# Delivering Specification-Based Learning Processes with Service-Oriented Architecture: A Process Translation Approach[*]

CHIEN-TSUN CHEN, YU CHIN CHENG, CHIN-YUN HSIEH AND TIEN-SONG HSU
*Department of Computer Science and Information Engineering*
*National Taipei University of Technology*
*Taipei, 106 Taiwan*

Service-oriented architecture (SOA) is increasingly popular for constructing eLearning systems. SOA encourages the separation of process from underlying services. The separation is also advocated by IMS Learning Design, an international standard for describing a learning process. Despite the apparent congruence and the recent proposals of several SOAs for eLearning, existing eLearning systems have yet to take full advantage of the process-service separation and are denied of a number of quality attributes promised. In this paper, we describe a new approach, which incorporates process translation and a three-layer SOA, for delivering specification-based eLearning processes. Specially, an eLearning course description specified in IMS Learning Design is translated into a process of Business Process Execution Language (BPEL). The translated BPEL process is then executed on a BPEL engine, which powers the process service layer of the SOA and keeps the learning processes separated from the underlying Web services. The architecture achieves extensibility, reusability, and ease of integration by allowing advanced eLearning standards to be incrementally implemented as a group of collaborating processes in BPEL. Translation rules from IMS Learning Design into BPEL are presented. The benefits of the proposed approach are investigated and compared with existing eLearning architectures.

*Keywords:* service-oriented architecture, business process execution language, web services, learning design, collaborative learning, eLearning

## 1. INTRODUCTION

The emergence of frameworks such as E-Learning Framework (ELF) [8], IMS Abstract Framework [12], and Open Knowledge Initiative (OKI) [16] marks a general trend for eLearning systems to move towards adopting service-oriented architecture (SOA). Under SOA, eLearning functionalities (*e.g.*, learning management, content management, assessment, portfolio, *etc.*) published as services can be discovered and composed − statically or dynamically − for delivering rich, flexible, adaptive, and personalized eLearning applications.

Recently, Business Process Execution Language (BPEL) [1] has become popular as a way to realize business processes by orchestrating services within SOA [17]. BPEL adds a process layer to the service stack for facilitating synchronous and asynchronous

communication among Web services and supports long running and stateful processes. By adopting BPEL, the underlying services are kept simple and self-contained and hence are more reusable. The benefits of the separation are also recognized by the eLearning community. A prominent example is IMS Learning Design (LD) [13], a widely referenced international eLearning specification. Once widely used, LD can raise the reuse level from one of content to one of pedagogy [23]. To this end, LD advocates the separation of a *learning process*, which describes the interaction among learners and tutors, from the *underlying services*, which provides learning management and collaborative services.

Despite the apparent congruence of BPEL and IMS LD on the advocacy of process-service separation, we are not aware of any existing eLearning system that truly fulfills the separation. It is still common for learning processes and underlying services to be located in the same architecture layer. For example, in ELF, the LD-related services are placed alongside LMS services in the same architecture layer even though the former are dependent on the latter [8].

This paper considers the implementation of LD under SOA centering on achieving the separation of processes and services. LD is a language for describing a learning process that *orchestrates* the activities conducted by roles played by learners and tutors. The activities center around a *unit of learning* (UoL) that includes *learning objects* (contents) and *learning services* (tool supports), *e.g.*, for facilitating interaction and collaboration. In a *run* of a UoL, learners interact with tutors and among themselves. Since a UoL can be completed by a learner in several runs, each of which potentially involving different participants, a run is stateful by definition. For example, individual learners' progress must be remembered by the system. The needs to orchestrate and remember place an LD process one level higher than the underlying services in the service stack.

In this paper, we propose a *SOA framework* with a *process translation* approach to implement standard-based eLearning systems with BPEL. In particular, the proposed eLearning SOA separates learning processes from implementation services. We design and develop an *LD-to-BPEL translator* which translates courses in LD into processes in BPEL. Once the translator is developed, it is possible to dynamically generate different learning processes at translation or run-time, providing flexibility to dynamically compose Web services for eLearning [11]. Besides, since essential services requested by LD such as learning state tracking and learning process monitoring are extracted as individual Web services, they can be reused by other applications to provide new functions. For example, it is possible to provide a Web service to record the history of learning state so that learner activities can be played back for analysis. Such a functional extension can be dynamically plugged into the framework in a streamlined manner. Thus, the functionality of the entire system can be extended without significantly interfering with the existing service elements.

The rest of this paper is organized as follows. Section 2 briefly introduces IMS LD and BPEL. In section 3, we present the proposed eLearning SOA. Section 4 shows the translation from IMS LD courses into BPEL processes. Related work is given in section 5. Finally, section 6 concludes this paper.

## 2. BACKGROUND

### 2.1 IMS Learning Design

IMS LD is an international eLearning standard for describing learning processes. The primary goal of LD is to "provide a generic and flexible language which supports pedagogical diversity and innovation while enabling exchange and interoperability of eLearning materials" [13]. The former is regarded as the *course design aspect* of LD while the later is viewed as the *course deployment aspect* of LD.

