CHAPTER IV
ARCHITECTURAL DESIGN OF LMS
4.1 INTRODUCTION

Software architecture for LMS not only depicts a higher-level description of the system, it is also useful for checking conformance of the developing LMS with learning technology standards. It enables us to check this standard conformance early in the design phase, instead of doing it late in the implementation phase [discussed in section 2.2.2]. This chapter introduces a new methodology for software architectural design, which is verifiable for checking conformance with LTSA standard (Sengupta & Dasgupta, 2015). The core idea behind an AD of LMS is to provide a high-level design of a system for authoring learning contents, designing and constructing courses, conducting online teaching and learning activities and storing content, course structures and learners' record. The AD of LMS models the system's behavior with help of the identification of major components and their relationships. It also favors the subsequent detailed design phases in identification of the subsystems and reusable components.

4.2 USING ARCHITECTURAL DESIGN FOR LTSA-CONFORMANCE

From the SE perspective, it is important that an LMS must conform to an accredited standard to ensure quality. IEEE LTSC group has introduced LTSA (Farance & Tonkel, 2003), which sets a pedagogically neutral, content-neutral, culturally neutral and platform-neutral standard for architecture of a wide range of systems related to learning technology. This standard provides a framework for understanding learning technology systems (like LMS) and promotes interoperability and portability by identifying critical system interfaces. It also specifies a checklist, known as conformance rules (Farance & Tonkel, 2003), which must be satisfied by an implementing LMS to claim conformance with LTSA. This set of rules includes identification of different entities and flows within the system.

Unfortunately, though LMS development has been taking place for many years, they are still designed without any consideration on LTSA architectural standard. This is because LTSA conformance rules are tested on the implemented product via the pro-forma implementation conformance statement (ICS), not on any early design artifact. Moreover, the design and development of contemporary LMS, is
primarily focused on implementing as many functionalities as possible, instead of looking at the quality of the features. As a result, they often produce inefficient systems or systems with poor software quality (Avgeriou, 2003). Certain quality features such as performance, availability, portability, and usability can be easily achieved if the design of the system conforms to the recommendation of Learning Technology Systems Architecture (LTSA) standard. Since quality attributes in a system depend profoundly on its architecture, methodology related to specifying and evaluating software architecture of LMS is always an important research area. However, if we check the conformity at an early stage of development like software design phase, it would understandably add many advantages: i) a non-conforming design could be altered for suitable compliance before starting any implementation ii) if the design follows any formal modeling, then the conformance checking process could be automated and iii) the conformance verification could be completed with less cost, time, and effort. Since LTSA framework specifies an abstract (high level) description about the system, intuitively a high-level design of the software like AD is more suitable for the conformance checking than a detailed design.

This work proposes a methodology to accomplish the conformance checking of AD with LTSA. In our approach, we introduce a new meta-model for UML based on Acme style to design the LMS architecture and exhibit how it could be tested for conformance with LTSA.

4.3 PROPOSED METHODOLOGY FOR CHECKING LTSA-CONFORMANCE OF ARCHITECTURAL DESIGN
are represented more formally by Conceptual Graph representation for the verification method. The proposed method tries to find out a design pattern from the AD that satisfies a particular conformance rule, given as input. The verification process also introduces a goodness measure of the conformance.

4.3.1 Proposed framework

Figure 4.1: Framework for architectural design and verification

The proposed methodology for LTSA-conformance checking integrates different established methods of SE. Figure 4.1 shows the framework of the proposed methodology that depicts how different methods are inter-related within the methodology. Designing the architecture from the NL written requirements specification requires support from the domain knowledge, which is represented by the defined ontology. The resultant AD is a design artifact, which is prepared with a new style of AD. This style, inspired by the work of Herbert et al. (1999), introduces new stereotypes for UML component diagram. Although current UML version can represent significant structural aspects of a system for software architecture, it still lacks in features to specify software architecture sufficiently formally to be the subject of logical analysis, proofs of correctness, and automated reasoning (Mattsson et al., 2008 and Sarkar et al., 2012). Our proposed stereotypes are based on the graphical equivalence of Acme style for AD. The reason of selecting Acme among the other ADLs is for being a simple language for the essential elements of AD that supports natural extensions for complex architectural features (Garlan et al., 2010).

