Information and Software Technology 45 (2003) 83–93

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

From use cases to classes: a way of building object model with UML

Ying Liang*

Department of Computing and Information Systems, University of Paisley, Paisley PA1 2BE, UK

Received 12 July 2002; revised 4 September 2002; accepted 12 September 2002

Abstract

In a use case-driven process, classes in the class diagram need to be identified from use cases in the use case diagram. Current object modelling approaches identify classes either from use case descriptions, or using classic categories. Both ways are inefficient when use cases can be described with many scenarios in different words. This paper represents a new approach that identifies classes based on goals of use cases without descriptions. The approach produces use case-entity diagrams as a vehicle for deriving classes from use cases and to show the involvement of classes in use cases of a system.

Keywords: Object modelling; Use case; Use case's goal; Class; UML

1. Introduction

Unified Modelling Language [5,25] (UML) is a visual modelling language adopted as a standard for object-oriented modelling and design in software development by the industry body Object Management Group (OMG). It was created mainly based on three object modelling techniques and methods (Booch [4], OOSE [12], OMT [24]) that have been used in industry for many years. It was also influenced by other techniques such as statecharts invented by Harrel [9] and programming languages such as C++ [30]. UML has been evolved with many versions since it started officially in 1994 [5]. The latest release is UML 1.3 which this paper refers to. UML consists of a set of diagrams that are used to represent models of a system in diagrammatic notation through stages of development, as shown in Table 1. Sometimes, sequence diagrams, collaboration diagrams and activity diagrams may be also used for assisting to represent use case descriptions at requirements capture stage.

Each of models is a blueprint of a system with a view that enforces the model to focus on entities in a given perspective of the system and ignore entities not relevant to the perspective [5]. The views with UML are relatively orthogonal so that each of the diagrams can show one aspect of a system. This helps solve the problem with a notation that tries to cover every aspect of a system and cannot be described as simple and easy to understand [11]. To show clearly when and where different kinds of diagrams are produced, they should refer to each other in interesting and useful ways and allow CASE tool vendors to provide coherent cross-referencing and consistency checking [10]. Such a way is normally defined as a modelling process in an object-oriented method [17,18,20,22].

Although UML is a modelling language without a modelling process, many people think that a use case-driven process is an effective way of modelling a system with UML [1,2,6,13,14,16,31] as it enables to show functional requirements using a use case diagram at requirements capture stage and to generate the other diagrams as requirements specification from this diagram at analysis and design stage. Functional requirements mean what services the system should provide in particular situations [28] and the use case diagram is thought good and useful in collecting functional requirements [19]. Use cases in the diagram are thought easier than objects for users to join in object modelling [2,5,15,21,23]. A use case diagram for a system includes uses cases and actors: use cases represent the functional requirements of the system, and actors represent the units or people in the organisation that are expecting to receive the values or responses delivered by the use cases. Fig. 1 is an example of such a diagram that has an Enrol course use case and two actors Course Admission

* Tel.: +44-141-848-3886; fax: +44-141-848-3542.
E-mail address: lian-ci0@paisley.ac.uk (Y. Liang).

0950-5849/03/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0950-5849(02)00164-7

Table 1
Diagrams in UML

Diagram	Model	View	Stage
Use case diagrams	Use case model	Use case view	Requirements capture
Class diagrams	Analysis model	Modelling and design view	Analysis and design
Sequence diagrams	Design model		
Collaboration diagrams			
Statecharts			
Activity diagrams			
Component diagrams	Implementation model	Implementation view	Implementation
Deployment diagrams	Deployment model	Deployment view	

and Academic Department in an education system. In general, all use cases of a system are at the same level of abstraction in the use case diagram [21].

In a use case-driven process, mapping use cases in the use case diagram into classes in the class diagram is a key activity [7]. Currently there are two typical approaches for carrying out the mapping-use case descriptions and classic categories [1,8]—which we call noun-oriented object modelling approaches in this paper:

- The first approach identifies candidate classes by considering all nouns and noun phrases in use case descriptions as candidate classes (e.g. A *student* asks the *Course Admission* for enrolling a *computing course* in an *education system*), and then selecting significant classes from the candidates [24].
- The second approach identifies classes by checking if nouns in use case descriptions fall into classic categories such as tangible things, roles, concepts, events, and so on [4,26,27]. For example, 'student', and 'course' in the above description are classes as they fall into roles and concepts categories.

A common problem in object modelling with these approaches is the discovery of too many classes at one time, for example, make every noun a class [1]. Our experiences of using these approaches in case studies and teaching UML also showed that the approaches had following problems in object modelling:

- A description of a use case may show one of its scenarios. This means that it shows one way that may be not the only way of achieving the goal of the use case [29]. The classes identified using this approach could be limit to this description only. Also different classes can be identified for the same use case when different words are likely used by different people when describing it.
- There might be too many nouns and noun phrases in a use case description to select significant classes from them easily.
- Different people may use different types of classic category in classifying the things in a system. This can cause conflict and difficulty of making agreement on classes.
- Something in a system may fall into many classic categories when it plays different roles in different parts. This makes use of classic category more difficult.

In addition, requirements elicitation includes elicitation of both functions and entities in a system. Use case descriptions emphasise function elicitation only. Therefore the analyst cannot elicit entities effectively from them. Also these approaches let the analyst rather than actors identify classes that often represent the interest of the analyst rather than the interest of actors. The classes identified by these approaches therefore may not be acceptable to the actors. All these problems may cause the representation of user requirements inappropriate and vague.

