AFRL-IF-RS-TR-2000-90
Final Technical Report
June 2000

LINQUISTIC ASSISTANT FOR DOMAIN ANALYSIS (LIDA)

CoGenTex, Inc.

Tanya Korelsky, Benoit Lavoie, and Scott Overmyer

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2000-90 has been reviewed and is approved for publication.

APPROVED:

JAMES F. REILLY
Project Engineer

FOR THE DIRECTOR:

NORTHRUP FOWLER, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JUNE 2000	Final May 97 - Nov 99

4. TITLE AND SUBTITLE
LINQUISTIC ASSISTANT FOR DOMAIN ANALYSIS (LIDA)

5. FUNDING NUMBERS
C - F30602-97-C-0088
PE - 62702F
PR - 5581
TA - 27
WU - 02

6. AUTHOR(S)
Tanya Korelsky, Benoit Lavoie, and Scott Overmyer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
CoGenTex, Inc.
840 Hanshaw Road
Ithaca NY 14850

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFTB
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2000-90

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: James F. Reilly/IFTB/(315) 330-3333

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
LIDA (Linguistic Assistant for Domain Analysis) helps analysts to develop object-oriented models of a domain, using a subset of UML. In order to develop such models, the requirements analyst or knowledge engineer often needs to analyze large volumes of text from "legacy documents" -- these might include user manuals of legacy systems, company policies, use cases, or transcripts of interviews with domain experts. LIDA facilitates this analysis by compiling a list of the words and multi-word terms in a document, and providing a graphical interface for the user to mark them as corresponding to elements of a model. It also lets the user validate models as they are created, through integration with CoGenTex's ModelExplainer tool, which generates textual descriptions of a model.

14. SUBJECT TERMS
Object Oriented Domain Models

15. NUMBER OF PAGES
52

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

Table of Figures for Part I

Introduction

Object-oriented analysis and design grows more popular every year. Books on the subject abound [3-7]. Traditional systems analysts and developers are working hard to gain the knowledge and expertise required to take advantage of this relatively new technology. Despite the advantages that object technology can provide to the software development community and its customers, the fundamental problems associated with identifying objects, their attributes, and methods remain, and are largely manual processes driven by heuristics that analysts acquire through experience. The primary tool for object identification and refinement is pencil and paper, with the results being transferred to CASE tools after the analysis is largely completed. There are some software tools associated with some methods of object identification and refinement, such as the CRC card approach [1], however, the CRC card approach is labor intensive, and difficult for an individual analyst to use.

Very often, an analyst is given a text document that describes the environment for which an information system is to be developed. This document might include business processes, user tasks, information about existing systems, and so forth. Information about system requirements and proposed system use appears in many forms, from rambling discourse on the operational concept of a proposed system, to loosely organized text descriptions of the business environment and the user's tasks, to highly structured step-by-step procedure descriptions. It is the job of the requirements analyst, regardless of development methodology, to understand, develop, and document this information in a form that can be analyzed by developers, and translated into a software design, and subsequently into code. Traditionally, systems analysts have developed the requirements and modeled them using notations such as process models (e.g., data flow diagrams), data models (e.g., entity-relationship diagrams), flow charts, and plain text. For some time, task descriptions in the form of scenarios have also been employed on traditional analysis efforts as well.

In response to these challenges, the object-oriented community offers the Use Case, and its associated notations, or some variant of that idea. For example, in the Unified Modeling Language (UML), the notations offered are Use Case texts, Use Case Diagrams, and Activity Diagrams [2]. The widespread use of Use Cases as a basis for object-oriented analysis is apparent both in academic literature and in industrial practice on object-oriented development projects.

Still, the process for developing objects from Use Cases and other descriptions is largely manual, and difficult. While a number of methods exist for Use Case development and specification, very few tools exist to assist analysts in making the transition from text descriptions, use cases, or scenarios, to other notations for object-oriented analysis, such as class diagrams and activity diagrams. Without a methodology and a tool to assist the analyst, it is often very difficult to identify classes, their attributes and methods, and their relationship to other classes in the problem domain.

This Final Report describes LIDA, a prototype tool built to provide linguistic assistance in the model development process.

In the following sections, we first give an overview of LIDA's functionality and present its technical design and the functionality of its components. The following section provides a comparison of the LIDA functionality with other research prototypes. The last section of the report presents a tutorial for LIDA produced in collaboration with Professor Scott Overmyer of Drexel University, who served as a technical consultant on the project.

LIDA: A Linguistic Assistant for Domain Analysis

LIDA (LInguistic Assistant for Domain Analysis) helps analysts to develop object-oriented models of a domain, using a subset of UML. In order to develop such models, the requirements analyst or knowledge engineer often needs to analyze large volumes of text from "legacy documents"—these might include user manuals of legacy systems, company policies, use cases, or transcripts of interviews with domain experts. LIDA facilitates this analysis by compiling a list of the words and multi-word terms in a document, and providing a graphical interface for the user to mark them as corresponding to elements of a model. It also lets the user validate models as they are created, through integration with CoGenTex's **ModelExplainer** tool, which generates textual descriptions of a model.

The LIDA prototype has the following features:

- State-of-the-art linguistic processing is used to group different forms of the same base word together to determine part of speech (noun, verb, adjective) and to detect multi-word terms.
- The full text, word lists, and evolving UML model are displayed in parallel, letting the user compare different views.
- Words and multi-word terms can be assigned a type in the model (Class, Attribute, Role, etc.) with the click of a button. The corresponding strings are color-coded in the text display and graphically displayed in the Modeler window.
- Words and multi-word terms can be sorted alphabetically: by frequency, by part of speech, or by assigned type.
- A context view displays only those sentences containing a chosen word or group of words.
- Completed models can be exported to a visual modelling tool Visio™.

LIDA System Architecture

LIDA's system architecture is shown in Figure 1.

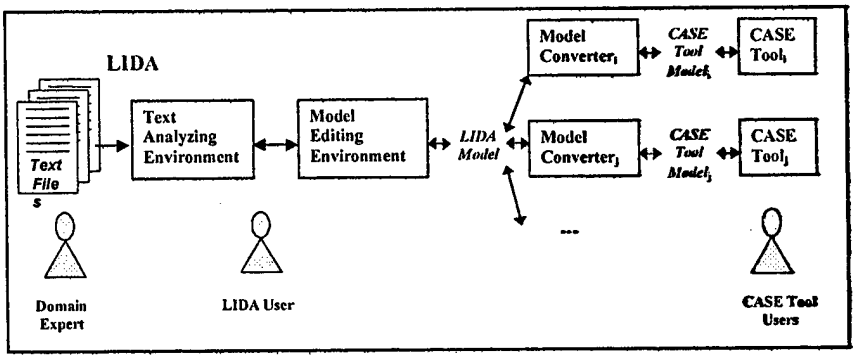


Figure 1: LIDA's Overall System Architecture

LIDA consists of two main components: a **Text Analyzing Environment** and a **Model Editing Environment**.

The Text Analyzing Environment offers LIDA's users the functionality needed for analyzing the lexical content of text files that domain experts may have written for specific domains and assists them in the process of identifying and marking the lexical items corresponding to candidate model elements. Model elements marked in the Text Analyzing Environment propagate to the Model Editing Environment where they can be assembled in a class model. While building the model using the Model Editing Environment, textual context of the model elements is directly accessible, and the addition of new elements to the model or the removal of elements from the model directly affect the marking of the text in the Text Analyzing Environment.

A model developed in LIDA can be saved as a file using LIDA-specific files. LIDA models can either be reloaded in LIDA or exported to a modeling tool for further model refinement. When loaded in LIDA, LIDA models can be applied either to the texts that were originally used for the development of the model or to different texts. The lexical content of a text after loading a model is marked following the information found in the loaded model.

Export of LIDA models to modeling tools and import of models into LIDA is made possible by using converters customized for these modeling tools. The current implementation of LIDA supports export to and import from the Visio™ Modeling Tool where the model converter is developed using Visio API.

The following sections describe in more detail the components of the LIDA system architecture.

LIDA Text Analyzing Environment

The Text Analyzing Environment illustrated in Figure 2 provides LIDA's central functionality. This section describes the functional features of the Text Analyzing Environment and the subcomponents of its interface.

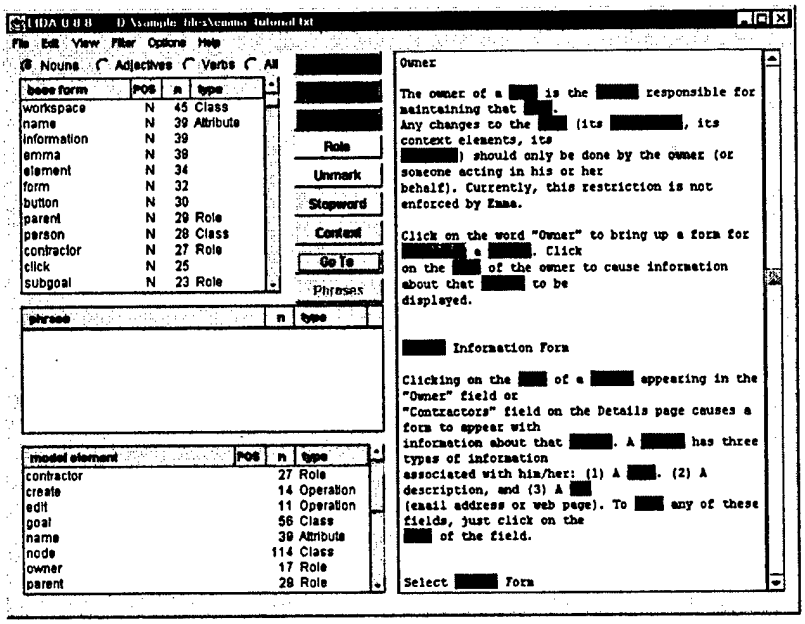


Figure 2: LIDA's Text Analyzing Environment Interface

Functionality

The main functional features of the Text Analyzing Environment include:

- Reading text from file (RTF or ASCII).
- Assigning part-of-speech to words using the MXPOST part-of-speech tagger, a software tool developed at the University of Pennsylvania and re-implemented in Java during the LIDA project.
- Retrieving base forms of words, counting word occurrences.
- Finding multi-word phrases for a given head word.
- Finding user-supplied multi-word phrases.

- Allowing the user to mark a word or a phrase as a candidate model element; all the occurrences of the marked word or phrase are highlighted in the text in a color associated with a model element type.

- Retrieving the textual context of marked words.

Main Interface Components

The main interface components of the Text Analyzing Environment are the following:

- Menus, which display options for opening text files.

- Text display, which displays the currently analyzed text.

- Base form table with part of speech filter buttons (Nouns, Verbs, Adjectives, All), which lists the lexical items contained in the text ordered by part-of-speech.

- Marking buttons (Class, Attribute, Operation, Role), which allow the user to mark terms appearing in the text as candidate model elements of different types (class, attribute, etc.).

- Unmark button, which allows the user to unmark previously marked terms.

- Context display, which displays all the textual contexts in which a given term appears.

- Model element table, which lists the terms marked as candidate model elements.

LIDA Model Editing Environment

The Model Editing Environment (or Modeler), illustrated in Figure 3, offers the functionality needed to build a model from the candidate model elements marked in LIDA's Text Analyzing Environment. It also displays the textual information (the textual context or textual description of a model element) from the Text Analyzing Environment, which is useful in the process of model building. This section describes the main functional features of the Model Editing Environment and the subcomponents of its interface.

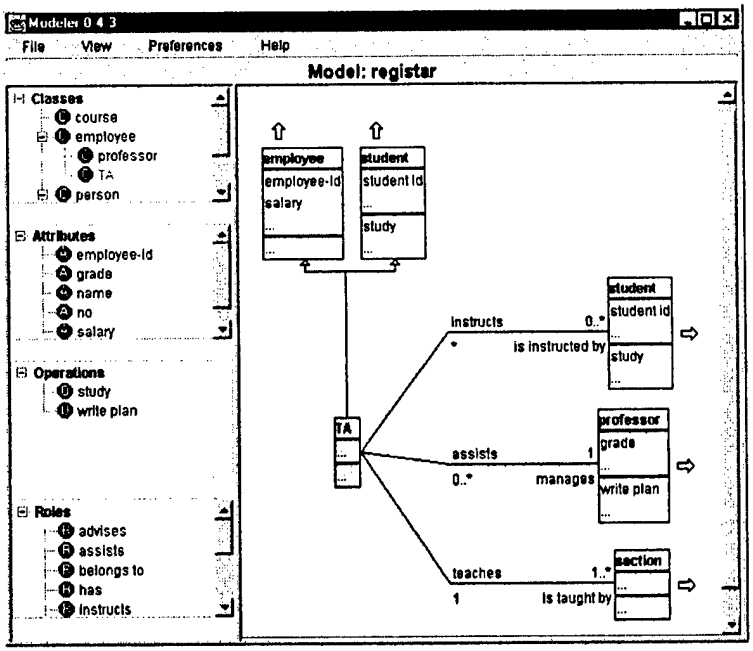


Figure 3: ***LIDA's Model Editing Environment Interface***

Functionality

The main functional features of the Model Editing Environment include:

- A display showing the candidate and actual model elements.
- Operations (e.g. "Declare an association between two classes") needed for incorporating model elements into a class diagram.
- A display showing the textual contexts in which the model elements appeared.
- An operation for automatically generating textual descriptions of model elements.

Main Interface Components

The main interface components of the Model Editing Environment are the following:

- Main menu, which contains functions for manipulating and customizing the information displayed in the Model Editing Environment.

- Pop-up menus associated with the model elements displayed in the Model Editing Environment, which contain commands for editing a model element, for displaying textual context of model element, and for automatically generating a textual description of a model element.

- Lists of class, attribute, operation and role model elements, both candidate and actual.

- Class diagram.

