ABSTRACTION IN AN OBJECT-ORIENTED ANALYSIS METHOD

Sai Peck Lee
Faculty of Computer Science and Information Technology
University of Malaya
50603 Kuala Lumpur, Malaysia
Fax: 603-7579249
email: saipeck@fsktm.um.edu.my

Colette Rolland and Jöel Brunet
Centre de Recherche en Informatique
University of Paris 1
17, rue de Tolbiac
75013 Paris, France
email: rolland@masi.ibp.fr

ABSTRACT

The recent interest in object-oriented analysis has resulted in the introduction of abstraction in the analysis phase of information systems development life cycle. Abstraction has been proven to be a powerful object-oriented construct through which software complexity can be reduced and software productivity can be improved. Our main aim in this paper is to provide the support of abstraction for both the statics and dynamics of an information system in an object-oriented analysis method. In this way, it allows the analyst to view the basic characteristics of major components of an information system at different perspectives.

Keywords: ***Object-oriented analysis, Information systems development, Conceptual model, Abstraction, Decomposition***

1.0 INTRODUCTION

The great advancement in object-oriented programming [1, 2], coupled together with the influence of artificial intelligence and databases [3, 4] during the past few years have given rise to the adoption of the object-oriented paradigm in *information systems (ISs) development methodologies* [5, 6, 7]. In fact, this has now become a common topic of discussion, in particular, among the circle of computer methodologists and systems engineers, hoping to exploit the useful features of this paradigm for a better development not only in design methods [8, 9], but also in analysis methods [10, 11]. There seems to have a wide consensus for the object-oriented paradigm in view of its adoption in many research areas. This is due to the fact that the object concept provides a more natural way for representing real-world objects.

This is the main reason that led to the emergence of *object-oriented analysis methods* within the framework of *object-oriented ISs development life-cycle* [12, 13, 14]. Moreover, the recent advancement in the field of *computer aided software engineering* (CASE) together with the wide availability of CASE tools [15, 16] in the computer market have helped not only the automation of analysis methods to become a reality, but also the facilitation and acceleration of the automation process [10, 12].

The consideration of the object-oriented paradigm in the analysis point of view can be noticed from the increasing development of *conceptual models* borrowing object-oriented concepts [11, 17, 18, 19]. However, the emergence of conceptual modelling have brought with it other problems to the analysis of ISs, among which, is the problem of providing abstraction at the conceptual level for dealing with the enormous information represented in the conceptual schemas of large applications.

Abstraction has been proven to be a powerful object-oriented construct and as a way of reducing complexity of software [20]. Software productivity is inversely proportional to software complexity [21]. As such, abstraction is a way of improving software productivity. The main aim of abstraction is to allow things to be represented in a more significant nature by omitting irrelevant details, and thus, leading to a better comprehension by the end user. This is important as the concepts formalised by many existing conceptual models do not model real-world objects as perceived by the end user. An efficient abstraction mechanism allows an abstract structure at the appropriate level of abstraction to be constructed from the basic building blocks of an IS. Besides this, the validation of a conceptual schema in different user points of view can be carried out with the help of such mechanism [22].

Other than providing a better comprehension of the information represented in the conceptual schema to the end user, abstraction also helps in the maintenance of the software in cases where there is a growth or an evolution of the specification that arises to the conceptual schema. This is normally the case that happens in most of the organisations in the fast-moving commercial world of today.

As such, numerous new concepts have been introduced in the literature to deal with abstraction, such as, *subsystem* [23, 24], *cluster* [23, 25, 26], *schema* [27], *semantic context* [28] and *natural object* [22], to name just a few, where each concept represents an abstract structure which is obtained depending on the technique considered. These concepts have been defined in order to handle the immense information of large systems.

In this paper, we intend to describe the support of abstraction in the O* object-oriented analysis method [17, 29], which has been classified as a *prescriptive method* that is supported by the O* conceptual model together with a set of methodological prescriptions (methodological guidelines) for the analysis of object-oriented ISs. Our aim is to provide the support of abstraction for both the static and dynamic aspects of an IS, so as to allow the representation of real-world objects as perceived by the end user at different perspectives.

The relevant research is summarised in the next section. This is followed by the description of the O* object-oriented method in Section 3.0. The proposed abstraction mechanisms for the statics and dynamics of an IS are respectively given in Section 4.0 and Section 5.0. Finally, some concluding remarks are given in the last section.

2.0 SURVEY OF ABSTRACTIO N MECHANISMS

In this section, we shall trace the historical evolution of abstraction and discuss some abstraction techniques that have been described in the literature. There is some clear cut in the application of abstraction in software development, such as, its application within the areas of *programming-in-the-small* and *programming-in-the-large* [21, 30, 31]. Application of abstraction dates back at the time when researchers were in search of a better technique for improving programming practice. In the early days of research, the development of *abstraction techniques* was centred around *high-level programming languages* and *abstract data types* [1, 32], that is, within the area of programming-in-the-small.

Within the area of programming-in-the-large, over the years, the application of abstraction has evolved to the present stage where it is widely applied in software architectural level of a system, such as, *functional decomposition* [33, 34, 35] and *object decomposition* [10, 12]. At this level of software design and reuse, abstraction focuses on essential properties of system organisation. It captures the basic characteristics of significant components of the system and their interactions. Hence, a special term called *higher-level abstraction*, was employed by Shaw [21] for describing this sort of abstraction.

Functional decomposition describes a large complex system as a hierarchy of subsystem functions. Such decomposition is supported by a wide range of *structured analysis techniques* [33, 35], among which *data-flow diagram* [18, 35] is the most common technique used in the decomposition process. On the other hand, object decomposition defines a system as a hierarchy of objects. Objects belonging to the same domain are grouped into a *subject area* [10]. In this way, it is possible to define a collection of objects sharing some basic properties and behaviour. Functionality of a system is no longer an important criterion in such decomposition. A subsystem is supposed to possess strong interactions between objects within it and weak interactions with objects from other subsystems. In other words, the internal coupling of a subsystem must be tighter than its external couplings with other subsystems.

