

## Crosscutting Runtime Adaptations of LD Execution

**Telmo Zarraonandia, Juan Manuel Dodero and Camino Fernández**

Universidad Carlos III de Madrid, Departamento de Informática, Escuela Politécnica Superior  
Av. Universidad 30 Leganes, Madrid, España 28911

tzarraon@inf.uc3m.es

dodero@inf.uc3m.es

camino@inf.uc3m.es

### ABSTRACT

In this paper, the authors describe a mechanism for the introduction of small variations in the original learning design process defined in a particular Unit of Learning (UoL). The objective is to increase the UoL reusability by offering the designers an alternative to introduce slight variations on the original design instead of creating a new one each time they want to reuse it. No changes or extensions to the Learning Design definition are required to perform these modifications. The use of design patterns to include the adaptations offers the possibility to easily introduce new operations, such as tracing the activity progress, for instance. The structure of a Learning Design player that is able to process the desired adaptation information and to apply it at runtime will be outlined. The player will be part of an architecture for the automatic adaptation of UoLs to their actual context of execution.

### Keywords

Learning Design, Unit of Learning, Adaptation, Adaptive Systems, Context

### Introduction

A Unit of Learning (UoL) is obtained when a description of an LD is included in a content package manifest (IMS, 2003b). The Unit of Learning encapsulates all the information required to go through the learning process, including both pedagogical information and information needed to locate and use the required resources. An appropriate tool called a learning design player can open the Unit of Learning and provide the participants with an appropriate interface to perform the activities during the learning process (IMS, 2003a). The IMS LD specification defines three levels of implementation and compliance (IMS, 2003b). Level A contains the core of the Learning Design, providing a vocabulary to specify a sequence of activities to be carried out for the learners and teachers who take part in the learning process, while Levels B and C allow the designers to define a more elaborated sequencing of the process (Jeffery & Currier, 2003).

A Unit of Learning is designed for execution in a particular environment and under the assumption that some specific conditions are satisfied. Its execution under different conditions does not guarantee that the expected results will be obtained and the whole process should be revised. However, sometimes light modifications – e.g. replacing the resource associated with a particular activity, modifying the presentation order, etc. – would make it possible to reuse most of the original processes in a new situation. For instance, we have developed a Unit of Learning for a particular course taught at our university. Now we have to use the same course to train the employees of a company. The computers used for training in the company are less powerful than those from the university and we know the bandwidth of the network is far smaller. Therefore, some of the activities proposed in the original Unit of Learning may not be suitable for the new conditions. However, the rest of the process remains quite similar. Later on we may want to reuse the course to train a different audience and similar situations can occur. The instructor is forced to create a new UoL for each of the courses even when they all implement the same learning process.

Furthermore, in practice, total reusability of the UoLs is not an easy objective to attain as, even when applied to very similar situations, the introduction of small variations on the learning process is usually required. From one semester to another authors may desire to replace some of the resources for more up-to-date files, to remove the activity environments to test learner's knowledge without helping resources, to introduce a presentation of the course for the current audience, to modify the question item order, etc. But like in the previously mentioned situation, each modification requires redefining the whole UoL.

Keeping different UoLs for each variation of the process could be an adequate approach when the number of UoLs is not large and their complexity is low, otherwise the costs of maintenance are high. On the other hand, designers can use level B elements to modify the UoL runtime behavior based upon the value of different properties. The system proposed by the ALFANET project takes advantage of this characteristic and is able to

recommend to each individual learner the most appropriate material taking into account the interactions performed by a group of similar learners (Alfanet, 2005). However, in any case, that material must be included into the course before its use. It is not possible for the designers to know in advance the range of adaptations that may be required to apply in the future, and for that reason the UoL will have to be redesigned each time a new modification comes into place. In addition, one adaptation can involve changes at different places of the UoL. As the number of adaptations applied to the UoL increases, it becomes more difficult to identify which change corresponds to which adaptation and operations like adaptation removal may become difficult to carry out.

Jacobson et al (1997) defined variation point as “places in the design or implementation that identify locations at which variation can occur”. Variation points can be bound to the system at different stages of the product lifecycle. Svahnberg et al. (2002) presented a taxonomy of variability realization techniques which defined different ways in which a variation point can be implemented. One of these techniques is the code fragment superimposition, where a software solution is developed to solve the generic problem; code fragments are superimposed on top of this software solution to solve specific concerns. This superimposition can be achieved by means of different techniques, as for example the aspect oriented programming approach, and provides the designer the possibility to bind the modification during the compilation phase or even at runtime.

Taking these concepts into the adaptation of the UoLs area we can define an alternative approach to the Level B usage. The authors can describe the desired adaptations on auxiliary specification files that could be processed together with the original UoL and applied at runtime giving the user the feeling that they were included on the original UoL. This way, we can maintain a single UoL definition and a number of descriptions for adaptations. Those files tie together all the changes involved on a particular adaptation and keep that particular concern separated from the main UoL functionality and the rest of adaptations.

An overview of the process is shown in figure 1. From several possible adaptations defined for a particular UoL, the designer chooses the one which best fits the current situation and applies it to the UoL. The introduction of the adaptive action can be carried out at design time (adaptation 1) or at runtime (adaptation 2, 3, 4). In the last case, adaptation could be applied to all the running instances of a UoL (adaptation 2), to all the users of a particular running instance (adaptation 3) or only to the personalized view of a particular user (adaptation 4).

