Reuse Patterns in Adaptation Languages: Creating a meta-level for the LAG adaptation language

Maurice Hendrix and Alexandra Cristea

The University of Warwick, Department of Computer Science
Gibbet Hill Road, CV4 7AL, Coventry United Kingdom
{maurice, acristea}@dcs.warwick.ac.uk

Abstract. A growing body of research targets authoring of content and adaptation strategies for adaptive systems. The driving force behind it is *semantics-based reuse*: the same strategy can be used for various domains, and vice versa. Whilst using an *adaptation language* (LAG e.g.) to express reusable adaptation strategies, we noticed, however, that: a) the created strategies have common patterns that, themselves, could be reused; b) templates based on these patterns could reduce the designers' work; c) there is a strong preference towards XML-based processing and interfacing. This has leaded us to define a new meta-language for LAG, extracting common design patterns. This paper provides more insight into some of the limitations of Adaptation Languages like LAG, as well as describes our meta-language, and shows how introducing the meta-level can overcome some redundancy issues.

Keywords: LAG; AHA!; Adaptive Hypermedia; Adaptation Engine.

1 Introduction

The use of adaptive systems [2] is increasingly popular, as it can provide a richer personalised learning experience. Commercial systems on the web (e.g., Amazon) or beyond (PDA device software) present at least a rudimentary type of adaptation. However, adaptation specification cannot be fully expressed by standards[1] yet, and most commercial and non-commercial systems rely on proprietary, custom designed, system specific, non-portable, and non-interoperable adaptation. An intermediary solution, until standards emerge, is the creation of *Adaptation Languages*, which, with their power of semantics-based reuse, appear as a reliable future vehicle [3, 4]. Once written, the same adaptation strategy can be used for various domains and vice versa. However, there are a number of limitations regarding adaptation engines, which ultimately influence the efficient authoring of adaptation strategies.

In Section 2 we define what we see as the main limitations. Moreover, we propose a *meta-language*, as a supplement to Adaptation Languages like LAG, showing how introducing it can overcome such limitations. This solution is compatible with *extant* adaptation engines, instead of requiring the creation of new engines.

[1] SCORM Simple sequencing allows basic adaptation. IMS-LD promises more for the future.

2 Adaptation Engine Issues and Limitations

The following are issues and limitations identified as influencing the authoring flexibility of adaptive hypermedia (AH) systems:

L1. Most adaptive hypermedia delivery systems determine the *adaptation on a per-concept base* [1]. A broad knowledge of the whole content at every adaptation step is (usually) unavailable, mainly due to run-time complexity limitations. Thus, adaptation strategies cannot specify complex inter-concept rules; e.g., a strategy with an arbitrary set of labels denoting topics of interest, displaying to the user concepts related to his topic, without limiting the possible topics at design-time.

L2. Adaptation engines don't (usually) allow for *non-instantiated program variables* [1]. Thus, authoring strategies which involve an unknown number of types, categories, etc., are currently not permitted. All domain-related variables need to be instantiated in the authoring stage.

L3. There are extreme difficulties arising when *combining multiple strategies* [1], which would facilitate authors to create their own behaviour by composing strategies. Adaptation engines update sets of variables based on some triggering rules, without knowing which high-level adaptation strategies these variables represent. An example of a combined strategy currently difficult to implement is one where the system checks whether the user prefers text or images, and then displays the preferred type of content, filtered via a beginner-intermediate-advanced strategy, where concepts are shown based on the user's knowledge.

3 Solutions to Adaptation Engines Issues and Limitations

A straightforward way of defeating limitations L1 and L2 would be to build new adaptation engines. The first scenario could be achieved by establishing which labels exist, in the *initialization* step. The second issue could be overcome by either allowing arrays of labels, or otherwise allowing multiple data to be stored in the label. However, in order to function with current systems, these issues should be solved in the authoring stage. For the third limitation (L3), the difficulty in application of multiple strategies, there are some efforts already to deal with this. The MOT to AHA! [1] converter, e.g., has already implemented an elegant solution (unique to our knowledge so far), in that it can apply multiple LAG files, with different adaptation strategies, with the order of execution set by priorities of the respective strategies (1: highest priority; any following number: lower priority).

Nevertheless, this method could override previous variables, thus, a unitary strategy merge, based on multiple labels for domain-related concepts and attributes, is preferable. Moreover, only simple types are currently allowed by most Adaptation Languages, for example arrays, due to lack of adaptation engine support. For example arrays or lists are not allowed.

However, we have noticed that a) strategies have common patterns that could be reused; b) templates based on these patterns could reduce the designers' work; c) a strong preference exists for XML-based processing and interfacing.

