile devices have the advantage of being personal, portable, always on and connected. It presents us with a massive opportunity to provide information and e-services to entire human population, not just in developed countries but also in developing and under-developed regions of the world. It can be the best enabling mechanism to empower billions of people with information and to bridge the digital divide.

While it is possible to access standard web content on some mobile devices, the user experience is often less than satisfactory primarily due to smaller screen size and lower bandwidth compared to a desktop or laptop. Therefore, Mobile Web Best Practices (MWBP) [2] specifies practices for delivering web content to mobile devices. These recommendations refer to the 'content' and not to the processes by which the content is created or delivered. We find emergence of following trends in creation and delivery of web content to mobile devices.

1. The static web content is either being re-developed or being transformed at runtime by using server side software such as Instant Mobilizer [4].
2. The dynamic web content is being provided by a new class of re-engineered, light-weight mobile specific versions of corresponding desktop web applications. Some of the notable examples are dynamic content sites such as BBC and ESPN; and dynamic applications such as Gmail, Facebook, and Twitter.

For the purposes of this paper, we define *Mobile Web Application* as "a web application specifically designed to deliver dynamic information from the database and provide simple transactional services to mobile devices via a browser".

The rapidly growing popularity and success of many mobile web applications has demonstrated the enormous potential of mobile web as an information appliance. Organizations are faced with an urgent need as well as a compelling reason to not only create mobile specific versions of certain parts of their current systems (legacy, client-server or web applications) but also to develop new mobile web applications which derive maximum benefit from this medium.

Though mobile web applications are simpler compared to desktop web applications, they are generally being built using the same web engineering methodologies and tools which are used for building desktop web applications. Thus, organizations around the world would require significant resources to design, develop, test and deploy mobile specific versions of their current as well as new applications. As a result, it might be difficult for many to quickly develop these applications.

A light-weight development methodology for developing mobile web applications which could supplement and co-exist with whatever methodology an organization might be using for their core systems could help address the aforesaid issue. This methodology would also have to be simple enough so that it can be used directly by system designers and analysts to develop such applications.

We attempted to find a solution using the DSM based approach. We analyzed the mobile web domain to identify various constructs and designed a simple, compact and extensible DSL called MobiDSL for defining mobile web applications. Though DSM based approaches usually involve generation of code which can be compiled and run, we chose to develop a Virtual Machine (VM) to execute MobiDSL models due to various benefits which are described later. Our approach aims to provide a reasonably sound framework for rapid prototyping, development, testing, deployment and life-cycle management of mobile web applications without the need of any web programming or provisioning of any special middleware.

The rest of the paper is structured as follows. Section 2 describes the background and related work. Section 3 describes

our approach, objectives and the VM. It also explains MobiDSL with a few succinct examples and finally provides a brief overview of MobiDSL's meta-model. In Section 4, we discuss various issues and describe the benefits of interpretive approach over generative approach. Section 5 presents a summary of contributions and future work.

2. Background and Related Work

Mobile internet access began with the Wireless Access Protocol (WAP), wherein the pages were composed in Wireless Markup Language (WML). The advent of WAP 2.0 permitted use of XHTML markup with end-to-end HTTP. To assist development community, W3C has launched *Mobile Web Initiative* [1] and published standards such as Mobile Web Best Practices (MWBP), XHTML for mobile (XHTML-MP), Cascading Style Sheet for mobile (CSS-MP), and MobileOK Basic Tests.

Mobile Web applications are generally being built using the same Web Engineering methodologies which are used for building regular web applications. Most methodologies follow a standard three tier architecture as described below:

a) A database tier – which stores the database and runs the database server
b) An application tier – which runs the actual application logic. The application server receives the user inputs from the client device via HTTP (over TCP/IP) and can use various API's to interface with HTTP server. It can interface with database server using various database API's to retrieve information or update transactions.
c) The client tier – which runs the application using a web browser. The browser renders pages composed in HTML. CSS is used to specify visual aspects rather than specifying them in HTML for each hypertext element. The client tier can also use technologies such as DHTML (to dynamically construct HTML fragments), JavaScript (to handle events) and Ajax (to asynchronously access the application tier).

There is a plethora of Web Engineering methodologies and frameworks ranging from scripting technologies such as ASP, JSP and PHP (to name a few) to enterprise class web technologies such as J2EE. There are also sophisticated web modeling based frameworks such as HDM, WAE, WSDM, UWE and WebML. The scripting based systems provide API for interfacing with HTTP server and Database Server and allow the developer to hand code the programs. Many systems also provide tools and templates to automate generation of boiler-plate code. The J2EE is a sophisticated enterprise class technology which divides application server into partitions, which can be run on different machines to achieve high degree of scalability. The modeling based frameworks provide facilities for various facets of modeling such as Content Modeling, Hypertext Modeling, Presentation Modeling and Customization Modeling. Some of them also provide tool support to partially or fully generate application code. There are also some XML based frameworks employing XML transformation (XSLT) such as Apache Cocoon.

