Malaysian Journal of Computer Science, vol. 10 No. 2, December 1997, pp. 72-81

ISSUES IN REQUIREMENT ENGINEERING OF OBJECT-ORIENTED INFORMATION SYSTEMS: A REVIEW

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

ABSTRACT

Information Systems (ISs) engineering, which covers requirements engineering and design engineering, has been oriented to adopting object-oriented concepts in its theoretical foundation as a basis for development tools to achieve higher quality and productivity with lower cost. In this paper, we present a review of the various problematic issues specifically in requirements engineering of object-oriented ISs engineering and we mention some existing research attempts to deal with such issues. Some of these problematic issues are related to the lack of expressiveness of the conceptual model used in most analysis methods and the rigidity of the Computer Aided Software Engineering (CASE) tools supporting these methods. We also discuss some important issues in design engineering which are closely linked to those in requirements engineering, such as the integration of analysis and design tools to provide a uniform ISs development support environment.

Keywords: ***Conceptual modelling, Information systems engineering, Object-oriented concepts, Tools Integration***

1.0 INTRODUCTION

Information Systems Engineering (*ISs Engineering*) was first introduced by J. Hagelstein [1]. It is divided into two phases: *requirements engineering* and *design engineering*. Requirements engineering involves knowledge acquisition of a domain description (description of a real world system) and knowledge representation by abstracting or conceptualising the relevant parts of the domain description to result in a conceptual schema of the required IS. The conceptual schema is the final product of conceptual modelling [2] in the requirements engineering phase. It contains conceptual representation of a real world system in which the essential information of the IS is preserved, while irrelevant details are phased out. Design engineering aims at mapping the conceptual representation of the conceptual schema to the design specification of the IS. Hence, the conceptual schema of an IS forms the basis for subsequent design and implementation of that IS.

Conceptual modelling of ISs refers to the process of ISs development that involves analysing the problem domain of a real world system and leads to the development of a conceptual schema, and then the specification of the desired IS. In an ideal case, the different phases of the ISs development life cycle should be aided by tools supporting different development methods [3, 4, 5, 6]. These tools need to be able to cooperate with each other in order to support the whole ISs development life cycle effectively. In relation to this, it is important to deal with the issue of tools integration [7] in ISs engineering.

The great advancement in object-oriented (OO) programming, coupled with the influence of artificial intelligence and databases [8, 9, 10], has given rise to the adoption of OO concepts in ISs development methodologies [5, 11]. The object concept provides a more natural way for representing real world objects. This was the main reason that led to the emergence of OO analysis and design methods within the framework of *object-oriented ISs (OOISs) development life cycle* [12]. As a result, towards the end of 1990s, ISs engineering has been oriented to adopting OO concepts in its theoretical foundation as a basis for development tools to achieve higher quality and productivity with lower cost.

In this paper, we present a review of the various problematic issues in requirements engineering of OOISs engineering and we mention some existing research attempts to deal with these issues. Some of these issues are related to the lack of expressiveness of the conceptual model used in most analysis methods and the rigidity of the CASE tools supporting these methods. We also discuss some important issues in design engineering which are closely linked to those in requirements engineering, such as the integration of analysis and design tools to provide a uniform ISs development support environment.

2.0 PROBLEMATIC ISSUES

The definition of a well-defined conceptual model for an analysis method is still problematic though there is a proliferation of analysis methods in recent years. It is claimed that a well-defined conceptual model should be easy to use and understand, besides possessing qualities such as high-level concepts for modelling objects and constraints, as well as the basic facility for checking the completeness and correctness of the conceptual schema of

the IS. Existing analysis methods [3, 13, 14] only partially solve certain issues related to conceptual modelling. They lack the necessary formalisation to define a formal specification language and abstraction mechanisms [13, 15] for providing the necessary underlying modelling support to lead to a more complete and precise specification of an IS at different perspectives. It has been argued that by having a conceptual model alone is not enough to substantiate the semantic power of a method, on the other hand, the model needs to be enhanced and enriched to include supplementary functionality to allow an IS to be modelled and viewed at different perspectives. Currently, most of the existing conceptual models only have a graphical language for modelling ISs. It is important to have a formal specification language to increase the semantic power of a conceptual model by providing an alternative way for specifying ISs at the conceptual level. The specification language allows different facets of an IS to be specified at the appropriate levels of detail.