For the creation of Adaptation Language strategies explicit knowledge about the content is needed. In a template version, this can be described and a pre-processor can then take both the content and the template strategy to create the concrete strategy for adapting the content. In the next section our solution will be described, it is based upon the LAG adaptation language and uses an XML-based template LAG language as Adaptation Language.

4 Meta-level Addition to LAG

To solve the limitations mentioned in section 2, we add a pre-processing step to the whole authoring process. This step takes a LAG template and the content, in the form of a CAF (Common Adaptation Format) file, and generates a new LAG file which extends the strategy sketched by the LAG template for the specific content described in the CAF file. For reusability, maintainability and to accommodate for future changes, we propose an XML-based notation for the template LAG files. Since CAF is already written in an XML-based notation, both documents can be used as input for an XSLT transformation which generates the resulting LAG file. Below we give the DTD (document type definition) for the template LAG file.

```
<!ELEMENT TLAG ((LAGfragment*, LIKE*)*)>
<!ELEMENT LIKE attribute CDATA value CDATA
(LAGfragment, MATCH, LAGfragment, (LAGfragment*, LABEL,
LAGfragment*)*) >
<!ELEMENT LAGfragment (#PCDATA)>
<!ELEMENT MATCH EMPTY>
<!ELEMENT LABEL EMPTY>
```

A template LAG file consists of a number of blocks of the following kind: a number of LAG fragments followed by a LIKE element. The fragments contain LAG adaptation snippets. The LIKE elements consist of an attribute and a regular expression against which it is matched, followed by a fragment of LAG program. The word MATCH represents the place where the LABEL needs to match the regular expression.

L1. *Problem: adaptation on a per-concept base*; a broad knowledge of the whole content at every step of the adaptation is (usually) unavailable.
Solution: such knowledge is not necessary in the adaptation engine. It is acceptable that this type of knowledge can be acquired as a one-off, at authoring time, as it is not to be expected that content labels will change at execution time. Therefore, the authoring strategy should contain this knowledge. As for an author it is difficult to manually extract all the pedagogical label types existent in a course, templates such as the DTD of the template LAG above can help in dealing with groups of labels (such as all labels containing 'beginner', i.e., '*beginner*'). An author can then generate the appropriate adaptation strategy (of which a snippet is shown above) in an easy and quick manner, making use of existing patterns in the authoring strategy itself.

L2. *Problem:* adaptation engines don't usually allow *non-instantiated variables*.

Solution: Unknown domain-related variables can be instantiated in the authoring stage, with the help of patterns specified via the LAG template language based on the above DTD. It is not necessary for an author to perform these searches manually; the two-step authoring system can extract unknown variables for him.

L3. *Problem:* extreme difficulties arise when *combining strategies*.
Solution: similar pattern extraction mechanisms have to be used in order to merge adaptation strategies. In (nearly) every system there is a limited number of weights and labels; this causes problems in combining a number of strategies greater than the number of weights and labels available. A solution to this can be to apply pattern matching on labels in order to be able to encode multiple strategies, by using the same label field. This thus enhances simple prioritization of strategies, as it allows the combination of multiple strategies which each requires specific labels.

5 Conclusions and Further Work

In this paper we have analyzed adaptation problems inherent in current adaptation engines, which reduce the power and generality of Adaptation Languages. We described and exemplified these issues with the help of the *LAG language*, currently one of the only exchange formats of adaptation language specification between systems. Moreover, we have moved one step further, by proposing improvements that can overcome run-time issues of adaptation engines, by solving them at the authoring stage. More specifically, templates can be used to create adaptation strategies, customized for the given domain models and pedagogical labels. For this purpose, we have proposed the *template LAG language*. The process is technically implemented by adding a pre-processor to the system setup, which has access to content at compile-time, which is not available at run-time. In such a way, more powerful adaptation strategies can be created for *existing* adaptation engines. The next step is to implement the pre-processor for LAG, merging efforts with new versions of AHA!

Acknowledgments. This research has been performed with the help of the EU *ALS* Minerva STREP, the EU FP7 *GRAPPLE* STREP, and was based on the *PROLEARN* Network of Excellence.

References

1. AHA! Adaptive Hypermedia For All, http://aha.win.tue.nl
2. Brusilovsky, P.: Adaptive hypermedia, User Modelling and User Adapted Interaction, Ten Year Anniversary Issue, (Alfred Kobsa, ed.) 11 (1/2), 87--110 (2001)
3. Cristea, A.I., Calvi, L.: The three Layers of Adaptation Granularity. UM'03, Pittsburg, US (2003)
4. Stash, N., Cristea, A.I., De Bra, P., Adaptation languages as vehicles of explicit intelligence in Adaptive Hypermedia, In International Journal on Continuing Engineering Education and Life-Long Learning, vol. 17, nr 4/5, pp. 319-336, InderScience, 2007.