The design is then tested for conformance with the LTSA specification. Since the proposed verification method applies an algorithmic approach, the NL-written...
conformance rules must be stated in some formal way. Therefore, the LTSA conformance rules are represented by conceptual graph diagrams (with equivalent lambda expression), which are then used for the verification phase. In addition to conceptual graph and AD, additional information about relationships between the domain-components is represented by the ontology. In the following subsections, we discuss about the different components of the framework in details.

4.3.1.1 Conceptual Graph for LTSA

4.3.1.1.1 LTSA layer I

LTSA layer I
LTSA layer II

This layer deals with Human-Centered/Pervasive Features. Considering that, the human nature has pervasive effect on system's modeling; this layer narrates an iterative process for learning activity.

1. Multimedia delivery to learner
2. Feedback necessary on assessment and corrective actions taken
3. Records Database: Learner’s performance history
4. Directs different learning styles and strategies based on knowledge library
5. Interaction between Learner and system to negotiate about learning styles

Figure 4.3: CG representation of LTSA layer II
CHAPTER IV - ARCHITECTURAL DESIGN OF LMS

LTSA layer III:

- Stores: Records Database, Knowledge Library.
- Flows: Behavior, Assessment, Performance (past, present, future), Indexes (query, content, and locator), Learning Content, Multimedia, Learning Style.
- A conceptual graph is insignificant at this level as it deals with only identification of components, does not reveal relationship between them. However, the relationships between these components are described within the conformance rules that are formally expressed with lambda expressions later in section 4.3.1.5 of this chapter.

LTSA layer IV:

- All stakeholders' perspectives should be incorporated within the LMS.
- Author's Perspective:
  - Figure 4.4(a): CG representation of LTSA layer IV - author's perspective
- Learner's Perspective:
  - Figure 4.4(b): CG representation of LTSA layer IV - learner's perspective

- Learner
- Learns
- MM content
- agnt

- Communicate
- Learner
- agnt

- Teacher

- Author
- Develop
- MM content
- agnt
Teacher's Perspective:

LTSA layer IV

This layer narrates the operational and system components:

- One operational component (e.g., HTTP) might be used by several system components (e.g., records database, knowledge library).
- One system component (e.g., delivery) might use several operational components (e.g., HTTP, HTML, MPEG).

4.3.1.2 Ontology

We use ontology as a lightweight formal method, to describe domain concepts and the relationships that hold between those concepts. The ontology depicts a tree representation of components of LMS domain based on the classifications of LTSA (e.g., Entity, Process, and Storage). Our proposed methodology considers only the hierarchical relationships between the nodes of the ontology.
4.3.1.3 Sample requirements

- R1. Authors should upload contents
- R2. Teachers should search for contents and create courses using the contents
- R3. Learners should enroll for courses
- R4. Learners should set learning preferences
- R5. System should deliver personalized course content to learners
- R6. Learners should intake courses
- R7. System should store learners' record
- R8. Learners should provide feedback on contents
- R9. Learner's performance should be evaluated by teacher
- R10. System should keep information about the learning sessions
- R11. System should assess learners based on their performance
- R12. System should provide suggestion to the learners
4.3.1.4 Architectural Design

The AD of the system from the above set of requirements is based on the component-connector style of UML. This style leads to a view of an abstract architectural description as a graph in which the nodes represent the components and the arcs represent the connectors (Garlan, 2000 (b)).

Figure 4.7: UML meta-model for the design components

Figure 4.7 represents the meta-model for the proposed extension to the UML component-connector style. Acme components represent computational elements and data stores of a system. A component may have multiple interfaces and ports.