According to [21], a good abstraction is ignoring the right details at the right time. Hence, a good abstraction mechanism should provide support for helping the analyst in abstracting the details that are significant at a certain point in time. We shall begin by studying numerous works carried out by some researchers that involve grouping of entities and relationships from the extended Entity-Relationship models such as those defined in [25, 26, 27].

Kozaczynski's work [25] involves the development of *entity class grouping* for defining a new entity class as a grouping of existing entity classes or subclasses. There are 2 types of entity class groupings: (1) *Homogeneous grouping* : it contains subclasses from the same specialisation tree. It combines two or more subclasses of a single root entity class by applying the operator *union*; and (2) *Heterogeneous grouping*: it contains subclasses from different specialisation trees. On the other hand, Teorey's work [26] involves the derivation of a *clustering technique* to produce a bottom-up abstraction of natural groupings of entities. It is possible to apply clustering repeatedly resulting in layered levels of abstraction. The highest-level entity cluster that represents the entire database conceptual schema is called root entity cluster. An entity cluster maintains the same relationships between entities inside and outside it.

Desfray's work [27] deals with the integration of various *synthesis mechanisms* in the Class-Relationship model, with the aim of providing abstractions for both classes and relationships. The notion of *schemas* has also been introduced. A schema corresponds to a set of classes belonging to the same domain. It is possible to define a domain from other domains through *use relationship* or *inheritance relationship*. In this sense, schemas are analogous to subject areas employed by Coad and Yourdon [10].

Abstraction has also been extensively applied in *view management* [36, 37]. Shilling [36] defines a view as a simplifying abstraction of a complex structure. Czejdo's work [37] involves the integration of views into the Object-Relationship model of OSA. The definition of view semantics is either derived from query expressions over a semantic-model instance or is specified as semantic submodels.

In spite of the numerous approaches providing some sort of abstraction support, there is not much research on the abstraction support that captures also the behavioural aspects of objects, which is very much stressed by Wand [24] in the formalisation of ISs concepts. This is important as it provides abstraction on interaction between objects, thus, allowing the user to perceive an IS at another perspective.

3.0 THE O* METHOD

The *O* method is an object-oriented analysis method*, which is composed of a conceptual model referred to as the O* model and a set of methodological guidelines. The background of the O* method and its model will be presented in the following sections. For reasons of space limitation, the methodological guidelines are not presented in this paper.

3.1 Background

Before we proceed to the details of the proposed abstraction mechanisms, we would like to give a brief presentation on the *O* method* [17, 29]. The O* method was developed within the framework of the ESPRIT II project named Business Class[1] [38]. The tool supporting the method allows the analyst to define a conceptual schema representing the description of a real-world system with the aid of the O* methodological guidelines. Moreover, a set of interactive and deferred controls corresponding to the verification of the conceptual schema have been built into the tool. The conceptual schema is divided into two main parts, namely, the static schema representing the structural aspects of an IS, and the dynamic schema representing the behavioural aspects of the IS, through the hypothesis that the behavioural aspects are responsible for causing state changes to the structural aspects of the IS.

3.2 The O* Model

The two main concepts of the *O* model*, namely, *objects* and *classes*, are adopted from the object-oriented paradigm. The main aim of the model is to adopt the concept of objects from the analysis phase right through the implementation phase. Moreover, the model also takes into account the *life cycles* of objects for the determination of *static links* between classes, besides the inheritance links of the object-oriented paradigm. Two types of static links are distinguished: *reference link* and *composition link*, where each link can be defined between two classes after identifying the dependency between the life cycles of objects in these classes. A reference/composition link can be of simple type or multiple type. For example, a simple reference link connecting two classes indicates that a referring object refers to one and only one referenced object. On the other hand, a multiple reference link indicates that a referring object can refer to 1 - N referenced objects.

Other than the structural aspects, the model has considered the concepts of *events* and *operations* to take into account the behavioural aspects of ISs at the conceptual level. Three groups of events are distinguished: *external events, internal events* and *temporal events*. External events correspond to the events occurring in the environment outside an IS. Internal events correspond to the internal state changes or rather the system responses of the IS, and temporal events correspond to the events whose occurrences depend on the description of time. On the other hand, an operation represents an action performed on an object of a class, thus, causing a state change to that object.

We shall represent the formalisation of the various aspects of the O* model through an example showing two main functions of a business firm, namely, order processing and inventory management. Having studied the requirement and the behaviour of these functions of the organisation, the structural aspects and behavioural aspects of the business conforming to the O* model are established respectively in the static schema and dynamic schema as shown in Fig. 1 and Fig. 2, with the aid of the O* methodological guidelines.