LD is an XML-based language and the design of a course is represented in a manifest file called imsmanifest.xml. Fig. 1 shows its static structure. In summary, LD borrows the theater metaphor to model the learning process. A course is organized as a tree, which is represented by a root node, namely, learning-design. A course can contain different learning scenario, called play, to fulfill a particular learner's need. Multiple scenarios can be carried out concurrently. Inside a play, an act controls the synchronization of the scenario. A scenario, in turn, can include a number of acts that are executed sequentially. Note that a scenario is essentially played by two types of actors, that is, learners and staffs (*e.g.*, instructors and assistants). LD has built-in constructs to specify activities of learners and staffs, respectively.
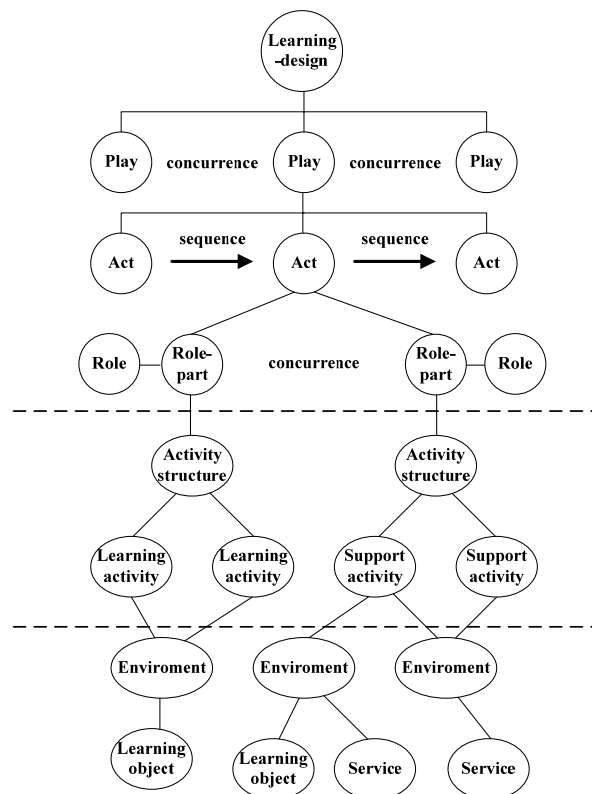


Fig. 1. IMS learning design structure.

An act is composed of learning activity, support activity, activity structure, role, and role-part. A learning activity is the basic learning unit of learners. It is usually used to reference to a physical learning material, including plain text, hypertext, graphics, streaming media, and so on. When a learning activity is bound to a collaboration service such as chat room, videoconferencing, bulletin board, and learning game, the learners in the same activity can collaborate with each other through the service. This binding is accomplished through the environment tag for the sake of reusability. For example, a particular service, said Microsoft NetMeeting, is used by a number of activities in a course. The course designer can link these activities to the same environment and bind the environment to NetMeeting. To replace NetMeeting with Skype, all the course designer has to do is to change the corresponding environment tag. Note that the functionality of a support activity is similar to a learning activity except that a support activity is performed by staffs.

An activity structure is a container of activities. It is used to organize and sequence activities. For instance, activities must be conducted sequentially if the structure type of the encompassed activity structure is *sequential*. A role-part binds a specific role to an activity or activity structure. Participants playing the roles are assigned to the activity.

To benefit most from LD, developers must understand the separation of the collaboration logic specified in an LD course and the functionality underlying the constituent learning materials. This course development metaphor is an instance of the "Separation of Material Preparation and Integration" pattern for eLearning course authoring [3], where LD provides an integration mechanism and a variety of learning contents as well as services act as material preparation methods. By separating integration from implementation, the responsibilities of course designers and learning content developers are clearly defined. The course designers focus on structuring and sequencing learning activities in a course to fulfill a particular learning goal whereas the learning content developers are responsible for implementing learning materials. As can be seen later, this style of course development is consistent with BPEL and Web services development.

## 2.2 BPEL

BPEL describes a sequence of activities and collaboration logic inside a business process [1, 10]. The actual functionalities of the activities are implemented by Web services. In this regard, a BPEL process is similar to an LD course. Both express the sequence and collaboration of a number of activities and the particular implementation of the activities is carried out by the underlying services.

In BPEL, actions or operations are executed through *activities* (Note that the term "activity" is used in LD and BPEL with different meanings). There are two categories of activities in BPEL: *basic activities* and *structured activities*. Basic activities include activities of invoking a Web service, receiving and replying messages, assigning a new value to a variable, throwing exceptions, waiting, and empty (doing noting). Structured activities, including sequence, switch, while, flow, control links, and pick are mainly used to control process execution sequence.

When a number of processes executed in a BPEL engine communicate with external Web services (called partners in BPEL), incoming events must be redirected to the right process. The BPEL correlation set mechanism meets the need.

In summary, BPEL is chosen to implement LD mainly because:

- BPEL is an open standard and its processes can be executed in different environments and exchanged among organizations,
- BPEL realizes the separation by expressing a business process's sequence and collaboration logic while leaving the underlying services to provide the business functions,
- BPEL provides synchronous (client-server) and asynchronous (peer-to-peer) invocations of the underlying web services, and
- BPEL tracks learner's progress through supporting long running and stateful process.

## 3. AN ELEARNING SOA

Successfully applying LD requires the support of client side *course authoring* and server side *course execution*. In this paper, we assume that LD-compliant courses have been produced by other authoring tools such as COLLAGE [5], CopperAuthor [6], elive LD Suite [9], and RELOAD [18]. Therefore, we focus on the development of a course execution environment.

A course based on LD is conducted within the execution environment provided by a *learning management system* (LMS) which, in turn, provides services for managing users and courses, controlling content repository access, and facilitating collaboration and adaptation. In the context of LD, an LMS is responsible for retrieving user profiles, binding a user to a particular role, retrieving course contents from the repository, loading the course, starting the learning process, invoking collaboration services, and tracking learning state. Thus, an SOA for eLearning must make provisions for these services to be created.