An interface is a point of interaction between the component and its environment, connected by a connector. An interface can be as simple as a single procedure call or can be more complex, such as a collection of procedure calls that must be invoked in certain specified orders. On the other hand, ports represent the state variables of a system on which the methods operate. The connectors represent interactions among the components. Connectors have properties that are defined by a set of roles. Considering the web based implementations of the contemporary LMS software (Kumar et al., 2010), we have restricted the roles within two types: initiator and concluer. Each role of a connector defines the message passing parameters between the participants of the interaction as query and response.
A configuration defines the participating components as attachments and a set of rules for communication as a protocol. Each protocol actually defines a map between triggering events and invoked methods. Figure 4.8 shows a sample AD on the proposed style. Figure 4.8(a) shows a magnified view of ports and interfaces within a component. Figure 4.8(b) shows a magnified view of a connector with message passing. Figure 4.9 depicts the AD for the above case study, built from the sample SRS using the proposed architectural style.
CHAPTER IV- ARCHITECTURAL DESIGN OF LMS

identifying its core components and their relationships. However, detailing of ports, interfaces, roles and protocols are not apparent from the SRS. As a result, designers make assumptions on the structure of the domain elements, which may lead to ambiguity and inconsistency in the design. To avoid such assumptions the proposed methodology relies entirely on the ontology description of the domain.

For example, the concluder flow RL4 in the diagram [Figure 4.9] is attaching the connector evaluation and component teacher. While the component and connector are directly identifiable from the SRS, the value of the query and response parameters of the role are not explicit in the requirements. The ontology, on the other hand, reveals the structure of evaluation that contains L_ID, Cr_ID, Session_info, Performance_info, and Evaluation_data. The designer, in order to avoid ambiguity, must set the parameters of the role in conformity to this set of sub elements. Table 4.1 shows the query and response parameters of the AD shown in Figure 4.9.

Table 4.1: Roles and parameters of the AD

<table>
<thead>
<tr>
<th>Label</th>
<th>Role</th>
<th>Query</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RL1</td>
<td>init</td>
<td>LO</td>
</tr>
<tr>
<td></td>
<td>RL2</td>
<td>init</td>
<td>LO_query</td>
</tr>
<tr>
<td></td>
<td>RL3</td>
<td>init</td>
<td>Course_content</td>
</tr>
<tr>
<td>Label</td>
<td>Role</td>
<td>Query</td>
<td>Response</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>RL4</td>
<td>concl</td>
<td>Performance_info, Evaluation_data</td>
<td>ACK</td>
</tr>
<tr>
<td>RL5</td>
<td>init</td>
<td>Enroll_info, ACK</td>
<td></td>
</tr>
<tr>
<td>RL6</td>
<td>init</td>
<td>Learning_style_preference, ACK</td>
<td></td>
</tr>
<tr>
<td>RL7</td>
<td>init</td>
<td>Course_info, Preference_info, Suggestion_info</td>
<td></td>
</tr>
<tr>
<td>MM_Presentation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RL8</td>
<td>init</td>
<td>Preference_query, Preference_info</td>
<td></td>
</tr>
<tr>
<td>RL9</td>
<td>init</td>
<td>Performance_info, Session_info, ACK</td>
<td></td>
</tr>
<tr>
<td>RL10</td>
<td>init</td>
<td>Feedback_info, ACK</td>
<td></td>
</tr>
<tr>
<td>RL11</td>
<td>concl</td>
<td>Suggestion_info, ACK</td>
<td></td>
</tr>
<tr>
<td>RL12</td>
<td>init</td>
<td>Learner_id, Learner_info: &lt;Learning_style_preference, Evaluation_data, session_info&gt;</td>
<td></td>
</tr>
<tr>
<td>RL13</td>
<td>init</td>
<td>LO_query, LO_list</td>
<td></td>
</tr>
<tr>
<td>RL14</td>
<td>concl</td>
<td>Suggestion_info, ACK</td>
<td></td>
</tr>
<tr>
<td>RL15</td>
<td>concl</td>
<td>LO, ACK</td>
<td></td>
</tr>
<tr>
<td>RL16</td>
<td>concl</td>
<td>LO_query, LO_list</td>
<td></td>
</tr>
<tr>
<td>RL17</td>
<td>concl</td>
<td>Feedback_info, ACK</td>
<td></td>
</tr>
<tr>
<td>RL18</td>
<td>concl</td>
<td>Content_list, Index_info</td>
<td></td>
</tr>
<tr>
<td>RL19</td>
<td>init</td>
<td>Content_list, Index_info</td>
<td></td>
</tr>
<tr>
<td>RL20</td>
<td>concl</td>
<td>Course_content, ACK</td>
<td></td>
</tr>
<tr>
<td>RL21</td>
<td>concl</td>
<td>Enroll_info, ACK</td>
<td></td>
</tr>
<tr>
<td>RL22</td>
<td>concl</td>
<td>Course_info, preference_info, suggestion_info</td>
<td></td>
</tr>
<tr>
<td>MM_Presentation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RL23</td>
<td>concl</td>
<td>Learning_style_preference, ACK</td>
<td></td>
</tr>
<tr>
<td>RL24</td>
<td>concl</td>
<td>Learner_id, Learner_info: &lt;Learning_style_preference, Evaluation_data, session_info&gt;</td>
<td></td>
</tr>
<tr>
<td>RL25</td>
<td>concl</td>
<td>Evaluation_data, session_info, ACK</td>
<td></td>
</tr>
<tr>
<td>RL26</td>
<td>concl</td>
<td>Assessment_info, ACK</td>
<td></td>
</tr>
</tbody>
</table>
Revisiting the objective of this work - to provide an automated conformance-verification methodology of the design artifact with LTSA rules - we propose an algorithmic approach for this verification process. It entails that both the design artifact and the LTSA rules must be represented in some formal way. We use Acme and conceptual graph (with Lambda expression) for formal representation of the design and the rules respectively. The Acme code equivalent to the UML design of the case study in Figure 4.9 is shown below (partially).