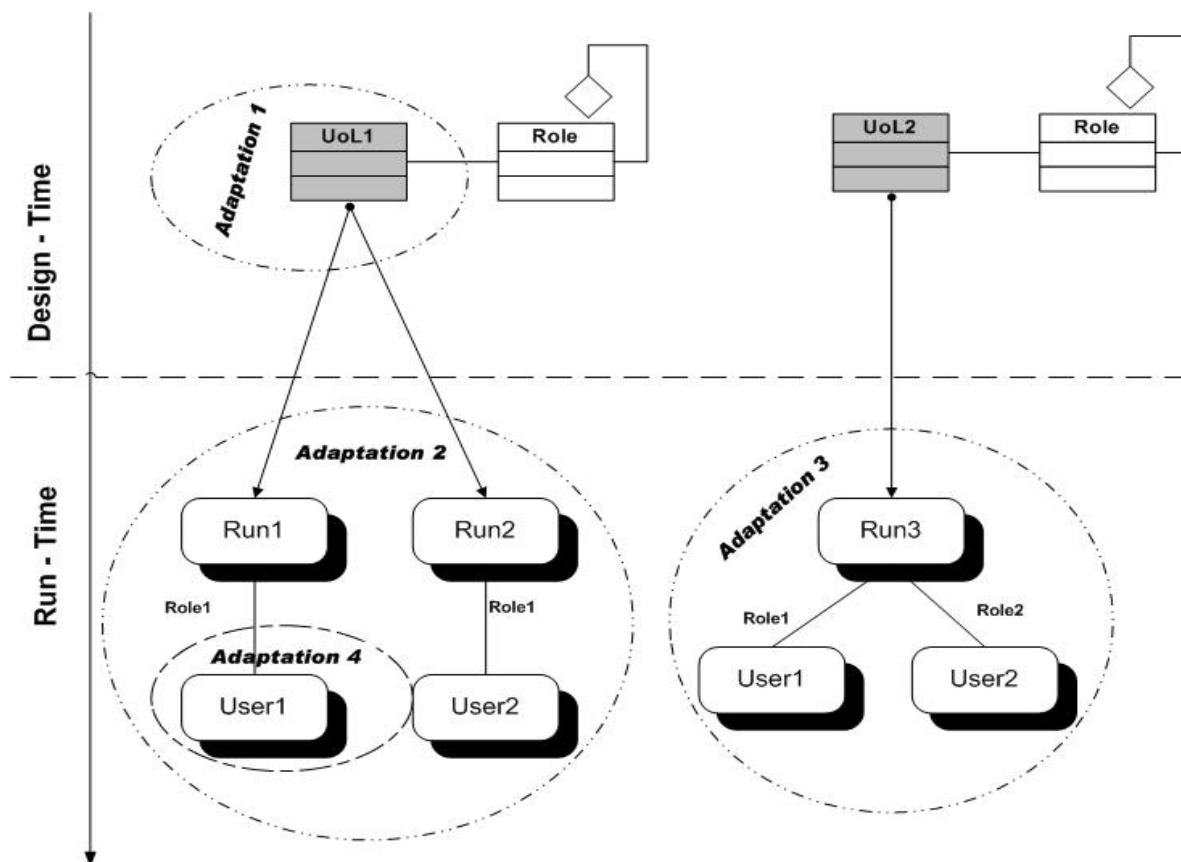


Figure 1. Overview of the adaptation process of a Unit of Learning

Two kinds of adaptive actions over a Unit of Learning can be defined: adaptations for its reuse in a specific situation and adaptations for its reuse in a context different from the original it was defined for. In the last case, the characteristics of the current environment of execution of the UoL can be captured, and the adaptation application automated.

In this document, we outline the architecture of a system capable of processing contextual information and automatically taking adaptive actions on a Learning Design. The core of the system is an LD Player able to adapt at runtime a UoL execution. The player is designed as an extension to the CopperCore runtime engine and is implemented with the help of different design patterns and an Aspect Oriented Programming approach.

The rest of the paper is organized as follows. First, a description of the adaptation capabilities of the Learning Design Player engine is presented. Second, its structure is outlined. Next, the implementation details are provided. Then, some examples of UoL adaptations are described and the architecture of the context adaptive system is introduced. Finally, some conclusions and further work are presented.

## Adaptation Definition

In principle, all the Learning Design elements that are defined in a UoL manifest file can be subject to adaptation. However, the adaptation process only makes sense when it involves just small modifications of the original process. The introduction of new roles, acts, complex conditions or structural mayor changes, requires the description of complicated adaptation files, making the redesign of the original UoL a more suitable approach. Taking this into account we limit the range of possible adaptations to the following subset of the elements in an LD.

- Level A: only activities, environments and resources can be adapted. This includes the definition and association of new resources with existing activities or environments, the introduction or removal of these types of elements and some modifications to their original definition.
- Level B: modifications of the definition of properties, conditions and the values of the elements “when-property-value-is-set” and “change-property-value”.

We define an *adaptation poke* as the description of a small modification of some elements in a learning design process. To describe an adaptation poke three different types of files can be specified:

- Adaptation command files: Containing the list of adaptation commands to be applied sequentially to the original Unit of Learning definition. It is the only mandatory file for the definition of an adaptation poke. Table 1 shows the possible adaptation commands. Note that for each command there exists an equivalent *commandT*, providing the same functionality but using the title of the elements instead of the id.
- Adaptation manifests: these are XML files containing the definition of new activities, environments, resources, properties or conditions. These definitions will be incorporated to the set of Learning Design elements definitions that are read from the original UoL manifest. If a condition element definition is found, it will override the condition definition of the original manifest. The schema definition of this type of file is based on the XSD binding for the Learning Design Definition Model (IMS, 2003c) for the above-specified elements.
- Resource files: new content files.

Adaptation pokes can be applied to the UoL at publication time if they are included into the content package. The Learning Design Player will process the files together with the original information and apply the changes before the information is presented to the participants of the process. Pokes must be numbered, and if more than one is included in the package the Learning Design Player will apply them sequentially.