Conceptual modelling is not only a time consuming and tedious process, it is also error prone. Some tools supporting existing methods are hard-coded and thus they are often too rigid for sustaining a change. The automation of analysis methods through customisation by using a meta-CASE tool attempts to alleviate these problems.
Having an efficient analysis tool is not enough to justify an effective ISs development, unless this tool is able to communicate with other development tools in the whole ISs development life cycle. Though there is a significant increase of development tools supporting various methods at different phases of the development life cycle recently, these tools are stand-alone in the sense that they do not share data and are not agreed on a common standard. As such, there is a need to integrate these tools together to form a uniform basis for an ISs development environment.

3.0 OO ANALYSIS METHODS: REVIEW AND PROBLEMATIC ISSUES

We begin by presenting some existing analysis methods based on two previous approaches (i.e., the Cartesian approach, and the systemic approach) [5] to ISs development and discuss the emergence of OO analysis methods based on the OO approach [3, 16].

3.1 Review of Analysis Methods

Within the context of software engineering, the term method has been defined as an explicit prescription for achieving a set of activities required by an approach to software development. In [17], a method is defined as a procedure or technique for performing some significant portion of the software life cycle. In short, a method consists of a well-defined and disciplined procedure tailored to the development of certain type of development product. It consists of a set of concepts and a set of methodological steps for aiding in modelling process to lead to a development product. Methods are often supported by a variety of tools, such as diagrams in the form of graphical notations, textual representations, etc.

The emergence of analysis methods in the 1970s aimed to deal with the issues of requirements analysis through conceptual modelling techniques. Basically, the existing analysis methods can be categorised in terms of the concepts employed from the different modelling approaches: the Cartesian approach, the systemic approach and the OO approach. Fig. 1 shows the modelling approaches viewed at three perspectives: data-oriented perspective, process-oriented perspective and behaviour-oriented perspective.

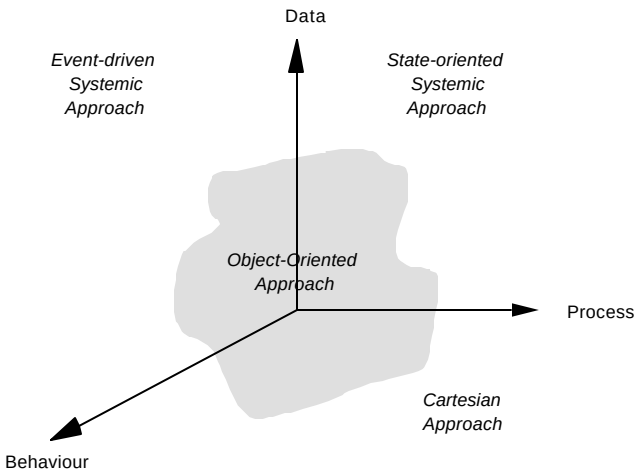

Fig. 1 The three perspectives of modelling approaches

The data-oriented perspective focuses on complete and thorough data analysis and its relationships; the process-oriented perspective focuses on analysis of functions/processes of real world systems; whereas the behaviour-oriented perspective concentrates on the dynamic nature of the data and events of the real world that have impact on the data recorded in the IS.

The Cartesian approach covers the process-oriented and behaviour-oriented perspectives, the event-driven systemic approach covers the data-oriented and behaviour-oriented perspectives, whereas the state-oriented systemic approach covers the data-oriented and process-oriented perspectives. None of these three approaches covers all the three perspectives. The shaded portion in the figure illustrates the OO approach as a breakthrough in integrating the three perspectives in current conceptual modelling techniques.

3.1.1 Cartesian Methods

The Cartesian methods are derived from the integration of the Cartesian paradigm with functional design approach. The dominant feature in the analysis procedure of these methods is that a problem domain is hierarchically decomposed into functions (or processes). These functions are further decomposed by successive refinement into

simple sub-functions. The decomposition continues to a point where the functions are manageable. As a basis of functional decomposition, the system is seen at a high level in terms of what it is intended to do, and then at a detailed design, how it will accomplish these process-oriented goals.