<table>
<thead>
<tr>
<th>Label</th>
<th>Role</th>
<th>Query</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
System Sample_LMS =

Component learner =

Port profile, preference, knowledgebase, performance;

Interface learner_intf =

enrollment(profile);
preference_set(preference);

Interface learning_intf =

content_display(preference,knowledgebase);
take_quiz(performance);
feedback();

Supported_Protocol = { prt_content_display };

Interface evaluation_intf =

take_quiz(performance);
feedback();

Component course =

Port LO_list, level, index;

Interface course_intf =

construct (LO_list);
```

...
CHAPTER IV- ARCHITECTURAL DESIGN OF LMS

126

bind(LO_list,index);

dispatch();

rate();

Supported_Protocol = { prt_content_display };

Connector display = {

Role initiator={

querystring= {course.name, learner.preference, LMS_agent.suggestion };

}

Role concluder={

responselist= course.LO_list[i].MM_content};

};

Configuration = {

Attachment course to learner;

Protocol prt_content_display = {

On learner.content_display Æ course.dispatch();

};

// and so on …

};

// and so on …

4.3.1.5 Design verification

The verification process checks conformance of the above designed software architecture [Figure 4.9] with the LTSA conformance rules. We use the Acme code of the design as a formal basis of the architecture in the verification process.

The NL-written conformance rules are represented formally using lambda expressions. However, these expressions are used here in the context of flows only, therefore we have considered only two types of relationships: frm and to, representing the source and destination of a flow respectively. Since, LTSA, domain ontology and AD style use their own set of taxonomies; we need to identify the proper mapping between the different elements of different artifacts.
We use mapping tables, Table 4.2 and Table 4.3, to show LTSA elements' equivalence with AD elements and Ontology elements respectively. Next, we classify and group the conformance rules stated in section 1.4.2 of Chapter I, based on the types of the identifiable elements, for the verification process and then express them in lambda expressions equivalent to conceptual graph representation.

### Identifying Elements

- **LE1.** LTSA learner entity
- **LE2.** LTSA evaluation process
- **LE3.** LTSA coach process
- **LE4.** LTSA delivery process
- **LE5.** LTSA learner records data store
- **LE6.** LTSA learning resources data store

### Equivalent lambda expressions

- **LE1.** \(\exists \text{Learner}\)
- **LE2.** \(\exists \text{Evaluation}\)
- **LE3.** \(\exists \text{Coach}\)
- **LE4.** \(\exists \text{Delivery}\)
- **LE5.** \(\exists \text{Learner_records}\)
- **LE6.** \(\exists \text{Learning_resources}\)

### The conformance-checking algorithm for identifying elements

