A Late Modelling Approach for the Definition of Computer-Supported Learning Process

Telmo Zarraonandia, Camino Fernández, Juan Manuel Dodero

Universidad Carlos III de Madrid
Departamento de Informática
Escuela Politécnica Superior
Av. Universidad 30 Leganes, Madrid, España 28911
{tzarraon, camino, dodero}@inf.uc3m.es

Abstract. The aim of this work is to bring together the traditional way of teaching and work using a computer-supported environment. This means increasing the flexibility of the learning processes application, giving the chance to instructors to introduce variations on runtime. Besides, learning processes are refined through their use, by making permanent the modifications which have shown to improve the learners' performance on the different learning objectives. We term this late modelling of learning processes. This paper describes the lifecycle of late modelling and proposes an architecture for implementing its runtime stages in the learning process described by means of the IMS Learning Design specification.

1 Introduction

When describing an educational process it is not always possible to know all its characteristics at design time. Many of them as, for instance, the ones related to synchronization and temporization of the activities cannot always be established before the proper execution of the learning process begins.

Regardless how carefully and precisely a learning process has been defined, its application to actual educational settings is not rigid, since it is very difficult to foresee all the potential reactions from learners. In practice, teachers take the learning process as a starting base, not to be followed blindly, and observe the evolution of the learners during its execution, introducing the appropriate adaptations in order to solve specific problems, reinforce the learning of some particular concepts and, more generally, guarantee the achievement of the original learning objectives. Furthermore, the adaptations proven to improve the original process results will be part of future applications. Due to the above, the learning process is refined through its use. We term this late modeling.

This work aims at increasing the degree of freedom of the teachers when applying a learning process on a computer-supported environment. Instructors will be provided with the possibility to introduce modifications in the learning process definition during its proper execution. The adaptive actions introduced could be evaluated against their original goal, measuring its influence on the learning objectives achievement and, accordingly, giving the teacher a chance to automatically include them in the original process. This way, instructors would imitate the way teachers work in real life: the gain obtained by the use of the process is kept within the process and, at the same time, is also used to refine it.

The rest of the paper is organized as follows. First, the late modelling lifecycle will be defined, describing the purpose and characteristics of each of its different stages. Next, the architecture of a system able to implement the runtime phases of the late modelling of IMS Learning Design [1] specified process will be outlined. Finally, some conclusions will be presented.

2 Late Modelling of Learning Process

The application of a learning process is in practice quite flexible as it is not possible to foresee all the potential reactions from the learners. Instructors take the learning process as a basis, and after observing the learners reactions, they may response providing extra examples, explanations to reinforce particular concepts, repeating activities, tuning the time-limits for completion of the assessments, etc. However, the more the instructors play the course, the less adaptations are required to be applied as the process is refined through its use. The experience gained from prior plays is comprised within the process definition and a wider range of learners' reactions response is captured. This means that the course model definition does not conclude at the design phase but rather when no more modifications are required to be applied. We term this process late modelling.

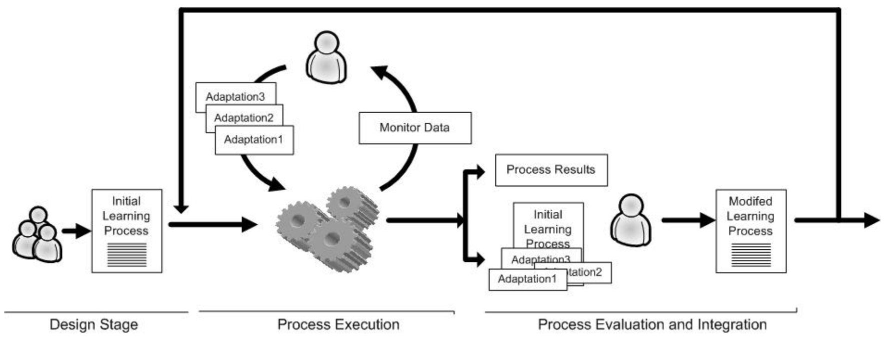

Fig. 1. Late modelling of learning process lifecycle

Figure 1 illustrates the different activities of the late modelling of a learning process carried out on a computer-supported environment. The process starts once an initial model of the course is been defined and its execution begins. Tutors observe learners interactions and introduce the appropriately tagged adaptations. The success of the applied adaptations is evaluated and once the process is finished, the learning objectives achievement will be measured. Based on that information a new version of the learning process could be generated, including the successful modifications introduced. This new version will go through the same cycle on its next plays until no more adaptations are required to be applied.