These methods are based on data flows. The most familiar tool supporting these methods is the *Data Flow Diagram* (DFD). Two well-known examples of Cartesian analysis methods are SA (*Structured Analysis*) [18, 19] and SADT (*Structured Analysis and Design Technique*) [18]. Later integration of these Cartesian analysis methods with OO design environments was partly because they have been widely used all over the world. The main motivation is to provide an OO development method that would be familiar to most analysts. However, the transformation from the DFD to the OO environment is not direct and difficult to accomplish. To alleviate this problem, some works provide heuristics for partial transformation, but this requires the analyst's intervention for decision making. Some variations of DFDs can be found in the literature.

The main drawback of the Cartesian methods is its inadequacy of data modelling features for representing data at a global perspective. Other disadvantages are related to the lack of theoretical work for providing a solid foundation to the concepts and techniques of top-down decomposition and the lack of methodological aids for ISs development. Furthermore, using DFDs is difficult to evaluate the consistency, completeness and quality of solutions.

3.1.2 Systematic Methods

The systemic methods focus fully on data modelling. They are based on the integration of the systemic paradigm with conceptual approach. In contrast to the Cartesian methods, the systemic approach is originated from systems theory. The analysis methods based on this approach emphasise the representation of relevant phenomena of a domain discourse through a conceptual model. Instead of analysing the real world in terms of functions as in the Cartesian approach, the systemic approach analyses the real world in terms of a set of entities or objects that have relationships between them and the interactions with each other. These entities or objects evolve with time. Its main aim is to interpret and comprehend a complex real world system by modelling its relevant phenomena.

The conceptual models of the systemic analysis methods are influenced by semantic networks used in artificial intelligence. They are originated from the data models such as the *Entity-Relationship model* (*E-R model*) [20] and the model of NIAM [21]. The E-R model uses three concepts: entity, relationship and attribute, while NIAM proposes two concepts: entity and binary relation. These models can be classified as semantic models because they express the semantic representation of knowledge. Even though the E-R model is easy to use, it lacks the representation of behavioural relationships of real world phenomena.

Due to the limitation of the above mentioned systemic methods to data-oriented representation, several subsequent works aimed at representing the dynamic aspects of real world phenomena. This led to the emergence of some analysis methods capable of integrating the structural and behavioural aspects of the IS in the same conceptual schema. Some of these early methods are REMORA [11] and ACM/PCM [22]. These methods have a common way in modelling real world objects: objects undergo state changes as time goes by and they exist within their life duration during which successive state changes take place. These objects have structural and behavioural properties. The structure of an object describes its internal and external relationships. The behaviour of an object expresses when and how objects undergo state changes. Time is an integral part of behaviour-oriented conceptual models. In CIAM method [23], the attributes of an entity type depending on time are defined as functions of time. Time modelling forms part of the definition of methods in ERAE [1], TEMPORA [24] and OBLOG [25].

3.1.3 Emergence of OO Methods

The main concepts employed in OO paradigm are object, class, inheritance and message communication between objects [12, 15]. From these basic concepts, other mechanisms were developed. Polymorphism is a concept linked to inheritance and by which a name can denote objects of many different classes that are related by some common superclass. There is still no consensus with regard to the useful concepts for conceptual modelling using the OO approach. The existing OO conceptual models are not rich enough for incorporating the static and dynamic aspects of an IS in a single conceptual schema.

The emergence of OO methods [3, 14] is mainly due to the need to integrate the three perspectives of Fig. 1 in the conceptual schema of an IS so that all aspects of the real world system can be fully captured. These methods are partly based on the approaches used in the previous sections. Existing OO analysis methods use the concepts of semantic models. The notions such as client-server and use link used in the OO design were also adapted to some OO analysis methods. We shall describe some important concepts used by most existing OO conceptual models briefly in the following, with reference and comparison made to the conceptual models using the Cartesian approach or systemic approach.

