Framework Enabling End-Users to Maintain Web Applications

Masayuki Nii, *Member, IAENG,* Kenji Tei, and Fuyuki Ishikawa

***Abstract*—INTER-Mediator is a framework for developing web applications and it can be applied to the IT systems of small organizations that do not have large budgets. Web pages based on INTER-Mediator can be synchronized with a database simply by using declarative descriptions, and no imperative code is required as far as their binding to a database goes. Although software engineers should do the initial development, end-users and web designers can be involved in later parts of the development, especially in the maintenance phase because certain modifications can be handled within declarative descriptions. In this paper, we show that this framework enables end-users without specialized programming skills to perform maintenance on web applications. We explain the mechanism of binding and template processing and examine how easy it is for web designers to learn how to use INTER-Mediator. If non-programmers like end-users, designers, etc., can participate in the system development process, the total cost can be reduced, and small- and medium-sized organizations will have more opportunities to introduce web-based business systems.**

***Index Terms*—web, framework, database, declarative, context.**

I. Introduction

BUSINESS applications should use databases to store data effectively. They also require web interfaces to enable access from various devices. Frameworks following the model view controller (MVC) architecture are widely used in web-based development [1], and programmers should take part in development because all functions are implemented in imperative descriptions. MVC is a flexible architecture for implementing complex business logic. However, if an application is fully built using imperative descriptions, any modifications to it have to be done at the source code level. This means end-users can't make modifications themselves, and they have to consign the whole process of development to software engineers. As a result, end-users with a limited budget might give up maintaining their software after the initial development.

We propose a web-application framework called INTER-Mediator [2] to enable end-users in business to maintain their software. If end-users and web designers can take up some of the development tasks, it becomes possible to cut costs. To involve non-programmers, INTER-Mediator uses declarative descriptions to create web pages connected to databases. We suppose that software engineers would handle the design of the database schema and develop it from scratch. On the other hand, end-users whoa are non-programmers can take part in the maintenance process.

A declarative description is relatively easier for end-users to understand than an imperative one. Six barriers to learning have been identified as affecting imperative programming [3]. Designing is the biggest barrier, and it means that the imperative programming requires the programmer to build the right algorithm. For example, if a list of queried records is to be shown, a program that repeats for every queried record should be described. Otherwise, if the declarative description has the possibility to do the job, a sophisticated enough framework could produce the same algorithm automatically. Declarative descriptions eliminate the need for writing any kind of repeating program with a programming language, thereby lowering the design barrier and making the task easier for end-users to understand.

We envision using INTER-Mediator in small- and medium-sized companies, or in sections of a larger enterprise as a way of reducing total costs. We further assume that the staff at these companies would use Microsoft Excel to create documents and send them to their colleagues by email. This state of affairs would cause confusion sooner or later, because they wouldn't be able to easily identify which file is the latest among the files scattered amongst mail boxes, file servers, and local folders. INTER-Mediator is an effective replacement for the helter-skelter of Excel tasks.

INTER-Mediator has sufficient features for developing a variety of business systems, and it has been launched on the Japanese market and used in schedule management systems for hospitals, print order systems for on-demand printing, and communication systems among construction companies. A hosting service company in Japan has also developed a converter from a FileMaker database to web pages based on INTER-Mediator.

The remainder of this paper explains how INTER-Mediator can help end-users to perform maintenance. Section 2 summarizes how to create web pages with this framework. The suitability of the framework to making system modifications is discussed in section 3. The binding and repeating mechanism is explained in section 4. Section 5 described an experiment we conducted to see if INTER-Mediator is easy to learn and use. Related work is discussed in section 6. Section 7 is a conclusion.

II. INTER-Mediator at a Glance

INTER-Mediator is a web application framework for building web pages that are bound to databases with declarative descriptions. The first release was at the start of 2010, and it's distributed with an MIT License from GitHub [4]. The supported database engines are PDO (PHP Data Object)-supported relational databases such as MySQL and

Manuscript received December 30, 2014.

M. Nii is a Ph.D. Student at the Graduate School of Information Systems, the University of Electro-Communications, Chofu, Tokyo, Japan. e-mail: nii@msyk.net

Dr. Tei is an Assistant Professor at National Institute of Informatics, Chiyoda-ku, Tokyo, Japan. e-mail: tei@nii.ac.jp

Dr. Ishikawa is an Associate Professor at National Institute of Informatics and a Visiting Associate Professor at the University of Electro-Communications. e-mail: f-ishikawa@nii.ac.jp

ISBN: 978-988-19253-2-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

PostgeSQL, and FileMaker Server. The web server requires PHP Ver.5.2 or over. The supported web browsers are based on HTML5, and the minimum version of Internet Explorer is Ver.8[1]. The function set of web application from the end-user's view point consists of 31 items [5]. INTER-Mediator supports almost all of them (the one regarding sessions is not supported yet because the alternative is available) and the use of declarative descriptions covers 25 items in them.

A. Development Example

To explain how to use INTER-Mediator and how it works, we present a sample application[2] and its sources in this section. The sample application is a simple asset management system that records assets and updates their lending logs. The schema for the sample has three entities: "asset," "lent" and "staff."

Figure 1 is the asset list page. The assets are stored in multiple records, and the table on the page has multiple rows associated with each record. As shown in the figure, INTER-Mediator not only shows the data in the database, but also generates elements for each queried record and repeats records.