The following section shows a new object modelling approach to mapping use cases into classes in a different way that intends to improve the effectiveness of generating class diagram from use case diagrams and the quality of requirements specification.

2. A goal-oriented approach to building the object model

The principle of this approach is to identify classes from use cases' goals rather than use case descriptions with the use case-driven process. Classes are the entities that participate in achieving the goals in the real world. They have their own features and can collaborate with use cases. Two entities refer to each other when one is created based on or act on the other in use cases. Goals and entities must be described by actors of use cases rather than analysts of the system because they are part of user requirements gathered from the actors. The approach provides a seven-step process to generate the class diagram from the use case diagram for a system, as the route illustrated in Fig. 2. Steps are represented using a circle containing a step number. Grey arrows are used to link the steps to the part of

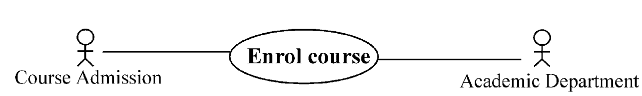

Fig. 1. A use case diagram.

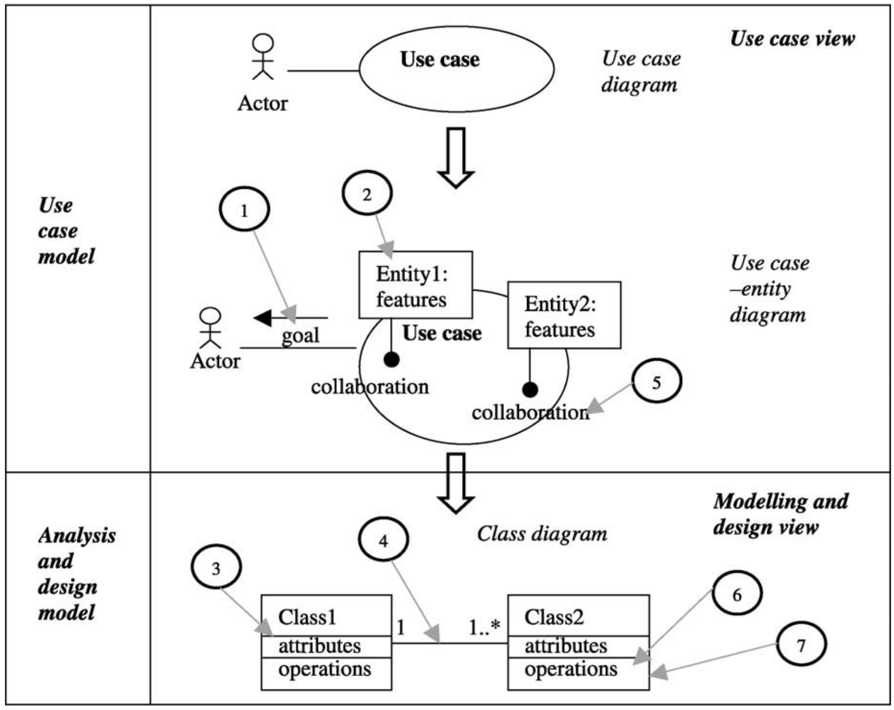

Fig. 2. Generating class diagram from use case diagram.

the diagrams they work on. More details of the steps are described below.

2.1. Step 1: determine the goal of each of use cases

Use cases are popular because they focus on functions that meet users' goal [3]. A user's goal means the goal of a use case: it is to yield observable result of a value to an actor [5]. Actors of use cases have goals (or needs) and use applications to help satisfy them [14]. The goal of a use case determines the scope of the use case. The goal of a use case meant in the approach is defined as follows:

> The *goal* of a use case is the objective of the use case's effort within a system. The goal usually is seen as either values that the use case supplies to its actors or responses that the use case replies to its actors. The goal is unique to a use case, no matter how it is achieved through routes of the use case.

This definition is consistent with the role of a use case in UML: it shows what a system does but not how it does [5]. The use case's goal must be described by actors of the use case. Following questions are asked to the actors when identifying the goal:

- Do you think what is the purpose of the use case?
- Which service do you expect the use case to provide?
- Which value or response do you expect the use case send to you?

The approach shows the goal of a use case explicitly using a use case-entity diagram. For example, two actors of the Enrol course use case in Fig. 1 described the goal of the use case as *new registration* that is a value delivered to the Course Admission and *the copy of a new registration* that is a response sent to the Academic Department. The goal is explicitly represented in a use case-entity diagram Fig. 3, using two goal flows that are attached by a value or a response. The goal of a use case in a system should be a goal of the system.

2.2. Step 2: identify entities using the goal of use cases

A use case involves entities in achieving its goal within a system. An *entity* is something that plays a role in the use case for achieving the goal. It may be involved in many use cases and have different roles in a system. It generally has its own features that make it different from other entities, and is involved in a use case by collaborating with the use case. Identify entities by asking actors of the use case the

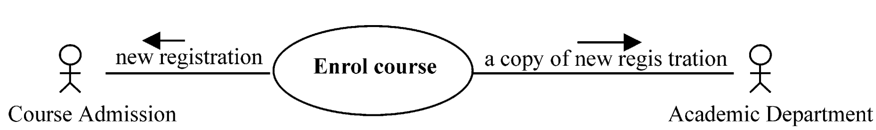

Fig. 3. A use case-entity diagram representing the use case's goal.

