Optimisation of an Intelligent ASP.NET User Interface

Plamen Paskalev, Georgi Panayotov

Abstract: *An existing software system, based on CLIPS scripts is analyzed. The main advantages and problems of using such an approach for WEB solutions, based on ASP.NET are discussed. A sequence of steps for interface transformation, modification of the internal structure and improving of the main algorithm of the WEB pages generation is overviewed. The effect on the intelligent, CLIPS-based control of the user interactions is discussed.*

Keywords: *Computer Systems and Technologies, Intelligent User Interfaces, Object oriented programming, CLIPS, ASP.NET*

INTRODUCTION

The paper [1] describes an approach for creating of user interface modules of an application, based on a platform independent script file, describing interface and interaction between the user and application. The user interface as long as parts of the business logic of the application are based on the expert system shell and language CLIPS [10]. An approach for building of Intelligent User Interface, based on generic rules and a knowledge base was realized for handling of complex control interaction [11]. Modules for visualization of the interface for each target platform have been developed. As far as the CLIPS engine and the script itself are used without changes in the different versions, these modules are responsible for parsing, creation and handling of the user interface, performing of the platform-specific tasks and communication with the CLIPS engine, which provides the connection to the business logic module, databases etc.

PROBLEM

During the testing of the real system in multi-user environment some problems and obstacles were found, which required a careful analysis of the current structure and looking for possible improvements of the system. The current paper describes this process of analyses and restructuring steps concerning the WEB solution as part of the software system.

WEB SOLUTION, BASED ON CLIPS SCRIPT

The WEB solution was created based on ASP.NET technology. It has following main characteristics [1]:

- Clear separation between business logic and GUI layers
- A script language was used for GUI description and preparation of user templates
- Preparation of modules for visualisation of the interface for each target platform
- Integration of a rule-based intelligent user interface agent with own knowledge base.

The following benefits were observed in this system, compared to the ones, built using standard approaches:

- *Platform Independent* – The user interface is built as a script, which is used without any changes in all the used platforms.
- *Logic implemented rules* – In contrast to the procedural programming languages, this approach gives much more instruments for realization of flexible user interface.

- *Fuzzy calculation logic* – CLIPS integrated fuzzy logic makes possible implementation of fuzzy rules in handling of UI.
- *Artificial intelligence* - intelligent control of the user interface was integrated, with capabilities to collect information concerning the user behavior and to modify dynamically the user interface using generic rules

Some serious problems were found due to the chosen design and implementation. Most of them depend on the web platform specific flaws and used development platform - .NET Framework 1.1. the following negative results were observed during the detailed analysis of the application:

- *Server timeouts* – because of the specific architecture of web applications, big delays in the business logic have to be speeded up or fully thrown away. Of course, there is another possibility – to use a special way of processing large business logic (to process data asynchronously from the web user interface for example). Big delays were encountered when the model was generated (fig.1) and the interface was rendered (the whole user interface description was stored in a single file, parsed by the application server, despite that most of it is not visible at all time). This point depends on the model, which structure and complexity is unknown at the time of developing.
- *Dynamic code generation* – very powerful mechanism, used to generate run-time logic which is converted to platform native code (MSIL). Despite of this, difficulties in debugging and implementing of new features were encountered.
- *Losing session state* – The session state has been lost, which contradicts to the web application design. This point is based on a well-known problem [2], which appears in ASP.NET applications when a code is been dynamically generated. This reason enforces restriction of dynamic code generation by the time of installation or total removing it from our conceptual design.