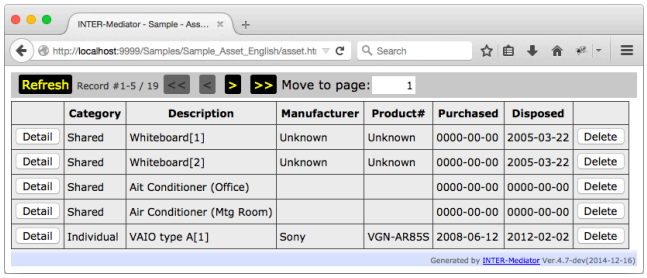

Figure 1. Asset List of the Sample Application

Figure 2 is the detailed page for each asset. It can contain any kind of form element. If the user edits the string of the text field and moves from one element to another by pressing the tab key, the edited string is sent to the database and the relevant field is updated. Moreover, this page has a one-to-many relationship with the "asset" and "lent" tables. Of course, all lent log items are for the asset shown on the page.

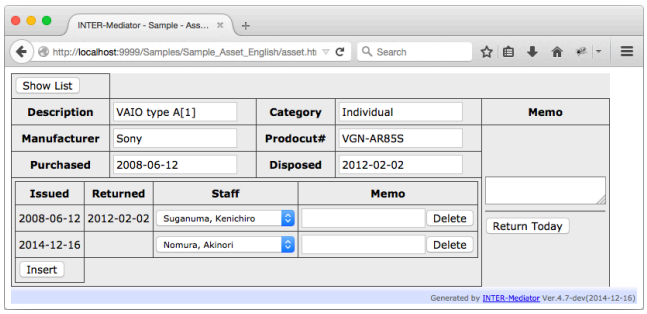

Figure 2. Details of the Asset

The "Return Today" button can set the returned date of the last item in the log to today, and a JavaScript program is required for it. Otherwise, showing the data in the database and updating with editing by the user (i.e. binding), repeating records within the table, showing associated records with their relationship, making buttons to add and delete a record, and the master/detail style user interface can be accomplished with declarative descriptions.

B. What Do We Need in Order to Create a Web Page?

Developing a web application with INTER-Mediator requires a "Database," "Page File" and "Definition File." The "Database" should be set up with a valid schema. The "Page File" is described in HTML, and it's a template of the web page. Many web frameworks use page templates because they separate the presentation layer from a logical layer [6]. The Definition File includes information to access the database in a declarative description. The Definition File is described using PHP's array expressions. Although PHP is an imperative description, the Definition File only requires pre-defined key's strings and their values. Moreover, no control descriptions such as repeating and conditions are required. Thus, the Definition File is mostly declarative. Moreover, a Definition File Editor that works on web browsers has been developed. As a result, developers can create a file without having to use an imperative style.

1) Definition File: Listing 1 shows part of the definition file named asset_context.php in PHP language. The Definition File should contain one IM_Entry function and its parameters, which are described in arrays and have information to work in a web application. The first array that starts from the second line is the "Context" of the database accesses. For example, the "view" key shows accesses to the "asset" table, and the "sort" key sets the "purchase" field for sorting records. The Context is identical to the input from and output to the database.

Listing 1. Example of Definition File (asset_context.php)

```
IM_Entry( array(
  array(  //The Context for Asset List
    'name'=>'asset', //Referencing name from Page File
    'view'=>'asset', //The entity name in database
    'key'=>'asset_id', //The primary key field
    'repeat-control'=>'insert delete', //Adding Insert/
        Delete buttons
    'navi-control'=>'master-hide',   //Master/Detail UI
    'records'=>5, //Show every 5 records
    'paging'=>true, //Pagination activate
    'sort'=>array( //Sorting fields and directions
      array('field'=>'purchase', 'direction'=>'ASC'),),),
  array(  //The Context for Details of Asset
    'name' => 'assetdetail',  //Referencing name
    'view'=>'asset',  //The entity name for querying to
        database
    'table' => 'asset', //The entity name for updating to
         database
    'navi-control' => 'detail-top', //Master/Detail UI
      [...omitted...]
  ),),
  NULL,
  array( /* Connection Information, omitted */ )
);
```

2) Page File: The Page File for the example is Listing 2, and the following explanations refer to the circled numbers in the source. The Page File is a page template described as a pure HTML5 source. A number of popular web frameworks have their own template language including special tags, and sometimes they prevent users from looking at their details. INTER-Mediator's Page File conforms to HTML5, and there is nothing to prevent it from being shown on browsers.

[1]Ver.9 or above are required for full support.
[2]All sources and schema definitions are distributed within the repository.

