New Features of the OPScript Language

Vicente Arturo ROMERO ZALDIVAR
University of Cienfuegos
Carretera a Rodas Km 4, Cienfuegos, Cuba
Jon Ander ELORRIAGA ARANDIA
University of the Basque Country
Apdo. 649 P.K. - 20080 Donostia - San Sebastián, Spain
Mateo Jerónimo LEZCANO BRITO
University of Las Villas
Carretera a Camajuaní Km 5, Santa Clara, Cuba
Mikel LARRAÑAGA
University of the Basque Country
Apdo. 649 P.K. - 20080 Donostia - San Sebastián, Spain

Abstract. In this paper the authors make a brief introduction to the YADBrowser project which includes the educational browser YADBrowser and its script language, OPScript. The authors expose also some features added to it, with them the author of an educational application can achieve more functionality and adaptation with less code. Also the YADBrowser reduces the interactions with the server reducing the response time and keeps a record of the actions and preferences of the user. These new features were added to facilitate the creation of dynamical applications, adaptable to student skills. They include "verbal" communication between objects, XML object models and reusable methods.

1. Introduction

Internet has been highly used as an effective means for publishing information of any kind. The Web is also a natural field for educational hypermedia applications [1]. Nevertheless it is hard to construct context-based adaptive Web applications for many reasons: static links between documents, the stateless nature of the HTTP protocol, etc. In despite of these problems authors are always creating applications, models, languages etc., to offer adaptive educational applications to students according to theirs knowledge and skills [2, 3].

In order to create these educational applications it is very important to have languages which easies the work of the developers, it is also necessary to consider that sometimes authors of educational applications do not have a solid knowledge of programming languages. So every tool must be powerful and easy to use for a wide set of authors. Automatic generation of code and code reuse are also desirable characteristics for a new language so many features can be added to a given application with less work. There are some works related with the development of new browsers or script languages with

educational purposes [4]. There are also reports of the development of browsers for general and specific domain applications. An interesting example is the Grendel browser [5], this browser allows the user to script the browser user interface, the browser's HTTP interactions, and the browser's rendering of documents. To do so the authors define a language, CrossJam, similar to Lisp, using it the user can perform a great degree of customization. Other authors have notice that collaborative browsing is more valuable than lonely browsing for educational purposes, so there are reports of tools for collaborative browsing [6]. Another issue is browsers adaptable to computing resources, location of the user, etc, [7].

There are other issues between authors; one of the most important of them is the production of frameworks which facilitate the creation of educational applications using current technologies. In this field, for example, the Avante architecture [8], and the MTeach framework [9] can be found. Another important issue is semantic relationships and consistency in hypermedia. There is also great interest in lesson components and courseware reusing [10, 11].

2. Brief Introduction to the OPScript Language and the Educational Browser YADBrowser

The OPScript language [12, 13, 14] is the script language of the educational browser YADBrowser which main objective is to develop a browser suitable for educational applications with a language that allows a fast development of these applications and other facilities like persistence of selected information in browser memory, lesson components reuse, etc. Some authors have stated the advantages of saving information in the main memory of a given browser [15]. Persistent additions can be very useful in educational applications, because they reduce the response time, the code needed to create a given application, etc.; for this reason the YADBrowser and the OPScript language includes many features to persist information in browser memory. When information is saved in browser memory, bandwidth and time can be saved. There is a great interest in bandwidth saving and many techniques including sharing information between browsers have been reported [16, 17]. Also, the YADBrowser includes an object model with a wide collection of objects adapted to educational applications; there are reports of the development of object models for Web applications to save effort and time [18].

An important member of the object model of the YADBrowser is the *TUser* class, only one object of this class is created for session and it is available in every page using client side code. This object keeps user generated information such as pages visited by him or her, score obtained in the solution of exercises and other information related to the user profile. Also the *TUser* class defines methods to add application specific information related with the user and to later request that information. This information can be useful to decide what content must be shown to the user, and content-based adaptive applications can make use of it to display information according to user knowledge and skills. For example the target of a given link could be redefined according to the level of a given user, beginner, intermediate or expert, or a graph of actions necessary to operate a medical equipment can vary according to the level of the user and to a given pathology selected by him. This information can be present in the *User* object and used on demand, by the way interactions with the server can be reduced because the information is always present in the browser overcoming this way the stateless nature of the HTTP protocol.