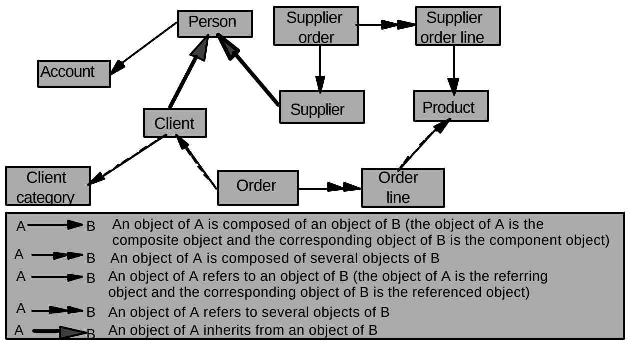
Person
Supplier order
Supplier order line
Account
Supplier
Product
Client
Client category
Order
Order line
A B An object of A is composed of an object of B (the object of A is the composite object and the corresponding object of B is the component object)
A B An object of A is composed of several objects of B
A B An object of A refers to an object of B (the object of A is the referring object and the corresponding object of B is the referenced object)
A B An object of A refers to several objects of B
A B An object of A inherits from an object of B

Fig. 1: Static schema

The static schema is set up with a set of atomic classes storing the persistent information of the organisation. These classes are interrelated among themselves through static links. A composition link between a composite class and a component class expresses a strong coupling of behaviour between a composite object and its component objects. It is derived semantically in such a way that the life cycle of a component object is totally included in the life cycle of its composite object. In other words, the existence of the component object depends on the existence of its composite object. Moreover, a component object belongs to one and only one composite object and their life spans are approximately the same. In contrast, a reference link defined between a referring class and a referenced class expresses a weak coupling of behaviour between a referring object and its referenced objects. The life cycle of the referring object is totally included in the life cycle of its referenced object. Moreover, a referenced object may be shared by several referring objects.

The representation of the behaviour of the business

[1] This project is supported by the European Commission under the contract 5311 of the second European Strategic Program for Research and Development in Information Technology (ESPRIT). Telesystems (France) is one of the partners in the project and University of Paris 1, as a subcontractor, involves in the development of the analysis environment.

activities is summarised as shown in Fig. 2. For example, the arrival of an order via a sales personnel is represented by an external event named *arrival of order*. Therefore, when this event happens, an object corresponding to the order and an object corresponding to the client who made the request for the order might be created depending on the triggering conditions defined on the triggers of the event. For instance, the condition *not c2* specifies the non-existence of an object of this client in the class *Client*. Note that the component classes are not represented in the dynamic schema as they have been encapsulated into the composite classes (see Section 5.3).

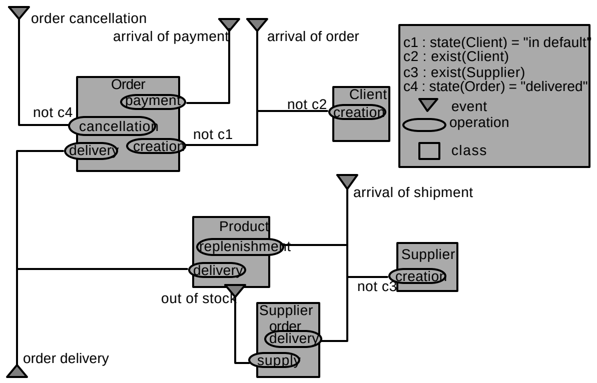

Fig. 2: Dynamic schema

Other than the dynamic schema, a *state transition graph* can be attached to a class for describing the local behaviour of its objects. It defines a set of distinct object states, in one of which, an object persists during a certain period of its life cycle. State changes of an object are specified by a set of state transitions described in the form of a triplet (*initial state, operation, final state*). This describes how an object undergoes state change at a certain point in time. The pair *initial state, internal operation* describes the object pre-condition, whereas *final state* describes its post-condition. An example of a state transition graph is given in Fig. 3.

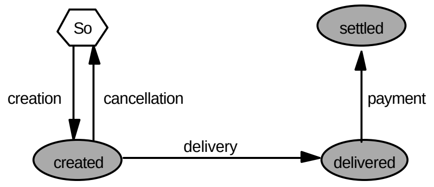

Fig. 3: State transition graph of the class *Order*

The state *So* specifies the non-existence of object and it is compulsory to define it in each graph. The state transition (*created*, *delivery*, *delivered*) stipulates that the operation *delivery* can be executed only if the object initial state is *created*, and as a result of this condition, the object will be in only a possible final state, *delivered*.

4.0 ABSTRACTION OF THE STATICS

Although the structural aspects and behavioural aspects of an IS can be modelled with the aid of the set of methodological guidelines, the model is enhanced through the integration of abstraction. Hence, the notion of *abstract class* that groups several classes is introduced in the model to provide a better representation of real-world objects. *Grouping* is an operation required to group several classes to form such a higher-level construct. It can be applied iteratively on classes and/or abstract classes to produce a higher-level abstraction. In some way, an abstract class is comparable to a subsystem, such as, each class is allowed to be grouped into only one abstract class.

4.1 Strategy

Our strategy has much similarity with the strategies used by Coad and Booch, where objects are grouped into subject areas, in particular, with the strategy of Coad [10], where a hierarchy of classes are grouped into a subject area in terms of aggregation. Moreover, our strategy allows grouping to be applied on a hierarchy of objects in terms of specialisation (see section 4.2).

One of the main static grouping strategies is to group classes that are semantically and behaviourally related, thus resulting in the formation of an abstract class that suppresses other irrelevant details and preserves only some salient features at a certain level of abstraction.

The static grouping mechanism allows abstract classes to be constructed not only from composite and component classes, but also from specialised and generalised classes, as well as from referring and referenced classes. This reduces the number of classes represented in the conceptual schema. An abstract class has a fuzzier semantics as compared to an atomic class, as all underlying objects and their static links are hidden in it. This is in accordance with the general abstraction principle, which states that the complexity of a problem can be reduced by omitting irrelevant details. Levels of abstraction can then be distinguished depending on the amount of details omitted, which the latter increases with the degree of nesting of abstract classes. An abstract class is diagrammatically represented by a dashed border.

Other than this, the static grouping mechanism is also capable of deducing static links on an abstract class through a set of *derivation rules*. Basically, these rules are stated as such:

1. The root is a class which is situated at the highest level of a hierarchy of classes in terms of inheritance, composition or reference.
2. The semantic significance of the abstract class is centred around the root class which gives the most general information on the underlying classes.
3. The name of the abstract class will be the name of the root class.
4. All static links coming into or going out of a class in the grouping will be naturally transferred to the abstract class.

