LAG 2.0: Refining a reusable Adaptation Language and Improving on its Authoring

Alexandra I. Cristea*, David Smits**, Jon Bevan* and Maurice Hendrix*

*Department of Computer Science, University of Warwick,
Coventry, CV4 7AL, United Kingdom
{A.I.Cristea, J.D.Bevan, M.Hendrix}@warwick.ac.uk
**Faculty of Mathematics and Computer Science, Eindhoven University of Technology
PB 513, 5600MB, Eindhoven, The Netherlands
davidsmits@live.nl

Abstract. Reusable adaptation specifications for adaptive behaviour has come to the forefront of adaptive research recently, with EU projects such as GRAPPLE[1], and PhD research efforts on designing an adaptation language for learning style specification [1]. However, this was not the case five years ago, when an adaptation language for adaptive hypermedia (LAG) was first proposed. This paper describes the general lessons learnt during the last five years in designing, implementing and using an adaptation language, as well as the changes that the language has undergone in order to better fulfil its goal of combining a *high level of semantics* with *simplicity*, *portability* as well as being *flexible*. Besides discussing these changes based on some sample strategies, this paper also presents a novel authoring environment for the programming-savvy adaptation author, that applies feedback accumulated during various evaluation sessions with the previous set of tools, and its first evaluation with programming experts.

Keywords: Adaptive Hypermedia, Adaptation Language, LAG, LAOS.

1 Introduction

Adaptation and personalization are considered to be both useful and desirable, and came to the fore with user modelling [2] and adaptive hypermedia [3] research. However, adaptive environments are notoriously difficult to author [4] for. Amongst all the components in adaptive environments, about which much has been modelled and written [5][6][7][8][9][10][1], the most difficult part is the specification (authoring) of the adaptive behaviour [8][12] [13][14][1].
Hence, reusability is desirable especially for the *adaptive behaviour specification*, in the sense of '*write once, use many*' [12]. Since 2003-2004 a few adaptation languages have been proposed; LAG [15] is, as far as we know, the first such language, followed by LAG-XLS [16] that caters for Learning Styles. Ideally, a single

[1] http://www.grapple-project.org/

common accepted standard would be best, similarly to content descriptions in the educational domain (e.g., LOM[2], SCORM[3]). In the GRAPPLE project, such an endeavour is being targeted. However, this is beyond the scope of the current paper.

Making a language a common standard and reusable is only the first step, the next is to allow different *levels of access* to the creation process. This targets the different types of authors that will be able to use the language (e.g., computer savvy or not). One such version of different access levels is given by the *LAG framework* [17]*:* *adaptation strategy* – accessible for all authors, via laymen-level descriptions, *adaptation language* –accessible mainly to computer savvy authors, *adaptation assembly language* – only accessible to 'hard core' computer savvy authors).

Another dimension is brought about by using *visualization* (e.g. the Graph Author developed for AHA! [18] is using visualisation in order to support authors) and *handling* support. In previous versions of the LAG language implementation, handling support was envisioned as not allowing an author to insert any wrong constructs [13][17]. In the GRAPPLE project, additionally to this restriction, the ultimate language to be used by the non programming-savvy author will be purely graphical [20]. Whilst this will hide most of the difficulty for the high-level author, it will also need to reduce the flexibility to some degree.

When major changes are needed, or when system-system interaction is required, underlying programming languages will support this. Currently, we consider supporting multiple adaptation language output as a desirable feature, besides developing new languages targeted at specific levels of access (transformed into wrapping levels).

Thus, this paper discusses general lessons learnt during the last five years in proposing, designing, implementing and using an adaptation language; the changes that the language has undergone in order to better fulfil its role as an adaptation language. Finally, this paper discusses the LAG language [8] as it currently stands, in view of the new extensions that have been performed which aim for it to better fulfil its goal of combining a *high level of semantics for authors*, with *simplicity* and *portability* as well as *flexibility*. Besides discussing these changes based on some sample strategies, this paper also presents an XML equivalent of LAG, which is to be used instead of the current language for portability between systems, as well as a novel authoring environment for the programmer or programming-savvy adaptation author, that applies feedback accumulated during various evaluation sessions with the previous set of tools. The outcomes of the tool, the adaptation strategies, can be used by any author [17]. This environment's first evaluation is also presented.

The remainder of this paper is organized as follows. Section 2 introduces the new elements in the LAG adaptation language via scenarios for adaptation. It also discusses alternative representations for the LAG language. Section 3 introduces the PEAL environment for authoring, by comparing it with the previous LAG language authoring environment, as well as with alternative solutions. The section concludes with a discussion of this environment and its first evaluation. Section 4 presents related research. In section 5, we draw general conclusions and pointers for further research.