Another important aspect is the markup language used by the YADBrowser which is a subset of HTML 4.01 [19]. Some additions have been made to this subset for achieving a better adaptation to educational applications requirements. Between these additions the more important is the pattern tag which allows defining some content that should be reused in several Web pages, this way it is possible to define a content once and use it several times, maybe adapting it when necessary. Patterns are another way of defining persistent information. Some authors have created new markup languages for domain specific applications some of them are XML based languages [20], the language defined by these authors, named MeML, was created for publishing mathematical content on Internet.

Initially the OPScript language was defined as a language without types, like JavaScript [21], nevertheless at present it defines the following types: integer, string, boolean and object. With these types it is possible to check better the correctness of programs, which is desirable even when code complexity can increase a little. The language is similar to Object Pascal but has some elements of the C++ language, for example variables declaration. OPScript has constructions to define what classes, methods or fields has to persist to be used in a coming page, decreasing this way the bandwidth needed and also the time to respond to a user generated event.

The experience obtained with the development of this project could show what characteristics are desirable in a language and a browser created specially for educational applications or what should be included in a standard browser to make it more suitable for educational applications.

In this article some features included lately in the OPScript language will be exposed, which make the language suitable to develop dynamical educational applications, reduce the amount of code necessary to implement them and the programming skill needed to create a given project.

3. Metadata and Reusable Methods

Metadata have proven to be very useful when used in programming languages, even in aspect oriented programming (AOP), they have proven to be very useful, for references about AOP see [22, 23]. They allow annotating a field, a class, a method, etc., metadata can be obtained later by reflection and special treatment can be done to a given programming element according to the metadata applied to it. In OPScript metadata can be defined by placing it in brackets before the element to be annotated. This is shown in the following fragment of code:

```
TStudent =
  class
    string [level] fLevel;
  end;
```

Later if it is needed to retrieve a field annotated by a metadata it can be used a set of functions present in every object, because they belong to the *Object* class and in OPScript every class inherits by default from this class. In OPScript metadata are the main support to the reusable method concept. A reusable method is one that can be called by objects of different classes, no matter which is the original class that defines the method. Using the sample code above the field could be retrieved inside a reusable method like this:

```
var
```

```
  string s;
begin
  ...
  s := GetStringFieldVal('level');
  ...
end;
```

The function used above to retrieve the field is applied to the calling object, no matter its class. When a reusable method is called it is executed as if it were a method of the calling object class. Actually what happens is that a class lends a method to another class and, no matter this fact, everything works properly. This can be confusing, but is very useful. To make a method reusable, it can not access the fields of the current object directly, but through metadata. Reusable methods decrease the amount of application code, making it simpler and they are an important component of the feature exposed below, "verbal" communication between objects.

4. Verbal Communication between Objects

This new feature makes possible the interaction of two objects without additional programming and with almost no knowledge between the objects. The relation between the objects is created automatically whenever it is necessary by the browser. So the creation of a Web application can be reduced, at least an important part of the process, to the inclusion of some objects in several pages and allowing the browser to link them automatically. With verbal communication, two objects can interact even if the interfaces of one or both objects change or even if any of them changes almost completely.

Verbal communication is important because it speeds up programming and makes the development of useful and real educational applications easier to people with less knowledge of programming languages. The following sample code shows how verbal communication works in OPScript:

```
TExercise = class
  needs Play;
  fields
    string [mediafile] fPath;
  methods
    procedure PlayMedia();
    begin
      ExecuteVerb('Play');
    end;
end;

TPlayer = class
  methods
    procedure DoPlay();
    var
      string Path;
    begin
      Path := GetStringFieldVal('mediafile');
      ...
    end;
  offers Play(DoPlay);
end;
```

In the sample code above, with the use of a hypothetical example, it is shown how two classes can communicate with each other by using verbal communication. See the use

of the keywords *needs* and *offers*, the *TExercise* class needs to interact with a class offering the *Play* verb, as long as the *TPlayer* class offers it, the link between both classes is made automatically. In the *offers* part after the verb and in parentheses, it must appear a method name, this is the method that will be executed when the verb is required.

