ISSN: 2277 – 9043
International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)
Volume 3, Issue 10, October 2014

A Knowledge-Based Web Development Kernel

Ahmet Egesoy

***Abstract*— Designing for the Web is more difficult than designing traditional software. There are special tools in this area but they are monotonically multi-paradigm and also provide limited freedom for the designer. This work is about the architecture of a project with a new approach. WDLDL language and its kernel services put the knowledge representation and manipulation at the heart of the architecture, and create the resultant code only through transformations.**

***Index Terms*—Design patterns, Model-driven, Web design, Software Engineering.**

I. Introduction

The Internet has an impact on almost every aspect of our lives by offering a wide range of remarkable technological amenities. This global information fountain has emerged as a social phenomenon as well as a technological marvel. The most fundamental and the most interesting sub-system of the Internet is the famous *World Wide Web* (commonly called as the *Web*), which is a very unique environment with its distributed, dynamic and heterogeneous form and content and its emphasis on visual appeal. The document-based but highly eclectic structure of the Web technology is more of a result of its own adventurous history than conscious design.

The design and implementation of Web applications is a multi-facetted task that exceeds the complexity of the conventional software development projects.

This work is about the design and development of the kernel of a knowledge-based design language called *WDLDL* (Web Design Logic Definition Language). *WDLDL* aims the representation of reusable design knowledge as design patterns so that they can be retrieved and instantiated when necessary. Support for the transformation and evolution of design artifacts are also within the scope of this project. The kernel which is the first step in the development of *WDLDL* aims to provide the basic relational modeling functionality together with support for basic I/O and computation.

The second section contains an introduction of the existing Web design tools and Web patterns; the third section contains information about the general architecture of the WDLDL project; Section IV contains information about using the BLUE Code scripting language and reaching the kernel of the system; Section V gives conclusions.

II. The Existing Web Design Tools

The state of the art tools for Web design are a group of development environments called as *Web Modeling Languages* (*WML*). These languages offer object-oriented methods for the conceptual design of the Web sites while employing custom models or XML-based notations for expressing the hypermedia characteristics. The two tools that need to be mentioned are OOHDM [1] and WebML [2],[3].

A. WebML

WebML (Web Modeling Language) is an object-oriented Web design language that was developed by the Polytechnic Institute of Milan. It deserves attention for being well-known and popularly used.

WebML reflects an object-oriented point of view not much far from that of UML (Unified Modeling Language). It is possible to translate to and from UML without much loss of information. In [5] Moreno et al. define the WebML syntax as a UML profile in order to form a bridge between the two technologies.

WebML is not exactly a multi-purpose Web development tool. It aims to support the projects where there is a data intensive application on the background and there is a set of dynamically-created Web pages on the foreground.

Fig. 1 depicts the multi-stage development process supported by WebML. The structural models are related to the structure of the data that is aimed to be accessed by the Web page. The representation is in harmony with E-R diagrams and UML class diagrams.

The transition from the structural model to the navigation-composition model is a step that does not involve much freedom since the logical relations between the parts of data determines the general topology of the site. The visual containers (called *content units*) that are linked to the database content and navigational links are the components that take part in the building of models of the navigational/compositional stage. The two relations that are significant at this stage are *navigation* and *composition*.

Content units function as visual components and has types like: Index Unit, Data Unit, Multi-data Unit, Scroller Unit, Multi-choice Unit and Hierarchical Unit [2],[3].

Next stage is the building of the site-view layer that determines the styles of presentation of information by the user-friendly interface.

As a result, WebML focuses on the creation of a visual interface for an object-oriented information system. And in spite of its effective usage as a Web development language, WebML is a language that is not really Web-based (or

Manuscript received Oct 18, 2014.
***Ahmet Egesoy**, Computer Engineering Department, Ege University, Izmir, Turkey, Tel:+90 2323887221*