ISBN: 978-988-19253-2-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Listing 2. Example of Page File

```
<html>
<head>
  ①<script src="asset_contexts.php"></script>
</head>
<body onload="INTERMediator.construct()">③
<div id="IM_NAVIGATOR"></div>④
<table> #Asset List with using 'asset' context.
  <thead>[... omitted ...]</thead>
  <tbody>
    <tr>
      <td>⑤</td>
      ②<td data-im="asset@category"></td>
      <td data-im="asset@name"></td>
      <td data-im="asset@manifacture"></td>
      <td data-im="asset@productinfo"></td>
      <td data-im="asset@purchase"></td>
      <td data-im="asset@discard"></td>
      <td></td>
    </tr>
  </tbody>
</table>
<table> #Asset Details with using 'assetdetail' context.
  <tbody><tr>
      [... omitted ...]
    <th>Purchased</th>
    <td>⑥<input type="text"
      data-im="assetdetail@purchase"/></td>
      [... omitted ...]
  </tr></tbody>
</table>
</body>
</html>
```

The sample has two pages, but these pages are contained in only one HTML file, meaning they are essentially a single-page application. ① The element tagged with SCRIPT to load the Definition File (Listing 1) should be described in the HEAD part. textcircled2 Some elements have a "data-im" attribute like "asset@name." This value indicates that the element binds with the "category" field from the "asset" context. ③ The JavaScript statement "INTERMediator.construct()" starts to generate the page. This statement is imperative, but it can be the same code regardless of any applications.

④ is a DIV element with an "id" attribute having the value "IM_NAVIGATOR." This element replaces pagination controls such as "Prev" and "Next" buttons. The "true" value associated with the key "paging" in Listing 1 is required to make these replacements. The asset list of Figure 1 has a "Delete" button on each line and an "Insert" button in the pagination control. The value associated with the "repeat-control" key lets the framework add these buttons.

The "navi-control" key in a Context directs the page to the Master/Detail style page. The Master by "asset" Context area automatically generates buttons to navigate the detail page, and ⑤ is the space for them. ⑥The Detail area by "assetdetail" Context generates the button to back the Master list.

C. INTER-Mediator is Simple

For the sake of comparison, we tried to develop the same application of the section II-A's example by using the PHP-based MVC framework "CodeIgniter [7]." Table I compares the metrics of INTER-Mediator and CodeIgniter. Every file was formatted using the same IDE tool, and comments weren't counted. CSS files are commonly used in both frameworks and weren't included in the metrics. The table shows that INTER-Mediator enables users to develop using fewer files and shorter lines than CodeIgniter permits. Their parts for displaying the data from the database mostly have the same metrics. Whereas INTER-Mediator does not need code for updating the database, CodeIgniter requires users to write php-based imperative code for Controller and Model.

TABLE I
COMPARISON USING METRICS FOR THE SAME FEATURED APPLICATION

INTER-Mediator	CodeIgniter
Page File (HTML file) • 1 file, 109 lines • (JavaScript is 14 lines) Definition File (PHP file) • 1 file, 72 lines	PHP files mainly described in HTML for View • 2 files, a total of 130 lines • (No JavaScript codes) PHP files for Controller and Model • 2 files, a total of 216 lines Edited configuration files • 2 files, a total of 6 lines
Total: 2 files, 181 lines	Total: 6 files, 352 lines

D. Development Style

The Page File in INTER-Mediator is a kind of prototype web page, and it's relevant to Mockup Driven Development (MDD) [8], a practice of agile development [9]. MDD means that a mockup of the HTML-based user interface is created before the implementation. The mockup helps to clarify the user's requirements [10]. The HTML mockup does not require specialized skills to create, so it can be made by end-users. INTER-Mediator's development style thus conforms to Mockup Driven Development.

III. SYSTEM MODIFICATIONS IN INTER-MEDIATOR

Although INTER-Mediator enables development with declarative descriptions, engineers should handle some of the tasks in the development process such as building the schema with domain analysis, improving the UIs with JavaScript code and adding server-side programs. As far as Page Files and Definition Files go, end-users can modify them since they are described in a declarative way. Some aspects of the maintenance task are just modifying the developed files. This means end-users can perform certain maintenance tasks on the INTER-Mediator-based application if they can be done within declarative descriptions. In this section, we discuss the range of modifications that can be made by end-users.

A. Categorizing System Modifications

Table II categorizes the modification tasks that can be made to web applications that are connected to a database. In INTER-Mediator-based development, ①〜③ are generally done in a declarative way. In contrast, ④〜⑥ should be done by engineers.

B. Modifications to Page Elements

① in Table II usually consists of minor modifications such as editing HTML code in a Page File. This kind of modification can be done declaratively in most other frameworks.

Suppose that end-users want to select the "Category" from a pop-up menu. To accomplish this, the Page File is changed

ISBN: 978-988-19253-2-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE II
RANGE OF WEB SYSTEM MODIFICATIONS

	Occasions	Examples
①	Page Elements	• Order of elements, Disapper them • Add a field already in database • Change the color of characters
②	Request to Database	• Modify the query criteria • Modify the sort condition • Add the button to create and delete
③	Response to Single Field	• Change the decimal place from 2 to 3 • Add constant strings before/after the data • Calculated property
④	UI Customize	• Button with a special procedure • Update any elements
⑤	Database Response	• Send mail after create record • Aggregation unsupported by database
⑥	Modify Schema	• Create a new view • Create a table or a field
*	Create New Page	• Create from scratch • After copying an existing page, apply ①-⑥

as in Listing 3, and the pop-up menu will be replaced with a text field. If only HTML codes are placed on it, the initial selection is not relevant to the real data of the "category" field. INTER-Mediator can bind the SELECT element to the field, and initially set a value corresponding to the field data. The field data will be updated after the user selects an item.