See also that this method has no parameters, to access the information contained in the calling object at execution time the metadata are used. This facilitates the communication with different objects of different classes, no matter their types or interfaces; of course at execution time there can be problems if an object does not have a field annotated with the correct metadata, but the programmer can prevent multiple possibilities and in the worst case to do nothing appear to be the better option. Finally, see that the *ExecuteVerb* method present in every object is the trigger to make the link with an object that offers a given verb. This method can be called in response to a user generated event or when the application reaches a given state.

It is important noting that any class can be included dynamically in the object model in response to a user generated event, the download of a new page, etc., anytime a new class is added to the object model the browser automatically looks for matching between classes offering verbs and classes needing them.

Due to verbal communication a given object can interact easily with more than one object during the lifetime of the application, without noting it. This can facilitate the adaptation of an educational application to the skills and knowledge of a given student, because a given object without any change can interact with different objects which have different information or behavior depending on the answer to a question or the selection of the correct option in a given exercise or situation exposed to the student by an educational application.

Verbal communication between objects reduces development time and complexity. It is known that the development of software components can considerably reduce the development time of any application and its complexity, now with verbal communication the objects can define how the communicate with each other, without the need of the developer knowing exactly how they interact, the parameters that are necessary to pass and what to do with the resulting data. Course, verbal communication has to evolve, but it offers many facilities which make it valuable in the development of educational applications.

Suppose for example that a developer with not too much experience has a wide set of classes, developed by experienced programmers, these classes can communicate with each other by using verbs; his work is reduced a great deal because verbal communication reduces the time to tie the model and the amount of errors that can appear supposing that the original set of classes does not have errors. As a side effect, objects that communicate with other objects by verbal communication are simpler and can be more easily reused than objects that do not.

5. XML Object Models

The YADBrowser allows extracting object models represented in files using the XML language. XML has many properties which make it suitable to represent object models: it is a well known language, simple even for people with less knowledge about computing technologies; it has an intrinsic hierarchical representation, etc. So XML is a natural language to represent object models which can take advantage of XML hierarchical representation to reduce the amount of declarations needed to express the interrelations and properties of a given object model.

To extract an object model from an XML file the YADBrowser follows the following conventions:

- Every XML node will be converted into an object of the language, every attribute of the node will be a field of the object with the value and type of the attribute, this way is almost unnecessary to implement explicitly a constructor for a given object.
- If in the XML file a node named “node1” belong to another node named “node2” the object corresponding to “node2” will be the owner of the object corresponding to “node1”.
- For all the nodes with the same name it is created a class which represents them all. Every class declares a list of objects in case it is needed to contain any object, especially objects corresponding with XML nodes. All lists have the same name, and methods to make operations with the list like adding, deleting, inserting, etc., are included in every class automatically, reducing the amount of code that should be included in the XML file to define the behavior of the model classes.
- Every node should include a property named “id”, which will be the name of the resulting object. If any XML node includes a property which value is the “id” of any object this will be interpreted as an existing relation between both nodes which will be represented in the resulting object model. So the topological structure of the model is constructed automatically.

XML files representing object models can be downloaded at any time according with application needs. To download an object model from a XML file dynamically can be done in OPScript with a piece of code as the following:

```
var
  Object Graph;
begin
  ...
  Graph := XMLModel.LoadModelFrom('XMLModels\BeginnerGraph.xml');
  ...
end;
```

In the sample code above “XMLModel” is an object of the OPScript language which has the method “LoadModelFrom”, it receives the path of a XML file and returns an object that corresponds with the root node of the XML file. To represent object models in XML files has many advantages:

- The dynamical addition of object models to the current application model. Initially the main components of a page can be downloaded and later a model which can vary according with applications needs, student skills or both, can be downloaded on demand.
- It is necessary to code less because of the advantages of the XML language mentioned above.
- It is a simpler way of defining an object model so persons with less knowledge of programming languages should find easier to define an object model using XML than using an imperative programming language.