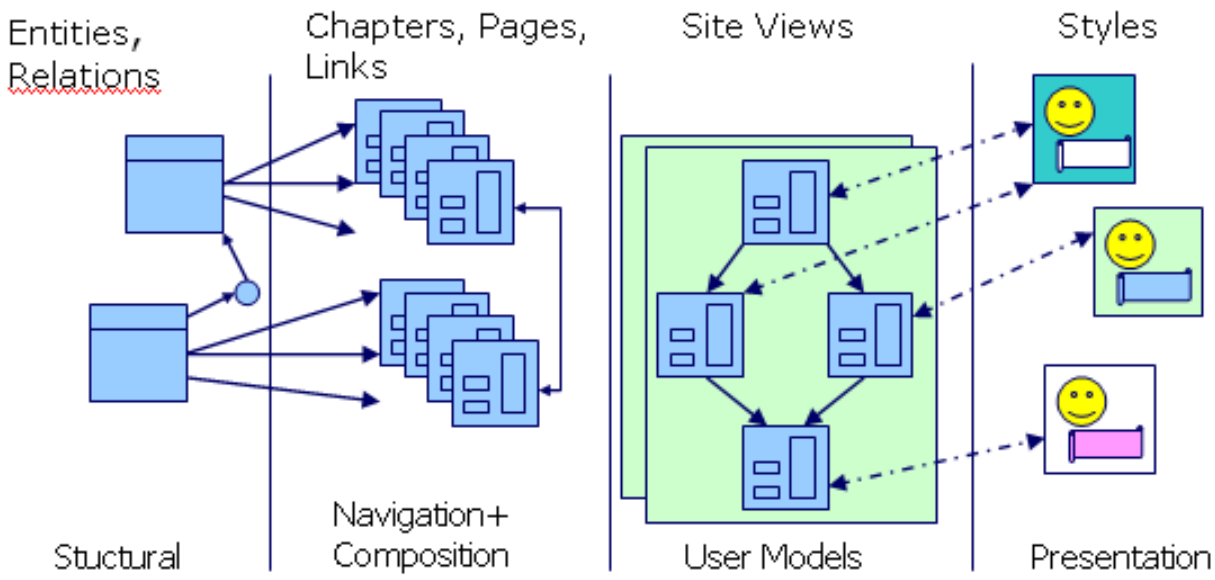

Fig. 1 WebML Models [3]

hypertext based) in terms of the paradigm it conceptually leans on. It is partially under the influence of the relational database paradigm and partially object-oriented paradigm.

A. OOHDM

OOHDM is an abbreviation for Object-oriented Hypermedia Design Method. As the name suggests the approach chosen in introducing OOHDM was to define it a new method rather than a language or tool.

OOHDM method aims to construct a balance between the computational requirements of conventional applications and the structural and navigational requirements of the hypermedia applications [6].

According to Schwabe et al. [6] the cornerstones of OOHDM are as follows:

1) Navigational objects are views of conceptual objects.
2) The navigational objects should be effectively abstracted by using navigational contexts.
3) All the interfaces should be separated from the navigational objects.
4) Some design decisions should be taken in the coding phase.

The development process phases of OOHDM are similar to those of WebML [1]:

1) Conceptual Design-Domain Analysis
2) Navigational Design
3) Abstract Interface Design
4) Implementation

Code development with OOHDM is an open ended task since it does not suggest any physical medium for the final implementation. At the implementation phase the necessary languages are chosen and the constructs of these languages are matched with the OOHDM objects. These *languages* may be some of the Web-related technologies and standards or they may also be hypermedia languages that are not related with the Web. Even conventional programming languages can be used for the coding phase. A database query language is usually added to this arsenal too.

A navigational OOHDM entity is defined below. This object inherits from a conceptual class just like a class would. Inheritance is used throughout the OOHDM development process (all four phases) effectively as a result of the object-oriented approach [6].

```
NODE Story [FROM Story:St]
 [INHERITS FROM Person]
author: String [SELECT Name]
        [FROM Person:Pr WHERE Pr Is
         Author of St]
auhtor_bio: String [SELECT Bio]
        [FROM Person:Pr WHERE Pr Is
         Author of St]
.... (other attributes "preserved" from
        the conceptual class Story}
toAuthor: Anchor (Is Author of)
END
```

In the third phase OOHDM supports user interface design. This is the last phase before coding. It is called as an Abstract Interface Design phase and it defines the content and the layout of the user interface without determining what exactly it looks like. The abstract interface design is realized through constructs called *abstract data views (ADV)*. An example of ADV can be seen on Fig. 2 [6].