Related Research

There is an extensive literature discussing the relationship between the linguistic structure of the requirements document and the model that is obtained from it, for example [9] – [12]. LIDA is inspired by the observations in these research efforts, but differs from these efforts crucially. These previous efforts are either purely methodological, or aim at automatic model extraction, which is currently beyond the state of the art for serious (i.e., non-demo) software engineering efforts. An interesting and ambitious example of this type of research is the COLOR-X system [13], which aims at including a large lexicon to aid in semantic model validation and provides a new modeling language specially developed for this purpose.
LIDA differs from these research efforts in the following key aspects:

- LIDA provides only state-of-the-art linguistic tools that perform extremely reliably with broad coverage (part-of-speech tagging and morphological analysis).
- LIDA includes a robust and usable GUI.
- LIDA is compatible with UML and UML-based COTS graphical modeling tools. It is designed to be compatible with actual software engineering practice.

Perhaps the system that comes closest to LIDA is the Grammalizer system, which is sold by the Dutch consulting group KISS, **http://www.kiss.nl**. Because it is a commercial product, information about this system can only be obtained from marketing literature, but it appears to use similar linguistic technology to LIDA (part-of-speech tagging and morphological processing). However, it appears that Grammalizer can only be used in conjunction with the KISS methodology, which includes mapping the natural language text to a set of predefined KISS-concepts, which then become the model. In LIDA, we make no assumptions about what sort of concepts will be found -- the analyst works with the result of LIDA's linguistic processing to create a model. We have designed LIDA to be flexibly integratable with different approaches to modeling through the use of UML-style models.

Tutorial:

Using LIDA to Assist Class Identification, Refinement and Validation

The following tutorial provides a methodology for class identification and elaboration, and a scenario-based, hands-on experience with how LIDA [2a] helps with this process.

The Methodology

There are a number of ways in which LIDA can be used to support the object-oriented analysis and design process, depending on where in the OOAD process the analyst is starting. These tutorials cover two situations: 1) starting with a text domain description, operational concept document, use cases, or other text system or task descriptions, and 2) starting with an existing class model, and using LIDA to explain it in text form. For example, if an analyst is starting with a text description of the concept of operation for a proposed system (in any text-based format), LIDA can be used to assist the analyst in identifying potential objects, attributes and methods, and elaborating on them based on the text description. On the other hand, if an analyst wants to validate an existing class diagram by viewing the diagram in the form of a text description, LIDA will translate the diagram into a text description that an analyst can use to compare the *current* model with the *intended* model for the proposed system or domain. This capability of going back and forth between text and graphical model during development of a model can be very helpful, and minimizes errors before model complexity obscures error states.

Most text documents that describe system functions contain many more nouns, verbs, adverbs, and adjectives than are useful in an OOAD effort. These words are necessary in prose and conversation to fluently describe the manner in which a system is to be used. The sheer number of words must be reduced to a subset that is directly applicable to the development of a system using classes. Prose is typically filtered in a context of use. In other words, knowing how the system is to be used helps the analyst identify which classes are relevant, and which classes are not. Prior to attempting to identify classes, the analyst should already have prepared a set of use cases or scenarios that represent the operational concept for the proposed system. The analyst should have this information in mind prior to using our methodology. By having this knowledge at hand, the analyst can make much more effective use of the capabilities of LIDA.

Hints regarding the identification of classes, their attributes, roles and methods will be included in the tutorial; however, if the analyst is unfamiliar with object-oriented analysis and design, this tutorial will be only minimally effective. LIDA is most effective in conjunction with a fully developed OOAD lifecycle model that includes a process, notation, tools and usage heuristics. Without a thorough education in OOAD, the user can expect only marginal results. Our methodology is consistent with the Unified Software Development Process, and LIDA uses the Unified Modeling Language (UML) as the preferred notation.

The full text and models used in these tutorials can be found in Appendix A. For questions about specific LIDA functions, we recommend using the online help facility from within LIDA.

Part 1

Identifying Classes Starting from Domain Descriptions, Operations Concepts, or Use Cases

The general flow of the class identification process is illustrated in Figure 1. Bearing the general flow of activities in mind will help the user make sense (and better use) of the tutorial. While this tutorial represents a single path through the methodology (one scenario) accomplished with the help of LIDA, the user is encouraged to experiment with their own variations on the method.

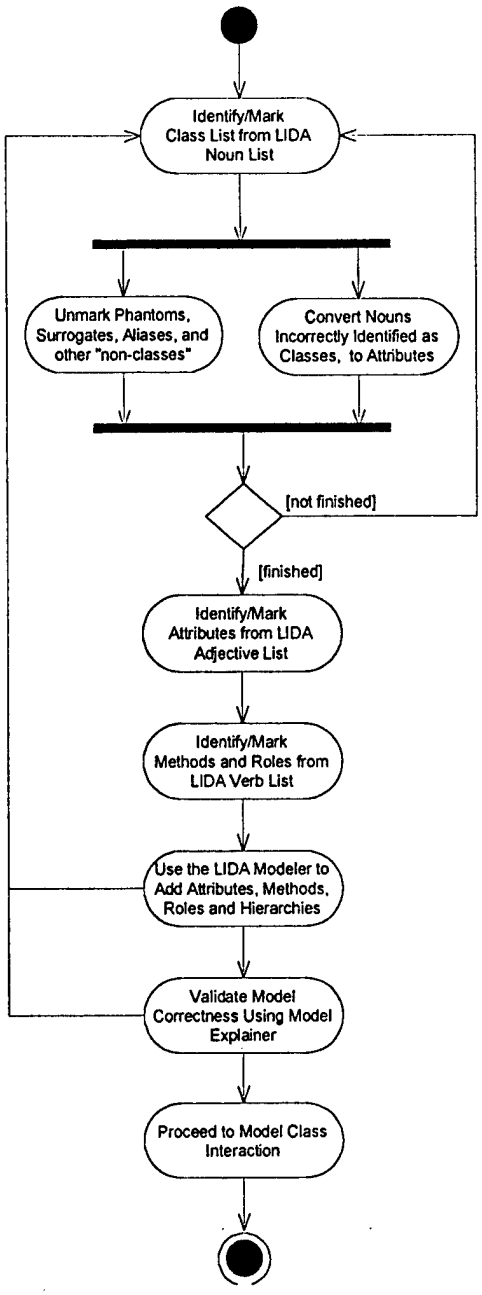


Figure 1. LIDA Class Identification & Modeling Process

The goal of this method is to utilize existing text descriptions of a problem domain, and from them, produce the initial class diagram with attributes, methods, and roles for export to an object-oriented CASE tool.

1. Start LIDA by Double-clicking on the LIDA file icon labeled LIDA(.bat)
2. The following screen will appear (Figure 2):

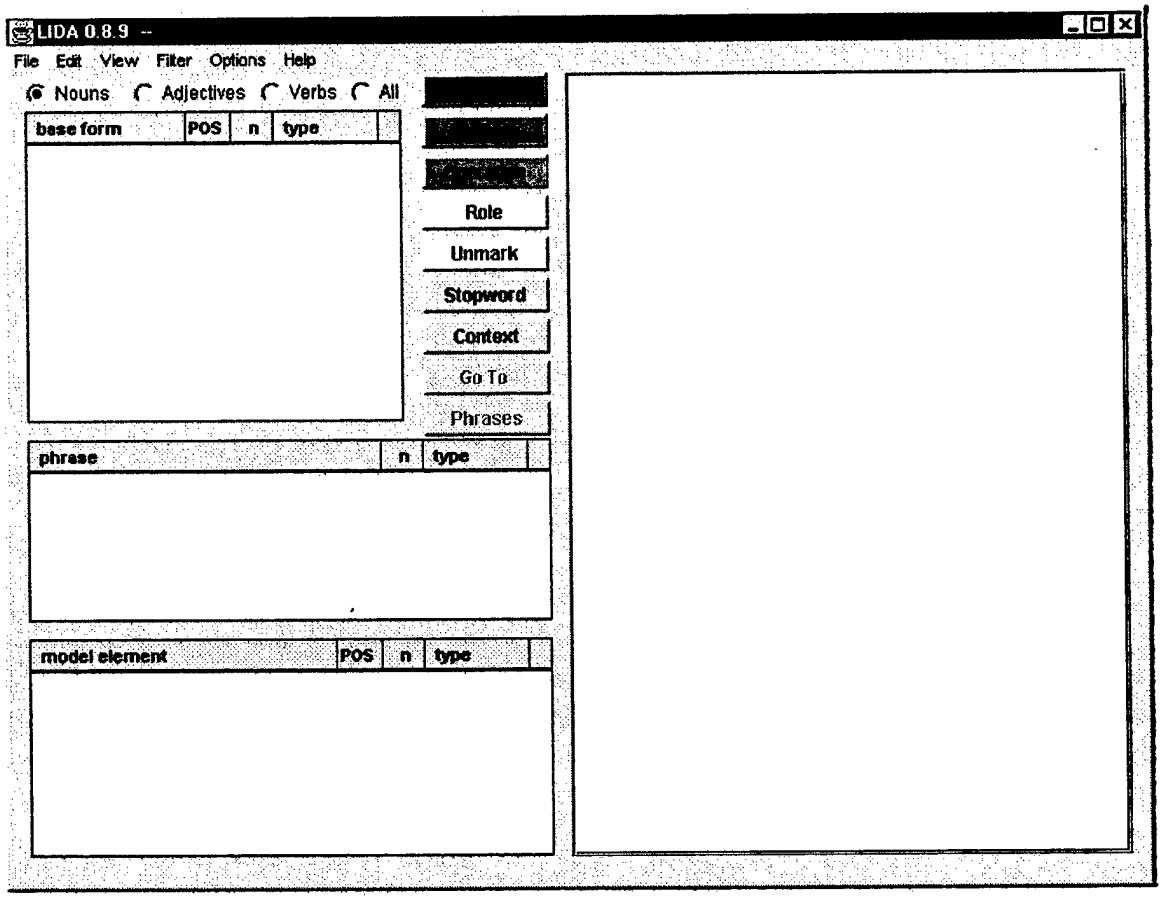


Figure 2 LIDA Main Screen

3. From the ***File*** menu, select ***Load Text ....***
4. For our tutorial, we will be using a text file that has already been tagged by the LIDA tagger. This is to say that the software has already identified the appropriate parts of speech in the document we will be processing. Figure 3 shows the Load Text dialog box. Within this box, find the file *Video_Store.txt* and select it so that it appears in the *File name:* field. Click the *Open* button. (Alternatively, double-click the file name for the same result.)
5. Figure 4 shows the LIDA main screen with the tagged Video_Store.txt file loaded.

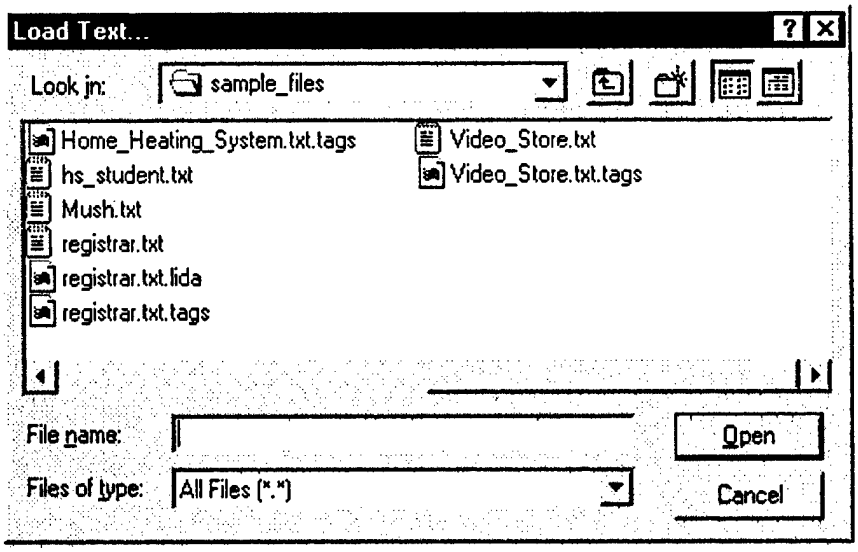


Figure 3. Load Text ... Dialog Box

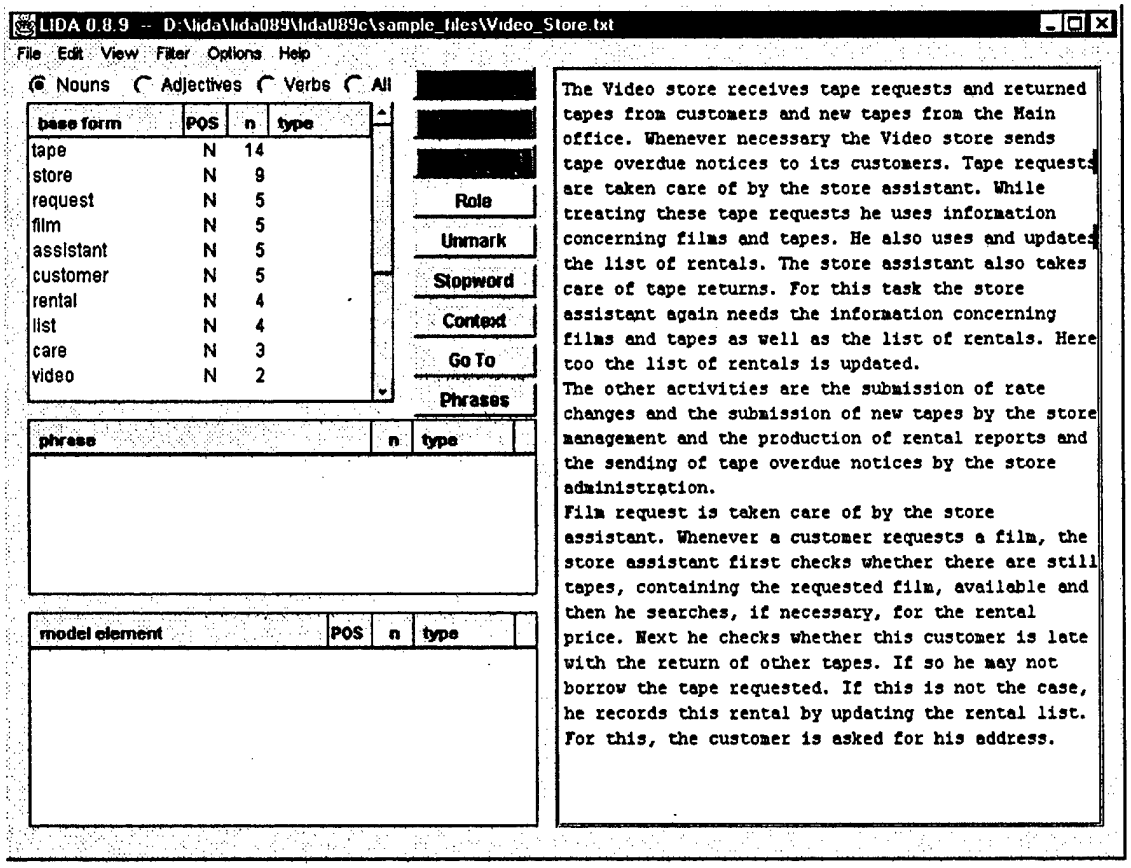


Figure 4. LIDA Main Screen with Text File Loaded

6. Notice in Figure 4 that the text file appears in the right-most window, and that the parts of speech appear in the upper-left-most window, with the nouns displayed in the default situation.
7. Most OOAD methods recommend the identification of candidate classes first, suggesting that the most frequently occurring nouns are likely candidates. In order to arrange the nouns in the order of occurrence, click on the "**n**" heading directly above the numbers of nouns. This will automatically sort the nouns in ascending order according to their frequency of occurrence in the document as shown in Figure 5. (Clicking again will reverse the order to descending.)