A service-oriented eLearning framework is a set of components and services based on which a variety of applications can be built [12]. Based on this definition, we propose a service-oriented architecture for eLearning as illustrated in Fig. 2. This architecture reflects the current trend in SOA that clearly separates process services from partner services.

### 3.1 The Partner Service Layer

The partner service layer provides reusable *building blocks* that can be used to compose a variety of eLearning applications. Web services in this layer are classified into *application independent services* and *application dependent services*. For instance, services such as logging, authentication, notification, and content retrieval are application independent. External collaboration applications such as chat room and videoconferencing are also considered application independent. Conversely, services directly related to a particular specification are application dependent, including, for example, services of LD manifest file analysis and LD course state maintenance.

It should be noted that external collaboration applications may not have Web service APIs. In this situation, we need to develop *proxy services* that export Web service APIs and forward requests to the applications. The proxy services also act as *adapters* between their clients (*i.e.*, Web services in the process layer) and their adaptees (*i.e.*, the
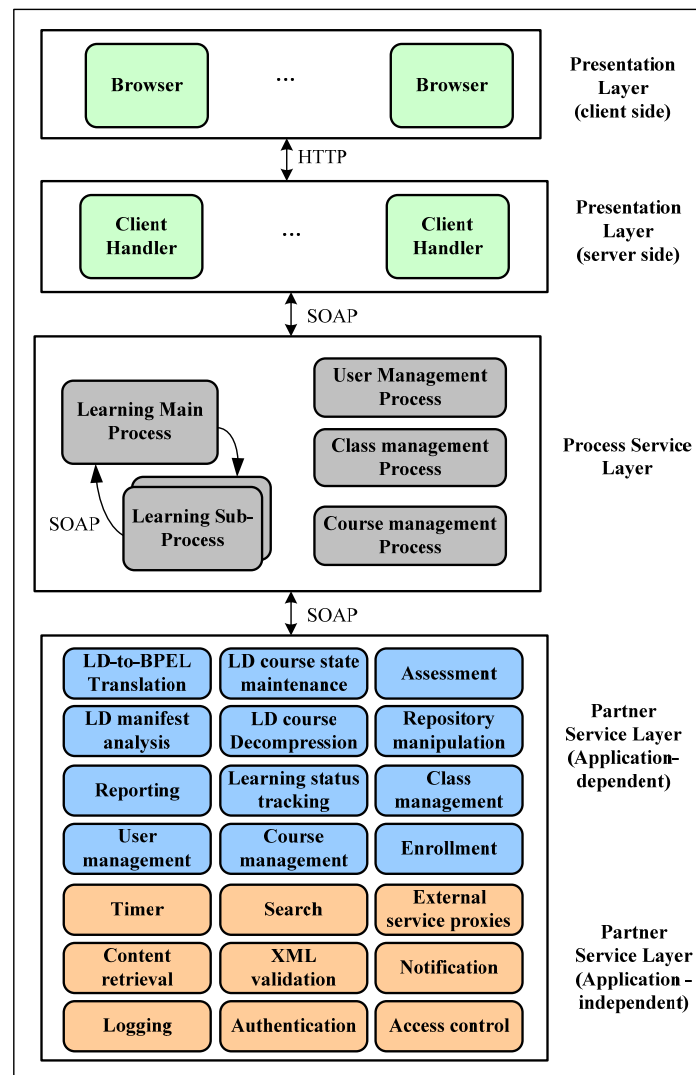
Fig. 2. The proposed service-oriented architecture overview.

applications). This provides a standard way in SOA to reuse existing applications in a coarse-grained manner.

## 3.2 The Process Service Layer

The process service layer delivers *end-to-end* eLearning functions by orchestrating the underlying partner services. This layer accomplishes an *eLearning federation* that contains a variety of applications running on different platforms. The orchestration and workflow logic are encapsulated by the processes and can be modified and extended in a standardized manner.

In our implementation, the LMS contains several pre-defined BPEL processes which make use of the partner services to provide LMS functionalities. For example, an "upload course" use case is realized by a BPEL process that orchestrates partner servers of course package decompression, manifest analysis, LD-to-BPEL translation, and content repository manipulation. These services are declared in the partnerLinks construct of BPEL and can be sequentially or concurrently invoked. Compared to general purpose languages such as Java and C#, controlling the execution of partner services is relatively easy in BPEL.

After uploading, the LD course is translated into a main process and one or more sub-processes in BPEL. The translation process and the functionalities of the main and sub-processes will be presented in section 4.

### 3.3 The Presentation Layer

The presentation layer separates the presentation from the process layer. In our implementation, a presentation layer component, Client Handler, is responsible for handling the participant's requests issued through a browser and translating the requests into SOAP messages for invoking the corresponding Web services. The Client Handler is also in charge of presenting user interfaces to the browser with XHTML and asynchronous JavaScript and XML (AJAX) technologies.

## 4. TRANSLATING IMS LD INTO BPEL

### 4.1 A Course Example

To facilitate the discussion of LD-to-BPEL translation, we use an English grammar course to illustrate a collaborative learning process in LD. The course will be translated into BPEL processes and required services will be implemented as Web services. The detailed translation steps will be discussed in the following. A complete example illustrating results of the translation is provided at http://pl.csie.ntut.edu.tw/~ctchen/ld-bpel.zip.

Fig. 3 (a) shows the static course structure which has exactly one play in which the scenario of learning is by way of reading and discussion. In the grammar course of Fig. 3 (a), there are two roles, the student and the instructor. The student can choose between the activities to browse course materials or to ask questions if the instructor is present. The responsibility of the instructor is to answer the student's questions via an online chat room. Figs. 3 (b) and (c) shows the course content seen by the student and the instructor, respectively. Thus, when people join the grammar course, the activities they perform are different according to the roles they play. In Fig. 3 (a), if the student selects the reading activity, she/he then must go through the units sequentially.