Within this context, an OO analysis method corresponds to a method tailored to support conceptual modelling in the requirements engineering phase of ISs engineering by following the OO approach. We propose that a well-defined OO analysis method should at least consists of:

- a conceptual model containing a set of OO concepts represented by a set of graphical notations for

representing the static and dynamic aspects of various phenomena of real world systems;
- a set of methodological steps for organising conceptual modelling process;
- a specification language for defining precise characteristics of an IS that are too complex to be represented graphically in the conceptual schema;

- **Object and Class**

All OO analysis methods have some kind of classification mechanism that groups objects of the same characteristics into classes. In certain methods, the concepts entity and entity types are used instead of the concepts object and class. One of the special features of the OO analysis methods is that all dynamic specifications for representing interactions of objects are encapsulated in classes. The class concept represents real world phenomena characterised by a collection of attributes and methods. The object concept has a rich semantics to represent the static and dynamic characteristics of real world phenomena. The entity concept only provides a static view of the phenomena.

- **Attribute**

In OO programming languages, the concept attribute takes its value from a data type or a class. The attributes of a class are uniquely defined. The attributes that take their values from a class correspond to association links or aggregation links. The data types or domains can either be pre-defined types (i.e., integer, real, Boolean, date, string, etc.), user-defined enumerated types, or pre-defined types to which a set of rules or an interval limit is associated. Complex attributes are defined in most of the models [8].

- **Static links: Association link, Aggregation link, Use link and Generalisation link**

Static links are static relationships that have certain time duration between objects. The following four main static links are used in OO analysis methods:

Association link

Association link is inspired from the relationship concept of the E-R model. It is related to the form of abstraction in which a relationship between member objects is considered as a higher level object. It is used in the conceptual models such as OMT [13], OOSA [12], PTECH [26] and OOA [16]. Though the E-R model is simple and widely used, it has some drawback for mapping into OO specifications such as the confusion in mapping the relationship types. Certain relationship types on which dynamic characteristics can be defined behave like classes, while other relationship types behave like the attributes of class. It is not possible to explicitly model behavioural aspects of real world using the E-R model.

Aggregation link

The literature of aggregation link that appeared in semantic models dated from [10]. It is related to the form of abstraction in which a collection of objects are viewed as a single higher level object. The aggregation link defines part-of relationship between an aggregate object and one or several component objects. It is a special case of the association link. The semantics between two objects *x* and *y* can be described in two ways: *x <has a> y*, or inversely, *y <is a part of> x*, where *x* is the aggregate object and *y* is the component object. For instance, a car is composed of a body, an engine and several wheels. There are, however, several variations of aggregation links.

In [10], aggregation is defined to be an abstraction by which an association between objects is interpreted as a high-level object. The same approach is used in the ACM/PCM model [22], where an author is modelled as an aggregate of person and paper in a conference management. In the OMT model, Rumbaugh [13] advocates that two objects having independent lives should not be related by an aggregation link, instead it should be related by an association link. For instance, a society is an aggregate of its states, but not of its employees. The aggregation link is said to be related to the dynamic aspects of objects, in such a way that an operation of an aggregate object can be applied automatically to its component objects. The aggregation link is also employed in the OOA model [16], the MADIS model [15] and in [3, 4].

Use link

The notion of use link is used in OO design methods. It was introduced to aid in ADA software design. The use relationship is inspired from software engineering principles. It is increasingly gaining popularity in OO analysis methods. A use link [3] establishes a path for message sending from one object to another. Its semantics is that a client object "uses" the services of a server or supplier object. It is also called client-server link as in the OOA model [16]. As a result of its support for message sending, the use link establishes a strong dynamic link between two objects besides being considered as a static link. In [4], association links or aggregation links used in the analysis stage are transformed to use links in the design stage.

Generalisation link

The form of abstraction called generalisation captures the commonalties among objects and ignores the differences among them. The need to differentiate the inheritance concept used in OO programming languages from the one (generalisation/specialisation) used in OO conceptual models have been noted by several authors. For instance, inheritance by specialisation that is used for representing a real semantic link, is differentiated from inheritance by factorisation for reusing object characteristics. In the former

case, every instance of the specialised class belongs also to the generalised class, whereas in the latter case, an instance of the subclass does not belong to the superclass. It is possible that the superclass does not have instances, in which case, the superclass defines only methods and attribute declarations used by the terminal subclasses in the inheritance hierarchy. In this context, late binding mechanism is adopted by certain OO programming languages for accessing the attributes defined in the generalised class by an object of a specialised class.

Other related concepts used are: cluster [10] for relating to a set of specialised classes in which the instances are disjoint; overriding, denoting that instead of executing the method defined in the generalised class, the method defined in the specialised class that has the same name as the method in the generalised class will be executed; and augmentation mechanism, by which the method of the generalised class is executed in addition to the method of the specialised class.