4.2 Abstraction of Specialised Classes

Abstraction of specialised classes is associated with the grouping of a generalised class together with its specialised classes. According to the derivation rules, a static link coming into a specialised class from any class outside the grouping will be naturally transferred to the abstract class. Fig. 4 shows a general example of grouping for the abstraction of specialised classes.

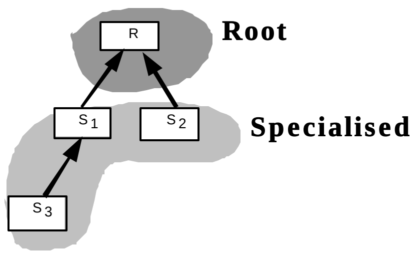

Fig. 4: General example of grouping at specialisation level

The root class *R* will be the ultimate abstract class if grouping is applied on this hierarchy. This is the most natural way of grouping as all characteristics of a generalised class, including also the behavioural characteristics, are inherited by the specialised classes. Therefore, the emphasis of abstraction is centred around the generalised class in such a way as to omit the additional characteristics possessed by the specialised classes. Hence, the abstraction extracts only the characteristics of the generalised class. In this way, the abstraction no longer gives the exact information on certain phenomena represented by the underlying specialised classes. The information given by this kind of abstraction will be less concise and normally relates a general statement. For instance, Fig. 5 illustrates that a client who makes a request for an order is a person, this fact can be abstracted into a person makes a request for an order.

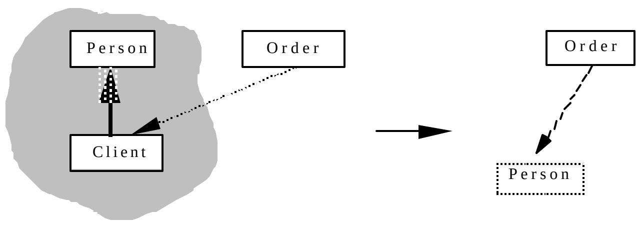

Fig. 5: Abstraction of the specialised class *Client* into the abstract class *Person*

Mathematically, it can be formalised as such : *″o Î Order, $p Î Person : o - -> p*. It can be stated as such : for each object in *Order*, *o* , it exists an object in *Person*, *p*, such as *o* refers to *p*. Obviously, it is less concise to say that an order refers to a person. It is true that a client makes a request for an order, but it is false to say that each individual does so. Therefore, the fact that an order refers to a person is only a general case. The semantics of this fact is imprecise and vague as to which particular individual makes a request for an order. However, the simple reference link transferred between the class *Order* and the abstract class *Person* retains the original semantics. This is because all instances of the specialised class *Client* which were previously referenced to by the objects of *Order* also figure in the generalised class *Person* according to the definition of the model. As the abstract class *Person* is centred around the generalised class *Person*, therefore, the reference link between the class *Order* and the abstract class *Person* is valid.

There is, however, a restriction for this sort of abstraction, which is taken care of by the following consistency rule:

R1 : Consistency rule for a specialised class
If it exists a multiple inheritance of a specialised class with some generalised classes, none of the generalised classes can be separated from the specialised class in the grouping. This means, a specialised class in the grouping cannot be at the same time a specialised class of an external class.

For instance, it is forbidden to have the case of Fig. 6.

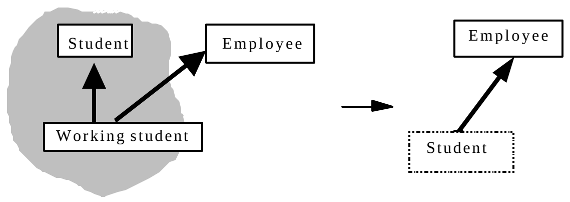

Fig. 6: Forbidden example

Mathematically, it is formalised to as such : *″s Î Student, $e Î Employee : s = e*. This is not allowed as such abstraction will generate inconsistent information. The grouping of the class *Student* and the class *Working student* generates an abstract class *Student* in which all instances of students are figured. However, the semantics of the inheritance link between the abstract class *Student* and the generalised class *Employee* indicates that all instances of *Student* should also be figured in *Employee*. This means that every student is also an employee, which is not true as not all students are employees. In conclusion, the inheritance link deduced between the abstract class *Student* and the class *Employee* violates the original semantics.

4.3 Abstraction of Component Classes

In any subject, either in the molecular model, economic model or any organisational model, interactions between objects within a component are always more frequent than interactions between objects from different components. For instance, in the molecular theory, intra-molecular forces are tighter than inter-molecular forces. This fact can be related to the high-frequency dynamics of a component involving internal interactions and the low-frequency dynamics involving interactions with other components.

A composition link defined between a composite class and a component class expresses such a strong dependency between these two classes. The behaviour of the objects in these classes is strongly coupled, such as the creation or deletion of a component object is usually done via its composite object. Besides this, there is a strong relationship in the natures of a composite object and each of its component objects. These objects normally originate from the same phenomenon. Hence, it is quite natural to integrate these objects into a unified whole, which is more abstract and closer to the real-world signifying this phenomenon. In this way, it is possible to group the root class and its specialised classes together with their components classes. Fig. 7 shows a general example of grouping for abstraction of component classes.