This section provides a description for each of the different activities that compose a late modelling process: to monitor the execution, to introduce adaptations, to evaluate the adaptations, to evaluate the process results, and finally, to integrate the adaptations.

2.1 Monitor the Execution

In order to detect potential problems and introduce the appropriate adaptive actions, it is fundamental for the instructors to be able to monitor the learner's interactions and progress during the learning process.

The more information tutors can obtain from the process execution, the better they will identify causes of problems during the learning process. For instance, if they can only retrieve information about the learner's score on the different activities, they may only be able to conclude that her performance is not being adequate. Otherwise, if they could retrieve information about which resources the learner has visited and how much time has she spent on each of them, they may be able to extract more accurate conclusions and produce appropriate recommendations and adaptations.

On the other hand, the comparison of information from the different learning process instances of the different participants allows the instructors to detect which problems are specific of a particular learner and which others are common to all of them. To facilitate this task, it is desirable to count on a system which allows the definition of watch points.

2.2 The Introduction of Adaptations

Based on the information retrieved from the monitoring activities, instructors will describe the process variations required to guarantee the process success.

Jacobson et al [2] defined variation points 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. Svahberg presented [3] 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 Approach [5], and provides the designer the possibility to bind the modifications during the compilation phase or even at runtime.

We can take these concepts into the adaptation of learning processes. Authors can describe the desired adaptations on auxiliary specification files that could be processed together with the original Unit of Learning (UoL) [1] and applied at runtime giving the user the feeling that they were included in 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 in a particular adaptation and keep that particular concern separated from the main UoL functionality and the rest of adaptations.

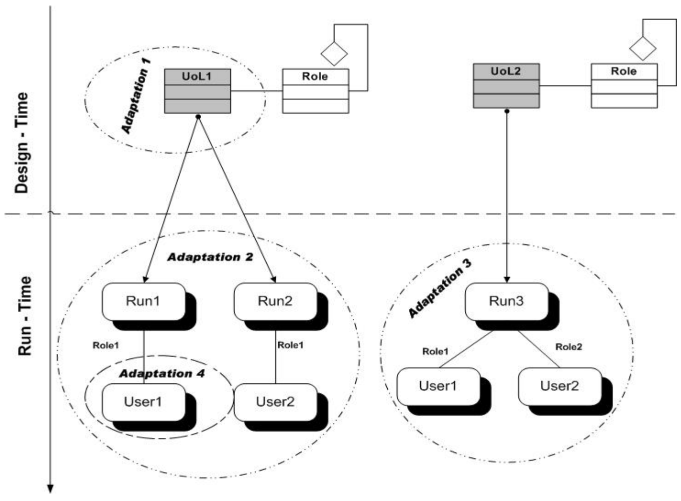

Fig. 2. Overview of different adaptation possibilities of a learning process

An overview of the process is shown in figure 2. 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).

In [4] authors defined adaptation pokes as descriptions of small modifications of some elements in a learning design process. The set of elements whose modification could be described by an adaptation poke were also defined. The authors also introduced the three different types of files which could be required to specify an adaptation poke: an adaptation command file (describing the adaptive actions), adaptation manifests files (containing the definition of new learning process elements), and resource files (corresponding to new content files). In the context of a late modelling process it may also be necessary to specify:

- Description of the adaptation poke objectives: The outcomes that instructors expect to obtain by introducing the poke.
- Description of the mechanism for evaluating the adaptation poke objectives: The way the grade of satisfaction of the adaptation poke objectives should be measured. This includes the definition of the moment when the evaluation must be performed.
- Condition of integration: a condition to determinate when the adaptation poke should be permanently integrated into the learning process.

Regarding the purpose of the adaptation, moment of introduction and possibility of integration, adaptation pokes can be classified into four different groups:

- Touch Pokes: These are specific variations whose introduction should not be considered during the rest of the process. They are always included at runtime and do not contain adaptation objectives or evaluation mechanism descriptions as they respond to an external requirement. For instance, when tutor need for any reason to set to 'completed' or 'incomplete' the status of a particular leaner activity, or due to the momentary unavailability of a required resource, tutors modify the time limit of an assessment activity.
- Readjustment Pokes: Here, the purpose is to readjust the learning process after an incidence external to the process nature occurs. They do not include adaptation objectives or evaluation mechanism, but their introduction must be taken into account when evaluating the process. They can be introduced in the process before or after the beginning of the execution. For example, the original course schedule may have been modified due to adverse atmospheric conditions. As a result, the available time for the completion of some of the activities has been decreased. Another example could be the modification of the original course program to cover a new subject in response to the students' interests.
- Corrective Pokes: A problem is detected during the learning process and some corrective actions must be introduced in order to accomplish the established learning objectives. It is necessary to describe the objectives of the adaptation as well as an evaluation mechanism. Due to its nature, their introduction only makes sense at runtime. Examples of corrective adaptations could be the introduction of complementary material, the adjustment of the time limit to complete a particular assessment, replacement of incorrect content, etc.
- Variation Pokes: They define a specific variation in the whole learning process. This means, they transform the original UoL to adequate it, for instance, to a different learner profile, context or agenda. They must be introduced before the execution of the learning process starts and they do not require objectives specification. This way, authors can create variation pokes for obtaining adequate versions of the UoL for distance courses, intensive on-campus instructions or just to remove the student's evaluations.

A corrective poke may become a variation poke for the next course execution. To illustrate this situation a whole process example will be shown: consider a course run during which instructors may have been required to introduce several corrective adaptations including complementary material. Once the process is finished instructors analyze the process data to find out the reason of the adaptations requirement. The analysis ends up with the conclusion that the learners profile assignment was incorrect: the learner's real profile did not match the expected one and most of the adaptations introduced aimed to cover knowledge gaps between the two of them. Designers decide to group the introduced corrective adaptations under a single variation poke to be applied to the original UoL in future instructions in order to adapt it to the new learner profile.

2.3 Evaluation of Adaptations

Once the adaptation has been introduced it is necessary to score it. This way, not only the grade of satisfaction of the adaptation objectives must be evaluated, but also its influence in other parts of the process must be considered. Evaluations can be specified using formulas composed of values retrieved by process monitoring mechanisms, direct tutor observation or from the leaner profile. Those values can refer to the student performance in the different process activities and learning objectives but also to the collaborative skills proven during the process, prior knowledge, etc. The moment when the evaluation should be performed must also be specified: once the learning process is finished, once a particular assessment activity is completed, etc.

Together with the evaluation formula, a list of learning objectives that can be influenced by the adaptation can be provided. Pokes with clashing related learning objectives should be thoroughly examined in order to detect possible correlations between their actions.

Note that the evaluation is not directly based on the learner's results but on comparing the expected consequences of the adaptation with the actual ones. Hence, the difficulty lies on the identification of what is a real consequence of the adaptation and what is not.

2.4 Process Evaluation

Once the learning process is finished, its results must be evaluated to identify strengths and weaknesses. This evaluation is mostly based on the information about the performance of the learners for each learning objective obtained once the process is finished. If most of the learners score low for a particular learning objective, designers may consider including complementary material, reviewing the pedagogical approach or reconsidering the calibration of the difficulty of the assessment activities. However, causes of low performance may also lay on external circumstances or incorrect learner profiles. It is necessary then, to establish the grade of reliability of the process results by comparing them other plays of the course data.

Following this idea, instructors can perform two different types of results data evaluation:

- Evaluate the overall results of all the participants of a particular course play: This gives instructors an idea of the success of the learning process definition for the current group of learners.
- Compare actual results with historical results: Historical data become more relevant the higher the number of plays accumulated, and it becomes a useful tool to identify weakness or improvements in our learning process. Average results before and after the introduction of a particularly important adaptation must be compared to gain an idea of its possible impact on the course success.

2.5 Integration

Once the process results have been analyzed, the integration phase takes place. This way, adaptive actions which have proved to lead to an improvement of the process become a permanent part of it. Instructors thoroughly examine all the corrective pokes evaluations selecting the ones to be integrated.

Conditions to establish the integration of the objective pokes can be specified to give the instructors the chance to automatize the process. These conditions will be based on threshold values for the adaptation evaluation results. Corrective pokes whose integration condition has been evaluated to true would be selected for its integration. The system will apply them to the original UoL definition following their introductory order. Each of the adaptation introductions is validated separately to facilitate the identification of dependencies with rejected adaptation in case of failure.

3 Implementation of Late Modelling in Learning Design Process

This section covers a proposed architecture for implementing the runtime phases of a late modelling of learning design specified process. The core of the architecture is a Learning Design Player able to interpret adaptation pokes descriptions and to introduce the specified modifications at runtime. A mechanism to guarantee the integrity of the modified UoLs must also be defined.