  Let us see how a model defined in a XML file looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<graph id='Root' CurrentNode='Start'>
  <methods>
    <!--
      procedure ProcessEvent();
      var
```

```
        integer i, event;
        Node Current;
        Link tmpLink;
        integer terminate;
      begin
        terminate := 0;
        event := GetIntegerFieldVal('eventdata');
        Current := this.CurrentNode;
        i := 0;
        while (i < Current.fItemsList.Count()) and (terminate = 0) do
          begin
            tmpLink := Current.fItemsList.Objects(i);
            if tmpLink.Event = event
              then
                begin
                  terminate := 1;
                  this.CurrentNode := tmpLink.Node;
                end
              else i := i + 1;
          end;
         if i = Current.fItemsList.Count() then Messages.Alert('Incorrect
Action');
      end;
      procedure Reset();
      begin
        this.CurrentNode := this.fItemsList.Objects(0);
      end;
    -->
  </methods>
  <verb name='ChangeEvent' offers='true' method='ProcessEvent' />
  <verb name='VReset' offers='true' method='Reset' />
  <node id='Start' name='StartNode'>
    <link id='LStartOn' node='On' event='1' />
  </node>
  <node id='On'>
    <link id='LOnPulsoElectrodo' node='PulsoElectrodo' event='10' />
    <link id='LOnOff' node='Off' event='5' />
  </node>
  <node id='PulsoElectrodo'>
    <link id='LPulsoElectrodoElectrodo' node='Electrodo' event='11' />
    <link id='LPulsoElectrodoOff' node='Off' event='5' />
  </node>
  <node id='Electrodo'>
    <link id='LElectrodoSincronismo' node='Sincronismo' event='12' />
    <link id='LElectrodoOff' node='Off' event='5' />
  </node>
  <node id='Sincronismo'>
    <link id='LSincronismoEnergia' node='Energia' event='3' />
    <link id='LSincronismoOff' node='Off' event='5' />
  </node>
  <node id='Energia'>
    <link id='LEnergiaEnergia' node='Energia' event='3' />
    <link id='LEnergiaPaleta1' node='Paleta1' event='6' />
    <link id='LEnergiaOff' node='Off' event='5' />
  </node>
  <node id='Paleta1'>
    <link id='LPaleta1Paleta2' node='Paleta2' event='7' />
    <link id='LPaleta1Off' node='Off' event='5' />
  </node>
  <node id='Paleta2'>
    <link id='LPaleta2Carga' node='Carga' event='8' />
    <link id='LPaleta2Off' node='Off' event='5' />
  </node>
  <node id='Carga'>
    <link id='LCargaDescarga' node='Descarga' event='13' />
```