```plaintext
Identification_of_Elements()
{
int flag:=0;
For each of the lambda expression in the form \(\exists C\) result:= Identify(T,C)
if result!= null
flag:=flag+1
}```
Next

Conformance_level:=(flag/No_of_Identifiable)* 100

}\n
Identify(T,C)\{
t= Find AD element type equivalent to T from Table 4.2
n= Find ontological term equivalent to C from Table 4.3
for each of the element D of type t in AD // from Acme code //
if D.name =n then
  return D
end if
next
return null;

\p

- **Identifying Data Flows**

- DF1. LTSA assessment data flow from evaluation to coach
- DF2. LTSA learner information data flow between evaluation and learner records
- DF3. LTSA behavior data flow from learner entity to assessment
- DF4. LTSA learner information data flow from learner records to coach
- DF5. LTSA learner information data flow from coach to learner records
- DF6. LTSA catalog information data flow from learning resources to coach
- DF7. LTSA locator data flow from coach to delivery
- DF8. LTSA learning content data flow from learning resources to delivery
- DF9. LTSA interaction context data flow from delivery to evaluation
- DF10. LTSA multimedia data flow from delivery to learner entity

- **Equivalent Lambda expressions**

- DF1. \[
\text{Process: Evaluation} \xrightarrow{\text{dataflow: assessment_data}} \text{Coach} \]
- DF2. \[
\text{Process: Evaluation} \xrightarrow{\text{dataflow: learner_info}} \text{Learner_records} \]
- DF3. \[
\text{Learner_entity} \xrightarrow{\text{interaction_context}} \text{Evaluation} \]
- DF4. \[
\text{Learner_records} \xrightarrow{\text{learner_info}} \text{Coach} \]
- DF5. \[
\text{Coach} \xrightarrow{\text{learner_info}} \text{Learner_records} \]
- DF6. \[
\text{Learning_resources} \xrightarrow{\text{catalog_info}} \text{Coach} \]
- DF7. \[
\text{Coach} \xrightarrow{\text{locator}} \text{Delivery} \]
- DF8. \[
\text{Learning_resources} \xrightarrow{\text{learning_content}} \text{Delivery} \]
- DF9. \[
\text{Delivery} \xrightarrow{\text{interaction_context}} \text{Evaluation} \]
- DF10. \[
\text{Delivery} \xrightarrow{\text{multimedia}} \text{Learner_entity} \]
Identifying Control Flows

CF1. LTSA learning preferences between learner entity and coach
CF2. LTSA query control flow from coach to learning resources
CF3. LTSA locator control flow from delivery to learning resources

Equivalent lambda expressions

CF1. [Entity:?Learner] Å (frm) [controlflow: ?learning_preferences] Æ (to) [Proces:?Coach]
CF2. [Proces:?Coach] Å (frm) Å [controlflow:?query] Æ (to) [DataStore:?Learner_resources]
CF3. [Process:?Delivery] Å (frm) Å [controlflow:?locator] Æ (to) [DataStore:?Learner_resources]
The conformance-checking algorithm for identifying data and control flows