3.1 LD Player

A Learning Design Player (LD Player) is the program that interprets a UoL. It presents the different activities and resources to the involved roles and controls their interactions. In [4] authors outlined the structure of a LD Player capable of combining, both at design and runtime, the original UoL information with adaptations descriptions included in the adaptation pokes. The proposed structure (Fig. 3) follows object-oriented design principles, establishing a correspondence between the elements of the Learning Design specification and the class concept from an OO approach. It also made use of design patters and an Aspect Oriented Approach [5]. This allows a separate specification of the elements of the structure and the definition of the operations that can be applied to it. Two of the possible operations that could be implemented were the modification of the elements definition (Adaptor class) and the retrieval of information about their stage (ProgressWatcher class).

A set of commands were defined for specifying the modifications that could be carried out for each of the elements. Designers used those commands to create adaptation pokes which were included in the content package or uploaded to a running instance. The AdaptationReader generates the appropriate Adaptor object and passes it to the execution engine to perform the required adaptation.

Following the same approach, another set of commands was defined to specify the elements' characteristics whose value could be retrieved at runtime. Designers introduced the appropriate command at runtime indicating the element identifier and

UoL running instance desired to be observed. A ProgressWatcher instance was then generated and the appropriate values obtained.

The adaptive LD Player was implemented as an extension to the CopperCore IMS Learning Design engine [6] and can be used to implement the main stages of a late modelling of learning design specified process: the introduction of adaptations and the monitoring of the execution. The first one clearly match the adaptive LD Player operational way and the second one can be performed using monitor services implementations [1], complemented with ProgressWatcher actions.

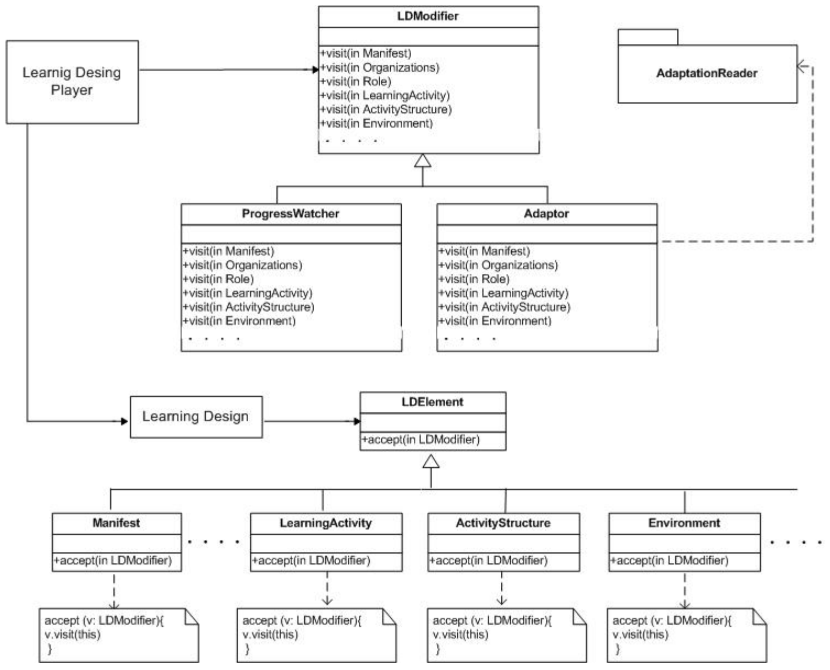

Fig. 3. LD Player Structure

3.2 Adaptation Validation

Some considerations must be taken into account to ensure that the adaptation by the LD Player previously described does not compromise the integrity of the original UoL. Every time a UoL is published in a particular player, a validation process is launched to guarantee its compliance with the IMS LD language definition and the availability of the referenced resources. Consequently, after the introduction of runtime adaptations, the same validation process should be repeated to ensure that the UoL definition remains valid.

An adaptation poke should be considered as a whole transaction and thereof its introduction should result on their complete application or invalidation. Some conflicts between adaptive actions within the same poke or between different pokes result in their application failure. In these cases, to maintain a copy of the UoL state before the adaptation to be restored can be enough to solve the problem.

However, some inconsistencies are harder to detect as the adaptive action does not cause failure. This type of validation could be solved by means of programming by codifying the characteristics and properties a valid Learning Design should accomplish. However, this is a hard task to undertake, and future changes or extensions on the LD specification would involve modifications on the program code. A more effective approach would consider the use of an ontology to capture the semantics of the Learning Design specifications to validate the consistency of the adapted UoL.