---

[2] ltsc.ieee.org/wg12/

[3] www.adlnet.gov/scorm/

2 The updated LAG grammar, via Scenarios for Adaptation

2.1 The LAG Grammar history and Lessons Learnt

The *LAG language* concept was first introduced in [17], together with the *LAG framework* (hence, the similarity in name between language and framework, although they are distinct entities). As sketched in section 1, the LAG framework distinguishes between *adaptation strategy* – accessible for all authors, via laymen-level descriptions; *adaptation language* – accessible mainly to computer savvy authors; an example of such a language is the LAG language, although *any* adaptation language fits at this level; *adaptation assembly language* – only accessible to 'hard core' computer savvy authors.

From the moment it has been proposed, the LAG language was supposed to fill in the 'missing link': it had to be and adaptation language, thus at a higher level than what the LAG framework called 'adaptation assembly language': it had to be re-usable, whereas adaptation assembly languages at the time were not. To give an example, it was possible then to write:

```
(a) IF Concept ('The Night Watch') has been visited
    THEN show Concept ('Rembrandt')
```

However, it was not possible to write:

```
(b) IF Concept (title) has been visited
    THEN show Concept (author)
```

Thus, even such simple generalizations were not easily available to authors, who would have to manually connect all concepts, instead of writing reusable rules.

Brusilovsky's taxonomy [3], used for defining the types of adaptation possible, also refers to such an assembly language level[4]. Take for instance the decision of *showing* a concept by stretchtext, versus showing it by regular text; or *hiding* it by removing, or by graying out. These are decisions which may be dependent on the capabilities of the adaptation and rendering engine. A given engine may allow for showing concepts or not, but not for applying strechtext (e.g., the AHA! engine [18]). Using such low-level requirements might make an adaptation strategy impossible to be used by different engines. Moreover, such a low-level requirement may have little to do with the pedagogy involved in teaching a course, for instance. A teacher author might decide that a certain piece of information is necessary for a student or not, but may leave the rest to the engine.

Thus, another condition for a language to be an adaptation language was that it had to be able to be converted to lower level assembly language, as per Brusilovsky's taxonomy, but that this exact conversion is to be left to the interpretation and specifics of the given adaptation engine (hence, the similarity with a programming language which is compiled into assembly language in order to run on a certain system). For the example above, any structure (b) as above, applied to a certain domain, could become something similar to (a). However, an adaptation language may not necessarily have IF-THEN constructs, as they themselves are relatively low level.

[4] Although the taxonomy can be used for writing reusable rules, it still only specifies low level actions performed on (usually specific) concepts.

Still, for compatibility with the engines of the time, the initial LAG language allowed for IF-THEN constructs, corresponding to *assembly language constructs*. Supplementary, however, it also defined higher level constructs, specific to the adaptation functionality, which are part of the *adaptation language* level within the LAG framework. Beside this important distinction, and essentially offering an instantiation of adaptation language ideas, it also defined what such a language should have: it should make use of the application domain (adaptive hypermedia) by

1) allowing it to be *simpler and with fewer constructs* than a regular programming language or a logic-based language (thus lowering the authoring threshold); Thus, elements were included in LAG only when considered necessary;and by
2) using constructs *specific for the adaptive hypermedia domain*, or *assumptions that can be safely made in that domain*. For instance, at the time it was safe to assume that most adaptive hypermedia applications have an underlying tree-like structure (as they were mainly designed for the educational domain, and, to some extent, inherited the organization into chapters-subchapters from educational books). This meant that, although, hypermedia, in theory, are graphs with any connections desired, in practice they were (and many of them still are) trees with given hierarchies. Hence, the GENERALIZE and SPECIALIZE constructs were born, the first to visit concepts higher in the hierarchy, thus of higher generality, and the latter, to visit concepts lower in the hierarchy, thus more specialized.

It was then not a working language, just a proposal, which in the following year has been developed [13] towards introducing, first of all, a tool for supporting this grammar. There, we also introduced the concept of *adaptation procedures*, i.e. code snippets that can be reused by other authors, similarly to how procedures or function calls are used in other programming languages – with the significant, simplifying distinction however that no parameter exchange would take place, and that the extra code would be pasted in its entirely in the place of the 'adaptation procedure' call. This made these procedures simpler than regular programming languages, as per requirement 1 above. Moreover, these snippets could be used not only by their initial designer, but also by other authors, effectively creating a tool for customized language extension. This allowed for a higher level of reuse of adaptation languages, whilst keeping the processing simpler than in regular programming.

The combination of requirements 1 & 2 above generated the following list of minimal constructs that should be present in high-level adaptation language:

(a) constructs allowing domain structure and composition related adaptation:

As said above, the domain structure and composition can be used to determine the adaptation process.

i. hierarchy-based adaptation: if a hierarchy is present, and concepts are grouped as concepts-subconcepts (such as concept 'The Night Watch' is a subconcept of 'Most Famous Paintings'), this hierarchy can be used to determine the order of appearance (such as the concept 'Most Famous Paintings' and its information should be shown before the concept 'Night Watch').
ii. other relations –based adaptation: the most commonly used relation in domain models in adaptive hypermedia is that of concept-subconcept, as above. However, other relations might be possible, especially in systems

importing RDF[5] structures, for instance. Adaptation languages should be able to use these relations in the adaptation process.

iii. domain-concept type-based adaptation: frequently, domain concepts have types (or other attributes). These also can be used in the adaptation process, thus should be accessible via the adaptation language.