To translate an LD course into the equivalent BPEL processes, a number of translation rules are established in sections 4.2 and 4.3. Specifically, an LD model tree is built from an LD course. Then, the translator performs a breadth first traversal and translates each LD tag into its equivalent BPEL construct according to the rules. The translator can translate any LD-compliant course into BPEL processes because the translation is rule-based rather than hard-coded.
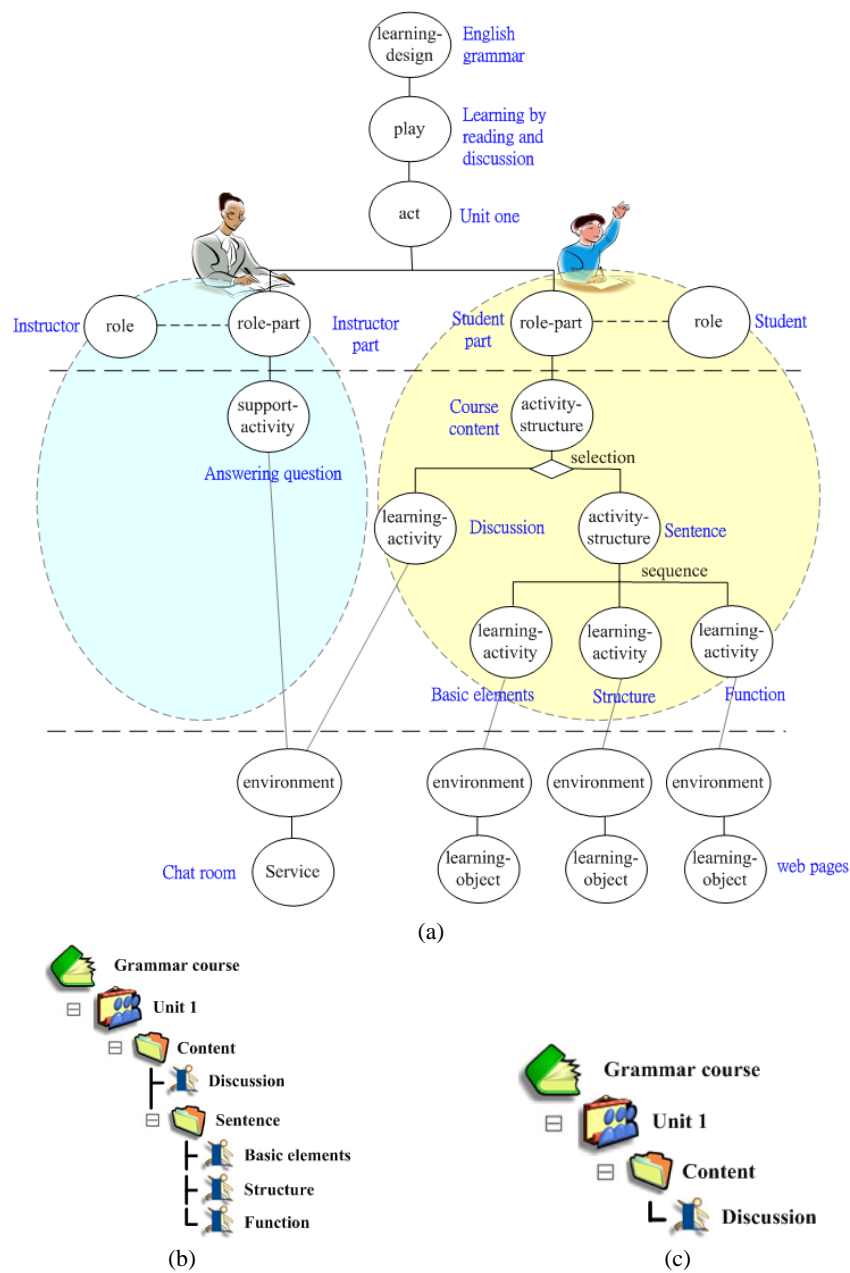
Fig. 3. (a) LD representation of the English grammar course, and the course as seen by (b) the student and (c) the instructor.

Information in LD tags of a course is mapped into a main process and a number of sub-processes. The main process sets up the course environment and awaits learning participants to join the process, upon which a sub-process is created. Thus, play, act, role-part that are applicable to all roles are mapped to BPEL constructs in the main process;

tags pertaining to the activities performed by the individual roles are mapped into constructs in a sub-process. For the English grammar course, a main process and two sub-processes are created. The instructor and the student share the same main process. However, participants interact with different sub-processes after they are bound to their particular roles.

A BPEL engine can host multiple concurrent processes. Since multiple participants are present and each participant interacts with the main process and a sub-process, an identification field is used in the message to correlate the participant's messages to its intended process. This feature is implemented with the BPEL's correlation set. Our implementation uses a group ID and course ID for correlating messages intended for a main process. For a sub-process, an extra participant ID is used. Note that the group ID is used to distinguish the groups of participants that engage in the same learning scenario. For example, the role constraint can specify a course to have exactly one instructor role and two to four student roles. When the second instructor role logins to the system, he or she is assigned to the second learning group.

### 4.2 Translation of the Main Process

The main process describes the behavior of a single play by using BPEL's sequence construct to model acts in a play since their execution is sequential. Before an act can be activated, the role constraint must be satisfied. Thus, we use BPEL's flow activity to concurrently monitor the constraint for each role.