Listing 3. Modification to the Page File(1)

```
/******* Before ******/
<input data-im="assetdetail@category" />
/******* After ******/
<select data-im="assetdetail@category">
  <option value="Indivisual">Indivisual</option>
  <option value="Shared">Shared</option>
</select>
```

Minor modifications can be done within the HTML template for any web application framework. If the modification requires more to it than simply changing the HTML descriptions, one should seek out the relevant code associated with the modification, even if the modification is a small one. The relevant code could be scattered among the controllers or the views. For example, if another field needs to be added to the page, it might require not only an HTML page to be added but also imperative modifications of the controller and model. Thus, we need to understand the whole code of the application. In contrast, INTER-Mediator can handle these modifications within the Page File.

C. Modification of Request to Databases

An example of ② in Table II is to narrow down the query results to the ones showing non-discarded assets instead of everything. To do this, a new "asseteffect" Context is prepared, as shown in Listing 4. It is duplicated from the already existing "asset" Context (Listing 1), and the value associated with the key "name" is modified. Moreover, a value associated with the key "query" is added, and the query condition means the "discard" field is less than "1990-1-1." This field should be blank if the asset isn't discarded, and this condition is equivalent to "the field is blank."

Other frameworks require one to make modifications to the SQL statement and/or imperative codes of the model or controller. In contrast, INTER-Mediator allows one to modify requests to the database in a declarative way.

Listing 4. Modification to the Definition File(1)

```
array(
  'name' => 'asseteffect',
  'view' => 'asset',
  'sort' => array(
    array('field' => 'purchase',
          'direction' => 'ASC'),
  ),
  'query' => array(
    array('field' => 'discard',
          'operator' => '<',
          'value' => '1990-1-1'),
  ),
),
```

D. Modification of Responses

Table II has two types of modification to responses, which are ③ for a single field and ⑤ for a single record or multiple records. ⑤ requires an imperative description.

1) Formatting Field Value: An example of ③ is formatting dates. MySQL returns the date as an ISO8601 style string like "2014-07-31." The aim of the modification is to present a date with a more natural style, e.g., "2014/7/31." Listing 5 shows the results of the modification. The array associated with the key "formatter" is inserted in the right place of the Definition File. The array has two elements: the first one means that the "purchase" field of the "asset" table should be converted into a "%y/%m/%d" style string with the MySQLDateTime class. Reverse conversion for updating field data is also supported.

Listing 5. Modification to the Definition File(2)

```
array(
        [... omitted ...]
  'formatter' => array(
    array('field' => 'asset@purchase',
          'converter-class' => 'MySQLDateTime',
          'parameter' => '%y/%m/%d'),
    array('field' => 'asset@discard',
          'converter-class' => 'MySQLDateTime'),
  ),
```

If we want to do the same thing in other frameworks, we have to modify the imperative code. INTER-Mediator has a bi-directional conversion feature for date, number, and HTML strings. As for number formatting, it supports decimal digits and the thousands separator.

2) Calculated Property: Another example of ③ is calculated property which is a read-only field in a record and has a value of calculated from other fields of the same record and/or other records. Listing 6 shows the definition of calculated properties. The array associated with the key "calculation" has an array with "field" and "expression" keys. After added these descriptions, for example, the element " " can be placed in the Page File, and it shows the value from the expression which is the average of other field values.

In other frameworks, usually additional properties should be defined using imperative descriptions. In contrast, INTER-Mediator can define them in declarative descriptions. Moreover not only defining a new property, it can assign the calculated value to attributes of an element.

ISBN: 978-988-19253-2-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Listing 6. Modification to the Definition File(3)

```
array(
  'name' => 'assetdetail',
              [...omitted...]
  'calculation' => array( // Added definitions
    array("field" => "avglength",
        // "avglength" can be used like a field
      "expression" => "average(lent@datelength)"),
  )     // Calculate the average of "datalength" fields
        // in the "lent" context
),
array(
  'name' => 'lent',
              [...omitted...]
  'calculation' => array( // Added definitions
    array("field" => "datelength",
        // "datelength" can be used like a field
      "expression" => "if(backdate = '', '',
            date(backdate) - date(lentdate))"),
  )     // Calculate the difference between 2 date
        // fields, "if" and "date" are functions.
),
```

E. Maintenance by End-Users

The above discussion shows that INTER-Mediator handles modifications ①-③ in a declarative way, whereas other frameworks require the imperative way. INTER-Mediator can thus help end-users in performing maintenance tasks.

IV. Binding and Page Generation

INTER-Mediator generates a web page from the Page File as a template. This process is run on the client-side. In regard to the template processing, how it binds the node to the database and how it repeatedly generates nodes from a record set are important considerations.

A. Binding for Automatic Updating

A template process is used to show the data from the field of the table in the database on an element of the Page File. The value of the "data-im" attribute in an element specifies the table and the field of the database. When the framework generates the web page, it refers to the "data-im" attributes and places the data from the field into the value or the attribute of the element. At the same time, the framework stores the key field value to identify which record is the origin of the data.

If the user edits the data in the binding elements, the framework can identify the record by using the pre-stored key field value. Before updating the database, the current value is queried and checked to see if it's the same as the previous value. This means the optimistic lock mechanism works internally. After that the framework updates the database with the edited value in the element.