- **Dynamic Links**

Dynamic features have been incorporated into some conceptual models such as REMORA [11], OBLOG [25] and MODWAY [27]. Dynamic links are used for characterising instantaneous relationships (interactions) that can occur between several evolving, related objects. In the functional approach, dynamic links can be represented by using DFDs. Two main categories of the systemic approach in which dynamic links can be modelled are procedural approach and declarative approach. The procedural approach uses the event concept of procedural type, while the declarative approach uses the event concept of declarative type. An event corresponds simply to something that occurs in the real world.

In the procedural approach, an event [15, 26] is expressed by an occurrence condition and a procedural part. The occurrence condition may define on the real world description external to the IS, on the states of the objects in the IS, or on the description of time. Three types of events are classified: external event, internal event, and temporal event. When the occurrence condition is satisfied, the procedural part is invoked to execute the actions on several objects simultaneously. The event concept following the declarative approach is used in the models such as OBLOG [25], MODWAY [27] and ERAE [1]. In this approach, the evolution of objects is caused by events. Each object has a life cycle consisting of the set of events affecting its life. This approach is closer to the perception of dynamics in the real world. Synchronisation is also proposed by different event sharing mechanisms such as in the OBLOG model and the MODWAY model.

- **Static and Dynamic Constraints**

A conceptual model must be rich enough to be able to define concepts for handling constraints on all possible worlds. In the literature, it can be noted that the stable properties and instantaneous aspects of ISs are constrained by constraints so that the consistency and reliability of ISs can be ensured. This makes the state of an IS to comply to the constraints described in a conceptual model. Some researchers equate constraints to rules that control the contents of the information base. However, there is still no accepted standard for modelling constraints, especially with respect to dynamic constraints. This area still remains a topic for on-going research.

Static constraints

Static constraints defined in a class are constraints that must be verified on the objects of the class at any moment, as a result, limiting the possible states of the objects. For instance, the static constraint of the aggregation link implicitly defined between a car and an engine, stipulates that an engine is always associated to a car. Other constraints that are not implied in the static links are constraints on attribute values (e.g., the attribute *age* of the class *EMPLOYEE* must have values greater than 17), or constraints on cardinalities of static links (e.g., the number of wheels of a car cannot be greater than 5). Some models allowing such constraints are OBLOG and TROLL [28].

Dynamic constraints

Dynamic constraints are constraints that restrict the evolution of objects with respect to time. The description of dynamic constraints normally involves the description of the temporal aspect. There are three commonly used approaches that allow dynamic constraints to be specified: the approach by state transition graphs, the approach by modal logic and the approach by temporal logic. The approach by state transition graphs is most widely used.

A state transition graph can be defined on a class. It describes the possible life cycle of objects in the class, i.e., the order of event occurrences affecting the states of the objects. The nodes of a state transition graph are object states. An object can be in one of these states during its life duration. The arcs of the graph correspond to the transitions from one state to another. An object undergoes a state change when it is affected by an event. The OMT model [13] uses the notion operation to describe how an object reacts in response to events, while OOSA [12] uses the model of Moore, by which the actions are not associated to transitions but to states.

In the temporal logic, two approaches are distinguished: the classical approach and the modal approach. In the classical approach, a time (interval) is assigned to every assertion. In the modal approach, some temporal modal operators are used to determine the time at which an assertion can be evaluated. A combination of the two approaches is used in some models. The OBLOG model and the TROLL model [28] allow formulas of the first order temporal logic to be established by defining temporal operators such as

sometimef() and *alwaysf()*, respectively signifying "at a given time in the future" and "always in the future". The OBLOG model uses a technique that describes the dependencies between events using the modal logic approach. Modal logic is used to express necessity. This logic represents what is possible (or impossible) to be done, and what is necessary to be done.

Comparatively, state transition graphs provide an abstract global view of the object behaviour. It is relatively easy to verify their completeness, but they must be associated to a mechanism that allows the representation of dynamic links. The advantage of the modal logic and temporal logic approaches is that dynamic constraints can be specified in the form relatively close to the natural language. However, it is quite difficult to verify their completeness.