However, creating a robust and scalable mobile web application using any of the above methodologies is not a trivial task. Many researchers have advocated use of *Domain Specific Language based frameworks* to develop robust, reliable and secure programs in a cost effective manner by using high level abstractions. We were inspired by the reported success of DSM approach in many application areas [5, 6] and the use of DSL paradigm in web engineering by some researchers such Nunes et al. [7], Martin et al. [8], E. Visser [9] and Ceri et al. [10].

We were particularly influenced by *Web Modeling Language (WebML)* proposed by Ceri et al. [10], which stands out for enabling high level abstraction of various models such as structural model, composition model, navigation model, presentation model, and customization model. However, whereas WebML's meta-model is very elaborate, we have combined the models into one simple model. Moreover, whereas WebML's tool generates the application code, we have implemented a VM to execute the model. Such a concept of executable DSL models has already been demonstrated in ModelTalk [11].

3. MobiDSL

3.1 Overview of Mobile Web Applications

The Mobile Web Applications aim at providing a compelling user experience. We find evolution of new design philosophy for mobile web, whose central theme is based on simple and minimalistic design concept [3].

A mobile web application consists of various web pages, which can be broadly classified as Home Page, Query Pages and Transaction Pages. These pages are hyperlinked to each other to enable navigation thorough the system. The links can be either Contextual Links (which pass certain contextual information while navigating to another page) or Non-Contextual Links.

The Home Page typically provides certain static choices (menu options). It may contain an authentication section to allow only authorized users to access the application. It can also contain choices created dynamically based on certain information in the database.

The Query Pages enable the users to access information from database based on some selection criteria. The selection criteria can be either based on search parameter inputs defined in the same page or query string passed to the page by a contextual link. The Query Pages can either display several records (List View) or a single record (Record View).

The Transaction Pages enable the mobile users to update simple transactions on the database. These transactions typically allow Create, Update and Delete facility on records in a specified table. They allow elementary validations to be performed on the data entered by users.

Some mobile applications also require special features such as Access Control for restricting access to some pages to a certain group of users based on their user role. Some applications might require personalization features, wherein user's preferences such as presentation skins, accessibility preferences and favorites are stored in a database table, and web pages are created accordingly.

The mobile web pages typically have different sections as:

- Page Header - which may consist of Branding (Logo, Name), Title and Navigation Tree
- Body - which can consist of menu options, search request section, query view section or transaction entry section
- Page Footer - which may consist of Branding, certain links or other relevant information

We find that Mobile Web Applications differ from regular Web Applications in several ways which are summarized below:

1. Device limitations - screen/keypad size and bandwidth
2. Different Usage pattern
3. Minimalistic design with Simple Layouts (no frames or nested tables or complex layouts)
4. No processing on client devices using DHTML, JavaScript or Ajax, as this would render the application unusable on a majority of mobile devices.
5. Lesser functional expectations

3.2 Our Approach

The simple nature of mobile web application presents an opportunity to create new lightweight frameworks suitable for this emerging medium. Our approach to creating a new framework for mobile web applications was based on the promise of DSM approach and on the premise that the mobile web domain can be analyzed to identify the core requirements and various design elements, based on which a domain specific language (DSL) can be designed. DSL is "a high-level software implementation language that supports concepts and abstractions that are related to a particular (application) domain" [9].

The driving factor for identification of language constructs was primarily based on "Domain Expert's and Developer's Concepts" approach [5]. Based on our earlier work in developing mobile web applications using scripting methodologies, we identified distinct features and recurring themes in these applications. We also discussed with an industry expert to understand the domain expert's concepts on creating mobile versions of current systems without any programming. We were also partly led by "look and feel" approach [5] as far as the user interface (hypertext) was concerned.