Identification_of_Flow()
{
int Flag:=0;
For each of the lambda expression in the form X\leftarrow (frm) \rightarrow Y\rightarrow (to) \rightarrow Z
where Y is the dataflow and X,Z are components or connectors
T_X := find the AD type equivalent -type of X from Table 4.2
N_X := name of the ontology element equivalent to X from Table 4.3
T_Z := find the AD type equivalent -type of Z from Table 4.2
N_Z := name of the ontology element equivalent to X from Table 4.3
A:= Identify the AD element where type== T_X AND name= N_X
B:= Identify the AD element where type== T_Z AND name= N_Z
R:= Identify the role in AD where element where type== T_X AND
name= N_X
//As Y is dataflow the type equivalent is always Role. For the other
two, we identify three possible cases

// Case 1: Both the elements are component
If (T_X == component AND T_Z == component) Then
For each connector C with A
If C has R AND C connects B Then
Pattern= A \rightarrow<Role_i> \rightarrow C \rightarrow<Role_j> \rightarrow B ;
m= (No. of Component + No. of Connector) in Pattern
n= No. of elements in the lambda expression
Flag=Flag+ n/m;
Break;
Else
If C has R but connects to some other Component D Then  // instead
of B
For each connection P from D
If P connects B then
Pattern= A \rightarrow<Role_i> \rightarrow C \rightarrow<Role_j> \rightarrow P \rightarrow<Role_k> \rightarrow D ;
Break;
End if
next
End if
End if
Next
End if

// Case 2: One element is component and the other is connector
If ($T_X == \text{connector}$ AND $T_Z == \text{component}$) Then
If $A$ has $R$ to connect to $B$ Then
Pattern=$A \rightarrow R \rightarrow B$ ;
$m= (\text{No. of Component + No. of Connector}) \text{ in Pattern}$
$n= \text{No. of elements in the lambda expression}$
Flag=Flag+ n/m;
Else
For each component $C$ connected to $B$ by $R$ using some connector $P$
If $C$ is connected to $A$ Then
Pattern=$A \rightarrow<\text{Role}_i> \rightarrow C \rightarrow<\text{Role}_j> \rightarrow P \rightarrow<\text{Role}_k> \rightarrow B$ ;
$m= (\text{No. of Component + No. of Connector}) \text{ in Pattern}$
$n= \text{No. of elements in the lambda expression}$
Flag=Flag+ n/m;
Break;
End if
Next
End if
End if

If ($T_Z == \text{component}$ AND $T_X == \text{connector}$) Then
If $A$ has $R$ to connect to $B$ Then
Pattern=$B \rightarrow R \rightarrow A$ ;
$m= (\text{No. of Component + No. of Connector}) \text{ in Pattern}$
$n= \text{No. of elements in the lambda expression}$
Flag=Flag+ n/m;
Else
For each component $C$ connected to $A$ by $R$ using some connector $P$
If $C$ is connected to $B$ Then
Pattern=B→<Role_i>→C→<Role_j>→P→<Role_k>→A ;
m= (No. of Component + No. of Connector) in Pattern
n= No. of elements in the lambda expression
Flag=Flag+ n/m;
Break;
End if
Next
End if
End if

//Case 3: Both the elements are connector
If (T_i == connector AND T_j == connector)
For each component C connected with A by R
If C is connected to B Then
Pattern= A→<Role_i>→C→<Role_j>→B ;
m= (No. of Component + No. of Connector) in Pattern
n= No. of elements in the lambda expression
Flag=Flag+ n/m;
Break;
End if
Next
End if
// end of Cases
Next
Conformance_level:=(flag/No_of_conformance_check)* 100
}
CHAPTER IV - ARCHITECTURAL DESIGN OF LMS