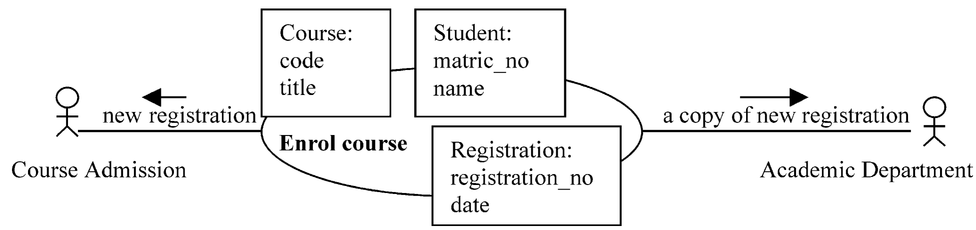

Fig. 4. Enrol course use case and entities involved.

following questions:

- What does the goal mean to you?
- Which terms do you think common in the vocabulary of the use case and the problem domain?
- Which entities do you think necessary and important to achieve the goal of the use case?
- Which specific features of the entities do you think in particular necessary and important to achieve the goal?

For example, the actors in Fig. 2 described that (a) the goal means a new registration on a specific course for a specific student; (b) 'registration', 'course', and 'student' are common terms in the vocabulary of enrolling course in an education problem domain; (c) the course and the student are necessary and important to generate a new registration; and in particular (d) registration_no, current date, course code, title, student matric_no, and name are specifically necessary and important to generate the new registration. The common terms described above mean the entities that are significant to generate new registration and must be specified for the system using a use case-entity diagram, i.e. Fig. 4. Rectangle boxes show those entities and their features that are involved in the use case particularly. A colon (:) is used to separate the entity name and the features.

In general, different use cases have different goals because they represent different functional requirements of a system. Therefore a use case may involve part, but not whole, of features of an entity when the entity is involved in many use cases of a system. For instance, there is another use case 'Create course' in the education system whose goal is *new course* described by the actor Academic Department. By asking the above questions, the actor also described that (a) the goal means a new course led by a specific course leader; (b) 'course' and 'course leader' are common terms in the education system domain; (c) the course and the course leader are necessary and important to create a new course; (d) course code, title, level, course leader's name, title, post, office and tel_no are in particular necessary and important to create the new course. Those identified entities and their features are shown in another use case-entity diagram, Fig. 5, in which the number of the Course's features has been increased from two to three. The whole features of an entity will be represented together within a class in a class diagram in the next step.

2.3. Step 3: specify entities as classes and their features as attributes of the classes in a class diagram

This step is to represent the entities and their features shown in use case-entity diagrams as significant classes and attributes in a class diagram (Fig. 6). They are regarded as significant classes because they must participate in achieving the goals of the system.

2.4. Step 4: specify references between entities as association relationships between classes

A use case may involve many entities, as shown in Figs. 4 and 5, that may have to refer to one another in achieving the use case's goal. Identify the references by asking actors the following questions:

- For each entity, do you think that it is produced based on, or its existence depends on, other entities in the use case? If so, it should refer to those entities.
- For each entity, do you think that it can act on, or be acted by, other entities in the use case? If so, it should refer to those entities.

For example, the actors described the references between the entities in Figs. 4 and 5 and multiplicity of the references as (a) a registration is generated based on a specific course for a specific student, and a course is created with a specific course leader; (b) a student can apply for registration on

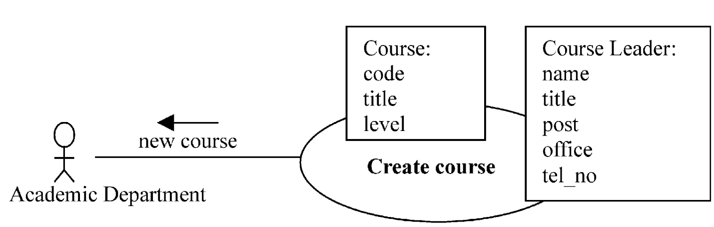

Fig. 5. Create course use case and entities involved.

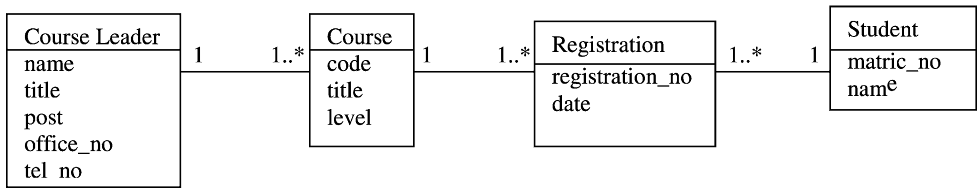

Fig. 6. Classes and association relationships.

different courses, a course can be acted by many registrations and one course leader, and one course leader can act on many courses. These references are represented as association relationships between classes in the class diagram, Fig. 6. The multiplicity (i.e. one to many, or many to many, etc.) of a reference is shown as the multiplicity (i.e. 1:1..*, or 1..*:1..*) of an association relationship.

2.5. Step 5: identify collaboration between use cases and entities

The entities involved in a use case are expected to collaborate with the use case for achieving its goal. Identify collaboration by asking actors the following questions:

- Do you think what the use case needs to do with each entity?
- Do you think what the use case needs to know about each entity?
- Do you think what each entity must contribute to the use case?

For example, the actors in Fig. 4 described (a) The use case assigns a matric_no to the student; it needs to know the name of the student; and the student provides its name to the use case. (b) The use case needs to know the code and title of the course; and the course provides the code and tile to the use case. (c) The use case creates a new registration with the current date and a unique registration number.