3.2 Review of Issues in Formal Specification Languages

An analysis method short of a specification language would be meaningless. In the requirements engineering phase, the purpose of the specification language is to aid in the conceptual specifications of an IS and to serve as a basis for rapid prototyping. The specification language provides the specification support of the method for building precise specifications of ISs.

The increasing demand for precise and complete specification of ISs and the lack of expressiveness of existing graphical languages, result in the need to have a specification language capable of providing the means for expressing all kinds of semantics of an IS to enhance the conceptual model. In [29], a specification language named O*L, which consists of three integrated sublanguages: O* Query Language, O* Procedural Language and O* Definition Language, is defined. Each of these sublanguages provides a specific aspect of the specification support. These sublanguages when taken together, not only allows a wide variety of systems to be precisely specified at the conceptual level, but also helps in code generation.

Much research has been dedicated to the definition of database language interfaces for defining database specification and for formulating high-level declarative ad-hoc queries in traditional database management systems, such as data definition language and high-level data manipulation language [9]. Numerous object language interfaces can also be seen in the literature recently with the increasing adoption of the OO paradigm. Among others, the object definition and manipulation languages of the standard ODMG (Object Database Management Group) model [30] have gained increasing popularity, especially in the area related to the development of object repositories. Rather than providing only a high-level language such as SQL for data manipulation, OO programming and database technologies are the main sources of inspiration and motivation for the definition of a standard language for object database models by the ODMG. The standard ODMG model consists of three components: *Object Definition Language (ODL)* as the data definition language for high-level definition of database objects, *Object Query Language (OQL)* as a declarative (non-procedural) language for querying and updating database objects, and C++ Language Binding for writing portable C++ code that manipulates persistent objects. Both SQL and O2SQL [31] provide the basis for the definition of OQL.

The specification language of a conceptual model must also be rich enough to describe constraints on all possible worlds. It must take into account the requirements stated in [32], i.e., it should be expressible in natural language; it should be able to express all types of constraints; it should be integrated with the rest of the conceptual model; and it should be able to express all constraints by using a few basic structures and it should be applicable for rapid prototyping. This includes the capability for the representation of time, for instance, for describing admissible histories of a system by using temporal logic; the capability for textual expression of typical patterns of constraints and the capability for declarative specification of the chronological order of event occurrences such as those used in ERAE and CPL.

Indeed, there is a great tendency for the development of specification languages to provide different integrated sublanguages for dealing with the different aspects of the specification environment - query aspect, procedural aspect and declarative aspect. This is especially useful for prototyping of ISs before the actual implementation.

4.0 ABSTRACTION: REVIEW OF PROBLEMATIC ISSUES

The emergence of conceptual models have brought with it the issue of providing abstraction at the conceptual level for dealing with the enormous information represented in the conceptual schemas of large-scale ISs. Abstraction has been proven to be a powerful OO construct and as a way of reducing complexity of software. Software productivity has been demonstrated to be inversely proportional to software complexity [33]. As such, abstraction is a way for improving software productivity. The main aim of abstraction is to allow real world phenomena to be represented in a more significant nature by omitting irrelevant details, and thus, leading to a better comprehension. This is important as the concepts formalised by most existing conceptual models do not model real world phenomena as perceived by the end user. An efficient abstraction mechanism allows an abstract structure to be constructed at the appropriate level of abstraction from the basic building blocks of an IS.

Due to the increasing demand of large and complex ISs, in contrast to the lack of abstraction support of existing conceptual models to cater for such needs, abstraction mechanisms are formalised as enhancement facility for

conceptual modelling to increase the semantic power of some conceptual models by allowing all facets of the elements defined in a conceptual schema to be represented at the appropriate levels of detail. Indeed, most existing analysis methods either provide abstraction support for only the static aspects of ISs, or do not provide a uniform integration of both the static and dynamic aspects.

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 [2, 19, 34] and object decomposition [3, 16]. At this level of software design and reuse, abstraction focuses on essential properties of systems organisation. It captures the basic characteristics of significant components of the system and their interactions. Hence, a special term called higher-level abstraction, has been employed by Shaw [33] 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 SA techniques [19, 34], among which DFD [34, 13] is the most common technique used in the decomposition process. On the other hand, object decomposition defines a system as a hierarchy of objects. In [16], objects belonging to the same domain are grouped into a subject area. In this way, it is possible to define a collection of objects sharing some basic properties and behaviour. Functionality of a system alone is no longer an important criterion in such decomposition.