The binding works on the foundation of the client-cached model objects that is the proxy of the queried data, and it's called the "Context Model." This contains the key field value, field data, and bound nodes. If the user changes the value in a bound node, the framework notifies to the model object and updates other bound nodes with the updated value. This synchronization works not only in one client but also in multiple clients with the WebSocket technology by Pusher [11].

B. Repeating and Relationship

1) Repeating for Record Set: Usually, multiple records are returned as the result of a query, and they should be presented in repeating elements such as in a table. The upper part of Figure 3 illustrates how the INTER-Mediator generates the rows of a table for displaying queried records.

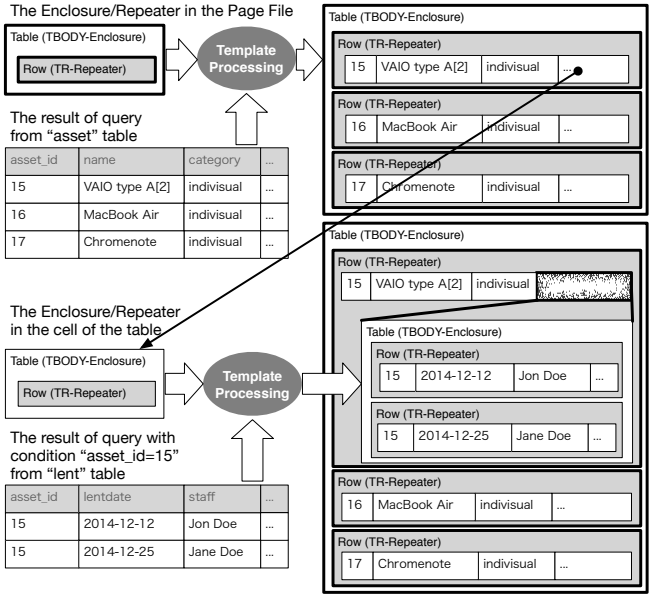

Figure 3. Template Process Involving Relationship

When INTER-Mediator generates a web page, it first traverses the nodes from the top BODY tag's element and seeks elements having the "data-im" attribute (① of Listing 7). After that, it identifies the TR tagged node (②) nearest the parent, and it's called the "Repeater." The parent of the Repeater (③) is the TBODY tagged element, and it's called the "Enclosure."

Listing 7. Part of Page File with Recursive Enclosure/Repeater Pairs

```
<table>
  ③<tbody>      ⇐ Outer Enclosure
  ②<tr>         ⇐ Outer Repeater
    <th>Manufacturer</th>
    <td>①<input type="text"
      data-im="assetdetail@manifacture"/></td>
      [...ommited...]
    <td colspan="4">
      <table><thead><tr>....</tr></thead>
        ④<tbody>  ⇐ Inner Enclosure
        ⑤<tr>     ⇐ Inner Repeater
            <td data-im="lent@lentdate"></td>
            <td data-im="lent@backdate"></td>
              [...ommited...]
        </tr>       ⇐ End of Inner Repeater
        </tbody>    ⇐ End of Inner Enclosure
      </table>
    </td>
  </tr>         ⇐ End of Outer Repeater
  </tbody>      ⇐ End of Outer Enclosure
</table>
```

After the Enclosure/Repeater pair is identified, the framework gathers elements having the "data-im" attribute under the Repeater, decides the Context name from the "data-im" attributes, accesses the database by referring to the specifications of the Context, and obtains the queried records. The framework removes the Repeater and stores it in a variable before the merging process. After that, it merges one

ISBN: 978-988-19253-2-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

record into a copy of the stored Repeater with matching the "data-im" attribute value and the field name in the record, and the merged Repeater is set to the child node of the Enclosure. This merging process is repeated for each record in queried ones.

As shown in Listing 7, the TBODY/TR tagged elements don't have any additional information. In this case, the framework automatically detects the Enclosure/Repeater pair, and the developer doesn't have to identify which ones form the pair. Other pairs like UL/LI, OL/LI, SELECT/OPTION and a special class of DIV/SPAN are recognizable as Enclosure/Repeater pairs.

2) Relationship: INTER-Mediator supports hierarchical Enclosure/Repeater pairs. If the framework detects another Enclosure/Repeater (④⑤ of Listing 7) within a Repeater (②), these called "inner" and "outer" Enclosure/Repeater pairs. After the outer Repeater processed for one record, the same process of gathering, accessing and merging for the inner Enclosure/Repeater runs. This process runs recursively.

The arrow from the upper part to the lower part of Figure 3 means any cell of the table has another TABLE tagged element. Suppose the inner table contains elements having a "data-im" attribute, inner Enclosure/Repeater generates bound elements for records as shown in the lower part of Figure 3. When the framework queries the database to see if there are inner Enclosure/Repeater pairs, it adds the relationship information in the Definition File to the query condition, and the associated records of the parent record appear in the inner table.

As described above, the simplest Page File simply involves adding "data-im" attributes for binding; it doesn't require a special description for repeating and relationship in a Page File. INTER-Mediator can generate a page corresponding to the structured data in the database by using simple declarative descriptions.

C. Algorithm of Page Generation

INTER-Mediator can generate a web page from the Page File with using queried data from the database. The algorithm of the page generating process is described in Algorithm 1. There are 4 procedures, and the underlined statements are calling any of these procedures. Other capitalized words in statements are variables. The PageConstruct procedure would be called just after loaded the Page File. The algorithm is implemented by using JavaScript. The contents of the Page File is treated as the DOM objects, on the other hand, commonly-used frameworks treat the template as strings.