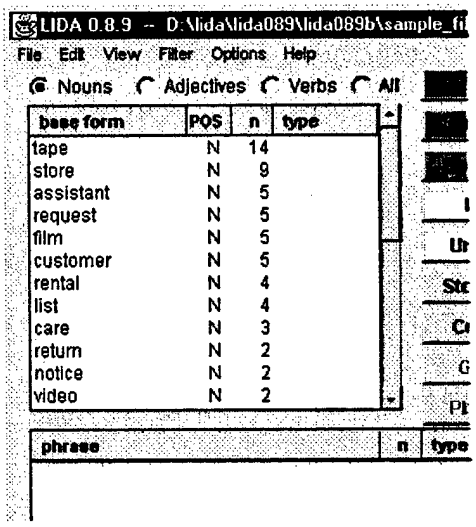


Figure 5. Sorted Nouns in Order of Occurrence

8. Examining the list of nouns, we find "tape" appearing 14 times in the document. Since this is a video rental store, and tapes are what are rented, this seems like an ideal candidate for the first class. Click on the word "tape" in the list of nouns, and then click the **Class** button.
9. LIDA highlights the word "tape" everywhere it occurs in the document. We recommend identifying the set of candidate classes first in a brainstorming fashion, then removing inappropriate classes and adding missing classes in subsequent passes.
10. The next most frequently occurring noun is "store", followed by "assistant", "request", and "customer". We highlight all of these in turn, and mark them as possible classes (or candidate classes).
11. The next word on the list is "film". We suspect that "film" and "tape" might be used to refer the same object (since we know that most video stores don't rent photographic film). In order to verify that this is the case, highlight the word "film", and click on the ***Context*** button. This results in the display of the context window as shown in Figure 6.

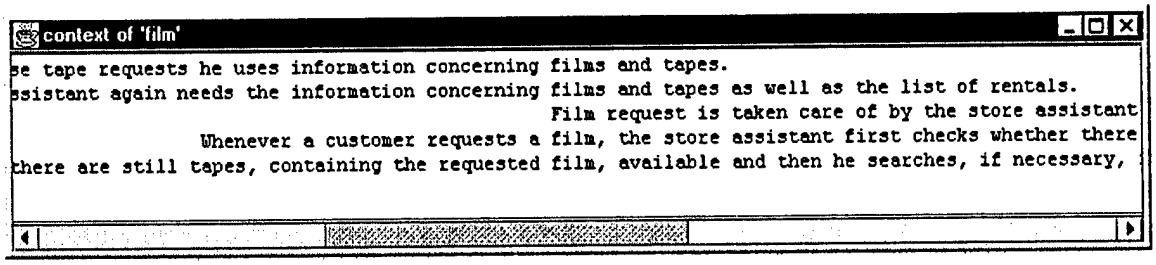


Figure 6. Context Window

Notice that the Context function locates the word of interest, in this case "film", in the center of the window, with the rest of the sentence in which the word appears on either side of the word. As we read through the context, although we notice that there is a possible "containment" relationship indicated in the last sentence, we decide that this is irrelevant to our analysis and that tape and film refer to the same concept, so we do not mark "film" as a class. Ideally, we might mark "tape" and "film" as synonyms, however, LIDA does not currently support this operation.

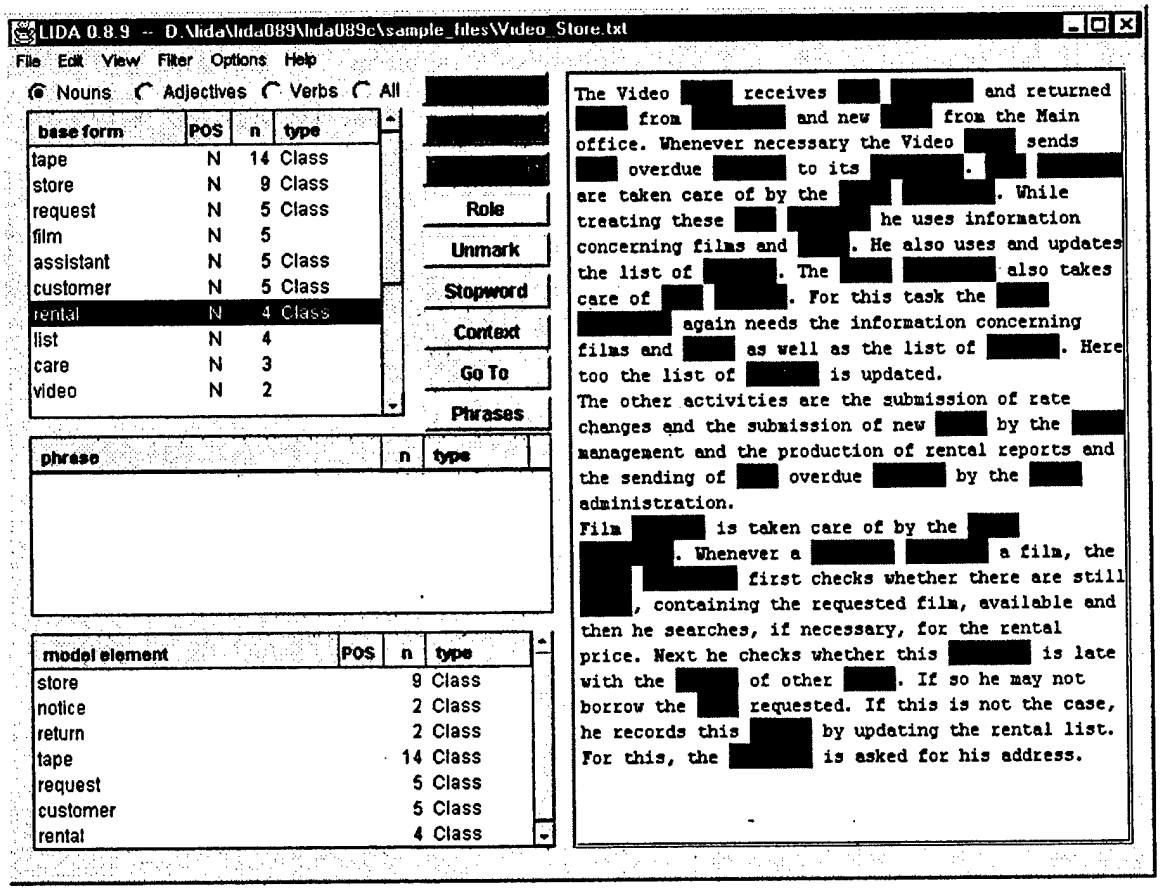


Figure 7. LIDA with Candidate Classes Marked

12. The next noun in the list that looks like a good candidate is "rental", since we know that there is likely to be a rental transaction in the system, and that transactions are candidates for classes. Highlight "rental" and mark it as a class. Similarly, we tentatively mark "return", "notice", "address", "report", and "price" as candidate classes. Figure 7 shows the appearance of the LIDA user interface at this point. Current model elements are shown in the lower-left corner. NOTE: It is also possible to highlight and mark words as model elements directly in the text window. This is very useful when giving the text a final read before moving from one step to the next.
13. Next, review the list of model elements, and revise it according to your knowledge (and your customer's knowledge) of the video rental problem domain. For example, we have identified "address" as a class, but upon examining the context by highlighting "address" and clicking the ***Context*** button, resulting in the display in Figure 8), it is apparent that "address" is really an attribute of "customer" (because of the possessive pronoun "his").

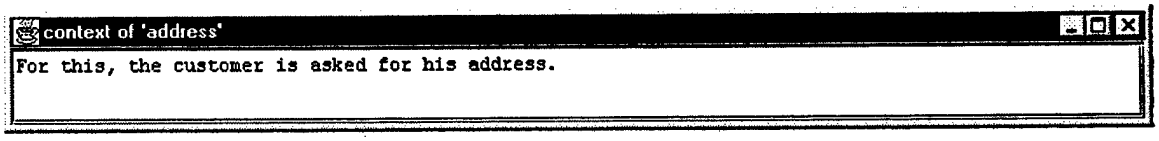


Figure 8. Context for "address"

To correctly mark "address" as an attribute, click on the Unmark button ("address" should already be highlighted). This action will remove "address" from the list of model elements, and unmark "address" in the main text. Next, using the scroll bar in the "Base Form" window (upper left), locate "address", and click on it. Click on the ***Attribute*** button, and notice that "address" is now highlighted in cyan in the main text, and appears again in the Model Element window as an attribute. The result is shown in Figure 9.

14. Examine the remainder of the list of model elements, and convert "price" to an attribute as we did with "address". Further, in thinking about the features we will be implementing in the proposed system, we determine that "store" isn't a system class or attribute (i.e., the video rental "store" doesn't play a real part in the system), and "assistant" is an actor in our scenario, rather than a part of the "system" per se. Remove both of these from the list of model elements. (Highlight each word individually, and then click the ***Unmark*** button.) Later, we may want to have an abstract interface class that equates to "assistant", but that class will be created in the design phase.
15. Finally, before moving to the next operation, we notice that "return" is marked as a class. Thinking about "return", we realize that it has few methods or attributes that are not associated with "rental", so we remove it as a class, and mark it as an attribute, possibly of "rental" (using the same procedure as with "price", etc). We may have to change this later, but it works for now in the context of our knowledge of the scenario for returning a rented video.

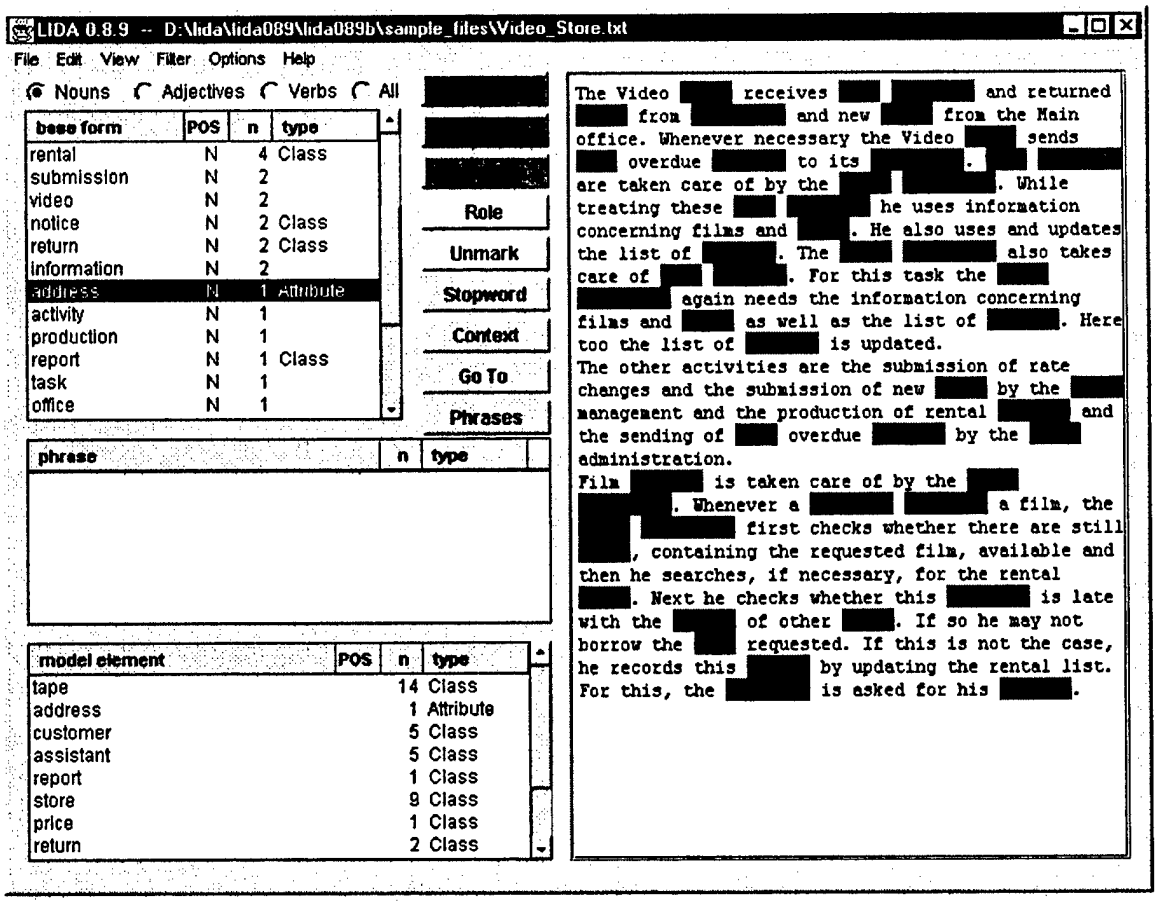


Figure 9. "Address" Marked as an Attribute

16. Having completed our initial marking of classes using LIDA, we now turn to possible attributes using tagged adjectives. First, click on the "Adjective" radio button underneath the main menu at the top of the LIDA window. This results in a list of adjectives. Next, click on the "*n*" above the number of occurrences of the adjectives, and the list will sort in ascending order of occurrences. The window should now appear as in Figure 10.