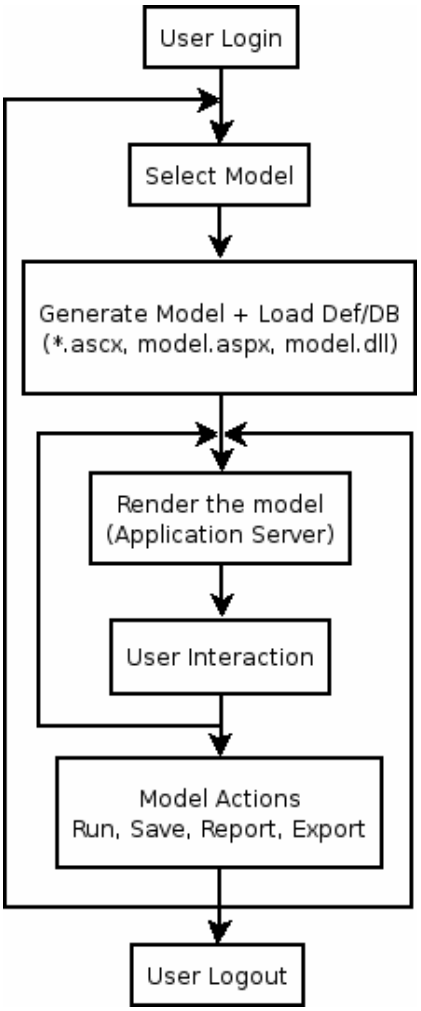

Fig. 1

- *Multi-user environment* – the system could be accessed by multiple users at the same time. There exists duplicating of static data and temporally files for all users (who may exceed thousands)
- *Model actions* – There exist three possible destinations for the data: *calculation* (interpreting the script rules by CLIPS, moving thorough the expert system logic), *storing to database* (using different ADO.NET Database Providers), *exporting to XML* (for further processing with XSLT, generating reports with Crystal Reports or just storing temporary model data). In the previous implementations they were messed up communicating with each other. This brings up a lot of problems in testing and implementing new features – as an example adding native support for other report engines.

- *User controls* – they form the user interface, interpreting the model structure. They were not separated logically, and they depend on each other. This makes the testing and adding of new ones a really difficult and time-expensive task.
- *Core vs. Presentation* – There existed strong dependency between the core and the presentation layers. This makes the implementation of new presentation layers difficult and expensive. This feature is needed because there are multiple browser dependent flaws that could be solved with duplicating the presentation layer. Another reason is the further plan to implement mobile interface for Pocket PC and Palm devices

RESTRUCTURING AND OPTIMIZATIONS

To surmount these problems, some restrictions were added and some subsequent transformations in the application design were performed. The primary goal is to keep using as much as possible from the current code and in the same time to change the ideology and main conceptions. The objective is to expand the system, removing the main problems discussed above.

A set of static data, stored in files created at the model installation/update in the system was defined. They depend on the model script, but they are changed rarely or never. The goals are minimizing the big delays and sharing the duplicated structures between users. The following types of files were defined (fig.2):

- *Tree* – represents the structure of the model in XML format. Next step is to directly transform it with XSLT and make a tree control on the client side
- *Data Schema* – hierarchical description in XML format of model data-holders. It serves to generate and initialize dynamic structures in memory when we start working with model. This structure fully describes the user session.
- *Default Values* – meaning of all data-holders' values created with data schema. They are used as a source when a new session for the model is created. Otherwise the values stored in database will be used instead.