We have designed a simple and concise textual DSL suitable for developing mobile web applications called MobiDSL, which enables a developer to define the specifications for each page at a very high level of abstraction without requiring any knowledge of web programming. We have used XML as the Meta Language as XML markup is simple, flexible and easily understood. MobiDSL allows us to define:

1. Page structure
2. Hypertext (text, widgets and links) in each section
3. SQL for data retrieval
4. Query result presentation
5. Validations and other business logic for transactions

3.3 Objectives of our framework

Our framework aims to:

1. Simplify mobile web application development by allowing designers, analysts and programmers to define application at a high level of abstraction
2. Allow the application to be run on most mobile devices by carrying out all processing and validations on server and using only XHTML-MP (with CSS-MP) on client device
3. Simplify deployment by adopting a simple three tier distributed RESTFUL architecture, which employs only basic protocols (such as HTTP) and basic API's (such as CGI and Pass Through SQL)
4. Provide scalability by allowing multiple application servers wherein a request could be serviced by any of them.
5. Optimize processing by caching to retrieve result sets of recently executed queries
6. Avoid middleware complexity normally seen in many high end web applications

3.4 The Virtual Machine

The Virtual Machine (VM) runs the mobile web application as embodied in the MobiDSL Model. It is implemented as a stateless, distributed and scalable Application Server. A schematic view of the MobiDSL VM deployment is shown in figure 1. VM uses the configuration information in sys.xml file to connect to Database Server and Memcached. On one side, the VM interfaces with HTTP server encapsulating the CGI API. On the other side, it interfaces with database and encapsulates the database access API. When the VM receives client input it identifies the page to be delivered based on page identifier in query string. It parses the corresponding MobiDSL model (<pageid.xml>), runs database queries based on search request and constructs the response XHTML page (with embedded mvm.css) as specified in model. In a nutshell, it handles complete server side processing for the mobile web application (encapsulating various web engineering technologies) based merely on the application's MobiDSL model. The VM also manages Sessions, Pagination, and a Navigation tree. The response pages created by VM are compliant with mobileOK Basic Tests.

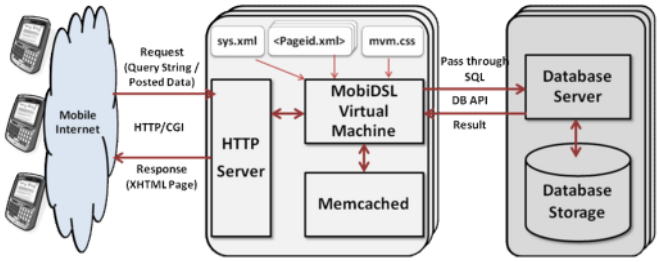

Figure 1. A Schematic View of MobiDSL VM Deployment

We have implemented this system on classic Linux-Apache-MySQL-PHP platform. The implementation follows a distributed three tier model, allowing any number of application servers to be connected to database server. A server based session management stores the session data in Memcached on server side, allowing that information to be retrieved from any application server. This makes each request-response cycle completely stateless, making it possible for each user interaction to be serviced by a different application server, making the system scalable and reliable. The VM also caches the result set of recently executed queries to optimize processing.

3.5 Sample Application

To understand MobiDSL, let us consider an example of a Mobile Web Pharma Sales Force Application used by Medical Sales Representatives (MSR). One of the tasks of the MSR is to visit various physicians (in the towns assigned to her) periodically to brief them about the company's products. The application enables an MSR to view the list of physicians and details of previous visits. It also enables her to record details of a new visit on-line. A simplified content model of relevant parts of the application is shown in figure 2.

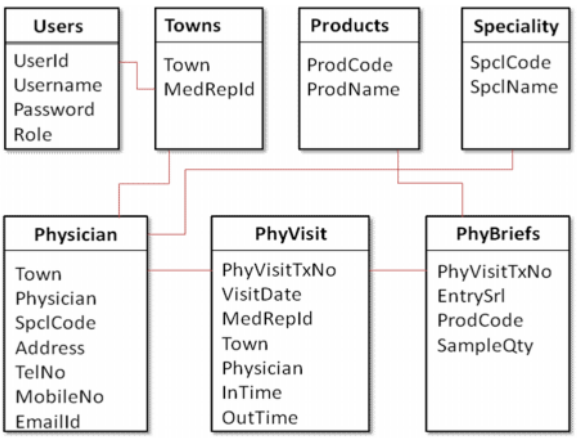

Figure 2. Content Model of Pharma Sales Force Application

3.6 The Physicians List Query

We first consider the physicians list query page/screen shown in figure 3. This screen enables the MSR to view a list of all physicians in the towns assigned to her. Moreover, it also allows her to optionally search based on part of name, specialty code or town.