```
    <link id='LCargaOff' node='Off' event='5' />
  </node>
  <node id='Descarga'>
    <link id='LDescargaSincronizar' node='Sincronismo' event='12' />
    <link id='LDescargaOff' node='Off' event='5' />
  </node>
  <node id='Off' />
</graph>
```

The fragment above shows a model created for a real application. It defines a graph, its nodes and the relations between them. The graph represents the steps a doctor must follows to employ a medical equipment used in case of a heart attack or other heart pathology. The graph is added to the application dynamically and is used to know if the user is following the correct steps for the equipment to function properly. The application, under development at present, is a tutorial to teach how to use the medical equipment. In the sample above *fItemsList* is the list automatically added by the browser, see also the methods it has, *fParent* is a field present in every object to access its owner and *Current* is a field that points to a given child, all of them are conventions that follow the YADBrowser when it loads a model from a XML file done to reduce the amount of code needed to define the model. See also in the sample above how it is possible to define a method or a set of methods that belong to a XML node.

6. Educational Importance of the New Features added to the OPScript Language

The new features of the OPScript language exposed in this paper can help authors building adaptive-hypermedia-based systems, but before seeing how this is possible it will be explained how all the features commented in this paper interact between them. The sample code above is a XML file that represents an object model, so it is an XML object model. In this file it is possible to see two verb tags. They are verbs offered by the *Graph* class defined in the XML file. These verbs offer theirs behaviour through two methods: *ProcessEvent* and *Reset* the first one looks for a valid node in the graph after the generation of an event by the user, the second restarts *Start* as the current node, this can be necessary to start an exercise again, etc., these methods are reusable methods and are lend by the *Graph* class to a class, not represented here, which will interact with the *Graph* class. To do so the counter part does not have to know what methods are offered by the G*raph* class, in fact it not even knows that it is interacting with a class named *Graph*, at a given moment it could continue interacting with another class which represents the valid answers to an exercise in any way, in fact in the real application it is possible to interact with different *Graph* classes according to the level of the student and to a previously selected pathology. Considering that the XML files can be downloaded on demand it is possible using these features to obtain a great degree of adaptation and in a dynamic way.

The idea behind this is that using the features exposed in this paper it is possible to, in a Web application, send to the student a page with selected static information, static in the sense that it is since the moment it arrive, present in the browser, and later according to her preferences or to some other characteristics it is possible to download an object model, that will be added to the already downloaded model, and will interact with it transparently. The so downloaded object model can be obtained using the user model represented in the server side of the application. In the sample code shown above that XML file corresponds to a beginner student, so the object model downloaded dynamically is related with the user model of the given student. The rest of the client side application, the one that has been

downloaded previously is able to interact with this XML model or with a model created to an expert student; its programming, definition, etc. can be the same. So the difference is in the XML model and what makes possible the interactions of the same objects with different models no matter the level or preferences of a given student are: “verbal” communication between objects and reusable methods.

The advantages for educational applications are the adaptation that can be obtained, the dynamism, because a XML model can be downloaded at any time without changing the current page, without a noticeable delay for the student and because the part that depend most on the student can be added to the application at any moment, it can be generated dynamically no matter what has been already send to the student through Internet because the communication using verbs is so flexible that it can adapt itself to almost any scenario.

7. Conclusions

In this paper authors have exposed the new features added to the OPScript language, these features have been added to achieve more adaptation using techniques in the client side of a Web application, this project could benefit a lot if it could be joined to models that could achieve the adaptation from the server side of a Web application, created to employ OPScript as the client side script language, like the LAOS model [24].

Authors have shown that applications could be more easily adapted to students’ skills and knowledge using the features exposed, because by using verbal communication a given object can interact with different objects in different moments during the application lifetime without special coding. By downloading whole object models represented in XML files the application can be adapted easily to the necessities, responses, etc., of the student. Code added automatically to those object models reduce the effort needed to develop the final application. Finally the features mentioned work based upon the reusable method and metadata features, which make method sharing and reusing between classes possible.