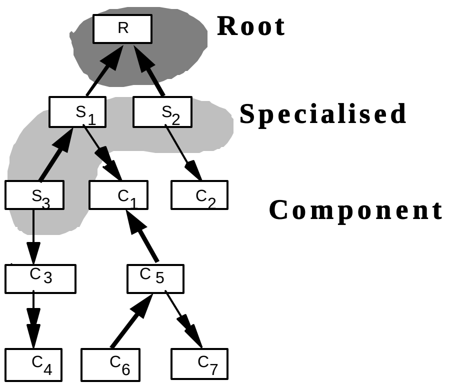

Fig. 7: General example of grouping at composition level

Notice that some of the specialised classes are also composite classes (C5, C6). As before, a static link coming into a component class from any class outside the grouping will be naturally transferred to the abstract class. For instance, Fig. 8 shows that a reference link comes from the class *Reservation* and terminates in the class *Room*, which the latter is grouped together with its composite class, *Hotel*. This results in the transfer of the reference link from the class *Reservation* to the abstract class *Hotel*.

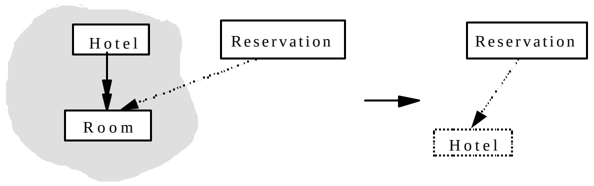

Fig. 8: Derivation of the reference link coming from *Reservation* and terminating in *Hotel*

Semantically, instead of saying, "A reservation of a room in a hotel", which is a more precise statement, we might say, "A reservation in a hotel", which is more fuzzy, as we no longer know that a hotel is composed of rooms. Also, the information on the room reserved is hidden in the abstract class *Hotel*, as it is of no interest to us.

A static link which is originated from or terminated in a class grouped might be converted to some other form before it is transferred to the abstract class. For instance, in certain cases, the static link transferred to an abstract class can become multiple, such as the case as shown in Fig. 9.

The abstract class *Order*, including the composite class *Order* and the component class *Order line* becomes multiple references to the class *Product*.

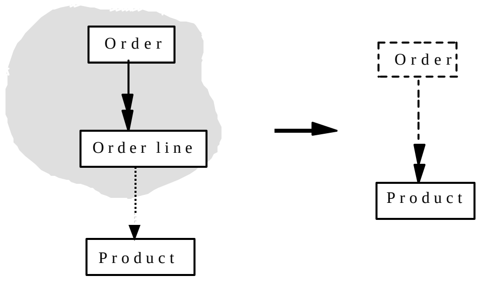

Fig. 9: The abstract class *Order* becomes multiple references to *Product* after the grouping

The fact that an order is composed of multiple order lines is of no interest to us after the grouping. The same case arises to the abstract class *Order* which gives a vague semantics in terms of its underlying structure. As such, a set of rules that derive static links on an abstract class can be set up as shown in Fig. 10.

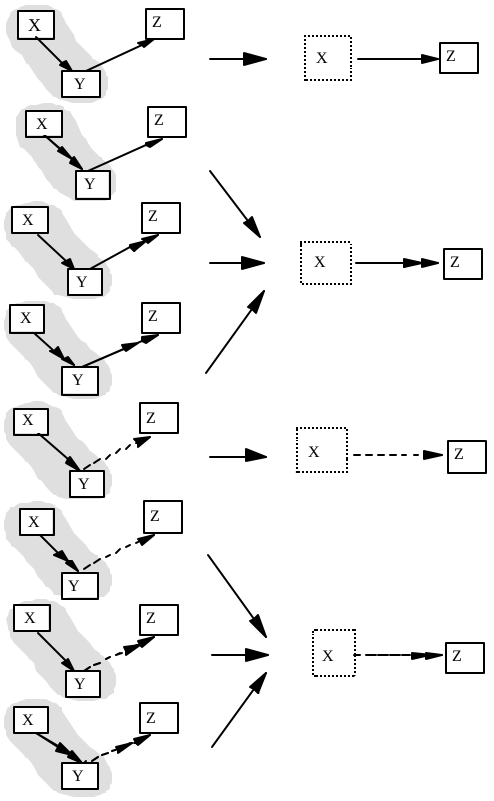

Fig. 10: Derivation of static links on an abstract class

There is a restriction to the derivation rules for abstracting component classes. The restriction is being taken care of by the following consistency rule:

R2 : Consistency rule for a component class
A component class in the grouping cannot be at the same time a component, specialised, or generalised class of an external class.

For instance, it is forbidden to have the case of Fig. 11.

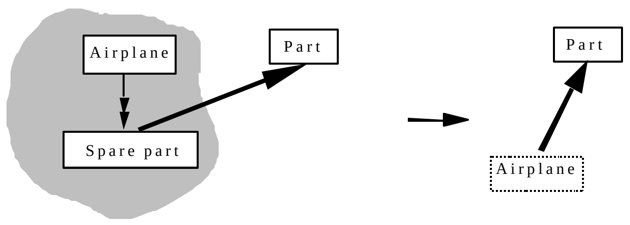

Fig. 11: Forbidden example

The fact that an airplane is composed of many spare parts which the latter in turn are parts, cannot be abstracted to the fact that an airplane is a part, which is semantically wrong.

4.4 Abstraction of Referenced Classes

A reference link defined between a referring class and a referenced class expresses a weak semantic dependency and a low-frequency dynamics between these two classes. The behaviour of the objects in these classes is loosely coupled as their interactions are much weaker. Indeed, each of these objects comes from totally distinct natures. For instance, an object of the class *Order* refers to an object of the class *Client*. These objects do not originate from the same phenomenon. In addition, their relationship is only at a superficial level in the sense that the actions suffered by the referring object hardly affect its referenced object.