To perform the adaptation on a running UoL instance, the adaptation poke files should be uploaded to the Learning Design Player specifying the UoL, run and user instance for which the adaptation should be applied.

In case of conflict between the definition of the poke and the UoL the whole poke will not be applied and the designer will be reported about the reasons. This is especially important when applying several pokes to the same UoL as the structure of the original LD may be modified and some activities, environments and resources associated to definition of the latest pokes may have been modified or even been removed. Besides, the modification of Level B properties and conditions can generate deadlocks; the original learning objectives can be changed and also must be taken into consideration the actual stage of the participants in the learning process in order to avoid inconsistencies. The final version of the adaptive LD Player will include a mechanism to automate the detection of such conflicts when possible.

Table 1. List of adaptation commands

<b>State Change Commands</b>	
<b>LEVEL A</b>	
<b>Actions on Activity Structure elements</b>	
Change title	ASchgTitle idActivity, newTitle
<b>Actions on Learning Activity elements</b>	
Change title	LAchgTitle idActivity, newTitle
Change the resource reference of an Item element of the activity description	LAchgDes idActivity, idItem, idItemRef
Set the <i>complete-activity</i> definition of the Learning Activity (user choice   time limit)	LAcamp idActivity, newoption
Change the resource reference of an Item element of the feedback description of the on-completion	LAoncomF idActivity, idItem, IdItemRef
<b>Actions on Support Activity elements</b>	
Change title	SUchgTitle idActivity, newTitle
Change the resource reference of an Item element of the activity description	SUchgDes idActivity, idItem IdItemRef
Set the <i>complete-activity</i> definition of the Support Activity (user choice   time limit)	SUcomp idActivity, data
Change the resource reference of an Item element of the feedback description of the on-completion	SUoncomF idActivity, idItem, IdItemRef
<b>Actions on Environment elements</b>	
Change title	ENchgTitle idEnvironment, newTitle
Change the resource reference of an Item element of the learning-objects definition	ENchgDes idEnvironment, idLO, idItem, IdItemRef
<b>Actions on Resources</b>	
Change the file reference of a resource*	REchg idResource, href
<b>LEVEL B</b>	
<b>Actions on Property elements</b>	
Change the initial value of a property	PRini idProp, value
<b>Actions on Learning Activities</b>	
Set the <i>complete-activity</i> definition of the Learning Activity to when-property-value-is-set	LAcomPv idActivity, propRef, propValue
Set the on-completion definition of the Learning Activity to change-property-value	LAoncomC idActivity, propRef, propValue
<b>Actions on Support Activities</b>	
Set the <i>complete-activity</i> definition of the Support Activity to when-property-value-is-set	SUcomPv idActivity, propRef, propValue
Set the on-completion definition of the Support Activity to change-property-value	SUoncomC idActivity, propRef, propValue
<b>Structural Change Commands</b>	
<b>LEVEL A</b>	
<b>Actions on Activity Structure elements</b>	
Add a new activity to the structure	ASaddAct idActivity, idActivity, pos
Remove an activity from the structure	ASrmvAct idActivity, idActivity
<b>Actions on Learning Activity elements</b>	
Add a new environment element	LAaddEnv idActivity, idEnvironment
Remove an environment element	LArmvEnv idActivity, idEnvironment
<b>Actions on Support Activity elements</b>	
Add a new environment element	SUaddEnv idActivity, idEnvironment
Remove an environment element	SURmvEnv idActivity, idEnvironment
<b>Actions on Environment elements</b>	
Remove a learning object element	ENrmvLo idEnvironment, idLO

\* This action replaces the file definitions of a resource of type “webcontent” with the specified *href* reference.



For each running instance of the Unit of Learning there will be only one instance of the *Adaptor* class, which inherits from *LDModifier* and encloses all the changes associated to the adaptation files, which are the changes that should be applied to the elements of the Unit of Learning for its actual execution. Once the original Unit of Learning information is read, the *accept* operation of the *manifest* element can be called, passing the *Adaptor* instance as an argument. This will trigger the manifest *visit* method of the *Adaptor*, which will retrieve, one by one, the different elements for which an adaptation has been defined and, in order to perform the appropriate changes, call their *accept* operation passing the *Adaptor* instance again. Figure 3 illustrates this process.