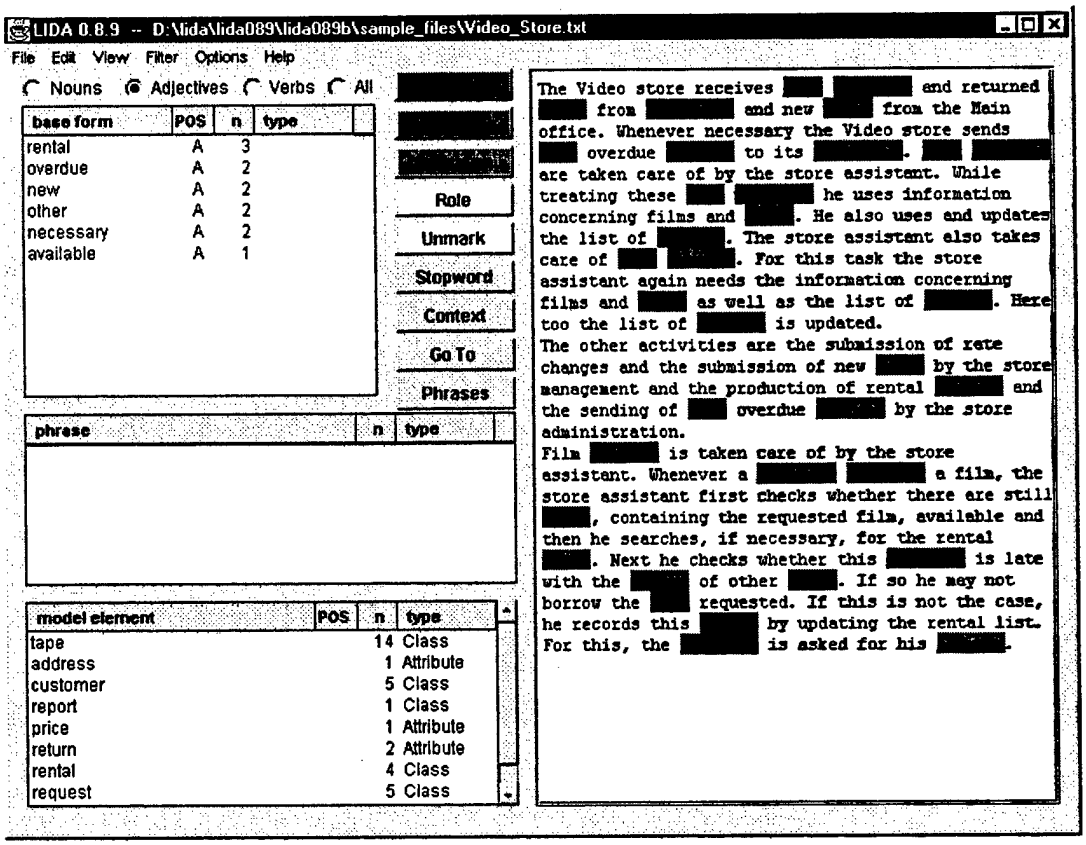


Figure 10. Adjectives in Descending Order

17. As shown in Figure 10, this list is rather short, and not very rich in possible attributes. The only 2 possibilities on the list are "overdue", a possible attribute of "rental", and available", a possible attribute of "tape". The other adjectives don't make sense as attributes in our context. Next, highlight and mark each of the two words, "available" and "overdue" as attributes.
18. Finally, we use our list of tagged verbs to help identify methods or operations associated with the classes we have previously identified. Using the same procedure as with the adjective list, click on the ***Verb*** radio button to display the tagged verbs, then click on the "**n**" to display the verbs in descending order of occurrence. Identifying methods at this point is much more difficult, and is typically done at a later iteration of the OOAD lifecycle (during analysis & design). Nonetheless, we can examine the list for candidates at this stage to help get as complete a class diagram as possible, early in the process. The verbs in order are shown in Figure 11 (upper left sub-window).

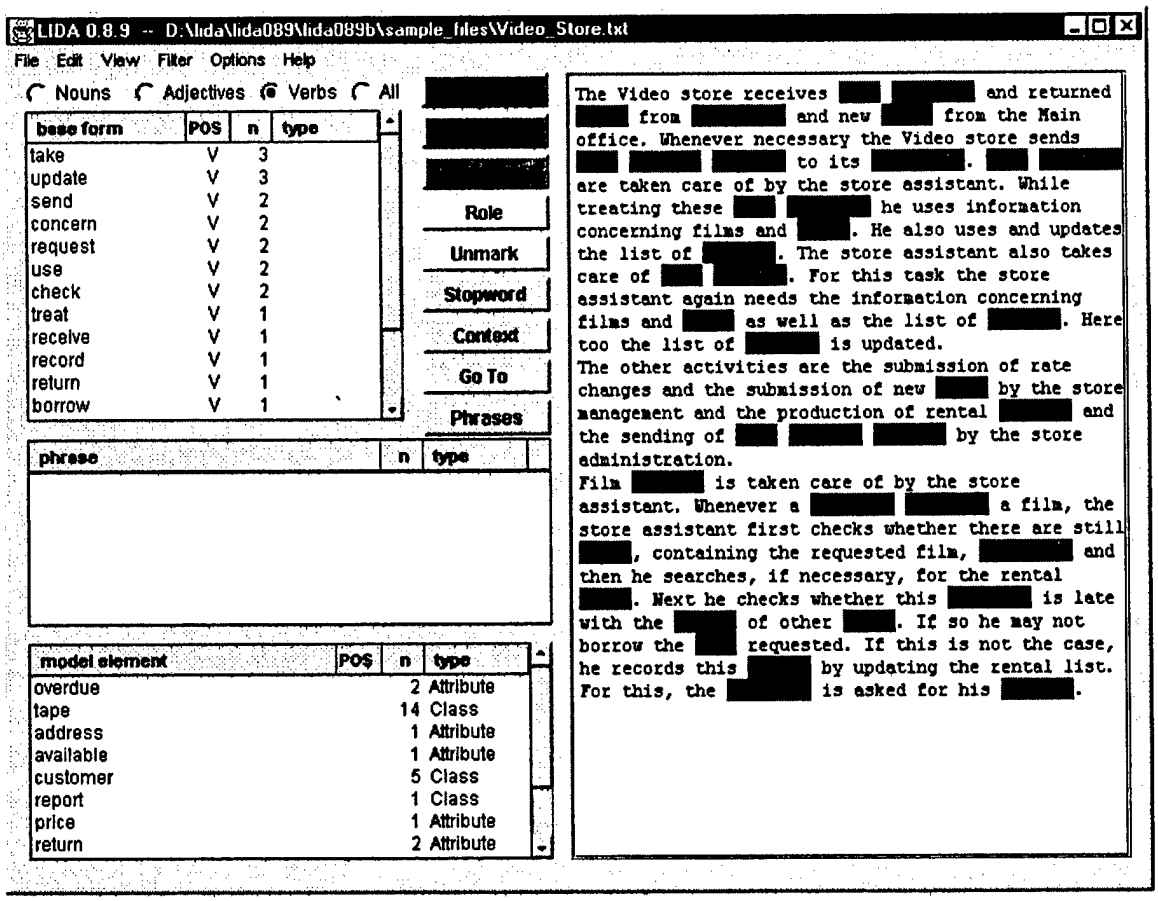


Figure 11. Verbs in Descending Order

19. With the classes in mind, we examine the list of verbs in search of behaviors (operations or methods) the classes might exhibit. We have tentatively identified six classes from the text. Go to the Model Element window, and click on "***type***", then on "***n***". This sorts the Model Elements by type in descending order so that we can view the list of classes more easily. Scrolling down to look at the resulting list of classes, we see that the first is "tape". Looking through the list of verbs, find behaviors that the "tape" class might exhibit. Using the context feature, we find that, according to the text, none of these verbs represent behaviors that are applicable to the "tape" class. One verb, "update" is likely to be a behavior exhibited by "tape", but not according to its use as described in the text. Using a bit of extra knowledge, mark the "update" verb as an operation.

    The next class is "notice". Scanning the list of verbs, we find a behavior that a "notice" might exhibit. Mark "send" as an operation. We follow this same general procedure for the remaining classes, making sure that any verbs marked as possible Operations are done so in the context of a class behavior. This procedure results in the following verbs marked as possible operations: "check", "request", "return", "record" and "borrow". The resulting LIDA window is shown in Figure 12. It is also critical, especially when differentiating between methods and roles to use the context feature to view noun-verb

collocations. Figure 13 shows the use of context to examine the relationship between the noun "notice" and the verb "send".

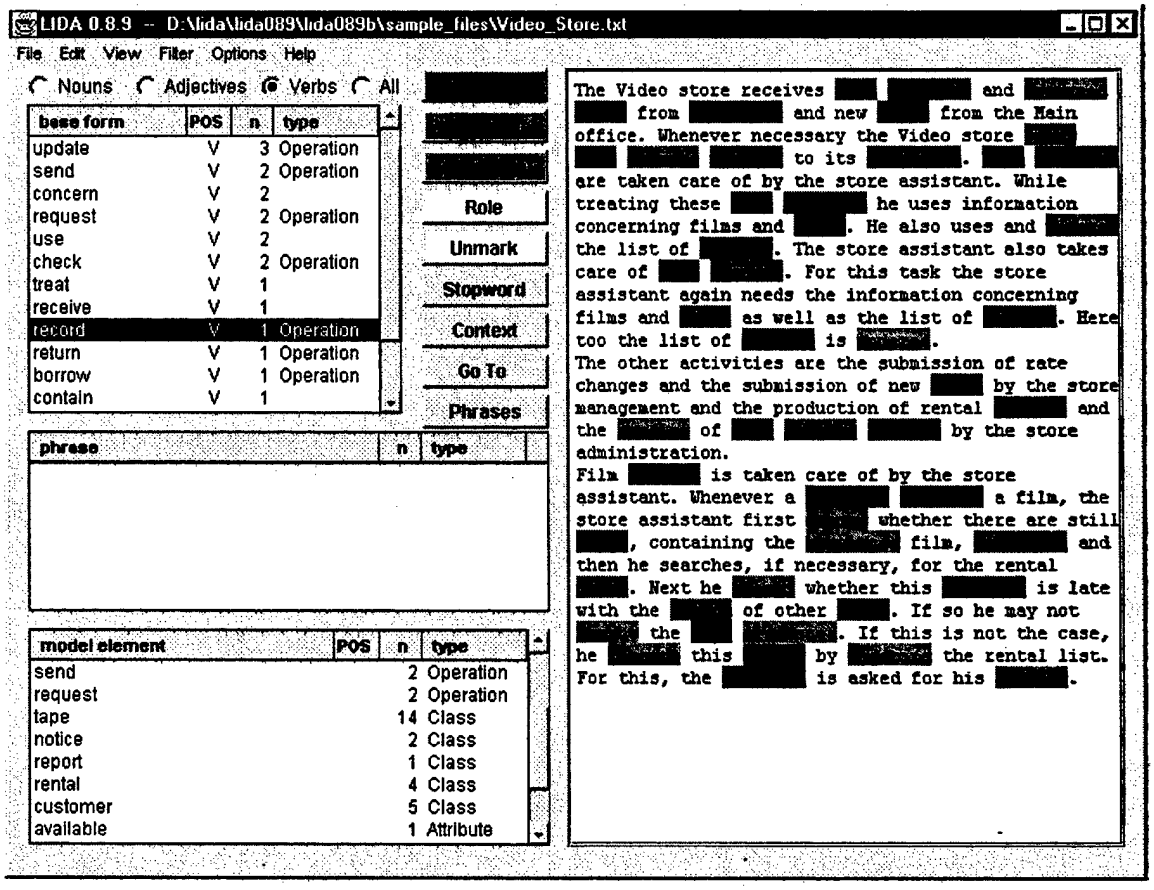


Figure 12. Classes, Attributes and Operations Marked

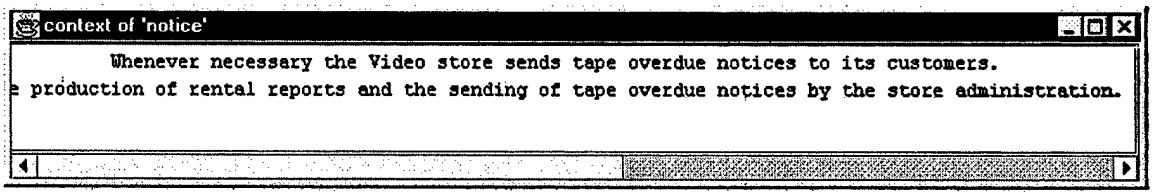


Figure 13. Using Context to Examine Verb-Noun Collocation

20. While looking over the highlighted text, we notice that a phrase is apparent "overdue notice", which may represent a sub-class, or special kind-of notice. To mark this as a class, click on the ***Nouns*** radio button to recall the list of nouns. Click on the "notice" base form, then click on the ***Phrases*** button to the right. A list of phases will appear that contain the word "notice". Cool, eh? Select the phase "overdue notices", and click the

Class button. The phrase will then be marked as a class. The result of this operation is shown in Figure 14. Notice that the first occurrence of the phrase "overdue notices" is highlighted together in the text window.