The term "linked node" means the HTML element which has the valid "data-im" attribute. The "context model" is a local cache of the queried data and already explained at IV-A. The dotted expression like "ContextModel.repeaters" means an object and its property. The property referencing with [ ] means the variable variables, i.e. the value of variable turns into the property name. The ← assigns the right value to the left variable. The ⇐ adds the right object to the left collection. The ⇏ removes the right object from the left collection.

The SeekEnclosure and ExpandRepeaters procedures recursively call each other. This realizes hierarchical Enclosure/Repeater pairs to bind to the associated records that are queried in regards of the context definition. In other words, the single Page File can show the data from multiple tables simultaneously, and they can associate with each other based on the relationship if the context definition contains it.

Algorithm 1 Page Generation

```
procedure PAGECONSTRUCT
    SeekEnclosure(node of body tag, NULL)
end procedure

procedure SEEKENCLOSURE(Node, Record)
    if Node is an enclosure then
        ExpandEnclosure(node, record)
    else
        for each ChildNode in child nodes of Node do
            SeekEnclosure(ChildNode, Record)
        end for
    end if
end procedure

procedure EXPANDENCLOSURE(EnclosureNode, Record)
    Repeaters ← copy of repeaters in EnclosureNode
    EnclosureNode ⇏ Repeaters
    LinkedNodes ← collect linked nodes in Repeaters
    ContextDef ← decide context from LinkedNodes
    if ContextDef == NULL /* in case of can't decide context */ then
        for each Repeater in Repeaters do
            EnclosureNode ⇐ Repeater
            SeekEnclosure(Repeater, Record)
        end for
    else
        ContextModel ← generate context model for ContextDef
        ContextModel.repeaters ← Repeaters
        if Record != NULL then
            ContextDef.additionalCondition
                ⇐ Record.keyValue and ContextDef.relationship
        end if
        RecordSet ← query result using ContextDef
        ExpandRepeaters(ContextModel, EnclosureNode, RecordSet)
    end if
end procedure

procedure EXPANDREPEATERS(
                ContextModel, EnclosureNode, RecordSet)
    if count of RecordSet == 0 then
        EnclosureNode ⇐ node for no record
    else
        for each Record in RecordSet do
            KeyFieldValue
                ← concatenate Record.keyField and Record.keyValue
            Repeaters ← copy of ContextModel.repeaters
            LinkedNodes ← array of linked nodes in Repeaters
            for each LinkedNode in LinkedNodes do
                LinkedNode.id ← unique numbered string
                TargetField ← target field of LinkedNode
                TargetValue ← Record.[TargetField]
                LinkedNode ⇐ TargeValue
                        (e.g. set to the value attribute)
                ContextModel.store.[KeyFieldValue].[TargetField]
                                        ← TargetValue
                ContextModel.binding.[KeyFieldValue].[TargetField]
                                        ⇐ LinkedNode.id
            end for
            for each Repeater in Repeaters do
                EnclosureNode ⇐ Repeater
                Repeater.id ← unique numbered string
                ContextModel.binding,[KeyFieldValue].repeaters
                                        ⇐ Repeater.id
                SeekEnclosure(Repeater, Record)
            end for
        end for
    end if
end procedure
```

V. Experimental Evaluation

INTER-Mediator aims to let web applications be maintained with declarative descriptions. The question is thus "can end-users easily learn INTER-Mediator?" To answer it, we ran an experiment in which participants studied how to conduct development with INTER-Mediator.

ISBN: 978-988-19253-2-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

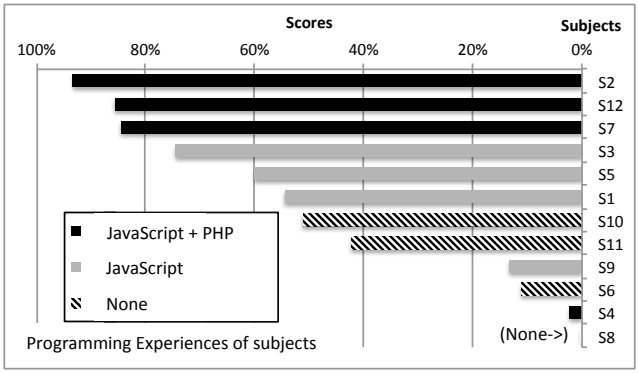

Figure 4. Score of Examination for each Subject

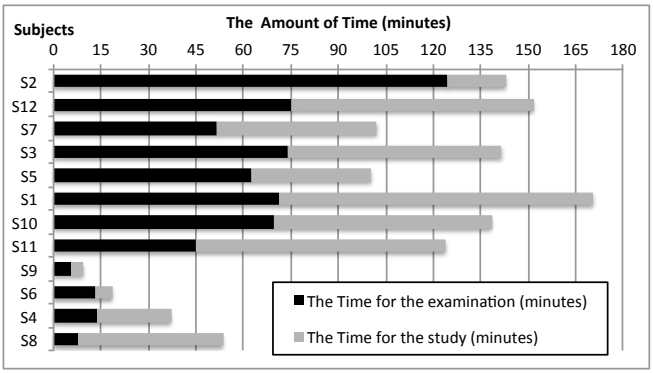

Figure 5. The Amount of Time for each Subject

A. Procedure of the Experiment

The experiment consisted of a study session followed by an examination. The study session was self-paced online learning about INTER-Mediator. Subjects were encouraged to read the web pages to learn how to create web pages connected to databases. There were ten web pages described in Japanese (a total of 25,000 Japanese characters). The content focused on the declarative descriptions; that is, some of the features requiring imperative descriptions weren't included. The time spent reading every page was measured.