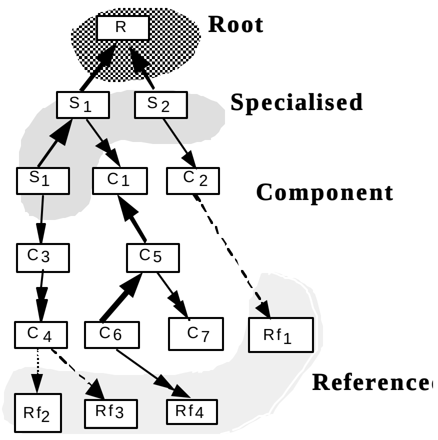

Fig. 12: General example of grouping at the reference level

However, this does not prevent abstraction to be performed on referenced classes, though abstract classes generated by this sort of grouping are of less importance. These abstract classes appear only locally on the static schema. The main aim of the grouping is to hide referenced classes. Fig. 12 shows a general example of grouping for the abstraction of referenced classes.

As before, there is a restriction to the derivation rules for the abstraction of referenced classes, which is being taken care of by the following consistency rule:

R3 : Consistency rule for a referenced class
A referenced class in the grouping cannot be linked by any static link with an external class.

For instance, in Fig. 13, it is possible to group the referring class *Client* and the referenced class *Client category* to form an abstract class called *Client* as *Client category* is not referenced to by any static link.

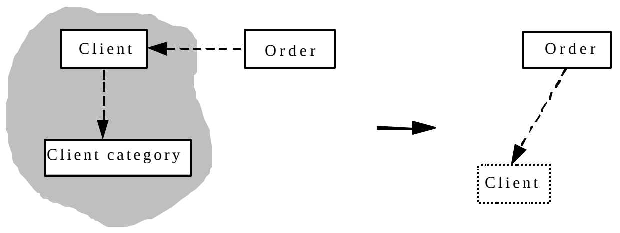

Fig. 13: Abstraction of the referenced class *Client category*

However, it is forbidden to group the class *Supplier order line* and the class *Product* in Fig. 1 to form an abstract class, as *Product* is referenced to by the class *Order line*.

A final conclusion can be drawn from the abstractions of component classes and referenced classes in the derivation of abstract classes. A static link derived on an abstract class will become multiple via the following derivation rule:

Let *R* be a root class and *T* be a terminal class of a path linking *R* and *T* in the grouping. Let E be an external class outside the grouping.

If there is a simple composition/reference link which
is originated from T and terminated in E,
If there is a multiple composition/reference link
in the path linking R and T,
Then the derived static link on the abstract class
will be a multiple composition/reference link.

Notice that the abstract class of the example in Fig. 9 is derived from this rule.

4.5 Illustration

It is possible to identify some groupings as shown in Fig. 14 on the static schema of Fig. 1.

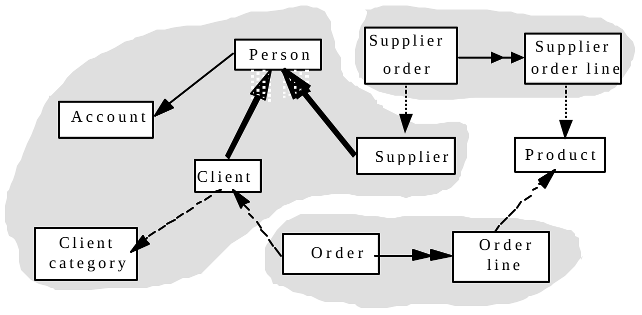

Fig. 14: Identification of groupings on the static schema

The application of the above abstraction principle gives rise to the outcome as shown in Fig. 15.

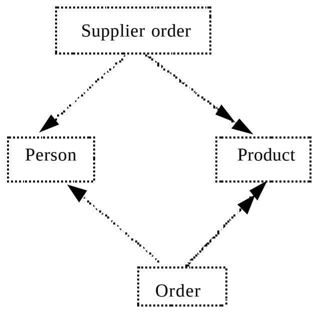

Fig. 15: Abstraction on the static schema

Notice that the simple reference link previously defined between the class *Supplier order line* and the class *Product* has been converted into a multiple reference link on the abstract class *Supplier order* and terminated in the class *Product*.

5.0 ABSTRACTION OF THE DYNAMICS

As it is the case where it is important to allow an IS to be perceived in multiple views: static view, dynamic view and architectural view, the support of abstraction for the dynamic perspective of an IS is also as important. It is through such abstraction that global behaviour of basic components of an IS can be characterised.

5.1 Strategy

To provide for the abstraction of behavioural perspective, the grouping mechanism must be able to deduce some *retrieved operations* for including in the views of specialised and composite classes in the dynamic schema. A retrieved operation whose form of light border is actually a virtual operation of the corresponding operation in a generalised/component class.

5.2 Retrieved Operations of Specialised Classes

Views of specialised classes will be represented on the dynamic schema so that reusability of generalised operations can be demonstrated explicitly. All the inherited operations of a specialised class will be generated in the view of the specialised class. For instance, Fig. 16 shows the view of *Client* in which the inherited operations are related to their corresponding operations in the generalised class, *Person* through *use links*.

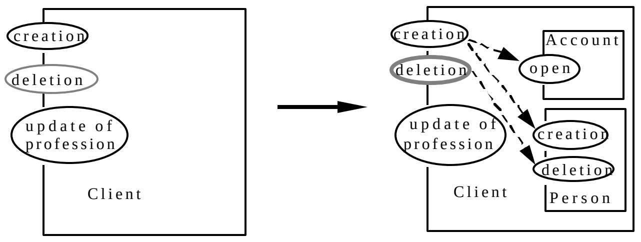

Fig. 16: View of *Client* and its internal definition

As each operation in the model possesses an operation type, it is possible to establish a semantic relationship between an operation of the composite class and an operation of the component class via a use link. Notice that there are two use links originating from the operation of creation in the view. This signifies that there is an *augmentation* in the operation *creation* of *Client*, and in this case, the generalised operation *creation* of *Person* will be executed before the opening of account by the operation *open* defined in the component class *Account*. In this case, a generalised operation is executed before any operation of the component class.