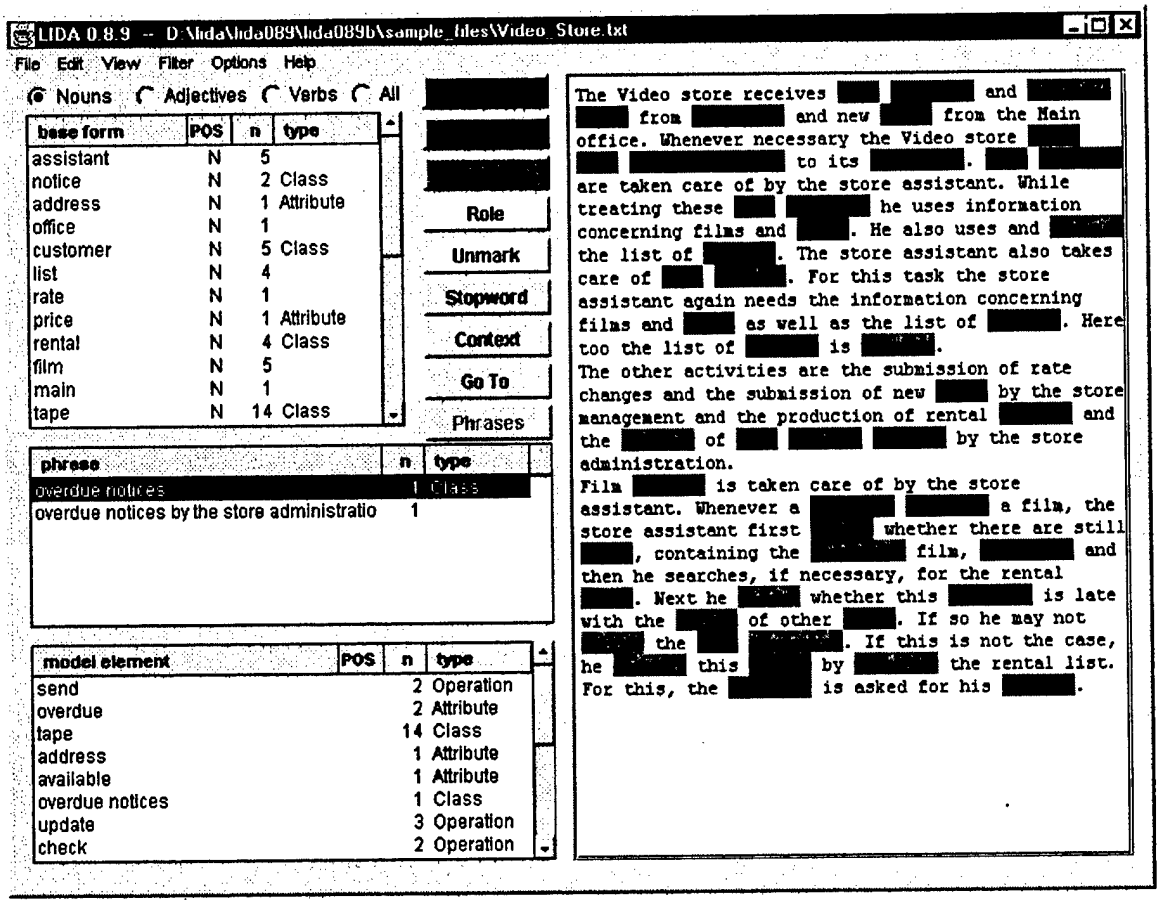


Figure 14. Using a Phrase as a Class.

21. Finally, examine the verb list to see if there are any possible roles that one class will play with respect to another class. Notice that a customer can "receive" an overdue notice and "receive" a tape that they have rented, so select "receive", and tentatively mark it as a role using the ***Role*** button.
22. Now that we have a set of initial classes, attributes, and operations, we are ready to use them to construct our model. On the ***File*** menu, select the ***Edit Model ...*** option. This will execute the LIDA Modeler, as shown in Figure 15. Notice that there are four sub-windows on the left side of the Modeler window, labeled Classes, Attributes, Operations, and Roles. Each sub-window contains those model elements previously identified using LIDA.

· Using Modeler, we will now associate these model elements into a preliminary class model. This window will appear in addition to the LIDA main window, and the two

windows may be utilized alternately by clicking on the window currently in the background, in order to bring it to the front.

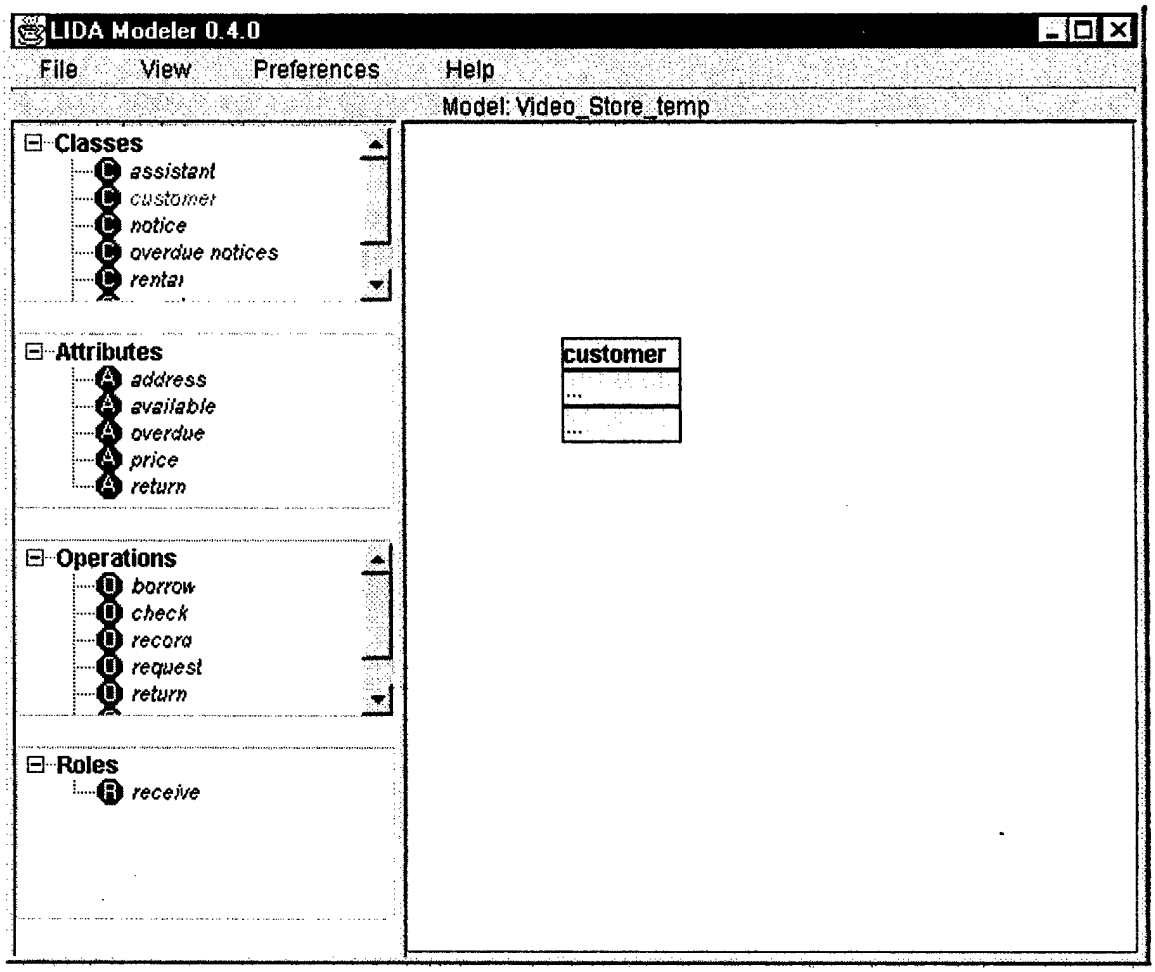


Figure 15. Initial LIDA Modeler Window

23. Starting with the "customer" class (shown above in Figure 15), we need to assign attributes and operations to the class. NOTE: The context of the class in the text is always available, automatically, via LIDA, if a "memory aid" is needed to assist with any assignment decisions.

    An attribute of customer that we are certain of is "address". Assign "address" as an attribute of "customer" by clicking on the "attribute" section of the "customer" class symbol in the main model display window. This is the center of the 3 rectangular widget areas as shown in Figure 16 below. Clicking on this box results in the display of the Attributes dialog box shown in Figure 17.

    Select the "address" attribute from the list, and click on ***OK*** to complete the assignment.

24. Next, click on the "operation" section of the "customer" class symbol in the main model display window. This is the bottom portion of the class symbol as indicated in Figure 18, below.

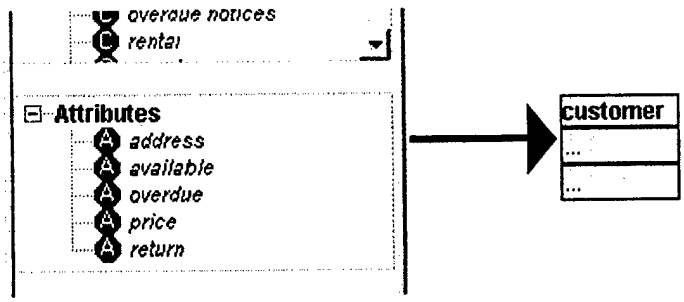


Figure 16. Attribute Box

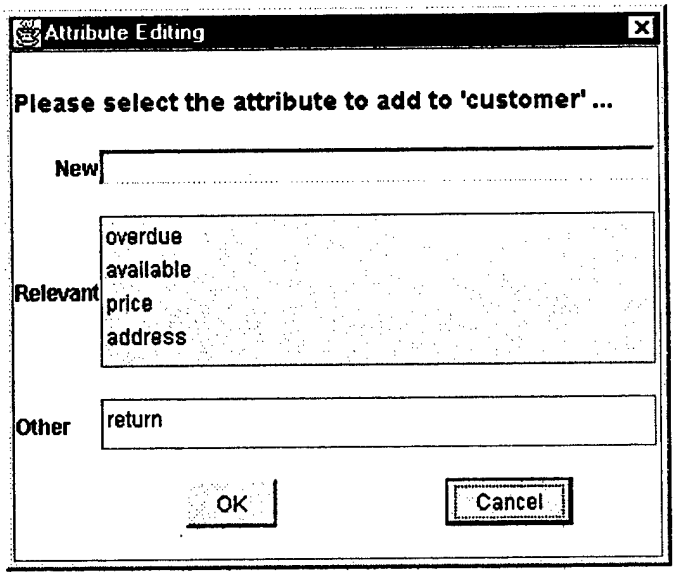


Figure 17. Attribute Editing Dialog Box

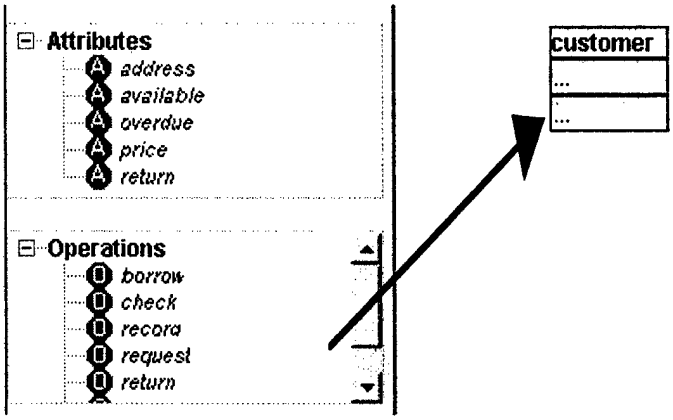


Figure 18. Operation Section of Class Symbol

Clicking on this part of the symbol results in the display of the "***Operation Editing***" dialog box. From this box, we find two operations that belong to the "customer" class, "borrow", and "return". Selecting first one, and repeating the operation for the other adds both operations to the "customer" class symbol in our model. The resulting model screen is shown in Figure 19.

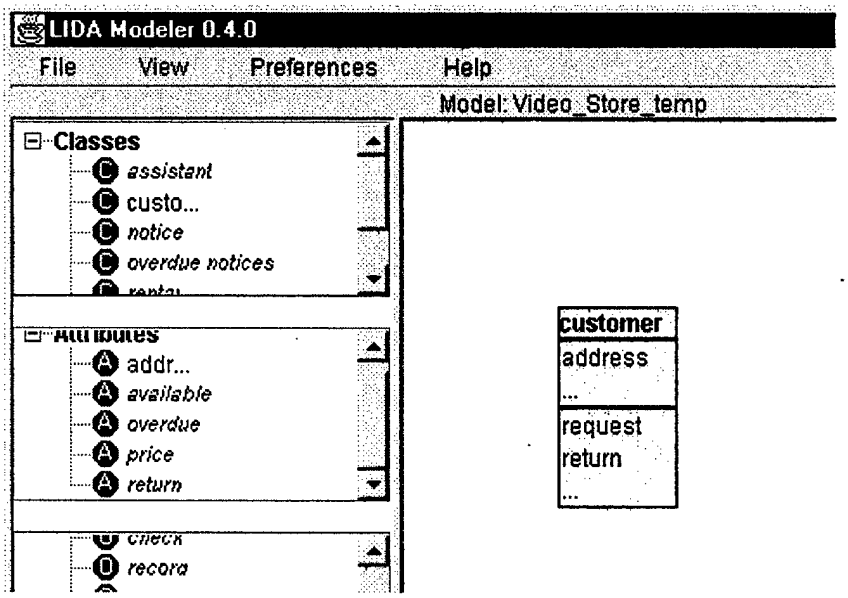


Figure 19. Customer Class with Attributes and Operations

25. Having selected the attributes and operations of the "customer" class from the available list, we can now add those not in the text (if we have sufficient information about the domain to do so), and we can associated the customer class with other classes in our model. For example, we know that in order to send a customer a late notice, we will need their name, and if we want to telephone them, we will need their phone number. To add attributes not in the text, simply click on the "..." blank in the attribute box of the class. LIDA will display the Attribute Editing dialog box as shown in Figure 20. Enter the name of the attribute, in this case "phone", and click the "**OK**" button. While the attribute will not be added to the base text in LIDA, it will be added to the model.

Continue this activity until the class is as complete as possible, given current domain knowledge.