Twelve subjects participated in the experiment, and at the time, they were working as a web designer, a coder, and web design professionals. They weren't end-users in straightforward sense, but like most end-users, they weren't skilled at writing imperative languages of computer programming. These web workers nonetheless had skill in HTML/CSS, and this is why the examination could focus on INTER-Mediator specific matters.

After they had finished studying, they took an online examination. It had eight questions. Five questions with 45 blanks were about INTER-Mediator related issues, and non of them were multiple choice. The time from the start to the end of answering was measured for each question. The examination was not easy for novices, and it was not similar to the study pages. Scores were scattered from very low to very high.

B. Experiment Results

Figure 4 shows the examination scores of each subject, and Figure 5 shows the amount of time for the study and the examination. The lower scoring 4 subjects (Group A; S9, S6, S4 and S8) scored under 15% and completed the examination within 15 minutes. The other subjects (Group B) with high scores spent 100 to 170 minutes in total studying the pages and taking the examination. Group A apparently abandoned their attempt to lean and answer. Group B's participants succeeding in learning, but their level of understanding varied. These results indicate that although some web workers would likely fail to learn INTER-Mediator, a 2 hour session would be sufficiently long for most to gain an understanding of the framework.

We asked them how often they used JavaScript and PHP in the survey after the examination. As far as JavaScript went, their responses included descriptions of short programs to accomplish their design work. Some participants had experience with PHP code, but their experience was only with modifications within the CMS application. In Group B, those with programming experience tended to get higher scores. Generally speaking, motivated designers tend to acquire skills beyond their speciality, and such skills often include programming. Moreover Group A had 2 participants who could program, and it's not always true that the programming skill helps to learn INTER-Mediator.

The survey after the examination shows that having knowledge about databases improved scores. One participant in Group A said he had no knowledge about databases, a middle score participant in Group B had the experience with Microsoft Access, and high scored participants discussed about relationship. INTER-Mediator is based on both web and database technologies. Obviously, the knowledge about HTML helps to understand, but the knowledge of the database is also important.

VI. Related Work

INTER-Mediator is not the only frameworks to use declarative descriptions. WebML [12] is a web site modeling language, and a model based development system using WebML has been proposed [13], [14]. WebML can specify the data structures used throughout the site, and it is systematic. So it requires the modeling skill with the language. In contrast, INTER-Mediator handles the HTML page directly, and it's easier to understand for end-users. Hilda [15], [16] supports declarative language and can separate the logic and presentation. It is a suitable architecture for developing web applications, but requires the same skills of software engineers. INTER-Mediator is more suitable than it as far as an end-user tool goes because of its HTML and key-value style descriptions.

Some frameworks aimed at simple development enable end-user development [17]. One example is XFormsDB [18]; it uses standardized XForms and XQuery. By contrast, INTER-Mediator uses HTML template; it's a simpler and more direct way to create a web page.

Several web application frameworks have template architectures to extend HTML. ColdFusion [19] is a popular one. Generally speaking, a web page connected to a database can be easily created with a small amount of HTML, but updating it requires tricky development. Moreover, some products don't have sufficient extensibility. ColdFusion has mostly imperative HTML extensions and improved backend processing. INTER-Mediator, on the other hand, doesn't extend

ISBN: 978-988-19253-2-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the HTML standard, and it updates databases automatically through the binding feature.

Over half of INTER-Mediator is in JavaScript, and it creates web pages from HTML templates and the data from a database on the client side. Nowadays, JavaScript-based frameworks, for example, AngularJS [20] and Knockout [21], are evolving into "front-end frameworks." They are basically client-side frameworks, works with HTML descriptions and little imperative code. INTER-Mediator has both server and client components, and most of its features can be used in declarative descriptions. INTER-Mediator has the Enclosure/Repeater feature and can handle it recursively in just one HTML page. In contrast, other frameworks require users to define another HTML page template for the inside of repeating elements [22], [23].

Some developers have integrated INTER-Mediator with other PHP frameworks [24], [25], [26], including CakePHP [27], CodeIgniter and Yii [28]. They are using INTER-Mediator as a View layer because it can describe page templates simply in HTML. The controller and model are composed of PHP frameworks.

VII. Conclusions

INTER-Mediator enables a database-driven web application to be maintained with declarative descriptions. The user should make an HTML5 description and access information can be edited by using a special editor application. The descriptions are not imperative, so end-users and designers can modify them. As a result, end-users and designers can be involved in the development of the web application, especially in the maintenance phase. The foundation for building a database-driven web page with just a declarative description is the binding and Enclosure/Repeater mechanism. Our experiment indicated that that end-users and designers could learn the knowledge needed to create and modify web applications with INTER-Mediator in a short period of time.

IT is beneficial in a competitive world [29]. However, current IT systems are extremely costly because most parts of the development have to be handled by software engineers. By involving end-users, the cost balance of IT can be changed, and small organizations with limited budgets can have more opportunities to develop their own IT systems.