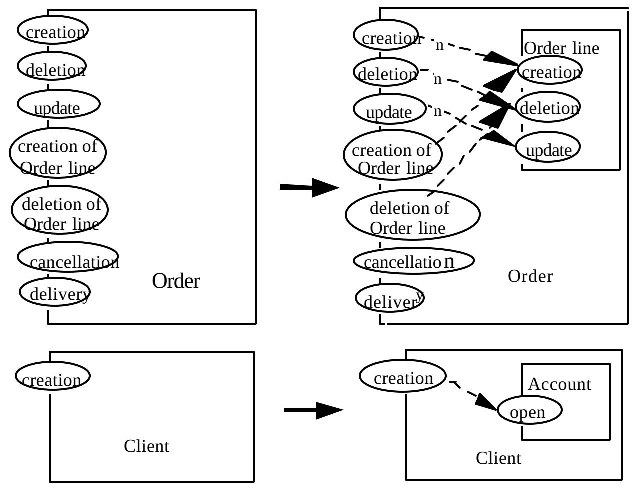

Fig. 17: Views of composite classes and their internal definitions

5.3 Retrieved Operations of Composite Classes

Abstraction of component classes can be demonstrated dynamically in the views of composite classes. The basic point is to associate each operation of each component class that deals with modification of a component object with a retrieved operation in the view of composite class. This means that any manipulation to a component object will be done via the corresponding retrieved operation generated in the view. A use link is used for relating an operation in the composite class to an operation in the component class. We shall demonstrate this through the examples as shown in Fig. 17. The first part illustrates the view of the composite class *Order* and the second part illustrates the view of the composite class *Client.*

The label *n* defined on a use link signifies that the operation of the component class will be executed *n* times. For instance, the creation of an object of *Order* will result in multiple creations of order lines. The number of order lines to be created depends on the argument *n* defined on the use link between the operation *creation* of *Order* and the operation *creation* of *Order line*. The same explanation applies to deletion and update. In this way, the actual behaviour of the component classes is hidden in the retrieved operations of a view, and as such, interactions between a composite object and a component object are not demonstrated explicitly in the view.

5.4 The Concept of Actor

The notion of actor class has also been introduced. An actor is a human or an artificial agent interacting with the IS for exchanging information. It is an instance of an actor class, the latter specifies the common characteristics of a set of actors. For instance, an actor class characterises the possible behaviour of its instances through external events and external operations.

An actor interacts with the IS by way of external events and thus causing state changes to the objects in the IS. An external operation represents an action triggered by a system response, which will be intercepted by an agent corresponding to the actor. For instance, it is possible to identify an actor class called *Sales personnel* for the previous example. Graphically, it is represented as shown in Fig. 18.

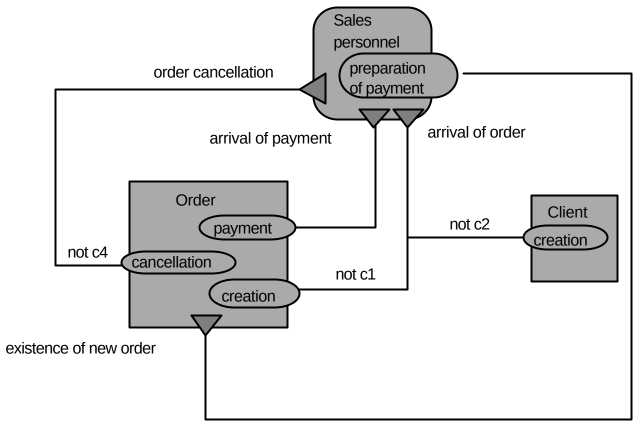

Fig. 18: Integration of actor into the dynamic schema

When the internal event *existence of new order* occurs, the external operation *preparation of payment* will be triggered. This response will then be intercepted by the actor *Sales personnel.*

6.0 CONCLUDING REMARKS

With the recent rapid progress in both software and hardware technologies, together with the increasing demand of providing the analyst with an easy-to-use supporting tool for the analysis of ISs, the integration of efficient abstraction mechanisms has become important research topic. The capability of the method to provide efficient abstraction of the statics and dynamics of an IS has been discussed. This allows the analyst to view an IS in different perspectives of the abstraction.

The types of static links serve as an important basis for the identification of the kinds of abstraction on the static aspects of the IS. The abstraction of component classes is based on the strong interactions between objects, whereas the abstraction of referenced classes is based on the weak interactions between objects. Grouping of a generalised class and its specialised classes provides the most natural way of abstraction and reusability.

REFERENCES

[1] B. Meyer, *Eiffel: The Language*. Interactive Software Engineering Inc., 1989.

[2] C. Lécluse and P. Richard, "The O_2 Database Programming Language" in *Proceedings of the Fifteeth International Conference on Very Large Data Bases, Amsterdam, 1989*.

[3] C. Cauvet, C. Rolland and C. Proix, "A Design Methodology for Object-Oriented Database" in *International Conference on Management of Data, Hyderabad, 1989.*

[4] W. Kim, "Object-Oriented Databases: Definition and Research Directions" in *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, No. 3, Sep. 1990.

[5] D. T. Lee, "Computer Information System Development Methodologies - A Comparative Analysis" in *National Computer Conference*, 1987.

[6] T. W. Olle, J. Hagelstein, I. G. Macdonald, C. Rolland, H. G. Sol, F. J. M. V. Assche and A. A. Verrijn-Stuart, *Information Systems Methodologies*. Addison-Wesley, 1991.

[7] C. Rolland and C. Richard, "The Remora Methodology for Information Systems Design and Management" in *IFIP Conference on Comparative Review of Information Systems Design Methodologies, North-Holland, 1982.*

[8] J. Martin and J. J. Odell, *Object-Oriented Methods: Pragmatic Considerations*. Prentice-Hall, 1996.