The collaboration identified above is added to the use case-entity diagram, as shown in Fig. 7. Each solid line shows the collaboration between the use case and an entity. One end of it links to a black ball with the collaboration, and another end of it links to the entity that fulfils the collaboration.

Identification of collaboration can help understand roles and responsibility of entities in the use case and also help check the references between the entities.

2.6. Step 6: specify collaboration as operations of classes

The collaboration between entities and a use case means what the entities must do for achieving the use case's goal. Thus they are significant operations of the classes and are specified in the class diagram explicitly, as shown in Fig. 8.

2.7. Step7: test use case-entity diagrams and class diagram

The approach tests classes in the class diagram against individual use cases instead of the whole system. This helps reduce the complexity of testing classes. The following guides can be used in testing the diagrams:

- Test use cases against functional requirements: check if (a) each of use cases represents a functional requirement, (b) all such requirements are covered by the use cases, (c) the goal of each use case is same as described by the actors.
- Test entities against each of use cases: check if (a) the entities are sufficient and necessary to achieve the goal, (b) features of each entity are what the use case needs to know about the entity, (c) collaboration linked to the entities are sufficient and necessary to achieve the goal of the use case.
- Test classes against the use case-entity diagrams: check if (a) the classes map all entities in the diagram, (b) attributes of a class are all features of the entity in different use case-entity diagrams, (c) operations of a class are entire collaboration between the entity and use cases, (d) each association relationship between two classes represents a reference between two entities, and (e) the multiplicity of it is same as the multiplicity of the reference.

The order of the seven steps can be either sequential or iterative in object modelling. You can go through seven steps together on all use cases at one time, or alternatively,

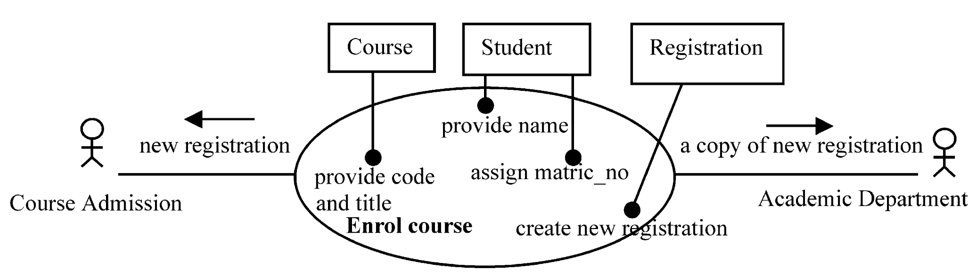

Fig. 7. Collaborations between Enrol course use case and entities involved.

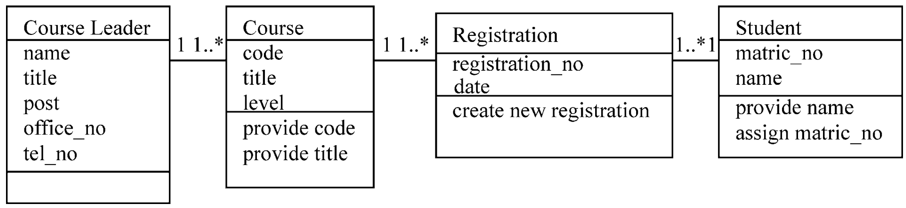

Fig. 8. Operations of classes.

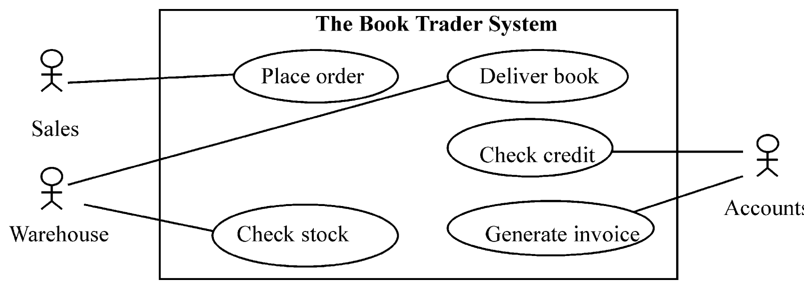

Fig. 9. A use case diagram for the book trader system.

go through them on one use case at one time and then repeat them again and again on the rest use cases.

3. A case study: the book trader system

This approach has been applied for a few case studies. This section shows one of them as an example—a book trader system whose use case diagram is shown in Fig. 9.

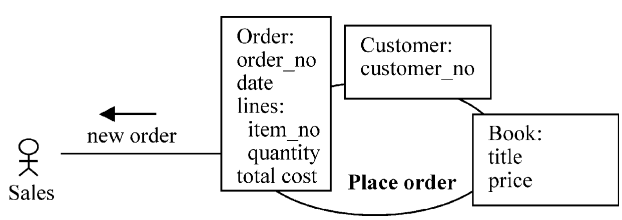

Fig. 10. Place order use case and entities involved.

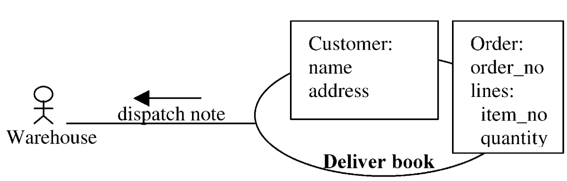

Fig. 11. Deliver book use case and entities involved.

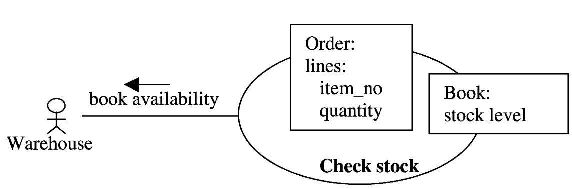

Fig. 12. Check stock use case and entities involved.