### 4.2.1 Play

Fig. 4 (a) shows the translation of an LD play tag into a BPEL Play scope. (A BPEL scope is a block structure used to compose multiple BPEL constructs into a single unit.) A play contains at least one act. Each act is translated into an Act scope. Since the acts are performed sequentially, they are placed inside a BPEL sequence construct. Note that there are additional scopes preceding and following the Act scopes. Before users can participate in a course, the course must be initialized first (represented by the Course Initialization scope.) After all acts are completed, the play is set to complete (represented by the Play Completion scope). Finally, the entire course is completed (represented by the Course Completion scope).

### 4.2.2 Act

Fig. 4 (b) shows the translation of an LD act into a BPEL Act scope, which contains two sequentially executed scopes: Role-Part Handling scope and Act Completion scope. The Role-Part Handling scope implements the role constraint and creates a sub-process for each participant. Its detailed implementation is discussed later. The purpose of the Act Completion scope is to complete the act when all participants finish their assigned activities. In the example, the English grammar course has one act. Thus, when the act is completed, the entire play is completed as well.
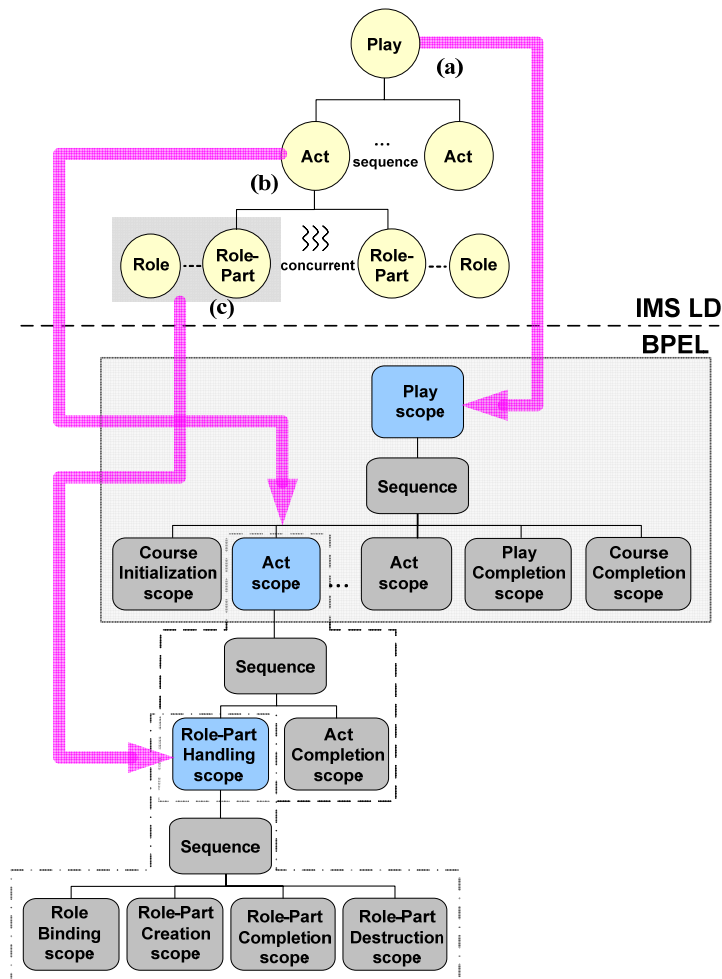
Fig. 4. Translation of the main process. The LD tags are mapped into the BPEL construct: (a) play tag, (b) act tag, and (c) role and role-part tags.

### 4.2.3 Role and role-part

Fig. 4 (c) shows the translation of a role and a role-part into a BPEL Role-Part Handling scope. The role tag defines all possible roles taken by participants in a particular act and the role-part associates each role to an activity. Its BPEL translation includes four sequentially executed scopes:

- *Role Binding*. Binds a participant to a role defined in the role tag by using a BPEL flow construct, which concurrently waits for a pre-specified number of participants to bind to each type of role. Fig. 5 shows the course fragment representing the role constraint of the English grammar course, in which exactly one instructor and two to four students are required for activating the act. Depending on the implementation, role

```
<imsld:roles identifier="LD-Grammar-Roles">
    <imsld:staff identifier="Instructor" min-persons="1" max-persons="1"/>
    <imsld:learner identifier="Student" min-persons="2" max-persons="4"/>
</imsld:roles>

<imsld:act identifier="ACT_Unit1">
    <imsld:role-part identifier="RP_Instructor">
        <imsld:role-ref ref="Instructor"/>
        <imsld:support-activity-ref ref="SA_Discussion"/>
    </imsld:role-part>
    <imsld:role-part identifier="RP_Student">
        <imsld:role-ref ref="Student" />
        <imsld:activity-structure-ref ref="AS_Content"/>
    </imsld:role-part>
</imsld:act>
```
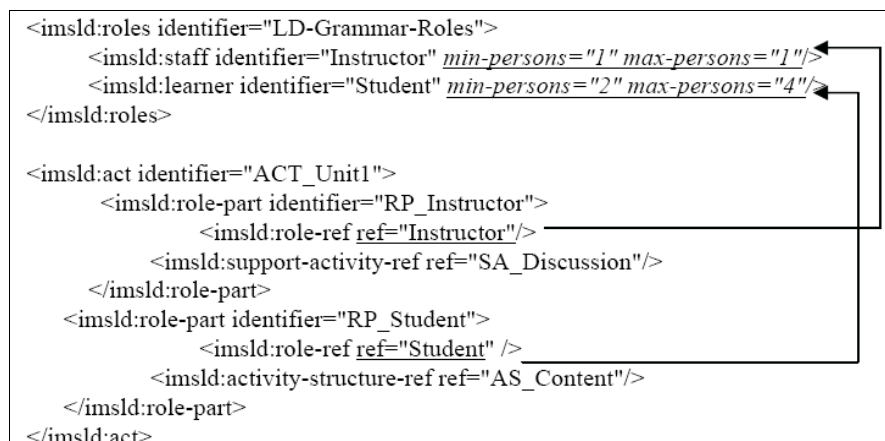
Fig. 5. LD constraints that bind participants to roles for the English grammar course.

binding can be determined by LMS automatically according to user profiles and course registration records at course loading time. Alternatively, participants are allowed to manually choose the role upon entering an act.