26. The next step is to decide the nature of associates between the customer class and other classes, given our knowledge of the scenarios under which the system will be used. LIDA makes this an easy process as well. First, with the cursor over the title of the "customer" class, click the *right* mouse button. This will display the menu shown in Figure 21. Left click on "Add Association". LIDA will then display a list of classes as shown in Figure 22. We know that customer and rental are associated, since customers are naturally engaged in rental transactions. Select "rental" from the list, and click on the "**OK**" button. Figure 23 shows the resulting graphical display of the "customer" class associated with the "rental" class. However, we are still not finished editing the association, since we have not specified the multiplicity of the relationship. As shown in Figure 23, the default multiplicity is *, or many-to-many. From our knowledge of renting videos, and from the base text, we can determine that this is incorrect, and that we must change the multiplicity of the customer to 1:1 or, "one-and-only-one", but that one customer can rent many videos. To do this, right click on the lower * multiplicity symbol associated with "customer". This produces a menu from which we select the "1", indicating only one customer per association. The result is shown in Figure 24.

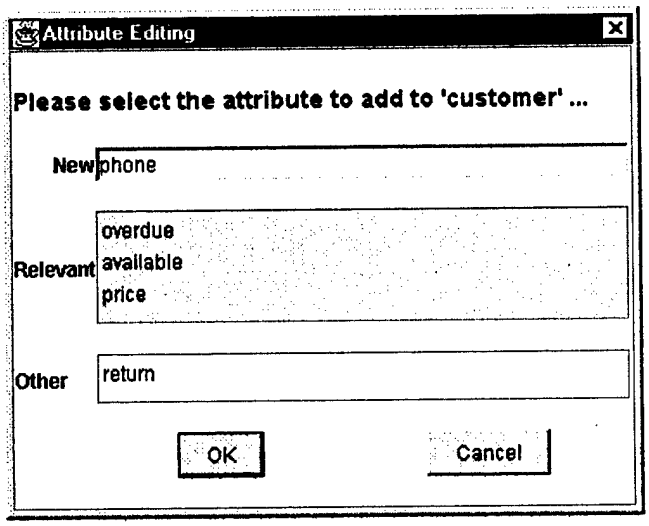


Figure 20. Adding a New Attribute, Not in the Base Text

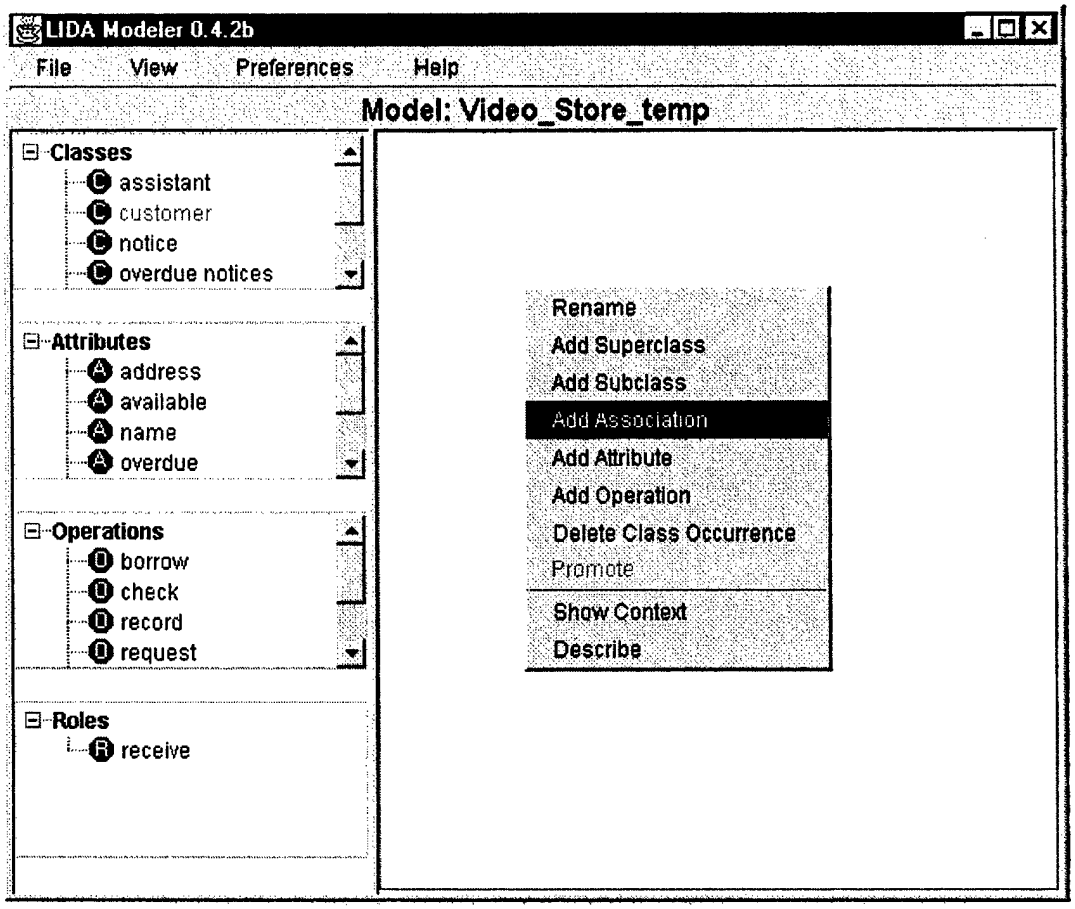


Figure 21. Class Option Menu

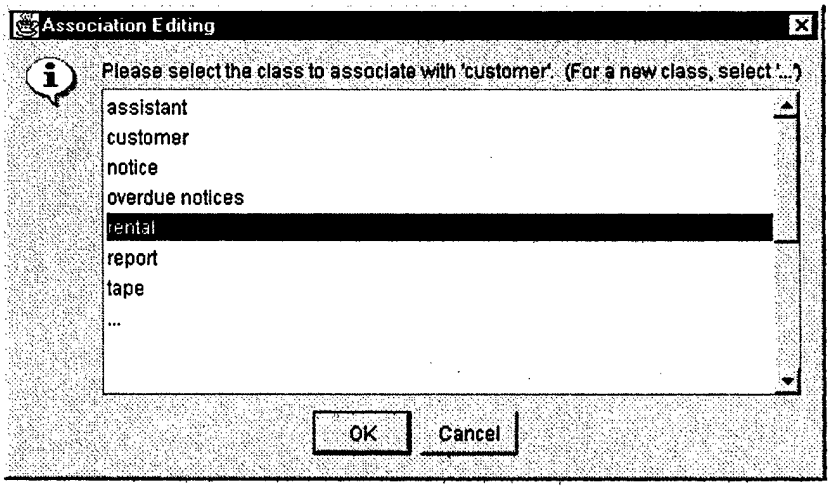


Figure 22. Association Editing Dialog Box

To validate that the model of the "customer" class is as we intended it to be, it's possible to describe the class in English using the LIDA Describe feature. To use this feature, simply click

on the title of the class using the **right** mouse button, and click on the Describe option at the bottom of the menu (Figure 21). This results in the display of an English language explanation of the class, generated by LIDA. The "customer" class is described in Figure 25.

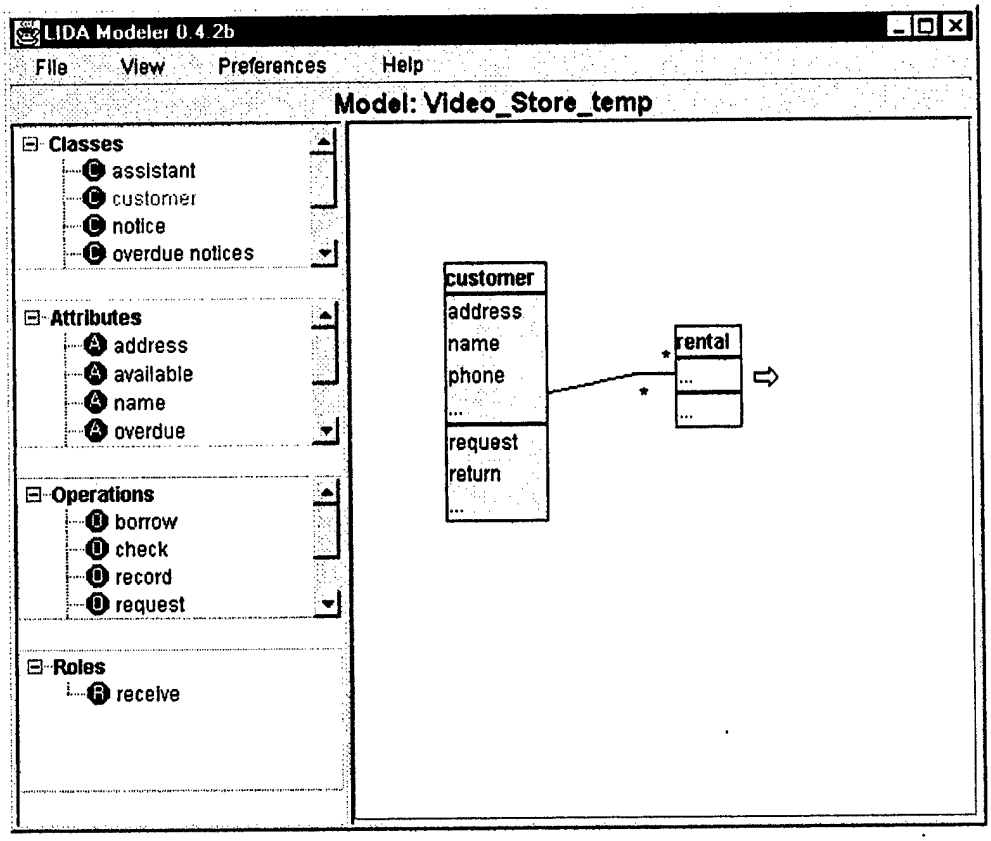


Figure 23. Association between 2 Classes

Also, notice that the right-button menu contains a number of other options that duplicate the methods described in this tutorial, and are available as alternatives that may be more convenient for the user.

27. To continue with model development, associate the remaining classes with their operations and attributes by clicking on the class name in the Classes window, and then either using the right-button menu or the attribute and operation areas in the class icon as we did with the "customer" class. When all of the word lists are used, and any additional attributes and operations have been added that are ***clearly needed*** and ***well understood*** by the analyst, proceed to build the associations between classes as described in Step 26. Most of the identification of additional classes, attributes, and operations will be accomplished in cooperating with users, customers and other domain experts. Now, the initial class model is finished.
28. To continue iterative refinement of the class model, and the development of associated models using UML notation, the basic class model should be exported to a more feature-rich diagramming tools. LIDA supports the export of models from LIDA to Visio™, a diagramming tool that supports the UML notation. Import of models from Visio to LIDA is also supported.

To export a model from LIDA to Visio™, follow the instructions in the LIDA User's Guide, *LIDA Integration with Visio™*, Version 0.4.2 [8]. The Visio model exported from the finished LIDA model in this tutorial is shown in Appendix B.

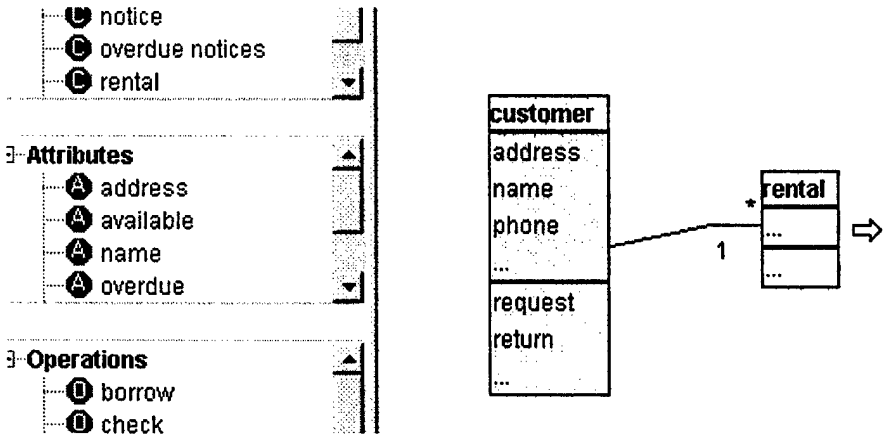


Figure 24. Association with Correct Multiplicity Values

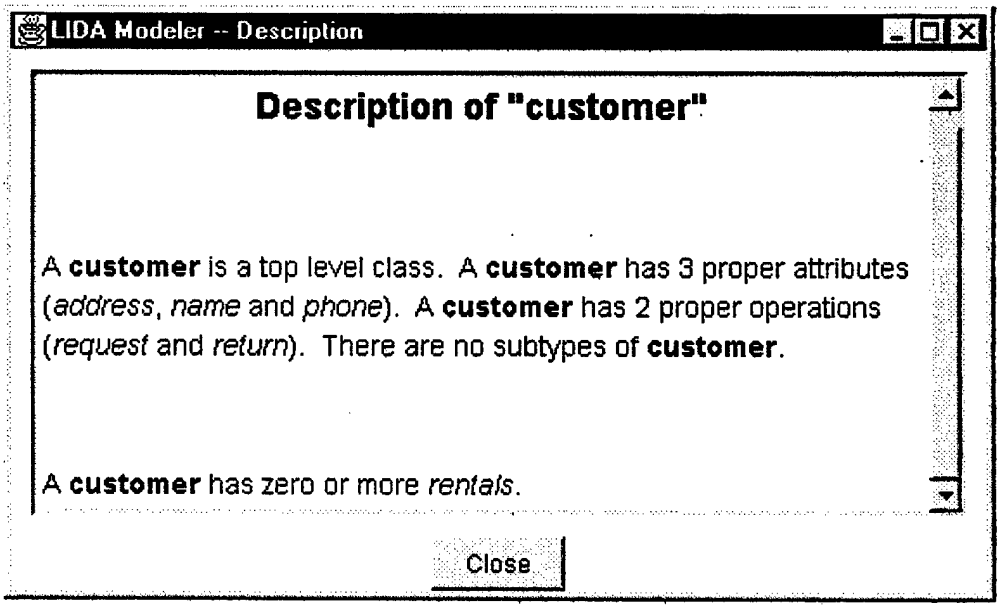


Figure 25. Text Description of the Customer Class

Part 2

Using LIDA to Explain and Assist in Validating an Existing Class Model

This part of the tutorial guides the user through the task of examining an existing class diagram using the import and explanation capabilities of LIDA. An analyst might want to use this capability of LIDA to closely examine, interpret, understand, or validate a complex model that is otherwise difficult to understand on it's face. The model we use for the tutorial is small, but more complex than the model utilized in the previous tutorial scenario.