As the abstract data views describe the user interface draft, they should also construct static links with the navigational design. Additional configuration diagrams are used for that purpose. In the implementation phase the abstract data views serve as, well-defined requirement documents.

Although the OOHDM approach is similar to the WebML approach in general, it puts more emphasis on navigational design. OOHDM designs are much more abstract than WebML designs and they contain many gaps to be filled by

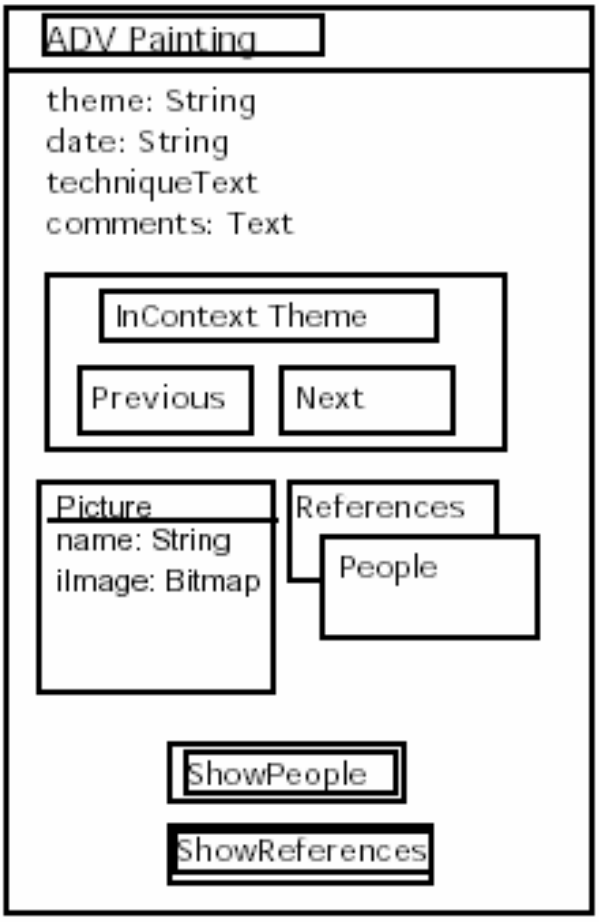

Fig. 2 OOHDM Abstract Data View [6]

the implementer. OOHDM is also more flexible and in that sense more in parallel with the hypermedia design philosophy. On the other hand both approaches are equally heterogeneous conceptually since the natures of the tasks that are required in the development process phases are quite different. It is our opinion that these web development technologies would benefit from a unifying paradigm not only because they currently involve four distinct phases, but because the contextual relations between *content* and *form* are not yet clarified.

B. *Web Patterns*

Web patterns are similar to the Design Patterns [7] of Object Oriented Software Design. In fact with their emphasis on ease of use and aesthetics [8] they are much closer to the original pattern concept than the Software Design Patterns. This kind of fidelity is uncommon in the concepts that are imported into the software development sector.

Web patterns are acquired through observations made on the Web itself and are the results of certain general Web dynamics:

- The users of the Web implicitly expect user interface standards that tend to make the structure of the artifacts easy to perceive and easy to use.
- There is the necessity to present the same content in a number of different forms and this requires the form and content to be separate. Form may variable just as content.
- The economy of navigation should be preserved. The users prefer the minimum number of clicks to reach a certain place.

A typical example of Web patterns by Rossi et al. is Set-based Navigation [8]. This is the kind of pattern that we may have encountered numerous times while browsing the Web. It involves a set of equivalent nodes that the user would like to navigate through. The user in this case may want to see each node at least once or may prefer to see just a subset of the content. The pattern ensures easy and economical navigation inside the set.

In the Fig. 3 a representation of Set-based Navigation pattern can be seen. The basic solution that the pattern provides is that the navigational facilities can be improved by using standard navigational elements on each node to move forward and backward and at the same time employing an index that provides individual access to all of the nodes at once.

The pattern concept is a good point to start for creating a unification paradigm for Web design. It should be strengthened however with linguistic concepts such as contextualization in order to automate pattern usage. It will probably take a lot of time and discussions to reach modeling elements with semantic soundness that enable machine intervention in their usage.

III. WDLDL PROJECT

The Web Design Logic Definition Language Project reflects our approach to Web design that supports the formalization of patterns as reusable models stored as script code that constitutes the base of the system.