In Step 1, actors described (a) the *new order* is the goal of Place order (Fig. 10), (b) the *dispatch note* is the goal of Deliver book (Fig. 11), (c) the *book availability* is the goal of Check stock (Fig. 12), (d) the *credit status* of a customer

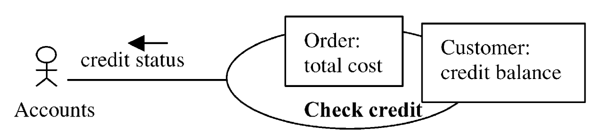

Fig. 13. Check credit use case and entities involved.

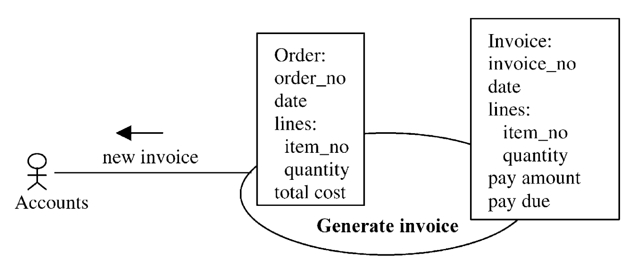

Fig. 14. Generate invoice use case and entities involved.

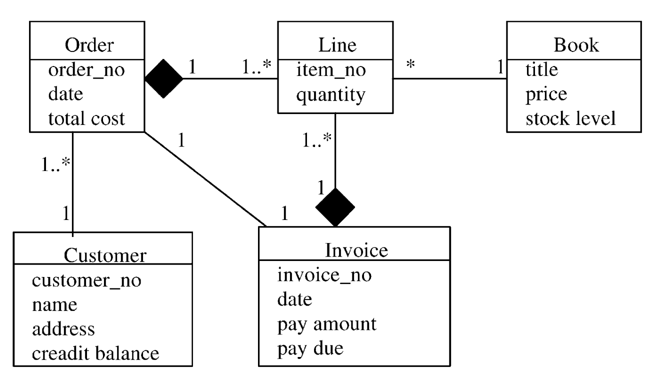

Fig. 15. A class diagram for the book trader system.

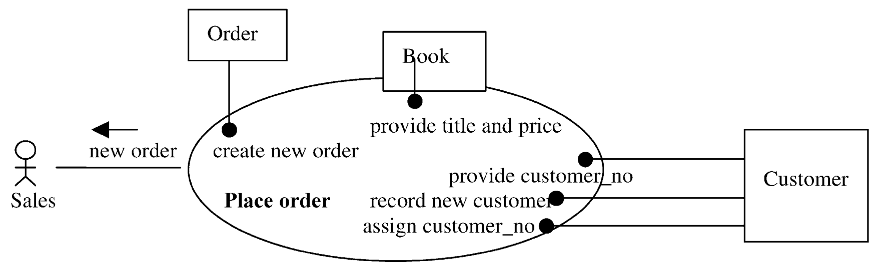

Fig. 16. Collaboration between the Place order use case and entities involved.

is the goal of Check credit (Fig. 13), and (e) the *new invoice* is the goal of Generate invoice (Fig. 14).

In Step 2, the actors described the entities and their features involved in achieving the goals of the use cases as follows:

(a) The goal of Place order means a new order to be created on books for a customer. Each order line is created on one book. The common terms and entities are 'order', 'customer' and 'book'. The features of the entities that are involved in this use case are as described in Fig. 10. Many orders can act on the same book or on the same customer.

(b) The goal of Deliver book means a dispatch note that is a copy of an order referring to the name and address of a customer. The common terms and entities involved in this use case are 'order' and 'customer'. The features of them involved in the use case are as described in Fig. 11.

(c) The goal of Check stock means the book availability that depends on an order line and a book. The common terms and entities in this use case are 'order' and 'book'. The current stock level of the book and the item_no and quantity of an order line are particularly involved as shown in Fig. 12.

(d) The goal of Check credit means the credit status that depends on an order and a customer. The common terms and entities involved in this use case are 'order' and 'customer'. The features of them involved in this use case in particular are as described in Fig. 13.

(e) The goal of Generate invoice means a new invoice to be created for an order. The common terms and entities involved in this use case are 'invoice' and 'order'. The features of them involved in this use case are as described in Fig. 14.

In Step 3, the above entities and their features are represented as significant classes and attributes in a class diagram, Fig. 15.

In Step 4, the references between entities are represented by association relationships between the classes in the class diagram, Fig. 15: (a) an order refers to a specific customer as it is created for the customer. (b) Many orders may refer to one customer if they are created for the same customer. (c) An invoice refers to an order exactly as it is created for that order only. (d) An order and an invoice may consist of many lines each of which is created on one book and refers to that book only. In this situation, the class diagram can represent the order and invoice and their lines separately with a special association relationship-whole-part relationship represented by a line ended by a black diamond, according to UML [5]. (e) Lines in different orders or invoices may refer to the same book.

In Step 5, the actors described collaboration between the use cases and the entities as follows:

(a) Collaboration between Place order use case and the entities: the use case creates a new order including lines. The use case needs either to know the customer number, or to record a new customer and assign a customer number to it. Customer needs to provide its number to the use case. The use case needs to know the title and price of each ordered book and Book must provide these features to it. The collaboration is shown in Fig. 16.