Table 4.3: LTSA terminology equivalence with ontology

<table>
<thead>
<tr>
<th>LTSA Terminology</th>
<th>Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learner</td>
<td>Learner</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Evaluation</td>
</tr>
<tr>
<td>Coach</td>
<td>LMS Agent</td>
</tr>
<tr>
<td>Delivery</td>
<td>Delivery</td>
</tr>
<tr>
<td>Assessment</td>
<td>Assessment</td>
</tr>
<tr>
<td>Learner's Record</td>
<td>Learner's Data</td>
</tr>
<tr>
<td>Learning resource</td>
<td>Learning Object Repository</td>
</tr>
<tr>
<td>Learner's Behavior</td>
<td>Session info</td>
</tr>
<tr>
<td>Assessment data</td>
<td>Assessment Info</td>
</tr>
<tr>
<td>Catalog info</td>
<td>Index Info, LO_list</td>
</tr>
<tr>
<td>Locator info</td>
<td>Index Info</td>
</tr>
<tr>
<td>Interaction context</td>
<td>Session Info</td>
</tr>
<tr>
<td>Learning Content</td>
<td>LO</td>
</tr>
<tr>
<td>Multimedia</td>
<td>Multimedia content</td>
</tr>
<tr>
<td>Query LO</td>
<td>LO_query</td>
</tr>
<tr>
<td>Learning Preference</td>
<td>Preference Info</td>
</tr>
</tbody>
</table>

The proposed conformance-checking algorithm has two modules: the first one - Identification of Elements() identifies the LTSA elements and the second one - Identification of Flow() identifies the data and control flow, within the AD.
First, the proposed algorithm checks for the AD-equivalent types of the LTSA elements and the equivalent terminology of the LTSA terms, using Table 4.2 and Table 4.3 respectively. In the second module, for Dataflow and Controlflow identification, our aim is to find out a pattern in the AD that satisfies the flow. The conceptual graph representation of the conformance rules describes each data and control flow using three concepts: source, destination, and flow. The equivalent lambda expression is treated as a formal basis of the conformance rule, which is used as input for the proposed algorithm. We define a new measure for conformance level, which indicates the percentage of successful identification of pattern within the AD for all the LTSA rules. However, the AD pattern that conforms to a rule, may not be found using a one to one mapping, instead may be constructed of more than three AD elements (reflecting multiple sources and destinations). In this context, we have defined a goodness measure for the individual conformance matching, which ranges from one to the ratio of number of elements to match ($n$) with number of elements found after matching ($m$). As the number of concepts in the lambda expression ($n$) is always three, the score depends on number of AD elements ($m$) used in the pattern. If a conformance rule is realized in a pattern that uses exactly three elements in the AD then the score is highest, i.e., 1. Conversely, if we do not find any pattern for a conformance rule then the score is zero. The score decreases from 1 to $n/m$ with the increase of number of elements in the AD pattern, as $m$ is always greater than $n$. The more elements used in the pattern to realize a rule, the more transitive the flow becomes in the design. The complete score of the conformance of a design-artifact (represented by conformance level) is the aggregate of the individual scores for each rule, represented in percentage.

4.3.2 Results and analysis

In this particular case study, we have represented the LTSA 'System Coach' element as an entity rather than a process. The 'LMS Agent' component of the AD actually represents the 'System Coach' of LTSA. Thus, in the element identification section, the conformance score remains 83.33% as five of the six LTSA elements, out of six, have an exact equivalent in the AD ('System Coach' is the missing one).
In the flow identification module, the lambda expression represents the relationships between three components. The data/control flow is the central component which is connected to the source component (by \textit{frm} relation) and to the destination component (by \textit{to} relation). The conformance algorithm first identifies the equivalent for these three LTSA elements from the AD then decides the strategy based on the types of the source (A) and the destination (B). We consider three different scenarios here.

Scenario one: if both A and B are component, we try to find out the connector that has the flow (role) joining these two components. We find a pattern \textit{<LMS Agent \Æ RL12 \Æ Learner model \Æ RL24 \Æ Learner Data Coach>} that matches the conformance rule DF5 in this way. If we do not find a connector that directly connects these components, we search if a connector is connected with A and some other Component D, and then we find a new connector that is connected to D and B. We find a pattern \textit{<Learner \Æ RL25 \Æ Preference \Æ RL23 \Æ Learner Data \Æ RL27 \Æ personalization \Æ RL8 \Æ LMS Agent>} that matches the conformance rule CF1 in this way.