[9] N. Prakash, “Specifying Operational Characteristics of Information Systems in OOD” in *Conference on Object Oriented Approach in Information Systems, Quebec, 1991*.

[10] P. Coad and E. Yourdon, *Object-Oriented Analysis*. Prentice-Hall, Englewood Cliffs, N. J., 1990.

[11] L. J. B. Essink, “Object Modelling and System Dynamics in the Conceptualization Stages of Information Systems Development” in *Conference on Object Oriented Approach in Information Systems, Quebec, 1991*.

[12] G. Booch, *Object-Oriented Design with Application*, Benjamin/Cummings. 1991.

[13] S. Shlaer and S. J. Mellor, *Object Oriented Systems Analysis*. Yourdon Press, 1988.

[14] A. G. Sutcliffe, “Object Oriented Systems Analysis: The Abstract Question” in *Conference on Object Oriented Approach in Information Systems, Quebec, 1991*.

[15] C. McClure, “The CASE Experience” in *The BYTE Magazine*, April 1989.

[16] G. Wilkie, *Object-Oriented Software Engineering: The Professional Developer's Guide*. Addison-Wesley, 1993.

[17] J. Brunet, “Modeling the World with Semantics Objects” in *Conference on Object Oriented Approach in Information Systems, Quebec, 1991*.

[18] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[19] K. Takagaki and Y. Wand, “An Object-Oriented Information Systems Model based on Ontology” in *Conference on Object Oriented Approach in Information Systems, Quebec, 1991*.

[20] P. J. Courtois, “On Time and Space Decomposition of Complex Structures” in *Communications of the ACM*, Vol. 28, No. 6, June 1985.

[21] M. Shaw, “Toward Higher-Level Abstractions for Software Systems” in *Data & Knowledge Engineering*, North-Holland, No. 5, 1990, pp. 119-128.

[22] P. A. Bres and J. P. Carrère, “Un Modèle de Comportement d’Objets Naturels" in *AFCET: Autour et à l’entour de Mérise, 1991*.

[23] P. Hsia and A. T. Yaung, “Another Approach to System Decomposition: Requirements Clustering” in *Proceedings COMPSAC’88, 1988*.

[24] Y. Wand and R. Weber, “An Ontological Analysis of Some Fundamental Information Systems Concepts” in *Proceedings of the Ninth International Conference on Information Systems, Minneapolis, Minnesota, Nov. 30-Dec. 3, 1988*.

[25] W. Kozaczynski and L. Lilien, “An Extended Entity-Relationship (E^2R) Database S*pecification into the Logical Relational Design”* in *Proceedings of the Sixth International Conference on Entity-Relationship Approach, New York, Nov. 9-11, 1987*.

[26] T. J. Teorey, G. Wei, D. L. Bolton and J. A. Koenig, “ER Model Clustering as An Aid for User Communication and Documentation in Database Design” in *Communications of the ACM*, Vol. 32, No. 8, Aug.1989.

[27] P. Desfray, “Object Oriented Structuring: An Alternative to Hierarchical Models” in *Technology of Object-Oriented Languages and Systems*, USA, 1991.

[28] T. Estier, G. Falquet, J. Guyot and M. Léonard, “Six Spaces for Global Information Systems Design” in *Conference on Object Oriented Approach in Information Systems, Quebec, 1991*.

[29] S. P. Lee, *Formalization and Automatic Support for Conceptual Modeling*. University of Paris 1 Panthéon-Sorbonne, 1994.

[30] K. Orr, C. Gane, E. Yourdon, P. P. Chen and L. L. Constantine, “Methodology: The Experts Speak” in *The BYTE Magazine*, April 1989.

[31] F. DeRemer and H. Kron, “Programming-in-the-large versus Programming-in-the-small” in *Software Engineering*, Vol. SE-2, No. 2, June 1976.

[32] D. Price, *Introduction to ADA*. Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[33] T. DeMarco, *Structured Analysis and System Specification*. Yourdon Press, New York, 1978.

[34] G. Maksay and Y. Pigneur, "Reconciliating Functional Decomposition, Conceptual Modelling, Modular Design and Object Orientation for Application Development" in *Conference on Object Oriented Approach in Information Systems, Quebec, 1991*.

[35] E. Yourdon, *Modern Structured Analysis*. *Yourdon Press/Prentice-Hall*, New York, 1989.

[36] J. J. Shilling and P. F. Sweeney, “Three Steps to Views: Extending the Object-Oriented Paradigm” in *OOPSLA ‘89 Proceedings*, *Oct. 1-6, 1989*.

[37] B. Czejdo and D. W. Embley, “View Specification and Manipulation for a Semantic Data Model” in *Information Systems,* Vol. 16, No. 6, 1991, pp. 585-612.

[38] Telesystems, “Analyst Workbench Tutorial” in *Esprit project 5311 (Business Class)*, Feb. 1992.

BIOGRAPHY

Lee Sai Peck obtained her Master of Computer Science from University of Malaya in 1990, her D.E.A of Computer Science from University of Paris VI Pierre et Marie Curie in 1991 and her Ph.D of Computer Science from University of Paris I Panthéon-Sorbonne in 1994. She is a lecturer at Faculty of Computer Science and Information Technology, University of Malaya.

Colette Rolland obtained her Ph.D of Computer Science from University of Paris I Panthéon-Sorbonne in 1971. She is a professor at Centre de Recherche en Informatique, University of Paris I Panthéon-Sorbonne.

Jöel Brunet obtained his D.E.A of Computer Science from University of Paris VI Pierre et Marie Curie in 1989 and his Ph.D of Computer Science from the same university in 1993. He is an associate professor of University of Paris I Panthéon-Sorbonne.