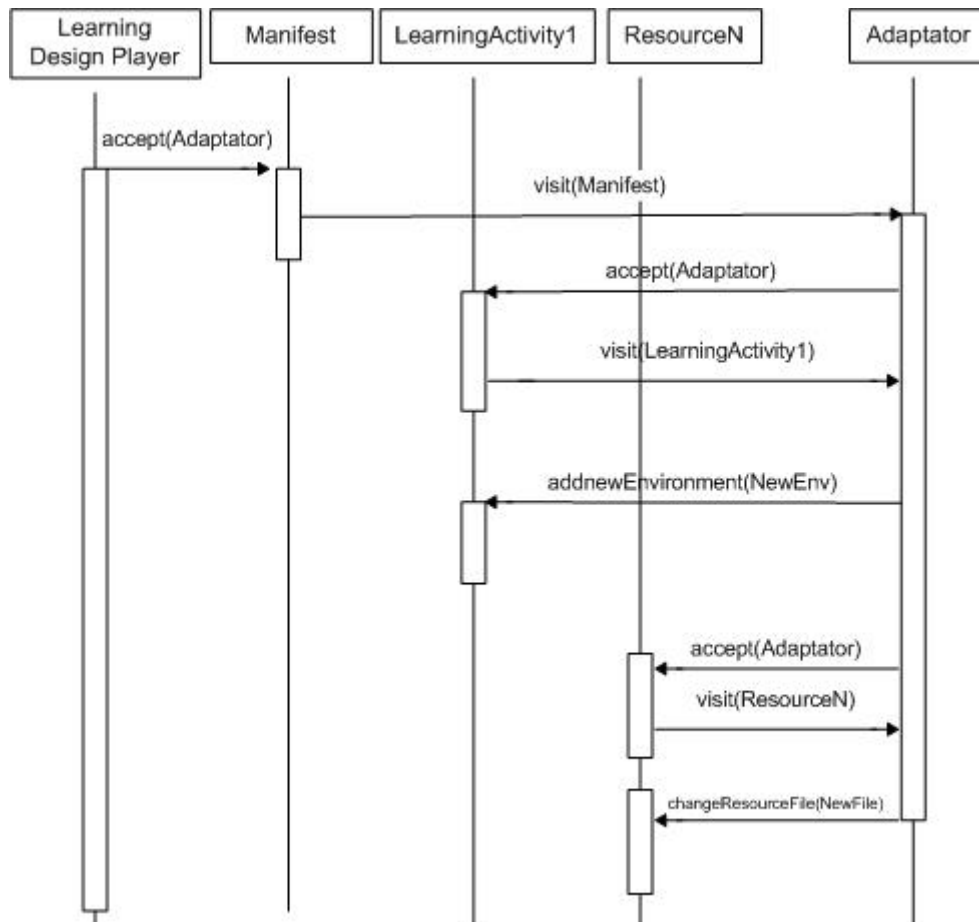


Figure 3. Sequence diagram of an adaptation poke application

Future adaptations can be easily performed. We only have to create the new adaptation files and include them in the content package together with the original ones or upload them to a running instance. The *AdaptationReader* will generate the appropriate *Adaptor* object and pass it to the execution engine for its application.

The addition of new operations over the elements of the structure is also smooth. For example, we can create a *Progress Watcher* class that inherits from the *LDModifier* and retrieves through its *visit* methods information about the different components of the Learning Design (property values, resources visited, time spent on a particular activity, etc). Since it extends the *LDModifier* class, we can pass this type of object as an argument to the *accept* operation of the Learning Design element classes. These will call the *visit* operation of the *ProgressWatcher*, what is performed without any change on the class interface. For the elements of the Learning Design the particular type of *LDModifier* received by the *accept* operation is transparent. The *Progress Watcher* implementation on the Learning Design Player could complement a *Monitor Service* actuation or constitute an alternative approach to its implementation on those UoLs not concerned with monitoring and for which the information retrieval is a requirement in a particular learning session.

The organization is easy to maintain and only if future specifications of the Learning Design should consider new types of elements, the interface of the *LDModifier* would be modified to include new operations for the new elements.

## Implementation

The adaptive LD Player is implemented as an extension to the CopperCore IMS Learning Design engine (OUNL, 2005). The CopperCore engine has been created by the Open Universiteit Nederland (OUNL) and is capable of processing the three levels of IMS Learning Design. It is not designed as a standalone application but to be integrated into existing e-learning infrastructures (Kraan, 2004).

## Aspect Oriented Programming Approach

The implementation of the design as previously described requires modifying some of the elements of the Learning Design to include the new operation *accept*. The first approach would be directly modifying all the elements involved in the process, which means modifying the CopperCore engine code. From the point of view of the software, flexibility and modularity, another possibility seems to be much more suitable: the use of an Aspect Oriented Programming approach.

Aspect Oriented Programming (Marcus, 2001) extends the object-oriented paradigm by introducing the concept of aspect, which encapsulates cross-cutting behaviors that affect multiple classes into reusable modules. Aspects are defined separately from the classes and methods that make up components at design time, and compilers and interpreters are in charge of the integration according to some supplied criteria and before the conversion into binary code.

Following these ideas, we can simulate the *accept* operation of the elements of the LD by defining our *LDModifiers* as separate aspects and establishing the conditions in which the normal execution of the elements will be intercepted to launch the new code. This way new aspects and launching conditions can be added or removed without any alteration on the Learning Design structure.

Furthermore, the use of this approach results in a more flexible structure, which allows the maintenance of the adaptive extension files separated from the specific CopperCore code. This facilitates the upgrade of the extension to new CopperCore engine versions.

## Adaptation Examples

In order to clarify these adaptations we illustrate some adaptations realized with examples taken from the OUNL DSpace repository (OUNL, 2003). The first one corresponds to a IMS LD Level A course and the second one to Level B. At this moment no authoring tool has been developed to automatically create the adaptive information files so manual editing is required. However, following the objectives of the process, its definition is straightforward. Next, we will explain the examples in detail.

### Level A adaptation. The Candidas example

“Candidas .The Great Unknown (I)” is an example of a simple course in level A developed by the Open Universiteit Nederland. It is composed of one act, one role and one single learning activity structure. The learner will go through an initial introduction to the material and different expositions; each one is followed by the corresponding questionnaire.