(b) constructs allowing goal-related adaptation:

Adaptive hypermedia goals can be related to the pedagogy involved, if an educational application is envisioned, or to a business goal, in an e-commerce application, for instance. These goals can determine how domain concepts can be used. A simple way of adding such information is via labels and weights overlayed over the domain concepts they refer to. For example, the concept 'The Night Watch' can be labelled 'visual' to be used in a strategy involving visual versus verbal presentation, could be labelled 'advanced' if it is to be used in a drawing and painting class, or 'beginner' if it is to be used in a class on famous paintings and painters. Thus, whilst this information is added to concepts in the domain model, it is independent to the domain. This type of independence between domain and goal (or pedagogic) model was proposed as a basis for adaptation language construction [8] and has found recognition as one of the design concepts in the GRAPPLE project.

i. label –based adaptation: see above.

ii. weight–based adaptation: an alternative to label-based adaptation, numeric values can be used to label concepts with respect to the goal. This alternative is not used very frequently currently.

(c) structure of the adaptation program:

i. Constructs defining the 'adaptation loop':

Unlike regular procedural programs, the concept-driven adaptation in adaptive hypermedia happens in a loop. Users can visit the same concept several times. It may be that similar, or evolving behaviour is needed at successive visits. Thus, similar to the collection of rules in expert systems, the programming constructs in adaptation languages can be triggered repeatedly, and in different orders. An adaptation language should allow for an 'adaptation loop', that defines the continuous interaction between user and system, and for mechanisms to ensure that the correct constructs are triggered at the correct time.

ii. Constructs allowing for an 'entry point':

As adaptive hypermedia content is often based on the Web, it suffers from the same draw-back as regular Web hypermedia: first time users may visit the site. Thus, an adaptation language needs to be able to define what these users will be seeing. This is different from the 'adaptation loop' above, where users already have some history of recorded behaviour in the system[6]. The most important difference between the 'entry point' and the 'adaptation loop' is that the 'entry point' is a one-off event. Constructs will be executed here only once.

High level language thus means here a language created from an authoring perspective: an author is concerned about how the content, as well as the goal

[5] http://www.w3.org/RDF/

[6] It is possible for this history of recorded behavior to be imported from a different application. In this case, direct entry into the 'adaptation loop' should be enabled.

description for the particular application, can be used to model adaptation. The actual particulars of how the adaptation engine searches, retrieves, and renders each of these actions is of lesser relevance to the author, and could potentially add to the authoring complexity[7]. As will be shown in the following, the LAG language allows for all these constructs envisioned to be present in an adaptation language.

A good update on the fundamental elements of the current basic LAG language is provided in [8]. There, handling of *overlay variables*, as well as *independent variables* is shown, for the different static representation layers supported: *domain layer*, *goal and constraints layer* (for representation of the goal of the application, such as pedagogical goal for educational presentations, and business goal for commercial applications), *presentation layer*, and *user layer*. These, together with the *adaptation layer*, that hosts the adaptation language, correspond to the layers as defined by the LAOS authoring framework for adaptive hypermedia [8]. Also there, the use of *generic variables* (for *any* domain or other static map) versus *specific variables* (for a *given* domain map or other static map) is described.

Further extensions comprised authoring extensions for *collaboration* [13], for *meta-level reuse* [21], in the sense of being able to describe meta-strategies triggering strategies [16], thus allowing reuse of strategies in an automatic manner.

In the remainder of this section, we illustrate with the help of scenarios[8] other recent developments of the basic language, grouped around the different types of adaptation which the language allows.

2.2 Hierarchy-based constructs for adaptation

As previously discussed, domain models in adaptive hypermedia often have a hierarchical structure. The generalize-specialize constructs initially proposed in LAG have been replaced with simpler ones, that can be used as attributes of the concept, such as *parent*, *child* and *level, order* (as inspired by XPath[9], in the spirit of using constructs of accepted standards where possible) The strategy shown below is a depth-first strategy, which shows the concept labelled as 'start' first, then the rest of the content in a depth first manner using the child-parent relations. The exact meaning of the constructs is given as comments in the strategy below:

```
initialization( // 'entry point': this defines what the user first
                // sees;
  PM.next = true  // allow for a 'next' button in the presentation;
   // please note that no information is given as to how to render
   // this 'next' button; this is up to the engine
  PM.ToDo = false  // don't allow for a 'To Do' list in
                   // the presentation
  PM.menu = false // don't allow for a 'Menu' in the presentation
  while true( // show the first, father concept, labelled 'start':
       if GM.Concept.label == start then (
         PM.GM.Concept.show = true )  ) )
implementation ( // 'adaptation loop': this defines the continuous
// adaptive interaction between user and system
```

[7] This statement is based on previous evaluation experiments and interviews.
[8] Available for tryout at: http://prolearn.dcs.warwick.ac.uk/strategies.html
[9] www.w3.org/TR/xpath

```
//if you visited the parent you should be able to
  if UM.GM.Concept.parent.access then ( // visit the child
        GM.Concept.show = true ))
```