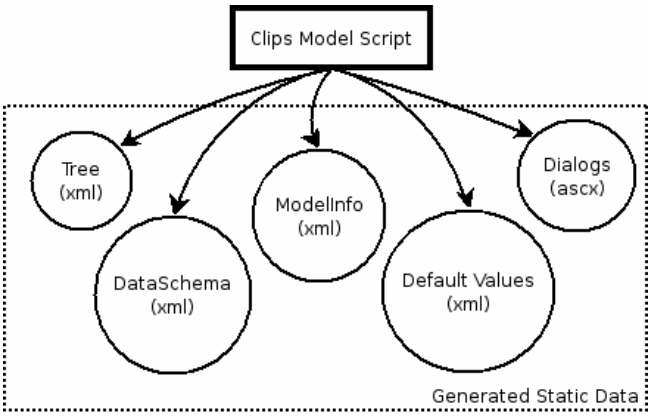

Fig. 2

- *Dialogs* – a set of *.ascx files (Web User Controls), used as template description of the interface. They are passed to application server (MS ASP.NET) that generates the interface in HTML passed to the client side. Includes all properties that are changed rarely or never, like positions, fonts, validations etc.
- *Model Info* – information for the model in XML format: name, version, description, author, copyright, uploaded by, upload date, etc. There is a possibility to implement simple source control versioning system. This will give additional control over the system and security at all. In addition, session data could be interpreted with script version that created it – backward compatibility

These additional files help interpreting the model faster and less expensive. In addition, several layers were defined in the conceptual design. They are invisible for the end user, but help adding new features and extending existing ones. These layers are:

- *Session Data Structure (SDS)* – dynamic data structures in memory generated with model data schema
- *Data Destinations* (SD) – several “destinations”, where the data goes to be processed, stored or showed to the user (CLIPS, Database, XML, View)
- Data Controllers & Handlers – help and unify the data transfer between the SDS and SD (fig.3)