#### *Adaptation Poke 1: Remove test activities.*

In this adaptation poke teachers want to remove the test activities from the Candidas course. To modify the original *activity-structure*, an adaptation command file will be introduced into the original UoL including the following orders:

Adaptation command file:

```
ASrmvAct      AS-learningactivity,    Test-1
ASrmvAct      AS-learningactivity,    Test-2
ASrmvAct      AS-learningactivity,    Test-3
```

ASrmvAct	AS-learningactivity,	Test-final
ASrmvAct	AS-learningactivity,	Feedback

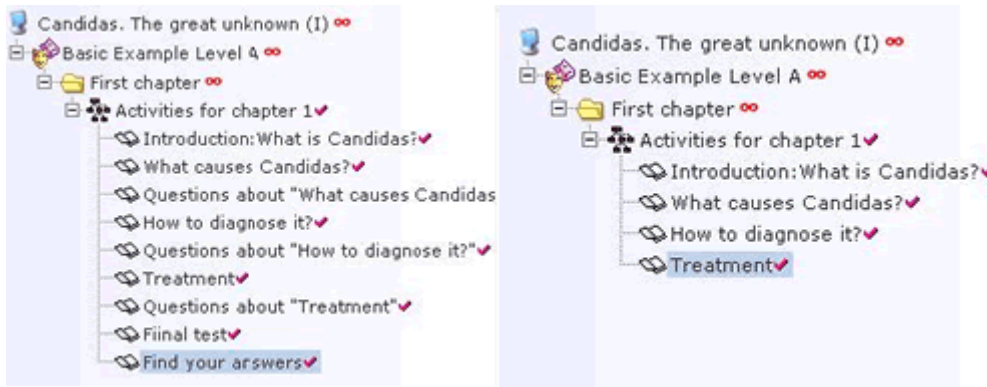


Figure 4. Structure of the course before and after the adaptation

Figure 4 shows two screenshots of the original course structure (left) and the structure after the adaptation poke is introduced into the UoL (right).

*Adaptation Poke 2: Remove test activities and include a teacher presentation.*

The objectives of this adaptation are to remove the tests and to add a teacher’s presentation including his contact information.

A new adaptation poke will be defined. It will be composed of an adaptation manifest file holding the description of a new activity “Presentation”, a new file resource associated with it, a new content file (*Presentation.html*), and an adaptation command file for adding the new activity to the activity structure.

Adaptation manifest file:

```
<imsld:activities>
  <imsld:learning-activity identifier=" Presentation ">
    <imsld:title>Presentation</imsld:title>
    <imsld:activity-description>
      <imsld:item identifierref="R- Presentation " identifier="I-
        presentation "/>
    </imsld:activity-description>
  </imsld:learning-activity>
</imsld:activities>

<resources>
  <resource identifier="R-Presentation" type="webcontent"
    href="presentation.html">
    <file href="presentation.html"/>
  </resource>
</resources>
```

Adaptation command file:

ASaddAct AS-learningactivity, Presentation, 1

The new adaptation poke will be applied on top of the one described in the previous point. Figure 5 shows the teacher’s presentation appearance after being added as a new activity of the course.



Figure 5. Teacher's presentation

### Level B adaptation. Learning to listen to Jazz example.

“Learning to listen to Jazz” is another example developed by the Open Universiteit Nederland. It describes a Level B course on the jazz music genre. It is composed of one act and two roles and the learner will have the possibility to follow the course by two different knowledge routes: thematic or historical. At the beginning of the course, the learner introduces some personal data, and after some initial testing she makes her choice about the knowledge route she wants to follow. Later on during the course, she will be asked about the grade of satisfaction on her decision and she will have the possibility to change to the other knowledge route.



Figure 6. Orientation activity environment before (left) and after adaptation (right)

### Adaptation Poke 1: Force the learner to go through the historical route.

The instructor wants to reuse the course but obliging the learner to go through the historical route. No initial knowledge or study approach test must be presented. For this to be achieved several modifications will have to be applied:

- The property related to the route election will be loaded with an initial value in order not to give the learner the possibility of choosing.
- The environment of that activity will be removed. This way the initial tests will be hidden.
- The activity ‘reflection in the meantime’ will be removed from the historical route activity-structure. This will block any possibility of changing to the thematic route during the course execution.

The adaptation poke will include the definition of an adaptation command file that contains the following three commands:

Adaptation command file:

```
PRiniT      option, historical
LArmvEnvT   orientation, What do you already know?
ASrmvActT   historical route, reflection in the meantime
```

Figure 6 to 8 show different screenshots of the course aspect before and after the application of the adaptation poke. Tests disappear from the 'orientation' activity (fig. 6) and no possibility to elect the knowledge route is presented to the user (fig. 7). The 'reflection in the meantime' activity is also removed from the 'historical route' activity structure (fig. 8).

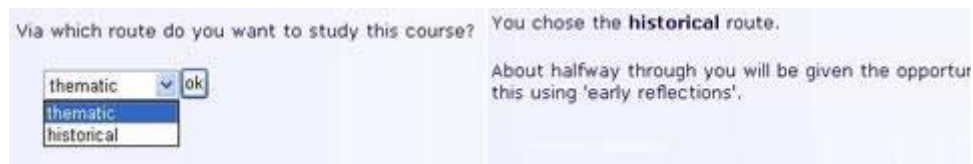


Figure 7. Orientation activity content before and after adaptation



Figure 8. Structure of the course before and after the adaptation

### Contextual Adaptations

It is very unlikely that a UoL could be reused exactly the same way in a different context from the one for which it was designed. Different learner characteristics, execution environments or time schedules, for instance, may require serious modifications of the definition of its Learning Design elements and a new redesign of the UoL considering the new characteristics of the new environment could be more appropriate.

However, other times, when the new context situation is close to the original one, reusability could be achieved by the introduction of specific variations on the process. This way, we can have an original UoL, and a set of possible modifications that will adapt the learning process to a predefined set of context situations. The characteristics of the actual context of execution can be captured and the best suitable adaptation applied in order for the learners to obtain a better-tuned process to follow.

This way, two possible sources of adaptation pokes can be found. On one hand, we have the adaptations the designer has defined for a learning process in order to solve specific situations in different courses. On the other hand, a set of context adaptations could be defined to make the Unit of Learning adequate for different environments of execution. Figure 9 describes the general process of adaptation of a Unit of Learning. The *ContextReader* gathers the features of the current environment of execution, determines which is the current

context situation and automatically applies the appropriate context adaptation. On top of that, the designer may apply her own modifications to the process to solve a specific problem or to optimize it.