Similarly, for breadth-first strategy, the level of a certain concept can be used to show all concepts of higher or equal level (the rest of the strategy is removed due to lack of space):

```
// if the current concept level is lower than the current user level,
// show the current concept (so only show users concepts up to their
// current level)
if GM.Concept.level <= UM.GM.level then (PM.GM.Concept.show = true)
```

2.3 Relation based constructs of adaptation

Domain models have been presumed to be hierarchical. However, more complex domains can have many types of relationships, next to, or instead of the hierarchical ones. To illustrate the use of related concepts, the LAG language supports a generic 'relatedness' relation between two concepts[10]. Here, only an excerpt of the strategy is shown. The exact meaning of the constructs is given as comments below:

```
// for advanced learners show the related concepts
if enough[11](GM.Concept.access // if a concept is accessed
    UM.GM.stereotype == adv , 2) // and the concept is labelled as
                                 // 'advanced'
then PM.DM.Concept.Relatedness.Concept.show = true // than show the
                           // concept related to the current concept
```

2.4 Type based constructs for adaptation

Domain concepts are defined in LAG as having attributes, which can have predefined types, or author-created types. Examples of predefined types are: introduction, explanation, conclusion, keywords, text, image, video. These types are intrinsic to the domain, and thus are not changed via the goal model. However, they can be used to guide the adaptation. This implementation only shows 'introduction' concepts and doesn't show the other concepts of type 'conclusion', till the introductions are read.

```
// DESCRIPTION: show first introductions, than conclusions;
initialization(
 while PM.GM.Concept.type != conclusion // make only introductions
  (       PM.GM.Concept.show = true )  // readable
 UM.GM.showall = 0)
implementation ( // if a concept is accessed and it is an introduction:
if enough (PM.GM.Concept.type == introduction
         UM.GM.Concept.access == true, 2)
// then increase the showall counter:
```

[10] The LAOS framework [8~~12~~] allows for multi-multi relations of any type; however, the current language only supports the relatedness relations, for compatibility with the MOT authoring environment [8].

[11] The 'enough' construct is based on games and provides a way to add several conditions. In some games, to pass a level, some conditions should be fulfilled, e.g., collection of objects. The exact objects may not be specified, only that there should be 'enough' of them to move on. Similarly, 'enough' in LAG means that enough conditions should be satisfied. The number after the conditions specifies how many – here, 2.

```
then ( UM.GM.showall += 1 )
// if the showall counter is greater than a threshold – here, 3,
// because we had three questions –
// and the type of the current concept is 'conclusion':
 if enough (UM.GM.showall > 3 PM.GM.Concept.type == conclusion, 2)
// then show the current concept:
 then ( PM.GM.Concept.show = true ) )
```

2.5 Weight and Label based constructs for adaptation

This strategy shows concepts based on their weights and labels. The idea in the LAOS framework [8] is that weights and labels are added in a separate layer from the domain map, in the goal and constraints map (GM), where they, for educational applications, represent pedagogic knowledge. Thus, a typical pedagogical division is to label concepts as beginner-intermediate-advanced. Labelling these concepts outside the domain model means that a different labelling is possible for the same domain model concepts, but a different instance of the goal map. Thus, some matrix multiplication concepts can be labelled as beginner concepts for maths students, but as advanced concepts for music students, for instance. The same strategy can be applied to both, as below.

```
implementation ( // count visits for each type of label:
    if UM.GM.Concept.access == true then (
    if (UM.GM.Concept.beenthere == 0) then (
      if (GM.Concept.label == beg) then ( UM.GM.begnum -= 1)
      if (GM.Concept.label == int) then ( UM.GM.intnum -= 1)
      if (GM.Concept.label == adv) then ( UM.GM.advnum -= 1))
    UM.GM.Concept.beenthere += 1
  ) // ------------------------------------------------------
  // Change stereotype beg -> int -> adv when appropriate:
  // ------------------------------------------------------
  if enough(UM.GM.begnum < 1 // 2 below means all conditions
            UM.GM.knowlvl == beg // should be satisfied
            ,2) then ( UM.GM.knowlvl = int )
  if enough(UM.GM.intnum < 1
            UM.GM.knowlvl == int,2) then (UM.GM.knowlvl = adv
  ) // show the concepts with the appropriate knowledge level:
  if (GM.Concept.label == UM.GM.knowlvl) then (
    PM.GM.Concept.show = true))
```

Thus, weights and labels for the goal model should be the default way for a strategy to express the adaptation. However, we have found out that some properties that are domain dependent can also be exploited. In the next section, we show how the types of domain concepts can be used for adaptation.