WDLDL is designed as a layered architecture. At the bottom there is a kernel that provides a library of model definition and transformation services. The common interface of these services is a functional scripting language called BLUE Code (Basic Linguistic Unification Environment). Any visual user interface or high level language is based on this layer and commands the modeling facilities through this language. This small language is also constitutes a gate through which various input-output components can connect.

In its current state BLUE Code is only partially implemented. However due to its basic structure that is also very symmetric with the kernel, it enables the prototyping and testing of the kernel functionality at a rather early stage of the development process. Each time a new functionality is added to the kernel, either a new command (function) is also added to the BLUE Code syntax, or an existing one is extended in order to meet the new functionality. The script interpreter is then used for testing and tuning the new code. Fig. 4 shows the architectural layers of WDLDL project.

The BLUE Code language that aims managing and interfacing the kernel has an easy to learn function-based LISP-like syntax. This language has constructs for creating various data structures and variables, assigning values to variables, performing basic operations and transformations

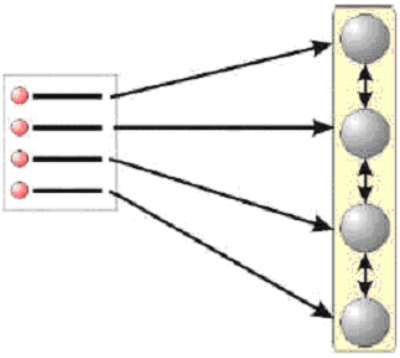

Fig. 3 Set-based Navigation Pattern [8]

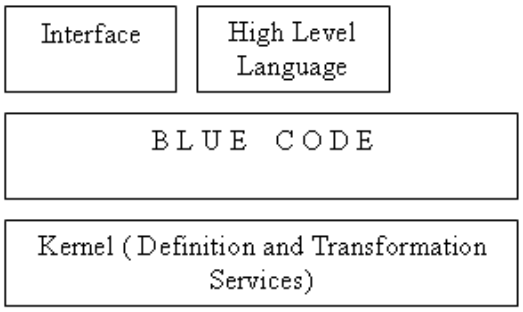

Fig. 4 WDLDL General Layered Architecture

and managing input and output. It also has the ability to store information in various forms such as trees, lists, sets, entity-relation graphs or script code. These can enter many operations as well as being assigned to variables, and some can also be transferred to (and from) persistent storage. Knowledge representation and management is at the center of the project. The choice of such a functional language approach as the backbone of the project is in many ways advantageous:

When system functionality is arranged as functions, although the language is not side-effect free, there is no contextual assumption that is required in order to assess the success (or failure) of the attempted system function. Each call is on its own for achieving its goal or if it fails providing the necessary feedback for explaining the reasons of failure. This coding standard provides an excellent modularization that enables the developer to modify code, in order to add, delete or change a certain behavior, without taking risky design decisions.

Secondly if system behavior looks suspicious or gets overly complicated at any time and the developer (or tester) fails to follow the execution sequence, it is possible to add new functions for monitoring the kernel with higher transparency with linear cost.

Although each function implements a very basic task, it is straightforward to use them together by nesting, in order to implement complicated tasks.

Functional languages typically have a very basic syntax that is composed of just a few kinds of lexemes. This makes it very easy to write a parser and as a result the language interpreter is very plain, lightweight and easy to maintain.

The Fig. 5 shows a data flow diagram that depicts borders of the BLUE Code subsystem with dashed lines. The diagram also shows the input/output scheme around the WDLDL kernel.

The WDLDL kernel is not just functional but also has some object-oriented and semiotic-influenced features. The run-time role of the BLUE Code interpreter is not unique to the interpreter but every linguistic element (variables for example) theoretically can adopt the role of being a local interpreter. In fact the activation of any function creates a *read* operation or a *write* operation or an *execution*. And any of these three operations requires three objects in the kernel, each accomplishing a different role. Table I shows the relation between the roles and the operations.