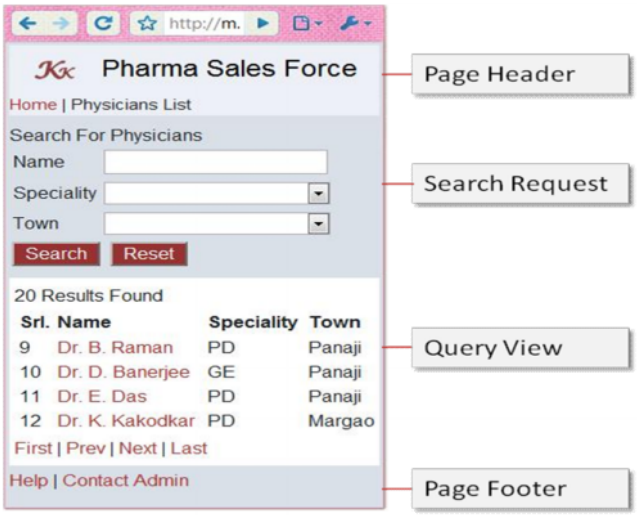

Figure 3. The Physicians List Page

The specification for this page as defined in MobiDSL is shown in figure 4. The root element `page` identifies the page and contains other elements such as `pageheader`, `searchrequest`, `queryview` and `pagefooter` corresponding to each section in the page.

The MobiDSL code for Page Header specifies a static image (stored on server in directory relative to location of VM), the page header title, a link back to the home page and the current page's title. Most of these specifications are self-explanatory.

The code for Search Request section specifies a text line, input widgets for search criterion and Submit and Reset buttons. The SQL statements are used to specify the options for input widgets such as `Specialty` (all specialties to be shown) and `Town` (only those towns which are assigned to MSR to be shown).

The code for Query View section specifies the SQL to be executed for retrieving required data from database based on the search criterion. Here, the SQL specifies the physician's list by using a join of the `Physician` table with the `Towns` table (for only those towns assigned to a MSR), and further filtering records based on search criterion. The search inputs submitted by the client device are stored in variables with names corresponding to input widget name prefixed by $. Using caret (^) in expressions such as `a.town=$town^` implies that if the input stored in `$town` is empty, the expression is to be ignored (reduced to logical true by VM). The data retrieved from the database using SQL can be presented in various layouts such as `para`, `dualcol` and `table`. The `resultrecord` element specifies presentation for each record and can contain one or more `resultcol` elements (data columns). A `resultcol` element can also specify a link which is invoked when user clicks on that column's value. It can pass that value (as well as any other value using `passvalues` attribute) as query string to the resource pointed to by it.

```
<?xml version='1.0' standalone='yes'?>
<page id="phylist">
  <pageheader>
    <image src="images/psf.gif" nobreak="true"/>
    <text class="title">Pharma Sales Force</text>
    <text href="mvm.php?pageid=home"
          nobreak="true">Home</text>
    <text expr="' | '" nobreak="true"/>
    <text>Physicians List</text>
  </pageheader>
  <searchrequest>
    <text>Search for Physicians</text>
    <input type="text" label="Name" name="physician">
    <input type="select" label="Speciality"
       name="spclcode"
       optionsql="select spclCode from specialty"/>
    <input type="select" label="Town" name="town"
       optionsql="select town from towns where
                  medrepid=$_userid"/>
    <submit label="Submit" />
    <reset label="Reset" />
  </searchrequest>
  <queryview layout="table" recordsperpage="4">
    <sql>select a.* from physician a, towns b
         where a.town=b.town and b.medrepid=$_userid
         and match(a.physician) against ($physician^)
         and a.spclcode=$spclcode^ and a..town=$town^
         order by physician</sql>
    <text expr=" $_reccount . 'Results Found' "/>
    <resultrecord>
      <resultcol label="Name" sqlcol="physician"
         href="mvm.php?pageid=phydet"
         passvalues="town;physician"/>
      <resultcol label="Speciality"
         sqlcol="spclcode"/>
      <resultcol label="Town" sqlcol="town" />
    </resultrecord>
  </queryview>
  <!-- pagefooter code omitted -->
</page>
```

Figure 4.The Specification for Physicians List Page in MobiDSL

3.7 Key Concepts

Sequence of Events and Processing. Though some MobiDSL element tags and properties might appear to be similar to HTML tags, they are not HTML tags. Whereas HTML is processed by the browser on the client device, MobiDSL code is processed by the VM (on Server) to handle client device inputs or create HTML output pages accordingly. The following points summarize the sequence of events and processing for the previous query page.