1. The tutorial starts, as shown in Figure 26, with a model that has been developed or imported to Visio. This model is of an Integrated Employee Training plan development system that has been described in terms of a set of classes, with attributes and methods. The basic idea is that an employee will develop a training plan based upon a set of required skills and employee skill deficiencies, and schedule classes in order to implement the training plan.
2. From the LIDA menu in Visio, click on the Export to LIDA option. You will see the warning message shown in Figure 27. Make sure you read it to see the notational features that are actually exported to LIDA. When you have finished reading the warning message, click the ***OK*** button to dismiss the dialog box.
3. Next, a dialog box is displayed that gives the analyst options for dealing with parameterized classes and class utilities. If there are none of these in the model you are importing (as in this case), or if they are not important to maintain for validation purposes, click on the Do Not Export radio buttons for each class, then click the ***apply*** button. The dialog box is shown in Figure 28.

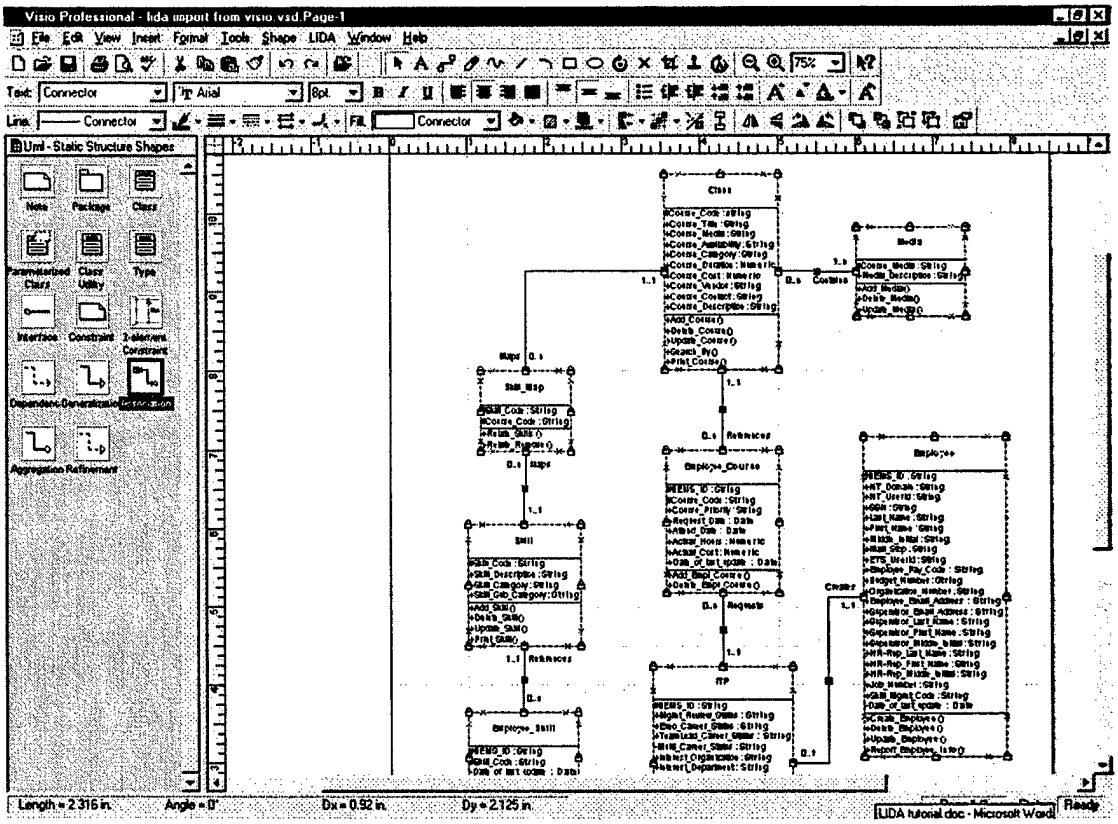


Figure 26. Visio Screen with ITP Model

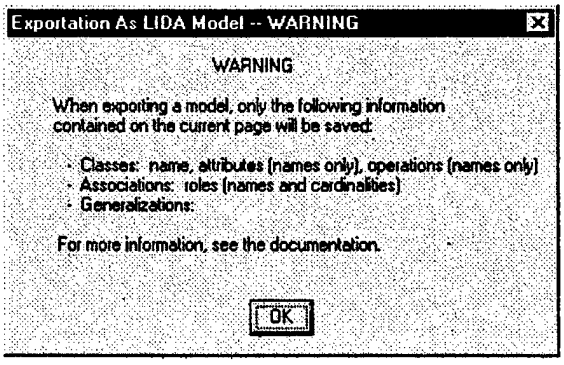


Figure 27. Visio Export Warning

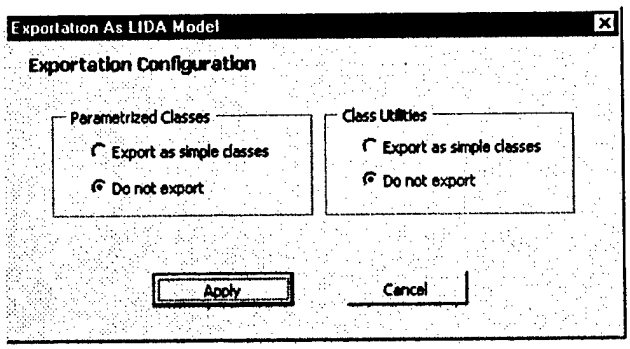


Figure 28. Exportation Configuration Dialog Box

4. Next, the exporter will display a dialog box asking for a name for the LIDA model being exported as displayed in Figure 29. Type in "trainingplan", and click the ***OK*** button. There will be an additional dialog box indicating that model export has been completed. Dismiss this box by clicking the ***OK*** button as well. You may now exit from Visio, since the model has been exported into a LIDA file.

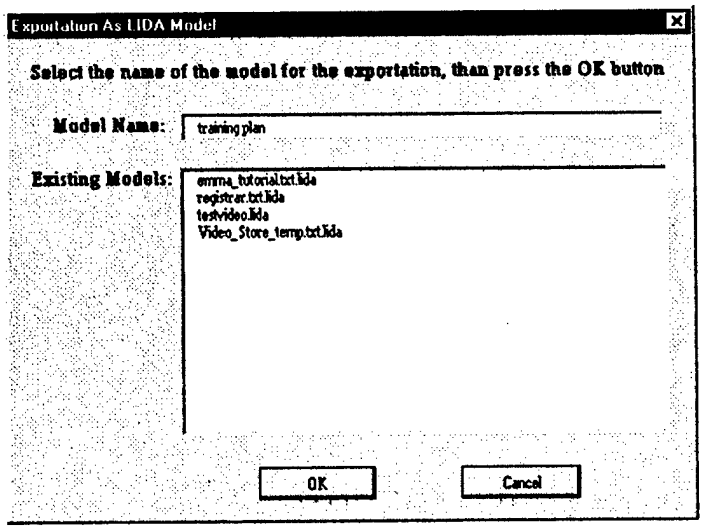


Figure 29. Naming the Exported Model

5. If LIDA is not already running, start it by double-clicking on the LIDA file icon labeled LIDA(.bat).
6. Select ***Load Model*** from the LIDA file menu, and select "training plan.lida" from the resulting "Load Model ..." dialog box as shown in Figure 30, and press the ***Open*** button.

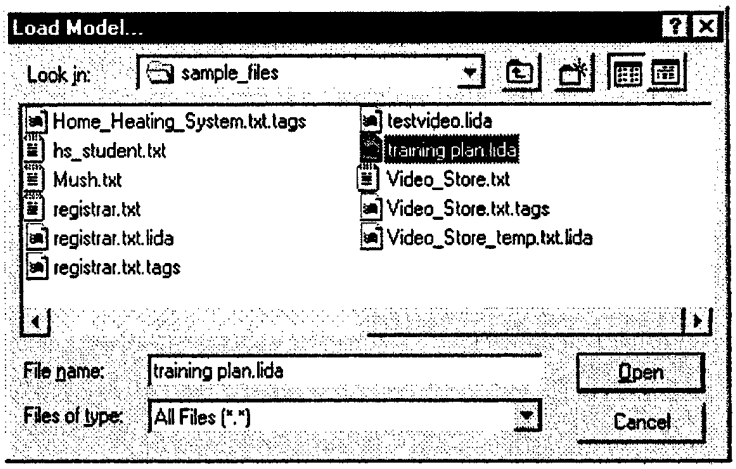


Figure 30. Load Model... Dialog Box

7. When the model has loaded (mouse cursor has reverted to an arrow), click on the Edit Model … option on the LIDA file menu. This will start the LIDA Modeler.
8. Notice that in Figure 31, the Classes, Attributes, Operations, and Roles have been loaded from the exported Visio model. Take a moment to scroll through the all of the lists and view the model components that are available for review.

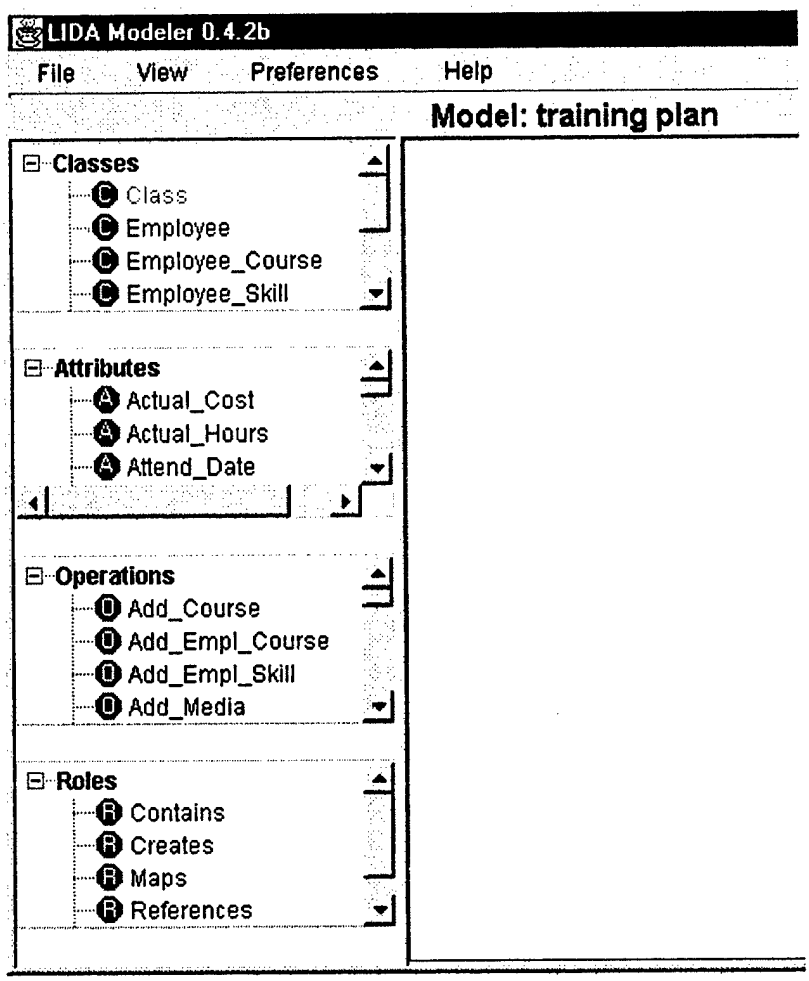


Figure 31. Loaded Model Elements

9. In the list of classes in the "Classes" window, double click on the class named "Class". The result of this action is illustrated in Figure 32. Notice that the "Class", "Skill_Map", "Employee_Course", and "Media" classes are displayed in the working window. NOTE: It may be a good idea to use the mouse cursor to grab the lower-right hand corner of the Modeler window and expand the window to better view the items displayed therein.
10. Examine the various classes, attributes, operations and roles for inconsistencies and ambiguities, as well as for missing elements. Using LIDA, you may decide that an English language description of a particular class would be useful in this process. Right-click on the name of the class of interest (in this example, "Employee_Course"), and select ***Describe*** from the resulting menu by clicking the option with the left mouse button. The resulting description is illustrated in Figure 33.