Table I. Roles and operations

Operation	**Active**	**Passive**	**Object**
Read	Reader	Read-Source	New Object
Write	Writer	Write-Target	New Object
Execute	Executor	Context	Code

The reading, writing and executing operations can be visualized as reflections of the three entities of the basic model of computation: *Input*, *Process* and *Output*. And the three roles labeled as *Active*, *Passive* and *Object* are concepts that originate from the basic model of communication that involves a *sender*, a *receiver* and a *message*. The Cartesian product of the two set gives nine different roles that any object can play in any operation. Of course not all BLUE Code objects would be suitable for playing all of the nine roles; for instance a plain string constant can not have an execution-related role. However all the nine methods are defined for any BLUE Code object, as they all inherit from a common ancestor, some with an implementation and some with a placeholder in order to maintain extendibility.

IV. USING BLUE CODE

The aim of the WDLDL project is to be able to produce, store and transform reusable designs and more primitive forms of information as patterns or graphs. The production of Web pages by using such stored information is the ultimate goal and that is also a form of graph to code transformation (also known as *model-to-code* transformation).

Patterns of various degrees of abstraction, concrete designs, and code itself are the artifacts that will be represented as graphs in WDLDL. At the moment as one of the prototype functionalities the BLUE Code language can be used in order to create and manipulate basic graphs and then a special transformer object can be employed for automatically generating static Web pages from such descriptive graphs.

The manipulation of graphs and their transformation into web pages is the first step of a development process that will gradually increase the level of abstraction. It is our view that WDLDL architecture is very suitable for such constant positive progress. The bridge between diagrams and Web material is a strategic achievement that makes Web material more manageable and the diagrams more visible.

In Fig. 6 a test graph is shown. This is a basic E-R graph

ISSN: 2277 – 9043
International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)
Volume 3, Issue 10, October 2014