## Context Model

Many definitions and different meanings for the term context and context-aware computing can be found in the literature (Chen & Kotz, 2000; Pasco, 1998; Schilit et al., 1994). In Dey (2001) the author defined context as “any information that can be used to characterize the situation of entities that are considered relevant to the interaction between a user and an application, including the user and the application themselves”. On another hand, Schilit et al. (1994) defines context-aware computing as “software adapts according to the location of use, the collection of nearby people, host, and accessible devices, as well as changes to such things over time”.

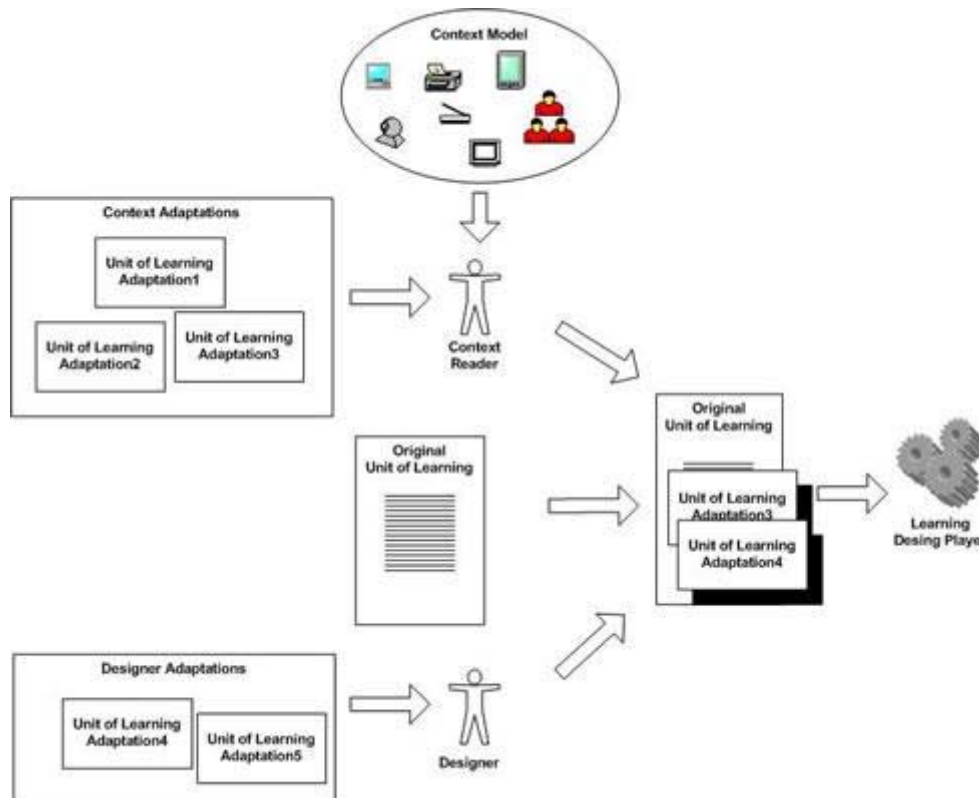


Figure 9. Overview of the context adaptation process

For our purposes, context will be any information that can be used to characterize the learning process, that is, any relevant information that could influence the execution of a Unit of Learning. This definition of context covers information about the computational environment (availability of different devices like printers or scanners, network connection bandwidth, computer characteristics, monitor resolution, etc) as well as physical and psychological characteristics of the user (age, background, disabilities, preferences, agenda) or information about the environment in which the learner is placed (location, weather conditions, other users availability, etc).

Every UoL is constructed having in mind a particular context, and to check its adequacy to the current context, different UoLs may require collecting different contextual information since the information that is relevant for a particular process may not be significant for another. Then, it is necessary to define which contextual data are significant for each learning process and how they should be obtained. We store that definition together with the definition of the UoL, and an ambient intelligence device can retrieve the actual values of the contextual information from different sources like sensors, operating system, user profiles, etc.

We consider a Learning Design context (Fig. 10) as any combination of context elements that can be classified in four types:

- Boolean Context Elements: Context elements with only two possible values: true or false. This type of element is adequate to denote the presence of connected devices, other learners, etc. We can define different

associated attributes to represent different characteristics and properties of the elements. For example: to represent the availability of a printer, we can define a Boolean context element named *Printer* with attributes *color* and *resolution*.

- Continuous Context Elements: In this category we group context elements whose value can be measured on a continuum or scale. The presence of the element is always true, but its intensity is variable. This kind of elements is appropriate to represent the lighting, noise level and other characteristics of the environment. In order to facilitate the work with these types of elements we can divide the range of their possible values into different sub-ranges. For example: for the context element *Lighting* we can define the ranges *Low* for values under 300 lux, *Medium* for values between 300 and 500 and *High* for greater values. If the actual value is 400, the *activeRange* would be *Medium*.
- Discrete Context Elements: This group includes context elements whose value is defined only for a particular vocabulary or classification. We can use this type of element to model characteristics of the user (marital status, background, occupation, etc), time and frequency terms (day of the week, months, etc), location (country, city, place), etc. As part of the definition of the element, it is necessary to provide the set of possible values or an URI to locate the specifically associated vocabulary.
- Aggregate Context Element: It is also possible to define a context element as an aggregate of other context elements. For example, we can define an element *Connection* that groups the continuous element *Bps* and the Boolean ones *Dial* and *Wireless*. The first element would indicate the speed of the connection while the other two would indicate if dialing its needed and if ubiquitous conditions can take place.