1. When this page is requested (as mvm.php?pageid=phylist) for the first time, the VM creates initial XHTML page with a list of all physicians in all towns assigned to the MSR (as all search inputs are empty at that time) in following steps:
   a) VM reads MobiDSL code for target page (`phylist`) and parses it. It saves parsed DSL in cache for re-use.
   b) VM begins to construct the XHTML page by generating the `<head>` section with page title and embedding the CSS file as inline `<style>`.
   c) VM then generates HTML for Page Header.
   d) Then, VM generates HTML for Search Request section. If the section contains any Select widgets, it populates options for these widgets from result sets of corresponding `optionsql`. The HTML for this section is embedded in `<form>` tag. VM also embeds control information like `pageid` & `sessionid` as hidden fields.
   e) To create HTML for Query View section, the VM first prepares the SQL statement (by substituting the values of variables used in SQL) and submits it to database server. The VM fetches the result set and constructs HTML for presenting the data in the specified manner.
   f) Then, VM generates the HTML for Page Footer section.
   g) It finally sends the fully constructed XHTML-MP page

with embedded CSS to the client device.

2. The browser on the client device loads the XHTML page. The user can see list of first four physicians. The user can use `Next` link to request the server to send a page with next set of records. The user can also select the link associated with physician name to navigate to Physician's Details Page. The user can also enter values for some search criteria and press `Submit` to request the server for a filtered set of data.
3. When the user submits the search criteria, it is posted to the VM. The VM retrieves control information such as `pageid` and `sessionid` from the posted data. It loads the parsed MobiDSL code from cache based on `pageid`. It retrieves search inputs from posted data and cleans it to guard against SQL injection attacks. The VM then prepares the SQL statement as explained earlier and submits it to the database server. It then fetches the result set and re-constructs the XHTML page and sends it back to the client device.
4. When the user selects any of the pagination link (`First`, `Prev`, `Next` or `Last`), a query string is sent to VM with appropriate information. The VM receives the query string and processes it in a manner similar to (3). The VM fetches the result set from cache and reconstructs the XHTML page with relevant records.

Variables. MobiDSL gives us the flexibility to refer to various variables in SQL statements or other expressions. These variables are created and managed by VM automatically in their respective context; and can be referred to in the specifications in appropriate contexts. In a nutshell, these variables are:

1. Authentication variables: These are variables like $_userid and $_userrole. They have global scope.
2. Query String Variables: Name-value pair collections from a contextual link are stored in variables with corresponding names prefixed by $. For example, if a query string is `field1=value1&field2=value2`, then two variables, `$field1` and `$field2` will be created. These variables are in scope of the page in which they are received.
3. Inputs Fields in a Page: The values received as posted data from a page are stored in variables with corresponding names of input fields prefixed by $. These variables are in scope of the page in which they are received.
4. SQL Query Result Set Control Variable: the `$_reccount` variable gives the count of records retrieved by query. It is in scope of `queryview` section.
5. SQL Query Result Record Level Variables: a) `$_currec` which gives the current record number. b) Result Columns for current record stored in variables with the column names prefixed by `$_sqlres_`. These variables are in scope of `resultrecord` specifications.

Expressions. In MobiDSL specifications, we can use expressions to define some attributes of various elements. These expressions can be any PHP expression comprising any PHP functions (in-built functions, library functions or user-defined functions) and any of the variables available in the given context. The ability to use expressions in several attributes such as following in the MobiDSL specification enables the developer to define various requirements with ease:

1. `expr` in text and `resultcol` elements can be used to define an expression to display required string or value
2. `hrefexpr` in image, text or `resultcol` elements to define an expression for a link associated with that element
3. various attributes such as `defvalexpr`, `disableifexpr`, `validexpr` etc. for Input field elements in a transaction

It is to be noted that the developer can provide additional functionality by developing application specific PHP functions which can be used in expressions. This feature makes MobiDSL reasonably extensible while keeping its core grammar to a minimum.

Pass Through SQL. MobiDSL allows us to define a pass through ANSI SQL to retrieve any data from the database. The SQL can be of any complexity involving any number of tables and can contain MobiDSL variables. The MobiDSL VM prepares the SQL statement by substituting the values of variables and then submits it to the database server in a pass through fashion. The SQL is executed as such by the database server and the result set is sent back to the VM. The SQL can be used in following contexts in the MobiDSL specification:

1. In `queryview` section to retrieve the data to be presented.
2. In `optionsql` attribute for select input widgets to populate the select options.
3. In `pageheader` and `pagefooter` to retrieve required control information from the database.

3.8 The Physicians Details Query.

We now consider the physicians details query page/screen shown in figure 5. This screen enables the MSR to not only view contact details, but also call or email the physician using a single click. It shows a list of previous visits to the physician and allows the MSR to view any previous visit transaction. It also allows her to initiate a new Visit Transaction.