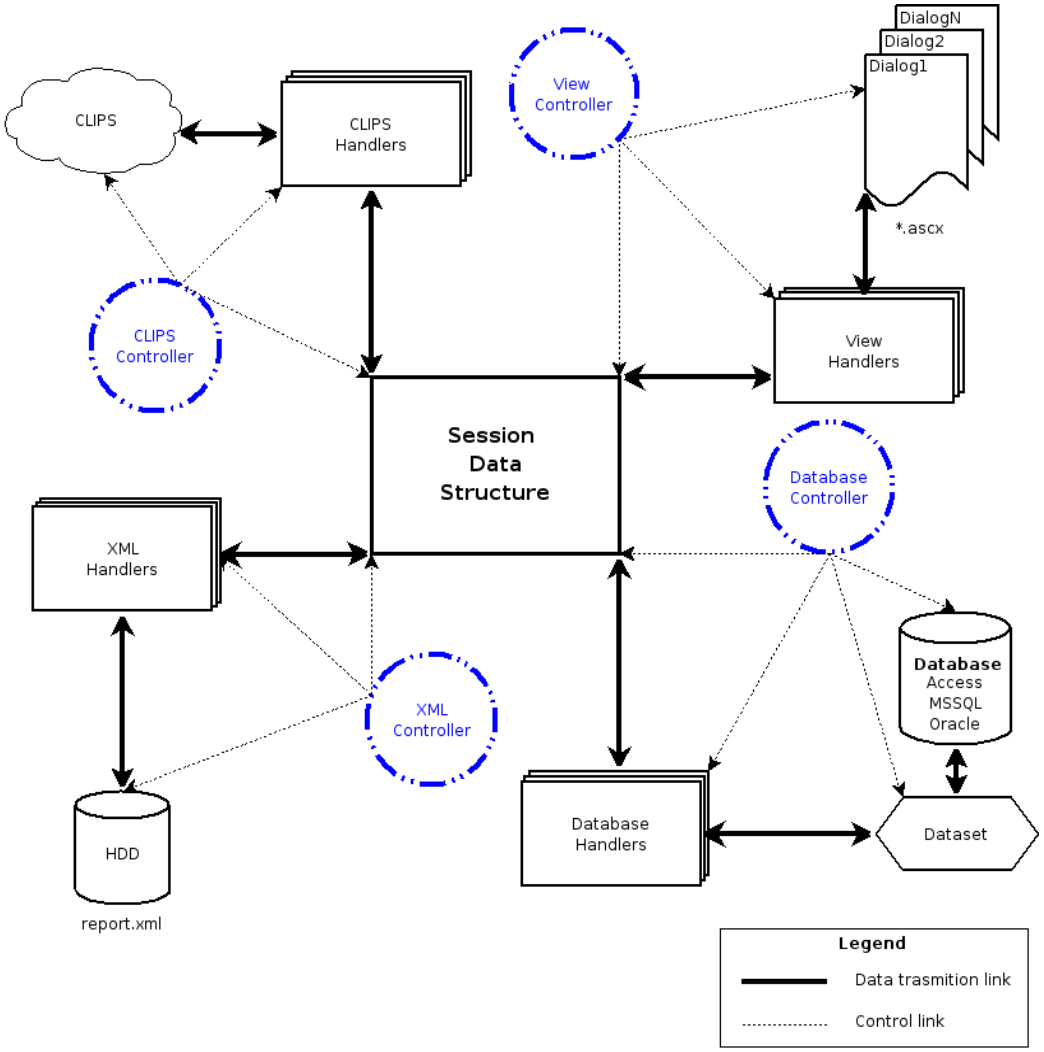

Fig. 3

The user interface interacts only with different *controllers*. There is a pair for every data destination – load and save. The *controller* initializes objects to communicate with specific data destination (as an example the CLIPS interface), iterates over all elements of session data structure and calls a *handler*, passing a reference to the element and data destination interfaces plus some additional information like current level of inheritance, that process the specific type of element. From this point implementation of different handlers is a very simple task. Handlers and controllers are regular methods of classes, grouped in array lists of delegates. Every element in the session data structure has got an index to these containers, so it is self-defining. In addition, a handler or a controller could miss in one or both directions (save, load). Handlers and elements in the session data state don’t know anything about controllers, so they could be added without changing the controller.

Using this scheme, the following interaction flow charts were achieved:

Administrator's point of view:

- Select existing or new model
- Collect information for the model and get the file from the client machine
- Process the input information, generate working files (fig.2), index the model globally in the system and make it available for end users.

End user's point of view:

- The user logins in, selects model and session
- According to the selected model the SDS is constructed using the description in the DataSchema.xml
- According to the selected session – new or existing, the SDS is filled up with data from DefaultValues.xml using XML Controller or from the current database connection using DB Controller.
- Showing the default dialog from the model, where data is being loaded dynamically using View Controller.
- User interacts with different dialogs, enters or changes data that is passed to SDS between post backs using View Controller.
- User selects one of the interactions with the model – calculation, storing to DB, exporting to XML, each of them realized using the corresponding controller: Clips Controller, DB Controller and XML Controller.

AFFECTS ON INTELLIGENT USER INTERFACE MODULE

In the existing system some aspects of the intelligent user interfaces were realized [1]. A module, responsible for handling of knowledge base with data for the user behaviour was created. Two processing levels are separated – education and implementation. Both levels included CLIPS rules groups. High level rules analyze user interactions and calculate statistical parameters, used in low level – implementation.

A set of interface description templates is read by the Inference Engine and used to define the initial control set contents and settings. A corresponding set of rules is loaded into the rule base. Templates and rules are read from script files. Additional module, included Generic rules, collect facts about user behavior, actions and decisions, and modify the existing scripts in real time. As discussed in [7], unobtrusive data collection techniques through the use of individual user profiles, capturing the user natural interaction gives better results in determining of the user's personal preferences compared to explicit collecting of data using complex and often boring questionnaires.

The transformations, discussed in this article resulted in increased productivity of the intelligent user interface module.

PROS AND CONS OF THE PERFORMED TRANSFORMATIONS

Pros:

- The system now could serve multiple users, each of them working with more than one model at the same time. This could be done, because the SDS is able to determine different sessions.
- Additional data destinations could be easily added, where the data is processed, stored or just showed to the user. As an example, the application can interact with a remote system using Web Services, .NET Remoting, Corba, etc. As a result, it can interact with systems based on different hardware and software platforms – C/C++, Java etc.