Concepts have been introduced and defined in the literature to deal with abstraction, such as subsystem [35], cluster [36, 37], schema [38], semantic context and abstract class [39], to name just a few, where each concept represents an abstract structure. These concepts have been defined in order to handle the immense information of large systems.

Abstraction has also been extensively applied in view management [40, 41]. Shilling [40] defines a view as a simplifying abstraction of a complex structure. Czejdo's work [41] involves integration of views in the Object-Relationship model of OSA. In spite of the numerous approaches providing some sort of abstraction support, there is not much research carried out to provide abstraction support for capturing also the behavioural aspects of objects, which are very much stressed by Wand [35] in his formalisation of ISs concepts. This is important as this kind of support provides abstraction of the interactions between objects, thus, allowing the user to perceive the IS at a different perspective.

5.0 CASE TOOLS: SURVEY, CUSTOMISA-TION AND INTEGRATION

To facilitate the modelling process, the method supporting conceptual modelling should be automated into a CASE tool. In fact, this realisation has been recognised not only in requirements engineering but also in design engineering that has led to a dramatic increase of tools supporting different aspects in ISs engineering since a decade ago.

5.1 Survey of CASE Tools

CASE tools emerged in the early 80s support only a limited number of tasks within a specific development phase. They mainly provide support for drawing models and code generation. For instance, the workbenches like Excelerator and Software Through Pictures provide modelling support for the E-R model and DFDs; while the code generators like Install/1 and Spectrum concentrate on generating application code [42]. The disadvantages of these CASE tools are two fold:

- they are rigid and inflexible - information engineers have to adapt their way of working to such tools.
- they are stand-alone and support only specific tasks.

The recent advancement in the field of CASE together with wide availability of meta-CASE tools in the computer market have not only helped the automation of analysis methods to become a reality, but also the facilitation and acceleration of the automation process [3, 16, 43]. The process of customising or building a CASE tool for a specific development method using a meta-CASE tool (CASE shell) is often called meta-modelling or methodology engineering [44]. The CASE shell architecture uses a meta-model as basis for supporting meta-modelling for the customisation of CASE tools. It aims at tailoring a CASE tool supporting a development method to a specific user-defined approach.

The emergence of meta-CASE tools has led to a proliferation at OO analysis and design tools [13, 15, 16, 29] to cater for requirements modelling in the early 1990s. Perhaps, this can be considered a great step forward in software development at the early phases of OOISs development life cycle. However, more often than not, the development tools supporting the various methods are stand-alone in the sense that they do not share data and are not agreed on a common standard. As such, this gives rise to the issues of tool integration with the aim of bridging the gap between tools supporting different phases of the development life cycle. As a result, the emergence of CASE tools at the beginning of 1990s also emphasised the integration of tools. In this respect, a CASE repository plays an important role for storing specifications from different CASE components and for sharing specifications among the integrated CASE components.

CASE shells are flexible and are tailored to customisation of CASE tools. They provide method independent support by integrating in their architecture a meta-model, and in certain cases, a meta-method for meta-modelling that can lead to the development of a CASE tool according to a specific user-defined approach. Method knowledge is

specified in the meta-CASE environment to be interpreted later. CASE shell architecture enables modification and extension of a tool's behaviour. Some commercial CASE shells are such as GraphTalk [45], Ramatic [44] and Finnish MetaEdit [46]. AWB [6, 43] and some existing methods defined in [6, 43] that are customised using a meta-CASE tool to provide modelling support for OO models.

5.2 Tools Integration

The need to alleviate the lack of coexistence of CASE tools gives rise to the emergence of another range of CASE tools tailored to the integration of tools in the late 1980s. This category of CASE tools often come with a built-in CASE repository to enable specifications from different integrated CASE components to be stored and shared.