- *Role-Part Creation*. The structure of the sub-process is determined by the structure of the associated activity. When each participant is bound to a role, a sub-process is instantiated.
- *Role-Part Completion*. The main process waits for the completion of all sub-processes.
- *Role-Part Destruction*. When all sub-processes are completed, the main process terminates each sub process.

## 4.3 Translation of the Sub-Process

The sub-process describes the behavior after a participant has been bound to a role. The generation of a sub-process includes the translation of learning-activity, supporting-activity and activity-structure.

### 4.3.1 Activity structure

An activity-structure has two primary features. From the structural aspect, it creates a tree-like organization of course materials. From the behavioral aspect, it supports two course navigation methods: selection and sequence.

Upon entering an activity structure, a participant can repeatedly take one of the following actions: (1) loading an activity or another activity structure that is part of the current activity structure, (2) completing an activity, (3) leaving an activity, and (4) leaving the current activity structure.

Fig. 6 (a) depicts the BPEL implementation of an activity-structure. We use a BPEL while construct to model the repetitive behavior. Then, a BPEL pick construct is used to receive messages sent by the participants and to determine which one of the above four actions is to be taken. These actions are represented by a Cluster scope, an Activity Completion scope, an Activity Exit scope, and a Structure Exit scope, respectively.
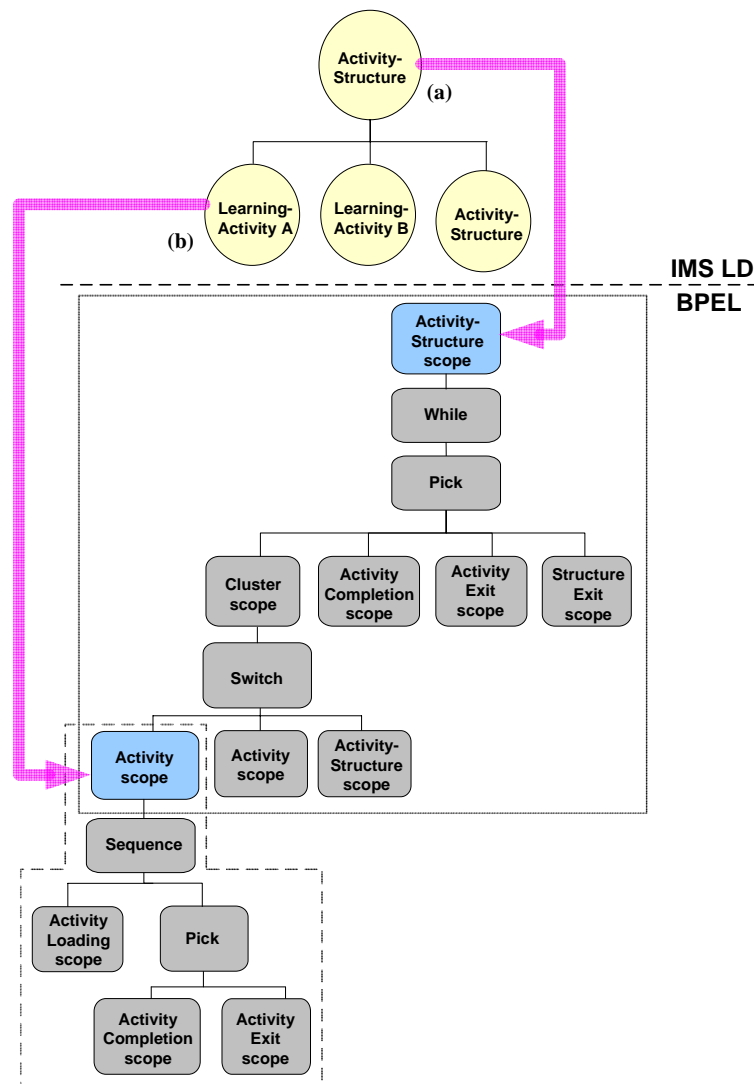
Fig. 6. Translation of the sub-process. The LD tags are mapped into the BPEL construct: (a) activity-structure tag and (b) learning-activity tag.

The translation of nested activity structures involves non-trivial implementation work. To simplify the implementation, we adopt the *cluster* concept from IMS Simple Sequencing specification [14]. Briefly, a cluster includes an activity structure as the root node and its immediate constituent activities or activity structures as the child nodes. All of the above four actions are applied to an *active cluster* which represents the current activity or activity structure taken by a participant.

**4.3.2 Learning activity and supporting activity**

Learning-activity is an atomic learning unit linking to a learning material. Regardless of its structure, a course must contain a number of learning-activity constructs. Fig. 6 (b) shows the BPEL translation consisting of two sequential scopes.