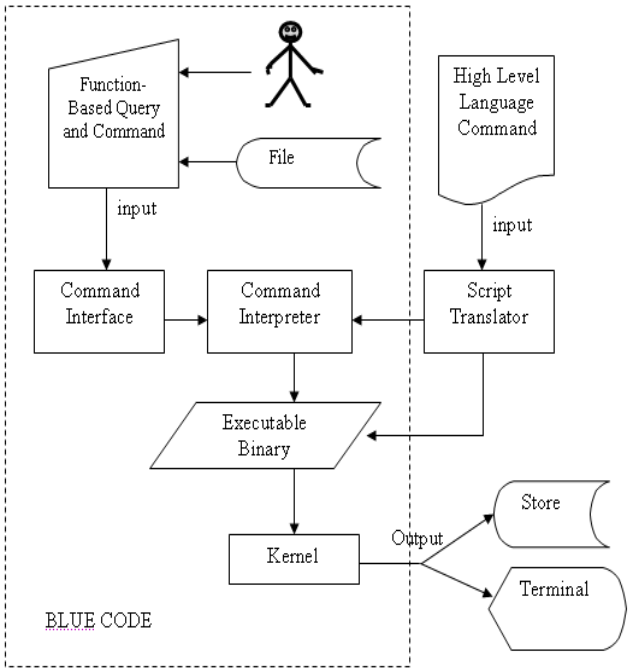

Fig. 5 WDLDL Kernel and data flow

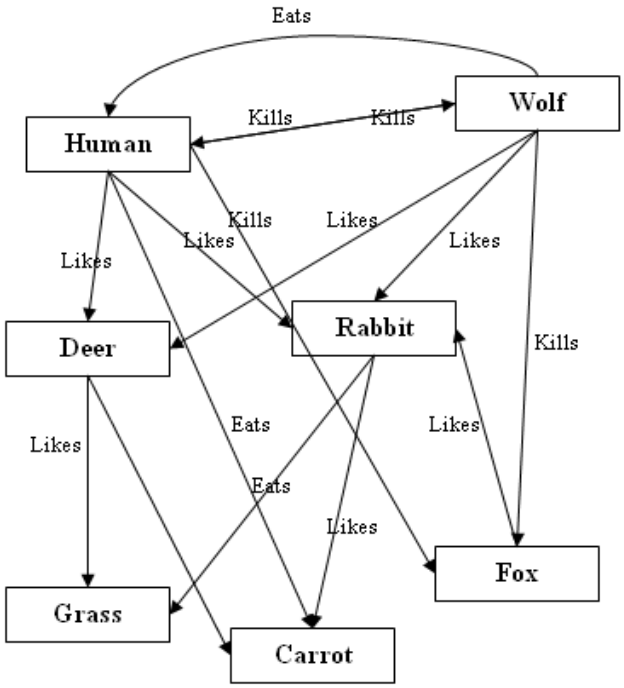

Fig. 6 The Ecology graph

that shows the ecological relations between some animals. The BLUE Code script for creating this graph and then turning it into a set of Web pages is as follows, (The original code including the function names has been translated into English for the ease of comprehension.):

```
(assign;web;(graphWeb)). //Web creator
(assign;grph;(graph)).
(add;grph;(rel;"Wolf";"Likes";"Deer")).
(add;grph;(rel;"Wolf";"Likes";"Rabbit")).
(add;grph;(rel;"Wolf";"Kills";"Fox")).
(add;grph;(rel;"Wolf";"Kills";"Human")).
(add;grph;(rel;"Wolf";"Eats";"Human")).
(add;grph;(rel;"Human";"Kills";"Fox")).
(add;grph;(rel;"Human";"Kills";"Wolf")).
(add;grph;(rel;"Human";"Eats";"Rabbit")).
(add;grph;(rel;"Human";"Likes";"Deer")).
(add;grph;(rel;"Human";"Likes";"Carrot")).
(add;grph;(rel;"Fox";"Likes";"Rabbit")).
(add;grph;(rel;"Deer";"Likes";"Grass")).
(add;grph;(rel;"Deer";"Eats";"Carrot")).
(add;grph;(rel;"Rabbit";"Likes";"Carrot")).
(add;grph;(rel;"Rabbit";"Eats";"Grass")).
(write;web;grph).
(execute;web). //Create the files
```

In BLUE Code notation the function name is written as the first item in a list of items that appear between parentheses. The items are separated by semicolons. The second item is the first attribute of the function and so on. Functions are freely nested in each other and type checking is performed at run time while each function is executed. Period is the end-of-line mark, and comments are C++ style. Quotation marks indicate string constants as usual. The *graphWeb* function in the first line of the script creates a *Web creator* which when loaded with a graph can create the corresponding Web artifact just by being executed by the script interpreter. So this type of structure can be a "Write target" and then "Code". The assignment that encloses the creation, immediately gives it the name "web".

Inside the second assignment a new graph is created by using the "graph" function (named as grph). In the next 15 lines new relations are created and added to this graph. Then the graph is written onto the graphWeb instance. The write operation on a graphWeb creates the Web version of of any complete graph or set of relations that are written onto it. If there is already a Web design artifact inside, the write operation inserts the relations into those that are already there. If there is nothing inside, it is created. When a graphWeb instance receives a message to be executed (that is what the execute function in the last line does) it creates all the required HTML files.

Currently the web creator obtains the necessary image files from a specific directory through a naming convention that uses the node names as a key, however the relational scheme can easily can be extended to include references to images inside the graphical model.

The creation of the HTML files takes place by calling the string manipulation services of the script interpreter. Fig 7 shows the command line interface of WDLDL through which BLUE Code scripts can be called. The last command given has been for running the Web creation test script and the command line interface responded by reporting the new files that are created in the process. The number 19 at the end is the number of lines of script that was executed. Smiley indicates success.

Fig. 8 shows two of the web pages that were created by the script. The relations defined by the graphical representation are transformed into hypermedia links. By adding different

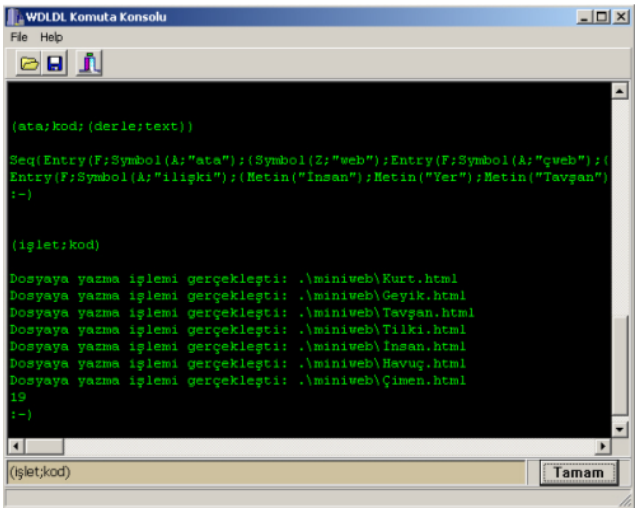

Fig. 7 BLUE Code Command Console

Fig. 8 Web page examples

kinds of web creators to the system both an aesthetical improvement can be achieved and the better representation of knowledge on the Web can be realized. The Web development approach that was described in this work is a part of a more general point of view in model-driven software development that uses the model concept as the central term for paradigm unification.