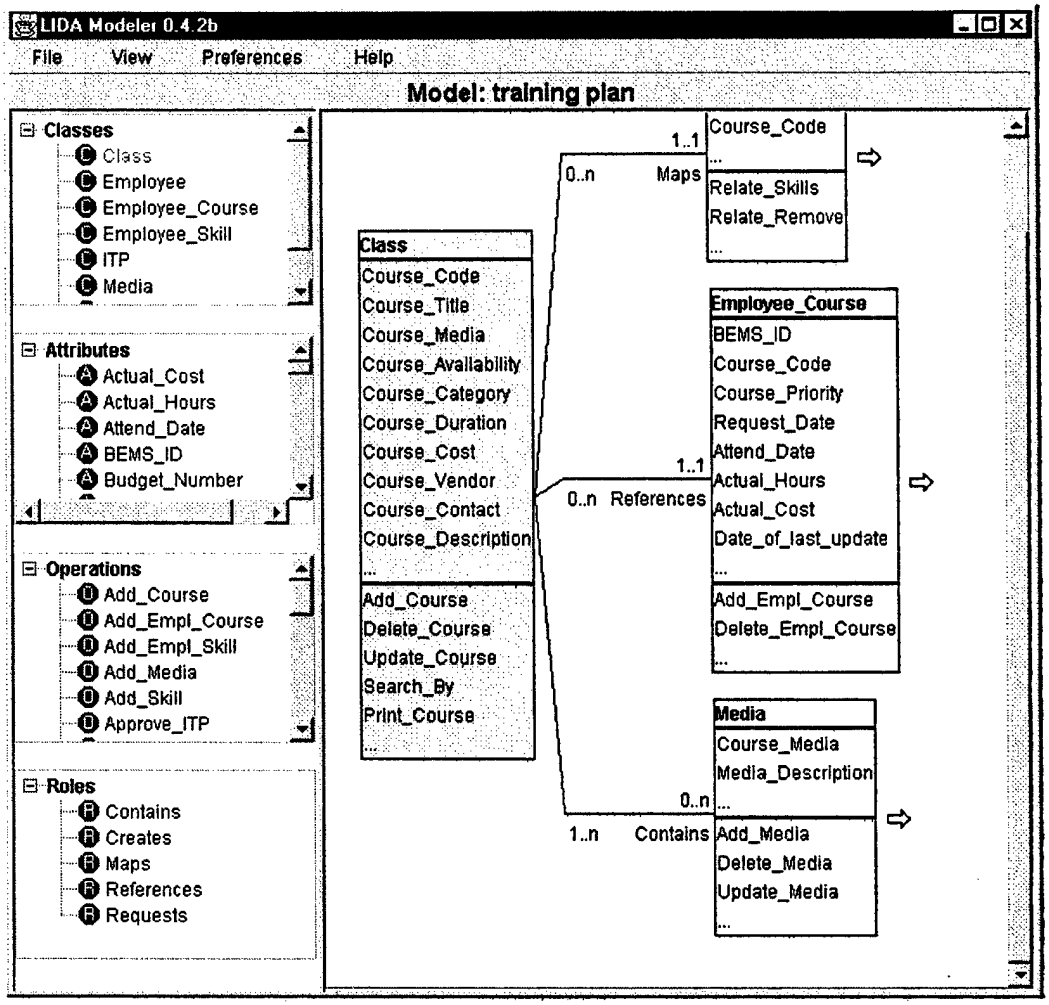


Figure 32. Imported Model with "Class" Displayed

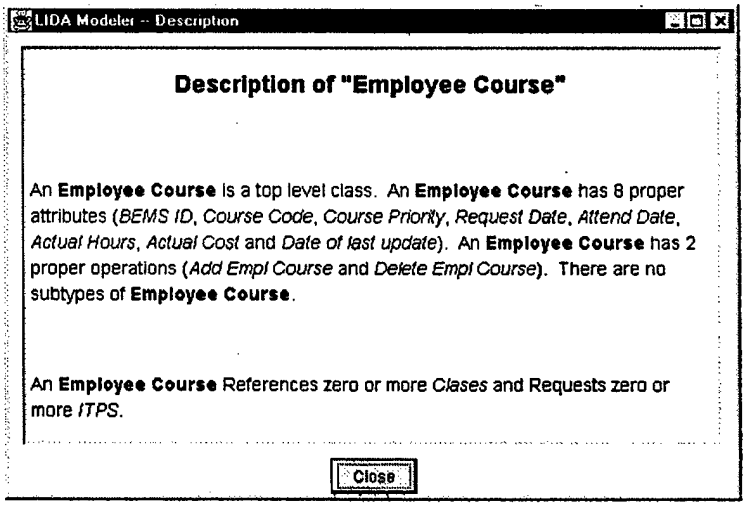


Figure 33. LIDA Text Description of Employee_Course Class

11. This process continues until the analyst is satisfied that all classes are properly defined, with appropriate attributes, operations, associations, roles, and multiplicity constraints.

This tutorial has demonstrated two ways in which LIDA can be used to assist in developing and validating class models for object-oriented analysis and design efforts. The reader is encouraged to experiment with extensions to the scenarios outlined in these tutorials, and discover as yet untried ways of further utilizing the capabilities provided by LIDA and the LIDA Modeler.

References

[1] Bellin, David, Simone, Susan Suchman, and Booch, Grady (1997) The CRC Card Book (Addison-Wesley Object Technology Series), New York:Addison-Wesley, ISBN: 0201895358.

[2] Booch, Grady, Jacobson, Ivar, and Rumbaugh, James (1998) *The Unified Modeling Language User Guide* (The Addison-Wesley Object Technology Series), Addison-Wesley, ISBN: 0201571684.

[2a] CoGenTex, Inc. (1999) LIDA User Guide. CoGenTex Technical Report. Ithaca, NY.

[3] Booch, Grady (1994) *Object-Oriented Analysis and Design With Applications* (Addison-Wesley Object Technology Series), New York:Addison-Wesley, ISBN: 0805353402.

[4] Peter Coad, Edward Yourdon (1991) *Object-Oriented Analysis* (Yourdon Press Computing Series), New York:Yourdon Press; ISBN: 0136299814.

[5] Jacobson, Ivar (1994) *Object-Oriented Software Engineering : A Use Case Driven Approach* (Addison-Wesley Object Technology Series), New York:Addison-Wesley, ISBN: 0201544350.

[6] Odell, James J. and Fowler, Martin (1998) *Advanced Object-Oriented Analysis and Design Using UML* (SIGS Reference Library , No 12), SIGS Books & Multimedia; ISBN: 052164819X.

[7] Schneider, Geri, Winters, Jason P., and Jacobson, Ivar (1998) *Applying Use Cases : A Practical Guide* (Addison-Wesley Object Technology Series) New York:Addison-Wesley Pub Co; ISBN: 0201309815.

[8] CoGenTex, Inc. (1999) LIDA Integration with Visio™. CoGenTex Technical Report. Ithaca, NY.

[9] Chen, P.P-S. (1983) English Sentence Structure and Entity-Relationship Diagrams. *Information Systems*, 29.

[10] Kristen, G. (1994) *Object Orientation: The KISS-method: From Information Architecture to Information System.* Addison-Wesley.

[11] Tseng, F.S.C., Chen A.L.P., and Yang, W-P. (1992) On Mapping Natural Language Constructs into Relational Algebra through E-R Representation. *Data and Knowledge Engineering,* 9.

[12] Rolland, G. and Proix, C. (1992) A Natural Language Approach to for Requirements Engineering. In P. Loucopoulos, ed., *Proceedings of the 4th International Confernece on Advanced Information Systems Engineering*. Springer-Verlag, Manchester.

[13] Burg, J.F.M. and van de Riet, R.P. (1996) Analyzing Informal Requirements Specifications: A First Step towards Conceptual Modeling. In: *Proceedings of the 2nd International Workshop on Applications of Natural Language to Information Systems*, Amsterdam, The Netherlands. IOS Press.

Appendix A - Sample Tutorial Materials

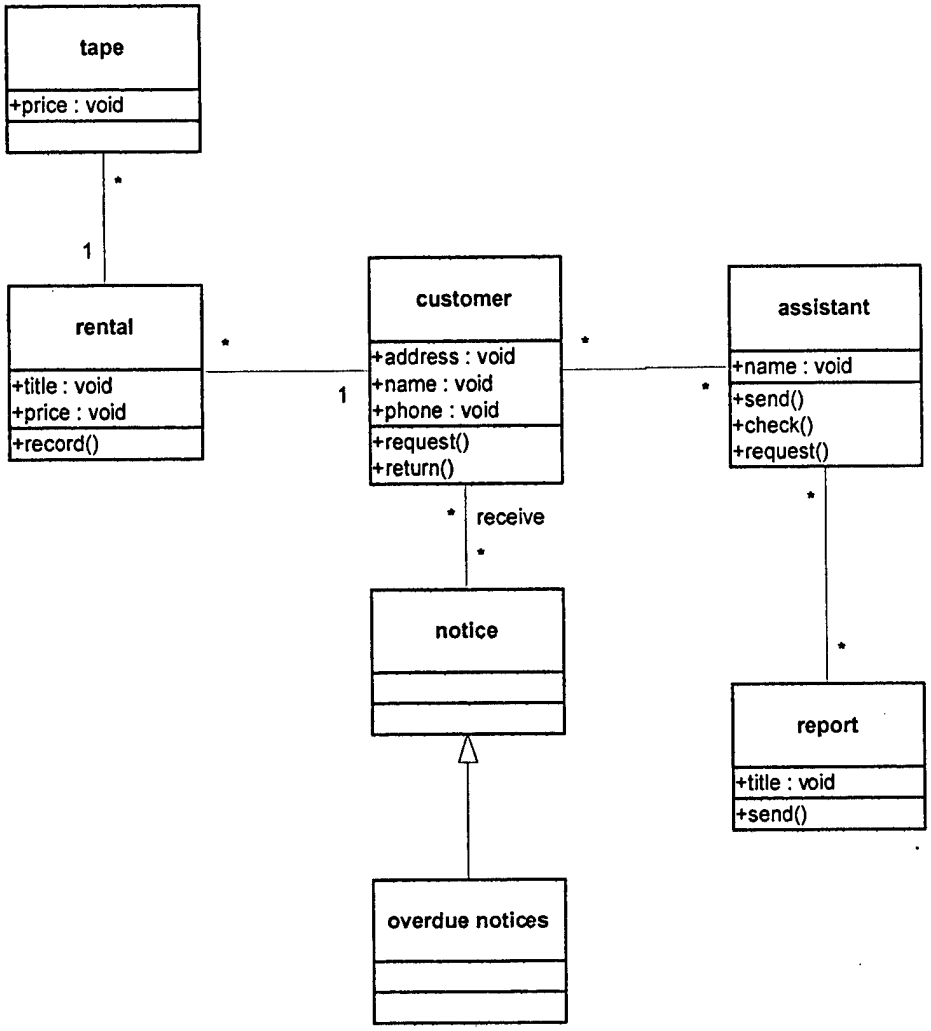
Video Store Operational Concept

The Video store receives tape requests and returned tapes from customers and new tapes from the Main office. Whenever necessary the Video store sends tape overdue notices to its customers. Tape requests are taken care of by the store assistant. While treating these tape requests he uses information concerning films and tapes. He also uses and updates the list of rentals. The store assistant also takes care of tape returns. For this task the store assistant again needs the information concerning films and tapes as well as the list of rentals. Here too the list of rentals is updated.

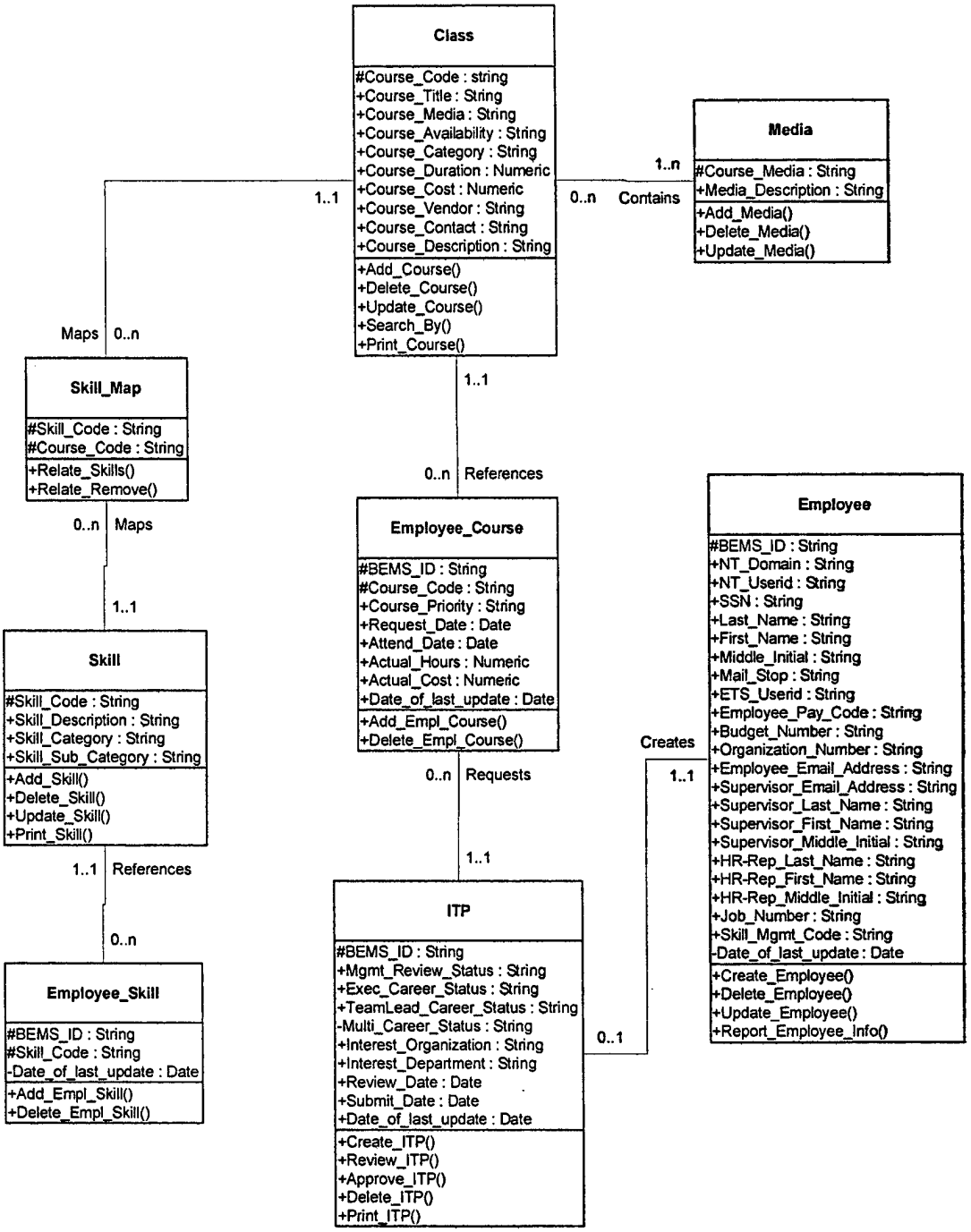
The other activities are the submission of rate changes and the submission of new tapes by the store management and the production of rental reports and the sending of tape overdue notices by the store administration.

Film request is taken care of by the store assistant. Whenever a customer requests a film, the store assistant first checks whether there are still tapes, containing the requested film, available and then he searches, if necessary, for the rental price. Next he checks whether this customer is late with the return of other tapes. If so he may not borrow the tape requested. If this is not the case, he records this rental by updating the rental list. For this, the customer is asked for his address.

LIDA Model Imported to Visio™



Visio™ Model Imported to LIDA



MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.