In this paper it have been included some portions of code of a developing project. This project will show to doctors and paramedics how to operate a medical equipment, this project is been created using all the features of the OPScript language mentioned in this paper.

References

[1] Montessoro, P., Pierattoni, D., Cicuttini, R.: MTeach: A Simple Production Framework for Context-Based Educational Hypermedia. Journal of Educational Multimedia and Hypermedia 12(4) (2003) 335-359.

[2] Manouselis, N., Sampson, D.: Dynamic knowledge route selection for personalised learning environments using multiple criteria. In Proceedings of the IASTED International Conference on Applied Informatics, (2002) 351–605.

[3] Atif, Y., Benlamri, R., Berri, J.: Learning Objects Based Framework for Self-Adaptive Learning. Education and Information Technologies 8(4), (2003) 345–368.

[4] Huang, R., Ma, J.: A Java Technology Based Shared Browser for Tele-Lecturing in University 21. [Online]. Available: http://csdl.computer.org/comp/proceedings/iccima/2001/1312/00/13120298.pdf (2001).

[5] Dennis, B., M., Harrison, M., A.: Grendel: A Web Browser with End User Extensibility. [Online]. Available: http://csdl.computer.org/comp/proceedings/compcon/1997/7804/00/78040074.pdf (1997).

[6] Hoyos-Rivera, G., J., Lima-Gomes R., Courtiat, J., P., Benabbou R.: The Web as a Tool for Collaborative *e*-Learning: the Case of CoLab. [Online]. Available: http://csdl.computer.org/comp/proceedings/icalt/2003/1967/00/19670312.pdf (2003).

[7] Henricksen, K., Indulska, J.: Adapting the Web Interface: An Adaptive Web Browser. [Online]. Available: http://www.dstc.edu.au/m3/papers/AUIC2001.pdf (2001).

[8] Theoktisto, V., Bianchini, A., Ruckhaus, E., Lima, L.: AVANTE: A Web Based Instruction Architecture based on XML/XSL Standards, Free Software and Distributed CORBA Components. UPGRADE 4(5), (2003) 29-38.
[9] Montessoro, P.L., Pierattoni, D., Toppano, E.: MTEACH: A simple framework for didactic and context-based hypermedia. Proceedings of SSGRR 2002W, International Conference on Advances in Infrastructure for Electronic Business, Science and Education on the Internet (2002).
[10] Boyle, T., Cook, J.: Towards a pedagogically sound basis for learning object portability and re-use. In: G. Kennedy, M. Keppell, C. McNaught, T. Petrovic (eds.): Meeting at the Crossroads. Proceedings of the 18th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education. The University of Melbourne. (2001) 101-109.
[11] Boyle, T.: Design principles for authoring dynamic, reusable learning objects. Australian Journal of Educational Technology, 19(1), (2003) 46-58.
[12] Romero, V. A., Mateo, L., Elorriaga, J. A.: The educative browser YADBrowser and its script language OPScript. In: Actas del II Congreso Internacional de Tecnologías y Contenido Multimedia. Tecnología y aplicaciones Web. La Habana (2004).
[13] Romero, V. A., Elorriaga, J. A., Mateo, L.: The Language for the Educational Browser YADBrowser. In: Actas del 6º Simposio Internacional de Informática Educativa. Cáceres (2004).
[14] Romero, V. A., Elorriaga, J. A., Mateo, L.: YADBrowser: A browser for Web based educational applications. To be published by Journal of Educational Multimedia and Hypermedia.
[15] Uehara, S., Mizuno, O., Kikuno, T.: An Implementation of Electronic Shopping Cart on the Web System using Component-Object Technology. [Online]. Available: http://www-kiku.ics.es.osaka-u.ac.jp/paper/data/pdf/10.pdf (2001).
[16] Xiao, L., Zhang, X., Xu, Z.: On Reliable and Scalable Peer-to-Peer Web Document Sharing. [Online]. Available: http://csdl.computer.org/comp/proceedings/ipdps/2002/1573/00/15730023b.pdf (2002).
[17] Xiao, L., Zhang, X., Andrzejak, A., Chen, S.: Building a Large and Efficient Hybrid Peer-to-Peer Internet Caching System. [Online]. Available: http://csdl.computer.org/comp/trans/tk/2004/06/k0754.pdf (2004).
[18] Hennen, D., S., Ramachandran, S., Mamrak, S., A.: The Object-JavaScript Language. [Online]. Available: http://acuity.cis.ohio-state.edu/Nois/Tguide/ojs.ps. (2000).
[19] W3C HTML 4.01 Specification (n. d.). [Onbline]. Available: http://www.w3.org/TR/html4/.
[20] Wang, P., S., Kajler, N., Zhou, Y., Zou, X.: WME: Towards a Web for Mathematics Education. [Online]. Available: http://ox.cs.kent.edu/~pwang/47wang.pdf (2003).
[21] Netscape Corporation. JavaScript Central (n. d.). [Online] Available: http://devedge.netscape.com/central/javascript/.
[22] Shonle, M., Lieberherr, K., Shah, A.: XAspects: An Extensible System for Domain-Specific Aspect Languages. [Online]. Available: http://www.cs.ucsd.edu/users/mshonle/p28-shonle.pdf (2003).
[23] Lieberherr, K. J.: Connections between Demeter/Adaptive Programming and Aspect-Oriented Programming (AOP). [Online]. Available: http://www.ccs.neu.edu/home/lieber/connection-to-aop.html (2004).
[24] Cristea A. I.: Automatic Authoring in the LAOS AHS Authoring Model. [Online] Available: http://wwwis.win.tue.nl/ah2003/proceedings/ht-2/

Acknowledgements

This work is partially funded by the University of the Basque Country (UPV00141.226-T-15948/2004), the Spanish CICYT (TIC2002-03141) and the Gipuzkoa Council in a European Union program.