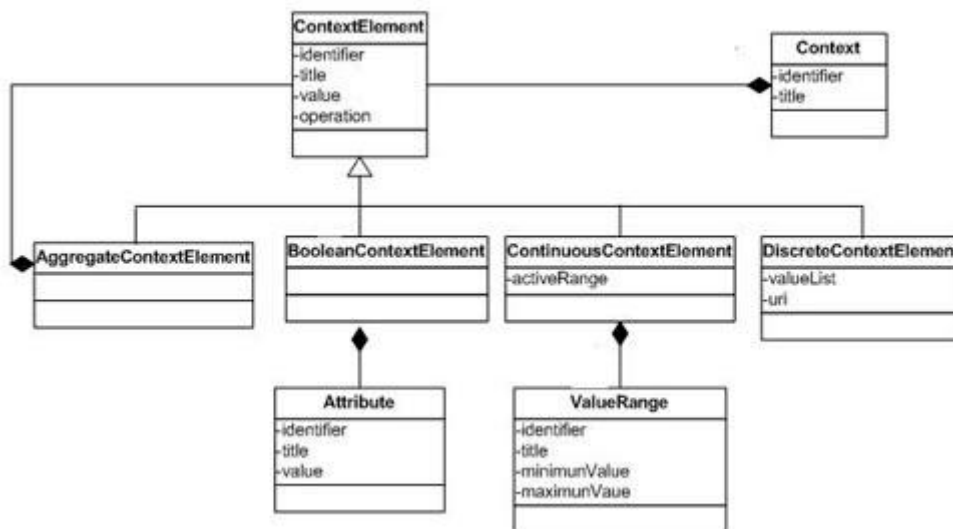


Figure 10. Context Model

The context of a Learning Design is formed by a particular collection of context elements. The same context definition can be associated to different UoLs, while in some other cases different UoLs may need different context definitions.

Before the execution of the process, the context elements must be populated with their current values for the session. The ambient intelligence engine will obtain those values by the execution of the appropriate operation, which is stored as part of the definition of each context element. The operation can consist in a specific call to the operating system or to different devices connected to the computer, to retrieve information from the learner profile, to query the user agenda, etc. Eventually, the user could be directly inquired.

## Context Situation Definitions

Once we have gathered information about the context of the learning session, we have to apply the appropriate changes to the Unit of Learning. Those changes consist of small variations to the original process in order to adapt it to the new situation.

The idea is to define context situations, i.e. sets of possible values for the context elements of the LD context definition, and associate them with a set of changes to the original elements.

For example, we have developed a UoL for an Internet course that covers an introduction to different Internet related concepts and the basics of web browsers, search engines, mail and instant messaging programs (Fig. 11).

Originally, the course was expected to be taken in a classroom and the scheduled time to complete it was two hours. Now, we want to reuse the procedure but giving the learners the possibility to follow the course from their homes. As the configuration of the home computers of the learners may not always be the same and some devices may not be available, it will be necessary to introduce some changes to the original design. On the other hand, the learners are not subject to time restrictions as they follow the course with their own computers and in their spare time. For that reason it is possible to increase the assigned time to some of the units to analyze them in depth.

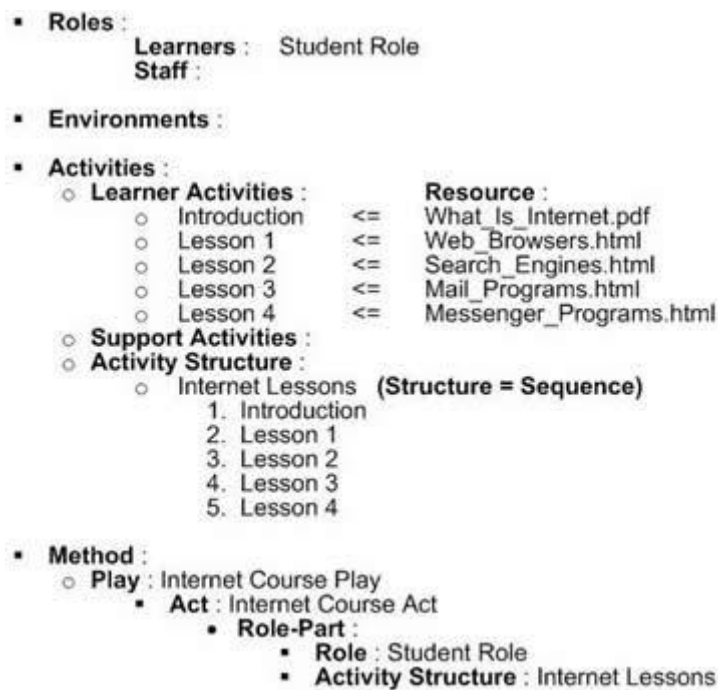


Figure 11. Unit of Learning schema for the Internet Course

Table 2 shows the elements of the context definition developed for the Unit of Learning, their possible values and the context situation names we have associated to their combinations.

If the ambient intelligence engine detects an Internet connection, a web camera, a low level of noise and an available time for the session of two hours, we can consider that the context of the learning session is quite similar to the one expected at design time. Therefore, no changes in the default process are required. On another hand, if no time constraints are detected on the user agenda, we can change the resource associated to the activity *Introductory* for a more extended document. If no web camera is available, we may want to replace the resource associated with the activity *Lesson 4*, which includes an explanation of how to proceed to configure that device for a messenger program. If the fourth combination of context element values is found, we can conclude that the user is on any transport or any other noisy environment. Her level of attention could be lower than on other context situations, and we may want to change the activities.