(b) Collaboration between Deliver book use case and the entities: the use case needs to know entire detail of Order and Order must provide its whole features to it. The use case needs to know the customer name and address; Customer must provide these features to it. The collaboration is illustrated in Fig. 17.

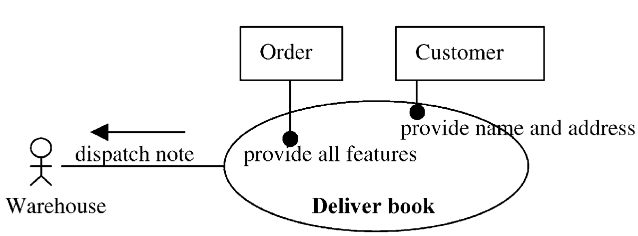

Fig. 17. Collaboration between Deliver book use case and entities involved.

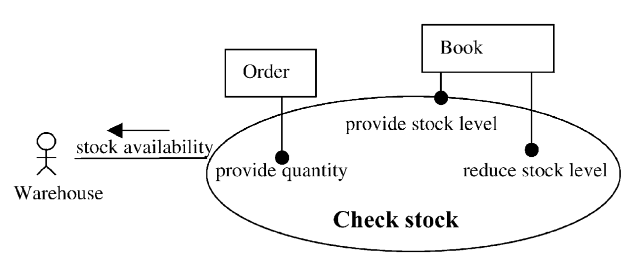

Fig. 18. Collaboration between Check stock use case and entities involved.

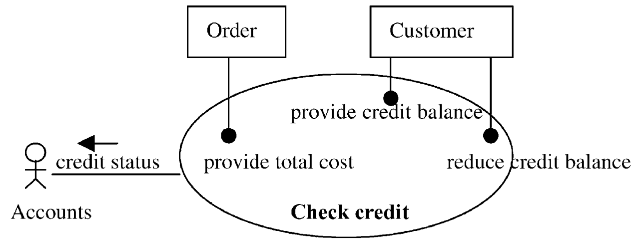

Fig. 19. Collaboration between Place order use case and entities involved.

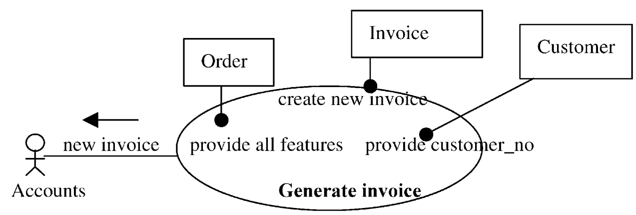

Fig. 20. Collaboration between Generate invoice use case and entities involved.

(c) Collaboration between Check stock use case and the entities: the use case needs to know the quantity in order lines; Order needs to provide this feature to it. The use case needs to know the current stock; Book must provide this feature to it. The use case needs to reduce the stock level when an order line is accepted. The collaboration is recorded in Fig. 18.

(d) Collaboration between Check credit use case and the entities: the use case needs to know the total cost of the order; Order needs to provide this feature to it. The use case needs to know the current available credit of the customer; Customer must provide the feature to it. The use case reduces the credit balance when the order is accepted. The collaboration is shown in Fig. 19.

(e) Collaboration between Generate invoice use case and the entities: the use case creates a new invoice. It needs to know entire details of the order; Order must provide whole features to it. The use case needs to know the customer number; Customer must provide this feature to it. The collaboration is shown in Fig. 20.

In Step 6, all collaboration described are specified as operations of the classes in the class diagram, Fig. 21.

4. Comparison and evaluation

The class diagram in UML is often generated from the use case diagram of a system through a use-case driven process. The most popular approaches carrying out this process are noun-oriented [13] object modelling approaches as discussed above. Other less popular approaches that also carry out this process are scenario-oriented [14] object modelling approaches that identify classes from events and interactions in use case descriptions. The goal-oriented object modelling approach presented in this paper aims to carry out the use case-driven process in a different way, that is, it identifies classes, attributes, operations, and relationships from use cases' goals rather than from nouns or events and interactions in use case descriptions. Table 2 compares them in details.

The goal-oriented modelling approach was applied in a few small and medium size case studies for the purpose of evaluation and comparison with other object modelling approaches. Table 3 evaluates the modelling process with the approaches by considering the effectiveness of developing class diagrams and the stability of the diagram, based on the experiences of the case studies.