In [7, 8], the authors describe an IMS LD-based ontology which captures the semantics of the IMS LD Level A specification as well as the restrictions to be verified between the LD concepts. As this ontology model defines formally these restrictions, it is possible to use it for the detection of inconsistencies on adapted instances of a learning design, because the detection of inconsistencies will happen when these restrictions are not verified. The IMS LD ontology was implemented in Frame-based Logic (F-Logic) [9], and the FLORA-2 reasoner [10] was used to check the axioms of the ontology when the concepts instances were introduced.

The process for detecting the inconsistencies can be resumed as follows: first, an adaptation poke is introduced into an UoL instance, and, as a consequence, its LD description is changed; then, the ad-hoc translator is executed to transform the XML schema representation of the adapted learning design into the F-Logic description; and finally, the FLORA-2 reasoner is invoked to answer the queries associated to the axioms that must be verified.

4 Conclusions

This paper has introduced the concept of adaptation poke as the specification of small adaptive actions that can be applied, even at runtime, to a previously defined learning process. The adaptive actions introduced are evaluated against their original goal, measuring its influence and, accordingly, giving the instructor a chance to automatically include them in the original process. The process of progressive refinement of learning processes through their use is what we call late modelling.

An architecture of a Learning Design Player that provides the means to implement the runtime stages of the late modelling in Learning Design specified process has been described. The player was designed as an extension to the CopperCore runtime engine and implemented with the help of different design patterns and an Aspect Oriented Programming approach. Once the UoL has been adapted, it must be validated in order to guarantee its compliance with the IMS LD specification. For that purpose an ontology that captures the semantics of the elements of the Learning Design specification is utilized. Although at this moment the ontology is only able to validate changes in Level A elements, future versions of the ontology will be able to represent Level B increasing the validation capabilities of the system.

To facilitate the evaluation of both the introduced adaptations and the process results, an evaluation model which works on top of the learning process definition can be used. Work is being carried on to produce an XML notation for the evaluations specification. At the same time, the XML language may also be adequate enough for the description of the adaptation commands. The adaptation description can be increased with new elements in order to support the adaptation evaluations definition.

Future lines of work include the development of an application to aid in the late modelling process. The application will provide facilities for the authoring of process evaluations and adaptations. By using a GUI interface, designers will be able to select elements of an IMS LD specified process and connect them with evaluation profile elements. Templates to facilitate the adaptation definitions and new learning process components specification will also be provided. The application will communicate with a CopperCore engine increased with adaptation capabilities in order to directly introduce the described adaptations into running UoL instances. The retrieval of data to populate the evaluation profiles will also be possible by means of the ProgressWatcher implementation. This will simplify the process progress monitorization tasks.

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

References

1. IMS Global Learning Consortium. "IMS learning design information model, version 1.0 - final specification". Electronically available from http://www.imsglobal.org/learningdesign/ldv1p0/imsld infov1p0.html, 2003.
2. Jacobson, I. Griss, M. Johnson, P. "Software Reuse. Architecture, Process and Organization for Bussiness Success". AddisonWesley, ISBN: 0-201-92476-5. 1997.
3. Svahnberg, M. et al. "A Taxonomy of Variability Realization Techniques", Technical paper, ISSN: 1103-1581, Blekinge Institute of Technology, Sweden, 2002
4. T. Zarraonandia, J. M. Dodero, and C. Fernández, "Croscutting runtime adaptations of LD execution", Journal of Educational Technology and Society, volume 9, number 1, 2005
5. Marcus, A. et al. "An overview of aspect oriented programming". Kent State University, Dept. of Computer Science, 2001
6. Open Universiteit Nederland (2005) "CopperCore v2.2.2 release". Electronically available from http://coppercore.org/
7. M. Lama, E. Sánchez, R. Amorim, and X.A. Vila. "Semantic Description of the IMS Learning Design Specification". AIED-Workshop on Semantic Web technologies for E-Learning (SW-EL 05), Amsterdam, 2005.
8. R. Amorim, M. Lama, E. Sánchez, A. Riera, and X.A. Vila. "An Ontology to Describe Semantically the IMS Learning Design Specification". Journal of Educational Technology and Society, volume 9, number 1, 2005.
9. M. Kiefer, G. Lausen, and J. Wu. "Logical Foundations of Object-Oriented and Frame-Based Languages". Journal of ACM, 1995.
10. G. Yang, M. Kiefer, C. Zhao, and V. Chowdhary. "FLORA-2: Users' Manual (version 0.94)". http://flora.sourceforge.net/docs/floraManual.pdf.