2.6 LAG language versus XML languages

XML languages can be preferable especially for system-to-system conversions. Whilst we don't believe they can completely replace programming-based approaches, as they are very verbose if used directly to program by hand, they are definitely useful for interfacing. Thus, export-import into such formats is desirable. Converting the LAG language to an XML specification is relatively straightforward: for each LAG

construct, an XML element can be defined, with sub-elements that enforce the prescribed grammar. Thus, a LAG adaptation strategy will contain:

```
<description> the layman description of the strategy </description>
<initialization> a user's first view of the system </initialization>
<implementation> the interaction loop user-system </implementation>
```

A more complex condition, such as the one using 'enough' in the type-based adaptation, can look as below. Obviously, the XML format is more verbose and shouldn't be used in direct programming by hand.

```
<if>
   <enough number="2">
      <condition>
         <attribute> UM.GM.showall </attribute>
         <operator> &gt </operator> <value> 3 </value>
      </condition>
      <condition>
         <attribute> PM.GM.Concept.type </attribute>
         <operator> == </operator> <value> conclusion </value>
      </condition>
   </enough>
   <then>
         <attribute> PM.GM.Concept.show </attribute>
         <operator> = </operator> <value> true </value>
   </then>
</if>
```

3 PEAL Environment for Authoring

3.1 The Problem

A previous online editor had been created[12] [15][17] (Fig. 1). It allows insertion of only predefined authoring constructs, thus providing *handling help*, as mentioned in Section 1 (in Fig.1, left side, the [add statement] link event only allows predefined constructs, such as the generalize construct below; the generalize construct in turn only allows condition insertion, via the [add condition] input; clicking this allows the [attribute][operator][value] construct to appear, which can only be populated by *generic concepts*, such as the 'UM.Concept.type==expert' on the left side of Fig.1, or by *specific concepts* – not shown here). Moreover, for the specific concepts, the environment allows direct database access to a domain model and goal and constraints model database, permitting selection of appropriate specific concepts directly from the respective instances[13]. Thus, it provides ample support for the authors, lowering the authoring threshold. Also, in terms of flexibility, it allows definition of both *adaptation strategies* and *adaptation procedures* (section 2.1). However, this environment is not currently up-to-date with the current LAG grammar. Most importantly, the environment does not allow for the separation of the interaction into initialization (what the user sees at the first interaction with the system, when nothing else is known about him) and the interaction part (called 'implementation' in

[12] http://elearning.dsp.pub.ro/motadapt/

[13] This idea has been revived in the GRAPPLE authoring tool, where adaptation tools can access the domain model authoring tools via a common shell [2016].

the LAG language; this describes the loop of interaction between the user and the system, and, like a rule list, is triggered as long as the conditions of the rules hold). The environment also does not directly support types of domain concepts, as in section 2.4, relatedness relations, as in section 2.3, and other minor extensions of the LAG language. For instance, the update rule for Fig.1, right side should read: `PM.DM.Concept.exercise_expert.show = true`; thus marking the fact that the 'show' variable (determining if to show or not something to the user) is part of the presentation model (PM), and the 'exercise_expert' is an attribute in of the current concept in the domain model (DM).

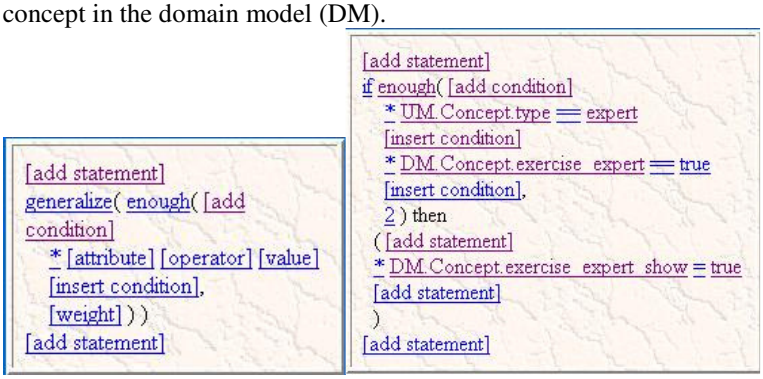

Fig 1. Initial Programming Environment for the LAG language.

Currently, due to the differences in grammar in the old on-line environment and the actual specification, we actually use simple text editors as adaptation language creation environments. However, this has several drawbacks, as for instance, the fact that errors are only spotted at compilation time; that the authors receive no help or support whatsoever whilst editing; that authors themselves need to keep track of the version of the language they use, and thus may be working with a version where some of the programming concepts are obsolete.

3.2 Objectives

Thus, the PEAL development set out with the following objectives[14]:

- Develop an online, AJAX based Programming Environment for the LAG language, based loosely upon the existing online editor, and developed from an existing open source project entitled CodeMirror[15].
- Save (to database or to files) and export (to file/database) Adaptive Hypermedia strategies written in LAG for either further editing or use in the AHA! delivery system [18] Database driven user access and strategy storage similar to the existing online system. Efficient storage system for the way in which AH Strategies are constructed by the user in LAG using this programming environment. Use the above storage mechanism to recognise basic LAG Grammar (e.g. statement construction, use of operations), and provide some warning system when strategies do not follow LAG Grammar.