Scenario two: if one of A and B is a connector (A) and the other is component (B), we check if they are directly connected by the flow (role); if yes, then the pattern is straightforward. Otherwise, we search for a new component C that is connected to B by the flow using some new connector P. We find a pattern \textit{<Evaluation \Æ R25 \Æ Learner Data \Æ R26 \Æ Assessment \Æ R28 \Æ LMS Agent>} that matches the conformance rule DF1 in this way.

Scenario three: if both of A and B are connector then, we must find out a component that is connected to both these connectors and one of the connectors is connected to that component by the specified flow (Role). We find a pattern \textit{<Delivery \Æ RL7 \Æ Learner \Æ RL9 \Æ Evaluation>} that matches the conformance rule DF9 in this way.

The AD patterns identified from some of the conformance rules along with the intermediate states of the parameters are shown in details in Table 4.4.
<table>
<thead>
<tr>
<th>#</th>
<th>Lambda expression</th>
<th>Equivalent Ontology</th>
<th>Equivalent AD type</th>
<th>Identified Elements</th>
<th>Pattern</th>
</tr>
</thead>
</table>
| 1  | \( X=\text{Evaluation} \)  
\( Y=\text{assessment\_data} \)  
\( Z=\text{Coach} \)  | \( X=\text{Evaluation} \)  
\( Y=\text{assess\_data} \)  
\( Z=\text{Coach} \)  | \( \text{Role} \)  | \( \text{Evaluation} \)  
\( \text{LMS\_Agent} \)  | \( \rightarrow \) |
| 2  | \( X=\text{Coach} \)  
\( Y=\text{learner\_info} \)  
\( Z=\text{Delivery} \)  | \( X=\text{Coach} \)  
\( Y=\text{learner\_info} \)  
\( Z=\text{Delivery} \)  | \( \text{Component} \)  | \( \text{LMS\_Agent} \)  
\( \text{Learner\_model} \)  | \( \rightarrow \) |
| 3  | \( X=\text{Learner} \)  
\( Y=\text{learning\_preferences} \)  | \( X=\text{Learner} \)  
\( Y=\text{preferences\_info} \)  | \( \text{Component} \)  | \( \text{Learner\_Data} \)  
\( \text{Preference\_Info} \)  | \( \rightarrow \) |
CHAPTER IV - ARCHITECTURAL DESIGN OF LMS

4.4 CONCLUSION

This chapter discusses software architectural design issues with respect to LMS. It proposes a methodology for AD, which is useful for a precise and correct description of the system at a higher-level of abstraction. It also enables us with automated conformance checking with LTSA standard. The proposed approach introduces a new set of stereotypes that extends the UML component diagram to depict an AD of a system based on Acme style. The Acme code, equivalent to the graphical AD style, serves as a formal basis of the AD in the verification phase. The LTSA conformance rules, originally stated in NL, are also converted into a formal representation in the form of conceptual graph diagrams for the algorithmic process of conformance checking. The domain knowledge, represented by the ontology, plays an important role in the AD development and the verification process. A partial view of the LMS domain ontology in the hierarchical form is used for this case study. In actual development, the proposed methodology entails a complete description of the domain to support the designing. Therefore, in our methodology, it is hard to incorporate a requirement that involves concepts beyond the structural definition of the available domain entities. However, such limitation could be avoided with a requirement refinement process, based on domain projection technique (Bjorner, 2010), preceding the designing phase. Finally, the proposed methodology endeavors for an automated process for the conformance verification and defines a measure for the
conformance level to understand the extent to which an AD artifact of LMS conforms to the LTSA standard.

After the architectural design phase, we proceed for a detailed design of the system. Since the higher-level description of the system in the form of AD uses formal modeling techniques like ADL, it is also easier to build a formal description of the system in the detailed design phase. We use UML as a common method for AD and detailed design, whereas VDM-SL is used for formal modeling at the detailed design level.