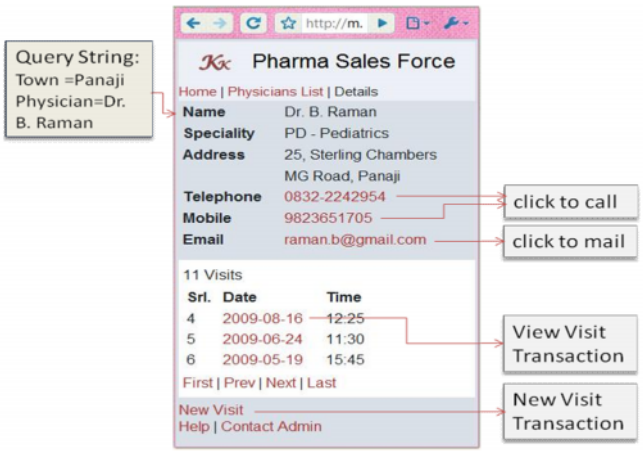

Figure 5. The Physicians Details Query Page

The specification is shown in figure 6. The `queryparams` specify query string parameters received by this page. The page contains two `queryview` sections:

1. Physician's Contact Details (Single Record View) - Here, the SQL specifies the desired selection from the physician table using query string parameters. The layout is set to be a two column layout where the first column shows label and second column shows data. The Telephone, Mobile and Email values are rendered as links using `hrefexpr` which allows us to specify PHP expression for defining the link.
2. Visit Details (List View) - The SQL specifies selection from `phyvisit` table. The layout is set as table showing the visit date and time. The visit date is rendered as a link to let the user to navigate to Visit Transaction in view mode.

```
<?xml version='1.0' standalone='yes'?>
<page id="phydet">
  <queryparams>
    <param name="town"/>
    <param name="physician"/>
  </queryparams>
  <!-- pageheader code omitted -->
  <queryview multirecord="false" layout="dualcol">
    <sql>select a.*,b.spclname
         from physician a,Speciality b where
         a.spclcode=b.spclcode and a.town=$town
         and a.physician=$physician</sql>
    <resultrecord>
     <resultcol label="Name" sqlcol="physician"/>
     <resultcol label="Speciality"
     expr="$_sqlres_spclcode.'-'.$_sqlres_spclname"/>
     <resultcol label="Address" sqlcol="address"/>
     <resultcol label="Telephone" sqlcol="telno"
        hrefexpr=" 'tel:'. $_sqlres_telno "/>
     <resultcol label="Mobile" sqlcol="telno"
        hrefexpr=" 'tel:'. $_sqlres_mobileno "/>
     <resultcol label="Email" sqlcol="telno"
        hrefexpr=" 'mailto:'. $_sqlres_emailid "/>
    </resultrecord>
  </queryview>
  <queryview layout="table" recordsperpage="5">
    <sql>select * from phyvisit where
         town=$town and physician=$physician
         order by visitdate desc</sql>
    <text expr=" $_reccount . 'Visits' " />
    <resultrecord>
      <resultcol label="Date" sqlcol="visitdate"
         hrefexpr="'mvm.php?pageid=phyvisit'.
                   '&amp;txmode=view'"
         passvalues="phyvisittxno"/>
      <resultcol label="Time" sqlcol="intime" />
    </resultrecord>
  </queryview>
  <!-- pagefooter code omitted -->
</page>
```

Figure 6. The Specification for Physicians List Query Page

3.9 The Physicians Visit Transaction

Now we consider Physicians Visit Transaction page/screen as shown in figure 7. This screen enables the MSR to enter details of her visit including the products that she briefed to a physician.

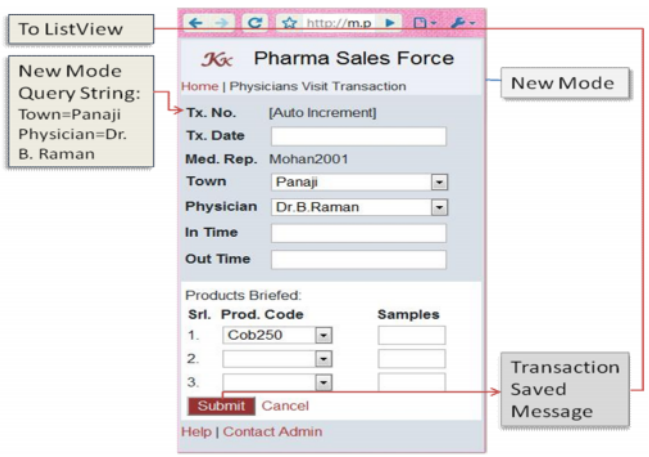

Figure 7. The Physicians Visit Transaction Page

The specification for this page is shown in figure 8. As before, the `queryparams` specify query string parameters received by this page. While the query parameter `phyvisittxno` is passed to load the transaction in View Mode, the `town` and `physician` parameters are passed in New Mode to serve as default values for these fields. This page contains a `simpletxn` section which contains two transaction blocks (identified by `txnblock`):