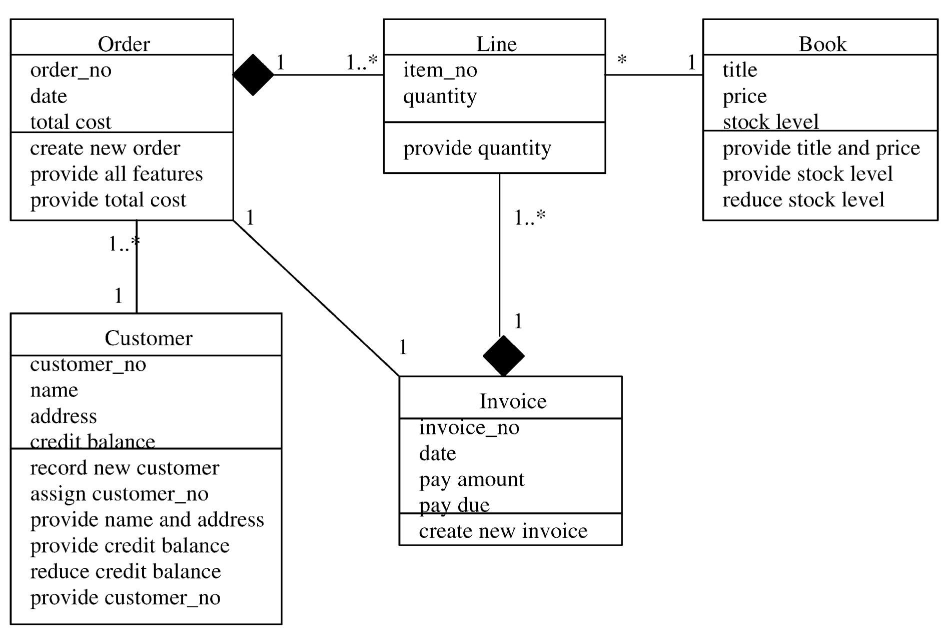

Fig. 21. Operations of the classes in the book trader system.

Table 2
Comparison of the approaches generating class diagram from use case diagrams

	Goal-oriented	Noun-oriented	Scenario-oriented
With use case diagram	Actors describe goals of use cases	Analysts write use case descriptions	Analysts write use case descriptions
Generate class diagram	Identify classes and attributes from use cases' goals Identify relationships between classes from references Identify operations from collaborations between use cases and classes	Identify classes and attributes from nouns in descriptions Identify relationships between classes from verbs Identify operations from attributes	Identify classes and attributes from events/interactions Identify relationships between classes from interactions Identify operations from events/interactions

Table 3
Evaluation of the processes of generating class diagram from use case diagrams

	Goal-oriented	Noun-oriented	Scenario-oriented
Effectiveness of the process	No need of finding candidate classes Goals of a use case do not depend on a route through the use case. Thus classes identified are not specific to one route Must draw class diagram based on use case-entity diagrams	Need of finding candidate classes A use case description depends on words used. Thus classes identified are specific to a description Cause confusions when a noun falls into many classic categories and when a noun may be a class or an attribute	No need of finding candidate classes A scenario of a use case depends on a route through the use case. Thus classes identified are specific to a route May draw class diagram based on sequence or collaboration diagrams
Stability of class diagram with the process	Stable as the goal of a use case is normally unique	Instable if existing many descriptions for a use case	Instable if existing many routes through a use case

Table 4
Experiences of using the approach in case studies

	Experiences
Goal-oriented process	Seven steps are easy to follow Goals help produce class diagram effectively as irrelevant classes and attributes and operations are impossible to be picked Not need prepared use case descriptions Goals help identify system boundary
Use case-entity diagrams	Make it easy to identify significant classes directly without wasting time on candidate classes Useful to identify proper attributes and operations for classes Enable to precisely illustrate how individual classes are involved in use cases They take time to draw, compared with other approaches. However, they can significantly reduce time of producing class diagram
Class diagram	Straightforward to generate it from use case-entity diagrams Easy to test it by referring to and walking through use case-entity diagrams

The case studies also showed that the scenario-oriented approach seemed closer to the goal-oriented approach than the noun-oriented approach in the sense that they both are very concerned about fulfilment of use cases. However, the new approach does not have a conflict with the reality in which most use cases have alternate routes in a system and a use case description only covers a specific route [6], as it uses goals to focus on what actors expect from a use case and ignore the routes through the use cases that produce output.

The case studies were done by people who also know other object modelling approaches with the use case-driven process. The experiences obtained through the case studies are listed in Table 4, regarding the approach, use case-entity diagrams and class diagram.

The case studies also showed that, similar to noun-oriented or scenario-oriented object modelling approaches, there might be many entities involved in a complex use case and this makes a difficulty of drawing use case-entity diagrams. In this situation it is necessary to partition the complex use case into small and simple ones, so that the complexity of the modelling can be reduced and controlled.

5. Conclusions

This paper represents a new goal-oriented object modelling approach that carries out the use case-driven process with UML. The approach identifies classes for a system from use cases' goals. It provides the use case-entity diagram for bridging the gap between the use case diagram and the class diagram, so that the derivation from the former to the latter is visible and effectively traceable and testable. It allows actors of use cases to describe the goals of use cases and the entities involved in achieving the goals, although the analyst can make recommendations based on their past experiences on developing other similar systems.

This paper shows an example of applying the approach in object modelling. Generalisation relationships are, however, not emphasised and discussed in particular in generating the class diagram with this approach, as they are best identified by the analyst rather than by actors in the modelling process. The author thinks that the approach in general has following special characteristics in comparison with other object modelling approaches that also carry out a use case-driven process:

(a) It focuses on use cases' goals rather than use case descriptions in identifying classes from use cases, so that classes identified are not limit to one description.
(b) It identifies entities that are involved in achieving use cases' goals, instead of nouns, events and interactions in use case descriptions, so that things to be considered are definitely significant to the system.
(c) It identifies attributes and operations of classes based on their involvement in individual use cases, so that features and operations irrelevant to use cases can be ignored in object modelling.
(d) It introduces a new use case-entity diagram that can explicitly show involvement of classes in individual use cases in a system, so that the complexity of modelling a large system with object orientation can be monitored and controlled effectively. The new diagrams therefore help represent user requirements precisely with use case-driven process.