Table 2. Context Situation Definition

Noise Level	Time Constraint	Internet Connection	Web Camera	Context Situation
Low	Yes	Yes	Yes	Classroom
Low	No	Yes	Yes	Home1
Low	No	Yes	No	Home2
Low	No	No	-	Home3
High	Yes	No	Yes	Transport

## Adaptation to the context

For each of the context situation an adaptation poke will be defined. The ambient intelligence device will retrieve the values for the context elements that are relevant to the UoL execution, and based upon those values, determinate the current context situation. Then, the set of adaptation files corresponding to its associated adaptation poke will be incorporated to the original UoL. The adapted UoL will be passed to the Learning Design Player for execution.

## Conclusions and Future Work

The aim of our research is to develop a mechanism to incorporate runtime adaptation capabilities to Learning Design execution. The purpose is to increase the reusability and flexibility of the UoLs by using a simple procedure to introduce the desired modifications into the original learning process. The concept of an adaptation poke has been introduced as the specification of small adaptive actions that can be included in the delivered package and interpreted by an appropriate Learning Design Player to obtain the adaptations applied at runtime.

A Learning Design player structure with adaptation capabilities has been described. The major advantage of our approach is its easy extensibility and the lack of disruption into the Learning Design definition. The implementation is being tested as an extension to the CopperCore Learning Design engine using the visitor pattern and an Aspect Oriented Programming approach. At the time of this writing we are defining a mechanism for the prevention of conflicts between the adaptation poke changes and the UoL definition, and the final set of adaptive commands is also being refined. The final version of this application will be available as open source software.

The architecture of a system able to automatically perform adaptive actions to modify a UoL to a context execution different to the one for which it was designed has been outlined. The core of the system will be the adaptive Learning Design player previously described.

Future research lines take into consideration that by following the ideas described above, other operations like learner progress observation, or checking particular conditions of actuation, can be easily implemented. This will open possibilities to define more complex adaptations, not only based on pre-defined information but also on the evolution of the learning process. In addition, and taking advantage of the straightforward of adaptations of this approach, a user-friendly authoring tool for its description will be developed.

## Acknowledments

This work is part of the MD2 project (TIC2003-03654), funded by the Ministry of Science and Technology, Spain.

## References

- ALFANET (2005) *Standards Contribution Report*, retrieved November 10, 2005 from [http://dspace.learningnetworks.org/bitstream/1820/345/2/ALFANET\\_D32\\_+Standards\\_Contribution\\_Report.pdf](http://dspace.learningnetworks.org/bitstream/1820/345/2/ALFANET_D32_+Standards_Contribution_Report.pdf)
- Chen, G., & Kotz, D. (2000). A survey of context -aware mobile computing research. *Dartmouth Computer Science Technical Report*, TR2000-381, 2000.
- Dey, A. K. (2001). Understanding and using context. *CHI 2000 Workshop on the What, Who, Where, When and How of Context-Awareness*, retrieved July 25, 2005 from <http://www.cc.gatech.edu/fce/ctk/pubs/PeTe5-1.pdf>.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns*, Reading, MA, USA: Addison Wesley.
- IMS (2003a). *IMS Learning Design Best Practice and Implementation Guide, Version 1.0 - final specification*, retrieved November 10, 2005 from [http://www.imsglobal.org/learningdesign/ldv1p0/imsld\\_bestv1p0.html](http://www.imsglobal.org/learningdesign/ldv1p0/imsld_bestv1p0.html).
- IMS (2003b). *IMS Learning Design Information Model, Version 1.0 - final specification*, retrieved November 10, 2005 from [http://www.imsglobal.org/learningdesign/ldv1p0/imsld\\_infov1p0.html](http://www.imsglobal.org/learningdesign/ldv1p0/imsld_infov1p0.html).

- IMS (2003c). *IMS Learning Design XML Binding, Version 1.0 - final specification*, retrieved November 10, 2005 from [http://www.imsglobal.org/learningdesign/ldv1p0/imslld\\_bindv1p0.html](http://www.imsglobal.org/learningdesign/ldv1p0/imslld_bindv1p0.html).
- Jacobson, I., Griss, M., & Johnson, P. (1997). *Software Reuse. Architecture, Process and Organization for Bussiness Success*, AddisonWesley.
- Jeffery, A., & Currier, S. (2003). *What is... ims learning design?* Retrieved November 10, 2005 from [http://www.cetis.ac.uk/lib/media/WhatIsLD\\_web.pdf](http://www.cetis.ac.uk/lib/media/WhatIsLD_web.pdf), 2003.
- Kraan, W. (2004) *Coppercore to power Learning Design implementations*, retrieved November 10, 2005 from <http://www.cetis.ac.uk/content2/20040126154220>.
- Marcus, A., Feng, L., & Schaffer, K. (2001). *An overview of aspect oriented programming*, Kent State University, Department of Computer Science, 2001.
- OUNL (2003). *DSpace Service*, retrieved November 10, 2005 from <http://dspace.learningnetworks.org/>.
- Open Universiteit Nederland (2005). *CopperCore v2.2.2 release*, retrieved November 10, 2005 from <http://coppercore.org/>.
- Pasco, J. (1998). Adding generic contextual capabilities to wearable computers. *Paper presented at The Third International Symposium on Wearable Computers (ISWC '99)*, October 18-19 1999, San Francisco, CA, USA.
- Schilit, B., Adams, N., & Want, R. (1994). Context-aware computing applications. *Proceedings of the IEEE workshop on mobile computing systems and applications*, Piscataway: Santa Cruz IEEE Press, 85-90, retrieved October 25, 2005 from <http://seattleweb.intel-research.net/people/schilit/wmc-94-schilit.pdf>.
- Svahnberg, M., van Gorp, J., & Bosch, J. (2002). *A Taxonomy of Variability Realization Techniques*, Technical paper, Blekinge Institute of Technology, Sweden, 2002.