Models in general can get very complex and it is often hard for the developers to follow. The visualization scheme described here may help providing models that can be followed and browsed as necessary.

V. CONCLUSIONS

The current approaches to Web design do not attempt to take the advantage of the flexibility that the Web provides. Instead they just try to carry the RDBMS (database) or object-oriented or document-based approaches to the Web domain. However the Web is more flexible than anything that it contains and in our opinion the full power of the Web will be unleashed when automats are enabled to work on content and structure in a similar fashion.

In this work Web design Logic definition Language (WDLDL) project is introduced and some of its architectural properties are given. The representation and manipulation of knowledge is the heart of the project. The WDLDL approaches designs as reusable models that can freely read, write or execute each other. The focus of this paper was on the kernel of the project and its script based interface the BLUE Code language. Together with being the most important elements of the architecture they are also the only major parts that are currently implemented.

In large-scale projects of modeling, model transformation or software development environments (in fact for large scale system programming in general) it is a good idea to develop at an early stage a functional scripting language that has full access to the kernel functionalities. Such a language is extremely useful for rapid prototyping of modifications and helps detecting design and requirement errors before they get too harmful.

In the current state of the WDLDL project the link between graphs and Web code generation has been established but this is only a technical issue. The real hard task is to adequately and elegantly represent the conceptual, navigational and aesthetical (related to layout) aspects of Web design as relational models. The advantage of WDLDL is that it is based on a sound functional language: BLUE Code which has very robust but flexible concepts and constructs (like reading, writing or executing whole graphs at once) that makes it easier to deal with complicated model transformations. The knowledge-based approach to Web page creation enables reusing the same information content in different transformations to create structurally different Web sites.

Future work will hopefully include attempting to unify the paradigms related to Web development. The design of the WDLDL is expected to evolve towards supporting a single model-driven point of view for unifying all the phases of Web development. The base of this project has been established accordingly.

REFERENCES

[1] D. Schwabe, G. Rossi, S. Barbrosa, “Abstraction, Composition and Lay-out Definition Mechanisms in OOHDM”, in Cruz,I.F., Maks, J. and Wittenburg, K. (eds.) Proceedings of the ACM Workshop on effective Abstractions in Multimedia, San Fransisco, CA, 1995.

[2] S. Ceri, P. Fraternali, A. Bongio, “Web Modelling Language(WebML): A Modelling Language for Designing Web Sites”, WWW9 Conference, Amsterdam, 2000.

[3] “WebML: the Web Modeling Language: Tutorial with Audio and Slides”, The Web Modeling Language homepage, Politecnico di Milano, http://dbgroup.como.polimi.it/brambilla/webml, accessed: 17 October 2014.

[4] A. Gu, B. Henderson-Sellers, D. Lowe, “Web Modelling Languages: The Gap Between Requirements and Current Exemplars”, The 8th Australian WWW Conference, 2002.

[5] N. Moreno, P. Fraternali, A. Vallecillo, “WebML Modelling in UML”, IET Software, vol.1, no.3, pp. 67-80, 2007.

[6] D. Schwabe, G. Rossi, “An Object Oriented Approach to Web-Based Application Design”, Theory and Practice of Object Systems vol. 4 no. 4, Wiley and Sons, New York, 1998.

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, “Design Patterns, Elements of Reusable Object Oriented Software”, 1st edn. Reading, MA, Addison Wesley, 1995.

[8] G. Rossi, D. Schwabe, F. Lyardet, “Patterns for Designing Navigable Information Spaces”, Proceedings. of PLoP98, Monticello, USA, 1998.

Ahmet Egesoy (PhD in computer engineering) is an instructor and Assistant Professor in Computer Engineering Department of Ege University Izmir, Turkey. Research interests include object-oriented programming, design patterns, model-driven software development, programming languages and related paradigms, philosophy of the language, semiotics and knowledge representation.