1. The first transaction block corresponds to `phyvisit` table. This block is defined as a parent block with single record set in dual column layout. While the `tablekeys` specify primary keys of the table, `loadkeys` define corresponding variables whose values should be used to retrieve the transaction in View or Edit Mode. The block contains several fields, the first of which is Tx. No, whose datatype is defined as `autoincr` (value of field to be generated by database server while saving the record). The next field is Tx. Date, which has default value of current date. The Med. Rep. field is a protected field with a default value of MSR's `userid`. Then, we have more input fields whose specifications are self-explanatory.
2. The second transaction block corresponds to `phybriefs` table. This block is defined as a child block with multiple records set in tabular layout. The first field here is Prod. Code which has a foreign key validation against `products` table. The second field is `Samples` which is optional.

While the transactional model uses *Controls* such as `Submit` button and `Cancel` link in New Mode, it uses `Edit` link, `Delete` link and `Back` link in View Mode. The behavior of these controls is fixed and the developer can only specify alternate labels for these controls.

```
<?xml version='1.0' standalone='yes'?>
<page id="phyvisit">
  <queryparams>
    <param name="town"/>
    <param name="physician"/>
    <param name="phyvisittxno"/>
  </queryparams>
  <!-- pageheader code omitted -->
  <simpletxn>
   <txnblock blocktype="parent" multirecord="false"
       layout="dualcol" tablename="phyvisit"
       tablekeys="phyvisittxno"
       loadkeys="$phyvisittxno" />
     <input type="text" label="Tx. No."
         name="phyvisittxno" datatype="autoincr"
         disableifexpr="1"/>
     <input type="text" label="Tx. Date"
         name="visitdate" datatype="date"
         defvalexpr="date('Y-m-d')"/>
     <input type="text" label="Med. Rep."
         name="medrepid" datatype="char" size="20"
         defvalexpr="$_userid" disableifexpr="1"/>
     <input type="text" label="Town"
         name="town" datatype="char" size="20"
         defvalexpr="$town" disableifexpr="1"/>
     <input type="text" label="Physician"
         name="physician" datatype="char" size="30"
         defvalexpr="$physician" disableifexpr="1"/>
     <input type="text" label="In Time" name="intime"
         datatype="time" size="5"/>
     <input type="text" label="Out Time"
         name="outtime" datatype="time" size="5"
         valdexpr1="$outtime > $intime"
         valdmsg1="OutTime must be morethan InTime"/>
   </txnblock>
   <txnblock blocktype="child" multirecord="true"
       layout="table" tablename="phybriefs"
       tablekeys="phyvisittxno"
       loadkeys="$phyvisittxno" />
     <title>Products Briefed</title>
     <input type="select" label="Prod. Code"
         name="prodcode" datatype="char" size="10"
         optionsql="select prodcode from products"
         fkeytable="products"
         fkeytablefields="prodcode"
         fkeyvaluefields="$prodcode"
         fkeyerrmsg="Invalid Product"/>
     <input type="text" label="Samples"
         name="sampleqty" datatype="num" size="2"
         required="false"/>
   </txnblock>
  </simpletxn>
  <!-- pagefooter code omitted -->
</page>
```

Figure 8. The Specification for Physicians Visit Transaction

3.10 MobiDSL Metamodel

We can see that MobiDSL has very high expressive power as it allows us to specify page structure, presentation (static/dynamic text, widgets), navigation, data retrieval, data formatting and transactional logic at almost same level of expression as required in communicating the specifications to a programmer. Further, the use of pass-through SQL enables system designers, analysts and programmers to leverage their knowledge to develop mobile web applications with relative ease without any web programming. Moreover, MobiDSL allows extensibility by allowing developers to use any user-defined function in various expressions. Figure 9 presents a simplified view of MobiDSL metamoddel, which depicts various design elements/constructs at a glance.

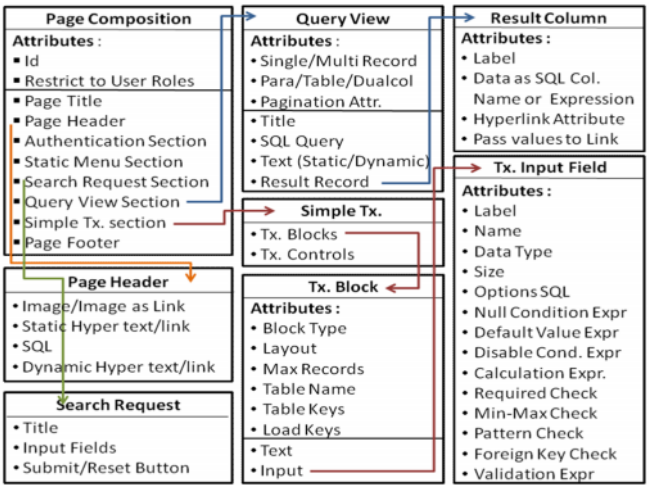

Figure 9. A simplified view of MobiDSL metamodel

4. Discussions

4.1 Application Life-Cycle Management

MobiDSL provides a reasonably sound framework for complete life-cycle management of mobile web applications:

Prototyping: As the MobiDSL specifications are at very high level almost mirroring the functional requirements, it is possible to create working prototypes using MobiDSL in similar time that might be needed to create a prototype using any prototyping tool.

Development: MobiDSL is compact DSL designed specifically for mobile web applications. The constructs provided in the DSL along with the ability to use pass through SQL and user defined functions make it reasonably adequate to cover most of the needs.

Testing: MobiDSL code, being declarative in nature, is far easier to debug than a procedural code. Moreover, the VM allows quick testing as the changes are reflected immediately in the application. This can save considerable time as the change-compile-build-deploy cycle is eliminated.

Deployment: The VM uses basic protocols (HTTP) and basic API's (CGI, pass through SQL). As a result, it can be deployed on commodity hardware or most of the existing infrastructure in an organization. It provides scalability by allowing multiple application servers, which can be added or removed without need to shut down the application. Finally, the VM does not require any special middleware typically seen in many high-end applications.

4.2 Generative vs. Interpretive Approach

We believe that a carefully crafted implementation of a VM can offer several benefits at speeds matching that of the generated code. Whereas generated code needs to be maintained, versioned, compiled and installed on an application server, these tasks are eliminated in the interpretive approach, leading to easier deployment and maintenance of the application.

5. Conclusions

Contribution. In this paper, we have looked into the question of how we can simplify the development, deployment and life cycle management of mobile web applications. The question is important because mobile web is a fast growing information delivery medium and it is vital for organizations to quickly develop systems for mobile platform with least effort. Our main contributions can be summarized as follows:

1. Identifying core requirements and design elements of mobile web applications
2. Designing a DSL for defining the complete specifications of a Page/Screen
3. Incorporating the concept of using SQL in DSL for Queries
4. Creating a VM to support the DSL

This research has resulted in development of a lightweight framework consisting of MobiDSL and its associated VM. It has been tested extensively by us, and was found to perform as per our design expectations on parameters such as coverage of problem domain, ease of development and deployment, speed of execution, scalability and reliability. This framework is also being used in industry to create mobile specific versions of certain parts of their enterprise application.

Future Work. MobiDSL being a nascent framework, is evolving continuously. Future work includes enhancing the DSL to increase functionality, providing client-side validations using JavaScript (based on device capability) and conducting a detailed comparative study of various metrics with respect to other popular web engineering methodologies.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. We also thank all those who have helped in this work.

References

[1] Mobile Web Initiative. Available: http://www.w3.org/Mobile/
[2] Mobile Web Best Practices1.0. Available: http://www.w3.org/TR/2008/REC-mobile-bp-20080729/.
[3] dotMobi Mobile Web Developer's Guide. Available: http://mobiforge.com/starting/story/dotmobi-mobile-web-developers-guide.
[4] Instant Mobilizer. Available: http://www.instantmobilizer.com/
[5] Janne Luoma,,Steven Kelly, Juha-Pekka Tolvanen : Defining Domain-Specific Modeling Languages: Collected Experiences. Available: http://www.metacase.com
[6] Arie van Deursen, Paul Klint, Joost Visser: Domain-Specific Languages: An Annotated Bibliography. SIGPLAN Notices 35(6): 26-36 (2000)
[7] Nunes, D. A.; Schwabe, D. : Rapid Prototyping of Web Applications Combining Domain Specific Languages and Model Driven Design. in 6th International Conference on Web Engineering (ICWE'06), ACM Press, Jul. 2006.
[8] Martin Nussbaumer, Patrick Freudenstein, Martin Gaedke: Towards DSL-based web engineering. WWW 2006: 893-894
[9] Eelco Visser: WebDSL, "A Case Study in Domain-Specific Language Engineering," in GTTSE 2007: 291-373
[10] Stefano Ceri, Piero Fraternali, Aldo Bongio: Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 33(1-6): 137-157 (2000)
[11] Atzmon Hen-Tov, David H. Lorenz, Lior Schachter: ModelTalk: A Framework for Developing Domain Specific Executable Models. CoRR abs/0906.3423: (2009)