- *Activity Loading*. This is the operation that retrieves the URL of course materials bound to an activity and returns it to a participant. As a result, the participant can retrieve the course materials, which can be a static web page, a videoconferencing service, a simulation game, *etc*. Fig. 7 shows the BPEL code fragment for implementing the Activity Loading scope. First, a participant's request (represented by the activityRequest variable) for loading a particular activity is received. Then, the LMS service is invoked by sending the requested activity ID. In return, URL of the activity is received (in the URLReturn variable.) Finally, the URL is redirected to the participant's browser.
- *Pick*. Once an activity is started, the participant can go on to complete the activity or leave the activity as represented by the ActivityCompletion scope and ActivityExit scope, respectively. These two scopes are placed in a BPEL pick construct so that at any time, exactly one of them is selected as desired. Note that the completion of an activity can be triggered manually by the participant or automatically if the activity's completion is time-limited [13].

```
<scope name="ActivityLoading">
    <sequence>
        <receive partnerLink="Participant" createInstance="yes"
              portType="tns:CoursePT"
              operation="loadActivity" variable="activityRequest">
         </receive>

        <invoke partnerLink="LMSServices" portType="ns1:LMSServices"
              operation="lookupActivity" inputVariable="activityRequest"
              outputVariable="URLReturn"/>

        <reply partnerLink="Participant" portType="tns:CoursePT"
              operation="loadActivity" variable="URLReturn"/>
    </sequence>
</scope>
```

Fig. 7. BPEL constructs of the activity loading scope.

## 5. RELATED WORK

Before SOA becomes a popular technology, only a few LD implementations based on application-centric architectures were available [4, 7, 15]. Recently, several eLearning SOAs have been proposed [8, 12, 16] where a number of eLearning services have been defined. However, in these architectures the process layer is not separated. In addition, until now little SOA implementation experience concerning LD or other specifications has been reported in the literature. Our work bridges a gap between SOA concepts and SOA realizations in specification-based eLearning.

### 5.1 CopperCore

Until now, publicly available LD systems are still rare. One of the best known LD systems is CopperCore [7, 15]. The design of CopperCore includes two main architectural components: an LD engine and an LD player. The separation also allows third-party LD players (*e.g.*, SLeD player [15]) to interoperate with CopperCore.

CopperCore is implemented with Java 2 Enterprise Edition (J2EE) architecture. Originally, EJB facades are provided so that CopperCore clients can make use of native Java call to invoke the session beans. Recently, CopperCore introduced CopperCore Service Integration [21] and exported a number of J2EE API's as Web services. As a result, CopperCore has evolved toward the SOA. However, the evolutionary approach has a key limitation so far as SOA is concerned. Since the application-centric architecture of CopperCore is largely preserved without being re-organized according to the principles of SOA, the entire LD service can only be treated as a large-grained stateful service. The fine-grained services that constitute the LD service will not be reusable in other application contexts.

In our architecture, the responsibilities of an LD system are reorganized. Compared to CopperCore and SLeD, the BPEL processes and the Client Handler constitute an LD payer. The dynamic aspects of course run, user, and role managements are extracted into BPEL processes, which reduce the state management requirements of LD-related partner services. In our implementation, all fined-grained partner services are stateless except the course state management service.

### 5.2 Existing eLearning SOAs

Several eLearning frameworks based on SOA have been proposed by international organizations, including E-Learning Framework (ELF) [8], IMS Abstract Framework [12], and Open Knowledge Initiative (OKI) [16]. These frameworks allow different eLearning reference models to be built by defining a variety of eLearning services and interfaces. However, it seems that current practices still primarily focus on the identification and specification of required eLearning services. A separated process layer is not yet defined in these frameworks. Our approach can be regarded as a bridge between existing eLearning frameworks and the state-or-the-art SOA. Essentially, the proposed approach differs from the existing eLearning SOAs in three ways: (1) learning processes are separated from underlying services, (2) specification-based learning processes are implemented as BPEL processes with a process translator, and (3) learning management functions that are traditionally hard-coded by an LMS are implemented via BPEL processes with the help of underlying services.

### 5.3 Other Process-Oriented Approaches

To support collaborative, adaptive, complex, and dynamic eLearning processes, researchers have suggested that there is a need for *learning process composition and execution languages* (LPCEL) [19, 20]. Compared to LPCEL, which tries to combine learning process description with execution in a single language, our work separates these two concepts and respectively uses LD and BPEL for learning process description and exe-

cution. While LPCEL proposes a new language to model and execute learning processes, our work reflects the mainstream approach where learning process description is separated from execution. Furthermore, regardless of the difference, the translation approach proposed in this paper is also applicable to LPCEL. Once LPCEL is publicly available, it is possible to design a LPCEL-to-BPEL translator so that LPCEL processes can be executed on a BPEL engine. By so doing, we can leverage the benefits of BPEL across application domains.

In general, eLearning processes can be seen as workflow processes and methods have been proposed to use workflow engine to execute eLearning processes [2, 22]. In this regard, using a standard workflow language or BPEL to describe and execute eLearning processes is merely a subject of language choice. However, current work on applying workflow engine to execute eLearning processes does not provide a solution to incorporate a variety of eLearning requirements. For example, is it possible to implement different kinds of LMS functionalities with workflow engines? Does the workflow engine solution support the staged implementation of a variety of learning needs? In addition, existing workflow solutions do not consider adopting standard eLearning specification such as LD in their systems. These issues are discussed and presented in this paper.

## 6. CONCLUSION

SOA is regarded as one of the most promising candidates for developing next-generation eLearning systems. However, the realization requires significant efforts from both eLearning and software engineering communities. In this paper we have shown that advanced eLearning standards can be implemented as BPEL processes with supporting eLearning Web services. Once each eLearning standard is individually constructed as Web services, it is possible to integrate these standards into a larger process to represent more complex eLearning activities. In this way, we can take the piecemeal growth approach to developing eLearning systems and applications.