[14] some typical to programming environments, some typical to for the LAG language
[15] http://marijn.haverbeke.nl/codemirror/

- Include basic templates for new adaptive hypermedia strategies when they are created. Recognition of complex LAG Grammar (e.g. nested statements, complex statements and user defined 'procedures'). Colourise reserved words, both those defined by the LAG Grammar and user-defined variables, including an effective storage system for these reserved words. Drag-and-drop standard code sections, user-defined code sections, list word-completion of standard words, list word-completion of user-defined words.
- Integration of basic demonstration content with strategy. Integration of user-defined content with strategy, including a strictly defined required format for user-defined content.

3.3 Programming Environment for Adaptation Language (PEAL)

The Programming Environment for Adaptation Language (PEAL) is shown in Fig. 1.

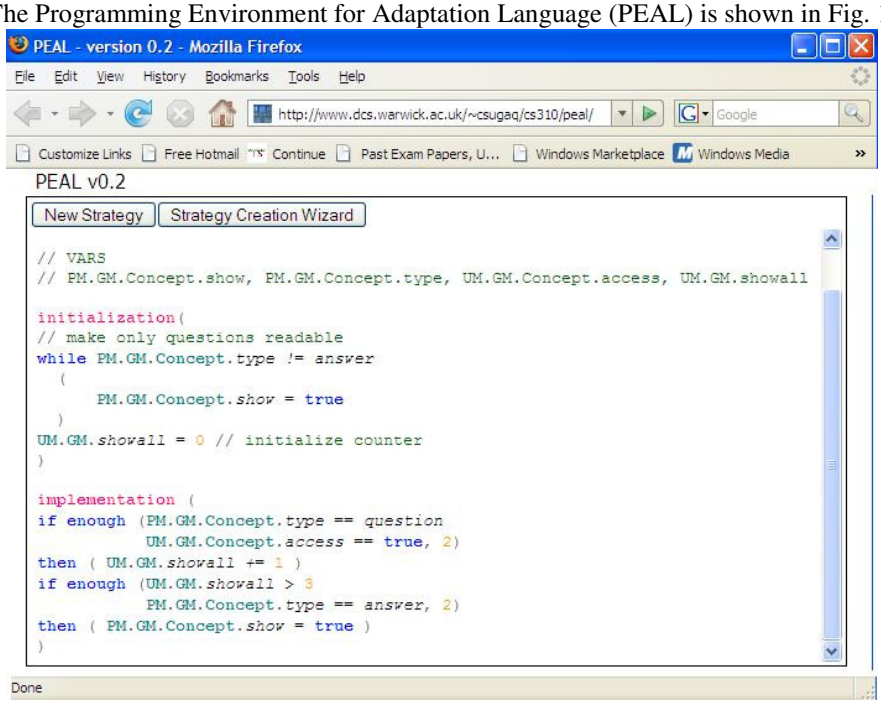

Fig 2. Programming Environment for Adaptation Language (PEAL).

The following components have been realized: User Access and File Storage; Basic Template for Strategies; Strategy Creation Wizard; Tokenizer/Parser Design; Reserved Word Storage; Configuration File; Production of an CAF file for each strategy. The major focus of this project was to create a parser for the LAG language which would recognise and colour correct syntax and grammar, and highlight incorrect syntax and grammar. This is a different take than previously of providing *handling help*, better known to the programmer communities: language constructs are identified and coloured respectively. In case the construct cannot be identified, the colour is bright red, signalling an error. The storage system with which the strategy

document structure is stored has been provided by the CodeMirror system in the form of a simplified DOM structure, so we have been working from that basis when designing grammar recognition. A detailed design of the user access and file storage system, including the database design to hold user details, a security system to ensure that passwords cannot be obtained and files cannot be accessed by any unauthorised users has been set up. We have also designed a method by which errors in the code can be easily highlighted. This may be further developed by supplying messages for the user by which they can more easily determine the problem. Currently the design for Reserved Words in the LAG language requires their storage in the parser and tokenizer. However, for greater extensibility, we will design a method using the Object-Orientation techniques provided by Javascript to store keywords externally to the parser and tokenizer in their own 'class'.Moreover, we have implemented: Predictive Word Completion; MOT2AHA Integration Support; Drag-and-Drop Coding. Predictive Word completion will be used in the wizard and also the editor.

3.4 PEAL Evaluation

PEAL was evaluated on a small scale, with the help of five programming-savvy persons, as they represent the type of author at which this tool is targeted at. Non-programmers are supposed to be using ready-made, reusable strategies, without getting into the details of programming. Nielsen [22] showed that 95% of the usability problems can be detected with just five users. Users were asked to identify the strengths and weaknesses of PEAL. Amongst strengths, the following were mentioned: "element suggestion and possible variables."; "syntax highlighting" "Coloured keywords."; "Simple screen."; "Use of other strategies."; "completion suggestions"; "code highlighting and indentation" . "save and reuse code fragments"; "highlighting and word completion"; "reusable code snippets". Amongst weaknesses, the following were mentioned: "Sometimes lines of code jump - automatically indented when the cursor moves." "No search function (or search and replace)". "If there are multiple errors only one is displayed in the status bar.". "Some display-size bugs exist with text entry box." The programmers were also asked to state which editor they would prefer to use to edit the LAG language, between regular text editors (used previously for editing the language) and PEAL. Without exception, they all voted for PEAL.