References

[1] A. Leff and J. T. Rayfield, "Web-Application Development Using the Model/View/Controller Design Pattern," in *Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing*, ser. EDOC '01, 2001, pp. 118–.

[2] M. Nii. INTER-Mediator. [Online]. Available: http://inter-mediator.org/

[3] A. J. Ko, B. A. Myers, and H. H. Aung, "Six Learning Barriers in End-User Programming Systems," in *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, ser. VLHCC '04, 2004, pp. 199–206.

[4] M. Nii and contributors. GitHub Repository for INTER-Mediator. [Online]. Available: https://github.com/msyk/INTER-Mediator

[5] J. Rode and M. B. Rosson, "Programming at runtime: Requirements and paradigms for nonprogrammer web application development," in *Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments*, ser. HCC '03, 2003, pp. 23–30.

[6] T. J. Parr, "Enforcing strict model-view separation in template engines," in *Proceedings of the 13th international conference on World Wide Web*, ser. WWW '04, 2004, pp. 224–233.

[7] CodeIgniter Project. Codeigniter. British Columbia Institute of Technology. [Online]. Available: http://www.codeigniter.com/

[8] E. Benson, "Mockup driven web development," in *Proceedings of the 22Nd International Conference on World Wide Web Companion*, ser. WWW '13 Companion, 2013, pp. 337–342.

[9] F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, and E. Astesiano, "On the effectiveness of screen mockups in requirements engineering: Results from an internal replication," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10, 2010, pp. 17:1–17:10.

[10] J. M. Rivero, J. Grigera, G. Rossi, E. Robles Luna, F. Montero, and M. Gaedke, "Mockup-driven development: Providing agile support for model-driven web engineering," *Inf. Softw. Technol.*, vol. 56, no. 6, pp. 670–687, Jun. 2014.

[11] Pusher Ltd. Pusher. Pusher Ltd. [Online]. Available: http://pusher.com/

[12] S. Ceri, P. Fraternali, and A. Bongio, "Web modeling language (webml): A modeling language for designing web sites," in *Proceedings of the 9th International World Wide Web Conference on Computer Networks : The International Journal of Computer and Telecommunications Netowrking*, 2000, pp. 137–157.

[13] M. Brambilla, S. Ceri, S. Comai, M. Dario, P. Fraternali, and I. Manolescu, "Declarative specification of web applications exploiting web services and workflows," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '04, 2004, pp. 909–910.

[14] S. Ceri, F. Daniel, M. Matera, and F. M. Facca, "Model-driven development of context-aware web applications," *ACM Trans. Internet Technol.*, vol. 7, no. 1, Feb. 2007.

[15] F. Yang, J. Shanmugasundaram, M. Riedewald, and J. Gehrke, "Hilda: A high-level language for data-drivenweb applications," in *Proceedings of the 22Nd International Conference on Data Engineering*, ser. ICDE '06, 2006, pp. 32–.

[16] F. Yang, N. Gupta, N. Gerner, X. Qi, A. Demers, J. Gehrke, and J. Shanmugasundaram, "A unified platform for data driven web applications with automatic client-server partitioning," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07, 2007, pp. 341–350.

[17] M. Laine, D. Shestakov, E. Litvinova, and P. Vuorimaa, "Toward Unified Web Application Development," *IT Professional*, vol. 13, no. 5, pp. 30–36, Sep. 2011.

[18] M. Laine, D. Shestakov, and P. Vuorimaa, "XFormsDB: an extensible web application framework built upon declarative W3C standards," *SIGAPP Appl. Comput. Rev.*, vol. 12, no. 3, pp. 37–50, Sep. 2012.

[19] Adobe Systems, Inc. Adobe ColdFusion 11 Family. [Online]. Available: http://www.adobe.com/products/coldfusion-family.html

[20] Google Inc. and community. AngularJS. [Online]. Available: http://angularjs.org/

[21] S. Sanderson. Knockout. [Online]. Available: http://knockoutjs.com/

[22] Hairgami_Master, jpmorin, and answerers. How can I make recursive templates in AngularJS when using nested objects? [Online]. Available: http://stackoverflow.com/questions/15661289/how-can-i-make-recursive-templates-in-angularjs-when-using-nested-objects

[23] Benny, nemesv, and answerers. Recursive template with knockout js. Stackoverflow. [Online]. Available: http://stackoverflow.com/questions/15525216/recursive-template-with-knockout-js

[24] A. Matsuo. IMCake. [Online]. Available: https://github.com/matsuo/IMCake

[25] K. Ito. Use INTER-Mediator from CodeIgniter (Japanese). [Online]. Available: http://agilmente.com/blog/2013/07/22/

[26] ——. Use INTER-Mediator in Yii (Japanese). [Online]. Available: http://agilmente.com/blog/2013/12/25/

[27] Cake Software Foundation, Inc. CakePHP. [Online]. Available: http://cakephp.org/

[28] Founded by Qiang Xue. Yii Framework. [Online]. Available: http://www.yiiframework.com/

[29] A. McAfee and E. Brynjolfsson, "Investing in the IT That Makes a Competitive Difference," *Harvard Business Review*, jul 2008.

ISBN: 978-988-19253-2-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)