For future work, we see a need to develop open source Web services for commonly used eLearning services so that SOA eLearning systems can be constructed by reusing the developed services. We are also using BPEL to model more eLearning specifications and to establish an engineering way to streamline the modeling process.

## REFERENCES

1. Business Process Execution Language for Web Services version 1.1, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.
2. M. Cesarini, M. Monga, and R. Tedesco, "Carrying on the eLearning process with a workflow management engine," in *Proceedings of the ACM Symposium on Applied Computing*, 2004, pp. 940-945.
3. C. T. Chen and Y. C. Cheng, "A pattern language for personal authoring in eLearning," published in the *11th Conference on Pattern Languages of Programs*, 2004.
4. M. C. Chen, C. T Chen, Y. C. Cheng, and C. Y. Hsieh, "On the development and implementation of a sequencing engine for IMS learning design specification," in *Proceedings of the 5th IEEE International Conference on Advanced Learning Tech-*

*nologies*, 2005, pp. 636-640.

5.  COLLAGE, http://gsic.tel.uva.es/collage.

6.  CopperAuthor, http://sourceforge.net/projects/copperauthor/.

7.  CopperCore, http://coppercore.sourceforge.net/.

8.  ELF, The E-Learning Framework Site, http://www.elframework.org/.

9.  elive LD suite, http://www.elive-ld.com/content/e56/e61/index_eng.html.

10. T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall PTR, Upper Saddle River, New Jersey, 2005.

11. M. Fluegge, N. P. Tizzo, I. J. G. Santos, and E. R. M. Madeira, "Challenges and techniques on the road to dynamically compose web services," in *Proceedings of the 6th International Conference on Web Engineering*, 2006. pp. 40-47.

12. IMS Abstract Framework: White Paper, version 1.0, http://www.imsglobal.org.

13. IMS Learning Design Specification, http://www.imsglobal.org.

14. IMS Simple Sequencing Specification, http://www.imsglobal.org.

15. R. Koper and C. Tattersall, *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training*, Springer, Berlin, Heidelberg, 2005.

16. OKI – Open Knowledge Initiative, http://www.okiproject.org/.

17. J. Pasley, "How BPEL and SOA are changing web services development," *IEEE Internet Computing*, Vol. 9, 2005, pp. 60-67.

18. RELOAD Project http://www.reload.ac.uk/.

19. J. Torres, J. M. Dodero, and C. Padrón, "A framework based on web services composition for the adaptability of complex and dynamic learning processes," *IEEE Learning Technology Newsletter*, Vol. 6, 2004, pp. 7-11.

20. J. Torres, J. M. Dodero, I. Aedo, and T. Zarraonandia, "An architectural framework for composition and execution of complex learning process," in *Proceedings of the 5th IEEE International Conference on Advanced Learning Technologies*, 2005, pp. 143-147.

21. H. Vogten, H. Martens, R. Nadolski, C. Tattersall, P. van Rosmalen, and R. Koper, "CopperCore service integration – Integrating IMS learning design and IMS question and test interoperability," in *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies*, 2006, pp. 378-382.

22. K. Weining, L. Junzhou, and Z. Tiantian, "A workflow based eLearning architecture in service environment," in *Proceedings of the 5th International Conference on Computer and Information Technology*, 2005, pp. 1026-1032.

23. M. Weller, A. Little, P. McAndrew, and W. Woods, "Learning design, generic service descriptions and universal acid," *Educational Technology and Society*, Vol. 9, 2006, pp. 138-145.

**Chien-Tsun Chen (陳建村)** is a Ph.D. candidate at the Department of Computer Science and Information Engineering, the National Taipei University of Technology, Taiwan. Prior to commencing his graduate study, he served as the CTO of CanThink Inc., which provides e-learning software solutions and services. His research interests include software architecture, pattern languages, and exception handling. He has actively contributed to

several successful commercial and open source software projects. Mr. Chen is a member of the Software Engineering Association of Taiwan (SEAT).

**Yu Chin Cheng (鄭有進)** received the M.S.E. degree from the Johns Hopkins University in 1990 and the Ph.D. degree from the University of Oklahoma, in 1993, both in Computer Science. He is currently a Professor at the Department of Computer Science and Information Engineering of the National Taipei University of Technology, Taiwan, where he served as the department chairperson from 2002 to 2004. Dr. Cheng teaches object-oriented programming, object-oriented analysis and design, and software architecture. His research interests include pattern languages for software design, software architecture and image analysis. He is a member of the IEEE Computer Society, the ACM, and the Software Engineering Association of Taiwan (SEAT).

**Chin-Yun Hsieh (謝金雲)** received his M.S. and Ph.D. degrees from the University of Mississippi and the University of Oklahoma, respectively, both in Computer Science. Dr. Hsieh held several positions at the National Taipei University of Technology, including the director of the computing center, the chairperson of the Electronic Engineering Department, and the director of the Library. His research interests include object-oriented software engineering, programming language theory and distributed systems. Dr. Hsieh is a member of the Software Engineering Association of Taiwan (SEAT).

**Tien-Song Hsu (徐天送)** is a Ph.D. student at the Department of Computer Science and Information Engineering, the National Taipei University of Technology, Taiwan, where he received the M.S. degree. His research interests include service-oriented architecture, eLearning, and software development. Mr. Hsu is a student member of the Software Engineering Association of Taiwan (SEAT).