In addition, goal flows are emphasised by this approach. Goal flows are different from data flows in conventional data flow diagrams because they show what the user expects from the system, instead of what the system will output based on input. They do not show any input to the system, because such data should be best considered at design stage. Goal flows can bring the following advantages into object modelling:

(a) Help realise the role of the use case diagram in UML: show functional requirements of a system only (i.e. what the system is expected to do).
(b) Help concentrate on what the system is expected to deliver rather than on how the system transforms from input into output.
(c) Help focus on real world conceptual classes that achieve the goal of a use case in a problem domain, rather than focus on software classes [14] (i.e. software components) that implement the use case in a software system. Such focus is in particular important and necessary in object-oriented analysis to build the analysis object model that consists of conceptual classes only and that is not a model of software components [14].
(d) Help test the class diagram against functional requirements of a system via use case-entity diagrams.

The new goal-oriented approach is also hoped to bring the following helps into system modelling:

(a) Help enhance reusability of entities involved in the use cases that have similar goal.
(b) Help identify classes gradually from use cases one by one at different times, instead of from all use cases at one time.
(c) Help appreciate and understand roles and responsibilities of classes in a system through identifying operations of classes based on collaboration between use cases and entities.
(d) The collaboration between use cases and entities, identified in Step 5, may be useful in modelling dynamic behaviour of the same system using sequence or collaboration diagrams, since the messages passed between objects could be identified from the collaboration.
(e) May help find a complementary process from requirements capture to object modelling with UML, so that more efficient CASE tools supporting object modelling with UML can be developed.

References

[1] P. Allen, S. Frost, Component-Based Development for Enterprise Systems, Cambridge University Press, Cambridge, 1998.
[2] S. Bennett, S. McRobb, R. Farmer, Object-Oriented Systems Analysis and Design using UML, McGraw-Hill, New York, 1999.
[3] G. Berrisford, Improving OO analysis methods, Journal of Object-Oriented Programming Mar/Apr (1998) 6–7.
[4] G. Booch, Object-Oriented Design with Applications, Second ed., Addison-Wesley, Reading, MA, 1994.
[5] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modelling Language: User Guide, Addison-Wesley, Reading, MA, 1999.
[6] I.K. Bray, An Introduction to Requirements Engineering, Addison-Wesley, Reading, MA, 2002.
[7] S. Gossain, Object Modelling and Design Strategies, Cambridge University Press, Cambridge, 1998.
[8] I. Graham, Use cases combined with Booch/OMT/UML: process and products, Journal of Object-Oriented Programming Jan (1998) 76–78.
[9] D. Harel, Statecharts: a visual formalism for complex systems, Science of Computer Programming 8 (1987) 231–274.

[10] B. Henderson-Sellers, OO diagram connectivity, Journal of Object-Oriented Programming Nov/Dec (1998) 60–68.

[11] A. Jaaksi, A method for your first object-oriented project, Journal of Object-Oriented Programming Jan (1998) 17–24.

[12] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard, Object-Oriented Software Engineering: a Use Case Driven Approach, Addison-Wesley, Reading, MA, 1992.

[13] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, Reading, MA, 1999.

[14] C. Larman, Applying UML and Patterns—an Introduction to Object-Oriented Analysis and Design and the Unified Process, Prentice-Hall, Englewood Cliffs, NJ, 2002.

[15] S. Lauesen, Real-life object-oriented systems, IEEE Software Mar/Apr (1998) 76–82.

[16] T.C. Lethbridge, R. Laganiere, Object-Oriented Software Engineering: Practical Software Development Using UML and Java, McGraw-Hill, New York, 2001.

[17] Y. Liang, M.A. Newton, H.M. Robinson, The use of object models for information systems analysis, Proceedings of the Fourth International Conference on Information Systems Development, Slovenia (1994) 625–634.

[18] Y. Liang, An approach to assessing and comparing object-oriented analysis methods, Journal of Object-Oriented Programming Jun (2000) 27–33.

[19] L. Mattingly, H. Rao, Writing effective use cases and introducing collaboration cases, Journal of Object-Oriented Programming Oct (1998) 77–84.

[20] D.E. Monarchi, G.I. Puhr, A research typology for object-oriented analysis and design, Communications of the ACM 35 (9) (1992) 35–47.

[21] B. Oesterich, Developing Software with UML, Addison-Wesley, Reading, MA, 1999.

[22] T.W. Olle, J. Hagelstein, I.G. Macdonald, C. Rolland, H.H. Sol, F.J.M. Van Assche, Verrijn-Stuart, An Information Systems Methodologies—a Framework for Understanding, Addison-Wesley, Reading, MA, 1988.

[23] D. Rosenberg, K. Scott, Use Case Driven Object Modelling with UML: a Practical Approach, Addison-Wesley, Reading, MA, 1999.

[24] J. Rumbaugh, J. Premerlani, M. Eddy, W. Lorensen, Object-Oriented Modeling and Design, Prentice-Hall, Englewood Cliffs, NJ, 1991.

[25] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modelling Language Reference Manual, Addison-Wesley, Reading, MA, 1999.

[26] S. Shlaer, S.J. Mellor, Object-Oriented System Analysis: Modeling the World in Data, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[27] S. Shlaer, S.J. Mellor, Object Lifecycle: Modeling the World in States, Prentice-Hall, Englewood Cliffs, NJ, 1992.

[28] I. Sommerville, Software Engineering, Fifth ed., Addison-Wesley, Reading, MA, 1996.

[29] P. Stevens, R. Pooley, Using UML Software Engineering with Objects and Components, Addison-Wesley, Reading, MA, 2000.

[30] B. Stroustrup, The C++ Programming Language, Second ed., Addison-Wesley, Reading, MA, 1991.

[31] M. Weisfeld, The Object-Oriented Thought Process, SAMS Publishing, 2000.