These initial evaluations highlight the usability of PEAL, the fact that such an environment is needed, as well as immediately pointed to some bugs that are to be fixed in the near future.

4 Related Work

The level of abstraction of semantics we envision in our work, that can lead to reuse of the adaptation strategies, is expressed in our paper via an adaptation language. An alternative to this is to express such sequences and interaction via *workflow languages*[16]. However, workflow languages have previously been shown to be

[16] http://www.yawlfoundation.org/

insufficient to express personalization at a level of expression as delivered by adaptive hypermedia [20]. Although not shown here, the LAG language can express various personalization strategies (based, e.g., on preferences, learning styles, goals) and is more powerful with the proposed extensions than other workflow languages.

A popular current growing competitor to adaptation languages and adaptive hypermedia expressivity is *IMS-LD*[17]. Research has shown however that IMS-LD is still not yet capable of delivering all adaptation functionality as defined by adaptation hypermedia [1], [10], and also has serious limitations when it comes to adaptively supporting collaborative learning [13].

The main question is how this contribution can be beneficial for other members of the community since LAG introduces various restrictions stemming from the assumptions on how several models need to be structured. Logic based languages or metamodelling seems to offer a bit more flexible option for doing this.

Other approaches, such as other adaptation languages exist (see, e.g., LAG-XLS [16]). The later language caters for learning styles, but it would need further extensions to cater for more extensive personalization as well as collaborative aspects.

Adaptation languages are based on the rule-based approach. Alternatively, reasoning mechanisms can be used to express the adaptive behaviour (e.g., description logic). Currently, logic-based languages are too broad, with a multitude of constructs that are not of use in current adaptive hypermedia, thus contradicting the first constraint as in section 2.1. The new developments in the Semantic Web offer new vehicles for reasoning such as RDF, OWL[18] (used also by adaptive learning systems, such as the Personal Reader [19]). They may provide viable alternatives for the future, but currently systems based on such mechanisms have serious performance problems when compared to other rule-based systems. Furthermore, whereas such approaches are very good in expressing modelling information, interrelations, etc., they lack direct and simple support for programmatic constructs required to express most of the behaviours we have discussed in this paper.

Meta-modeling, the process of designing languages through meta and meta-meta notations, such as DTD (document type definition) on top of XML, etc. are useful approaches for describing the domain content, and the different overlay structures (such as the goal structure). They are less useful however for defining adaptation.

Yet another direction of adaptation representation is the family of "assembly-level" adaptation languages, such as used in systems as AHA! [18], Interbook (Word-document-based) [25], WHURLE (LP: lesson plan) [25]. The problem with such languages is that, not only do they lack many of the required features as outlined in previous sections, but they are also extremely verbose and difficult for non-experts as well as experts alike to express high-level, reusable adaptation strategies in, rendering them a questionable choice for employing as the basis of adaptive specifications They may, nevertheless, serve as an appropriate "end-of-line representation", in essence serving as a possible "output format" for higher level languages such as LAG. The same is true for specifications such as IMS LD already mentioned above.

Our approach is also related to Pattern languages [1]: extracting snippets of adaptive behaviour (here, for collaborative adaptation) that are to be reused in

[17] http://www.imsglobal.org/learningdesign/
[18] http://www.w3.org/TR/owl-features/

different context (e.g., by different learners, teachers; groups of learners or teachers; with different course materials, etc.).

5 Conclusion and Future Work

In this paper, we have described the current developments of the LAG adaptation language, one of the first adaptation languages, discussed in the context of use and application of adaptation languages in general. We have sketched also an XML equivalent of LAG, which can be used for enhanced portability. Moreover, we have shown current progress in the design, implementation and small-scale evaluation of PEAL, a new environment for LAG language specification, which builds upon lessons learnt from previous implementations.

Other Lessons learnt, which can be of use to the research community at large, and connected to future work are as follows: 1) authors expect the authoring environment to be online, same as their environments for domain and other static map editing; they do not expect to have to install systems on their own computer. This is why both the initial LAG environment and PEAL are online editors. 2) An adaptation language as a human-computer interface is useful only for programming-savvy authors. Other authors should not be expected to program from scratch, but only reuse existing strategies. Hence, the strategy description in natural language is vital (strategy meta-data, as depicted in the 'description' part of the LAG language strategy). 3) It is easier to ask authors that have some level of programming knowledge to extend or modify existing strategies, thus adapting the strategies to new requirements gradually. 4) Simplifying the language can lower the threshold for authoring, but might have rejection effects for experts. 5) Similarly, visualization of the adaptation language can significantly lower the threshold for authoring novices. 6) Adaptation languages are useful also for computer-computer or system-system interfaces. Thus, they will not disappear, but instead will be the 'hidden' knowledge behind fancy interfaces, which are directly authored from scratch only by experts. 7) XML languages or XML compatibility is desirable, due to the automatic processing provided for web-based systems. 8) Whilst having a generic standard for all adaptive systems might sound ideal, a practical solution might be to mimic other use of standards elsewhere: e.g., LMS export to and import from various e-learning standards (LOM, IMS-CP, IMS-QTI, etc.). Similarly, adaptive systems, be they authoring or not, might need to be able to export various adaptation languages, in order to be compatible with a wide variety of applications.