- New user interfaces can be added quite easily – desktop, mobile front-ends. In addition, one can define different front-ends for a variety of browser types and versions. This goal is achieved with changing only the View Controller and its handlers.
- New user controls can be developed that could be tested and deployed less time-expensive. In addition, there could be different versions of controls deployed with different version – standard, professional, advanced. This goal is achieved by developing a standard Web User/Custom Control and corresponding handlers that support it. This could be packed in one file and deployed to the working system.
- The simultaneous work with different expert systems (FuzzyClips, etc.) is flawless. They could bring additional features as a result of rising cost-of-ownership for the system
- As an opportunity to growth, the system can be distributed to work between multiple computers. This approach will increase the overall system loading and shorten the response times (load balancing). In addition, the main sections can be duplicated to increase the system reliability (High Availability).

Cons:

- Duplication of data – could be fixed if the system is binded to specific external ones (CLIPS), but the homogeneous design will be thrown away. This solution could bring additional difficulties in extending and supporting the system.
- Change the model structure on the fly requires some extra processing. This complicatedness is excusable with the achieved flexibility and scalability of the system.

CONCLUSIONS AND FUTURE WORK

The practical application of the discussed approach has led to the following conclusions:

- The using of script based User Interfaces extends the flexibility of the software design process.
- Developing of platform - specific parsers gives the opportunity to use the same scripts for different platforms.
- This approach gives wide range of opportunities in implementing of rule-based intelligent UI modules.

REFERENCES

[1]. Esposito, Dino, Building WEB solutions with ASP.NET and ADO.NET, Microsoft Press, 2002

[2]. KB324772 PRB: Session Data Is Lost When You Use ASP.NET InProc Session State Mode, http://support.microsoft.com/default.aspx?scid=kb;en-us;324772

[3]. Giarratano, Joseph C., Ph.D. CLIPS User's Guide Version 6.20, March 31st, 2002.

[4]. Kasabov, Nikola, K. FOUNDATIONS OF NEURAL NETWORKS, FUZZY SYSTEMS, AND KNOWLWDGE ENGINEERING, MIT Press, Cambridge, Massachusetts, London, England, 1996.

[5]. Jeff Webb with Microsoft Corporation, MCAD/MCSD Self-Paced Training Kit: Developing Web Applications with Microsoft Visual Basic .NET and Microsoft Visual C# .NET, Second Edition, MS Press, 2003

[6]. Richter, Jeffrey, Applied Microsoft .NET Framework Programming, Microsoft Press, 2002

[7]. Jungsoon Yoo, Gervasio Melinda, Langley Pat, “An Adaptive Stock Tracker for Personalized Trading Advice”, in Proceedings of the 2003 International Conference on Intelligent user interfaces

[8]. Mandel, Theo, “The Elements of User Interface Design”, Wiley, 1997

[9]. Lieberman, Henry, “Integrating User Interface Agents with Conventional Applications”, in Proceedings of the ACM Conference on Intelligent User Interfaces, San Francisco, 1998

[10]. Dryer, D. Christopher, “Wizards, guides, and beyond: rational and empirical methods for selecting optimal intelligent user interface agents”, in Proceedings of the 1997 International Conference on Intelligent user interfaces

[11]. A. Antonov, V. Nikolov, Y. Yanakieva Rule-based Rating Model, CompSysTech’2002, proc. of the International Conference on Computer Systems and Technologies, Sofia, Bulgaria, II.7-1, II.7-6.

[12]. A. Antonov, Y. Yanakieva, V. Nikolov Rule-based Rating System, CompSysTech’2002, proc. of the International Conference on Computer Systems and Technologies, Sofia, Bulgaria, II.8-1, II.8-6.

ABOUT THE AUTHORS

Eurorisk Systems Ltd.
31, General Kiselov Str.
9002 Varna, Bulgaria
Plamen Paskalev
E-mail: E-mail: ppaskalev at eurorisksystems dot com
Georgi Panayotov