Acknowledgments. This research is supported by the GRAPPLE IST project IST-2007-215434 and was initiated within the PROLEARN Network of Excellence.

References

1. Stash, N. Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System, PhD. Thesis. Eindhoven Univ. of Technol., The Netherlands. 2007.
2. Rich, E. User modeling via stereotypes, Cognitive Science, 3 (4), 329-354. (1979)
3. Brusilovsky, P. Methods and techniques of adaptive hypermedia, Journal of User Modeling and User Adapted Interaction, 6, (2-3), 87-129. (1996)

4. Brusilovsky, P.: Developing adaptive educational hypermedia systems: From design models to authoring tools. In: T. Murray, et. al (eds.): Authoring Tools for Advanced Technology Learning Environment. Dordrecht: Kluwer Acad. Publishers, 377-409 (2003).
5. Berlanga, A., et al.: Modelling adaptive navigation support techniques using the IMS learning design specification, Hypertext, Salzburg, Austria, 148 - 150 (2005)
6. Cannataro, M. et al., Modeling Adaptive Hypermedia with an Object-Oriented Approach and XML. In Proc. of WebDyn'02, Honolulu, Hawaii, May 2002.
7. Ceri, S. et al.: Web Modeling Language (WebML): a modeling language for designing Web sites, Computer Networks, 33(1-6) 137-157.(2000)
8. Cristea, A. and De Mooij, A. LAOS: Layered WWW AHS Authoring Model and their corresponding Algebraic Operators, WWW'03, Budapest, Hungary. (2003)
9. Koch, N. and Wirsing, M.: Software Engineering for Adaptive Hypermedia Applications?, AH workshop at UM'01, Sonthofen, Germany, July 13-17 (2001)
10. Specht, M. and Burgos D.: Modeling Adaptive Educational Methods with IMS Learning Design. Journal of Interactive Media in Education (2007)
11. Cristea, A.I., et al.: Towards a generic adaptive hypermedia platform: a conversion case study, J. of Digital Info. (JoDI), Spec. Iss. on Personalis. of Comp. & Services, 8(3). (2007)
12. Cristea, A.I., C. Stewart, C.D.: Automatic Authoring of Adaptive Educational Hypermedia, book chapter II in "Web-based Intelligent E-Learning Systems: Technologies and Applications", ZongMin Ma (Ed.), Info. Science Publishing (IDEA group); 24-55 (2006)
13. Ohene-Djan J. A Formal Approach to Personalisable, Adaptive Hyperlink-Based Interaction. PhD thesis, Dept. of Computing, Goldsmiths College, Univ. of London. 2000.
14. Stash, N., et al.: Adaptation languages as vehicles of explicit intelligence in Adaptive Hypermedia, In IJCEEL journal, 17, (4/5), 319-336, InderScience, 2007.
15. Cristea, A.I., Verschoor, M.: The LAG Grammar for Authoring the Adaptive Web, ITCC'04, April, 2004, Las Vegas, US, IEEE (2004)
16. Stash, N., et al.: Adaptation to Learning Styles in E-Learning: Approach Evaluation, Proceedings of E-Learn 2006 Conference, Honolulu, Hawaii (2006)
17. Cristea, A.I., Calvi, L.: The three Layers of Adaptation Granularity. UM'03, US (2003)
18. De Bra, P., et al.: The Design of AHA!, Proceedings of the ACM Hypertext Conference, Odense, Denmark, August 23-25, 2006, 133 (2006)
19. Cristea, A.I.: Adaptive Course Creation for All, ITCC'04 (International Conference on Information Technology) April, 2004, Las Vegas, US, IEEE(2004)
20. Hendrix, M., et al.: Defining adaptation in a generic multi layer model: CAM: The GRAPPLE Conceptual Adaptation Model, ECTEL, 2008
21. Hendrix, M. and Cristea, A.: A meta level to LAG for Adaptation Language re-use, A3H: 6th Int. A3H, AH 2008. Hannover, Germany. (2008)
22. Nielsen, J. Usability Engineering, Academic Press Inc, p 165 (1994)
23. Alexander, C.: A Pattern Language: Towns, Buildings, Construction. USA: Oxford University Press. ISBN 978-0195019193 (1977)
24. Dolog, P., et al.: The Personal Reader: Personalizing and Enriching Learning Resources using Semantic Web Technologies, AH 2004.
25. Eklund, J., Brusilovsky, P.: InterBook: An Adaptive Tutoring System UniServe Science News Vol. 12. March 1999. 8--13 (1999)
26. Moore, A., et al.: WHURLE - an adaptive remote learning framework, ICEE-2003, July 22-26, Valencia, Spain (2003)