Three main integration architecture stated in [7] are CASE framework, Integrated CASE tools (ICASE), and Integrated Project Support Environment (IPSE). The CASE framework is an open architecture for integrating tools supporting different development phases from multiple sources to form a cohesive and uniform tool set that covers the whole development life cycle via a uniform tool interface. The well-established norms such as PCTE (Portable Common Tools Environment) [47] developed in the context of ESPRIT program, IBM's AD/cycle [48] and ISO/IRDS all employ such an architecture. ICASE aims at supporting a well-defined approach throughout the whole development process via an integrated meta-model. Oracle*CASE developed by Oracle Corp. (UK) and Foundation developed by Andersen Consulting [42, 44] have these features. IPSE is a framework that shares several characteristics with the CASE framework and ICASE, with additional features such as administrative support functions like configuration management, multi-user support and multi-user access control.

Kronlöf [49] uses a different technique for tool integration, whose contention is that method integration is a prerequisite for successful tool integration. According to his strategy, a new method will be constructed for a given purpose from two or more existing methods. The aim of method integration is to combine the strengths and reduce the weaknesses of the selected methods when applied to the engineering process in question. By using the *Computer Aided Method Engineering (CAME)* tool, method engineering is supported with respect to the storage, retrieval and assembly of method fragments. The fragments of several ISs development methods stored in the so-called *method base* can be selected by some facilities of the tool.

To bridge the gap between the development tools supporting different phases of the development life cycle, for instance, AWB [6, 43] was integrated in the Emeraude PCTE [47] to cooperate with a design workbench tool and a scaleable browser tool. The objective is to develop an ISs development support environment that covers the whole development life cycle. In this case, the open architecture and conducive environment of the Emeraude PCTE facilitates the development of a CASE repository for integrating AWB with the design workbench tool and the scaleable browser tool.

6.0 SUGGESTIONS AND CONCLUSIONS

A review of the various problematic issues in requirements engineering of OOISs engineering has been presented. There is a need to define a well-defined conceptual model to overcome the lack of expressiveness of the conceptual model used in most existing analysis methods. It is recommended to develop a CASE tool supporting a certain development method via customisation using an efficient meta-CASE tool so as to sustain any changes that might be made to the tool. This is to overcome the tedious process of programming it in a hard-coded manner and also to eliminate any unnecessary errors that could be introduced. We also discuss some important issues in design engineering which are closely linked to those in requirements engineering, such as the integration of analysis and design tools so as to provide a uniform ISs development support environment.

The motivation of this paper is to propose a sound and stable foundation for the development of OOISs in the requirements engineering phase. The OO concepts are adopted due to their closeness to represent real world phenomena. We propose that the formalisation of conceptual modelling with object-orientation should encompass:

- an OO analysis method, covering a well-defined conceptual model with OO concepts and including concepts such as assertion and exception at the conceptual level for expressing constraints involving pending actions that cannot be controlled on all possible worlds;
- customisation of the CASE tool supporting the analysis method;
- abstraction mechanisms as an enhancement facility, providing the means for abstracting inter-related objects of an IS at different perspectives;
- a specification language, whose usefulness ranges from high-level specification to procedural specification of ISs at the conceptual level;

Fundamental concepts for modelling complex objects, system behaviour and temporal aspects should be provided by the conceptual model so that all kinds of phenomena of a real world system can be represented in the conceptual schema of the desired IS. The analysis method should be theoretically sound in order to establish a sound and stable foundation for conceptual modelling. The CASE tool supporting the method should be integrated with other development tools to form a unified whole for supporting the whole ISs development life cycle in a uniform way.

In order to take into account the complexity issue of complex real world systems and to facilitate the maintenance of ISs, the formalisation rules of the abstraction mechanisms should provide abstraction support for both the static and dynamic aspects of an IS for abstracting the relevant parts and inter-related objects of the IS at different levels of detail.

Owing to the need for precise specification of ISs, integrated sublanguages of the specification language should be formalised. Each sublanguage should provide an aspect of the specification support. These sublanguages should provide facilities ranging from high-level specification and enquiry, to procedural specification of ISs at the conceptual level, so that all kinds of semantics of the conceptual schema can be completely expressed.

As for further improvement, a reuse model should be incorporated into the OO methodology to serve as a basis for reusable components so that the development of ISs will be reuse-oriented. With a reuse model, it is also possible to incorporate different conceptual models on top of a single repository of reusable components.

REFERENCES

[1] E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaert and A. Rifaut, "A Knowledge Representation Language for Requirements Engineering" in *Proceedings of the IEEE*, Vol. 74, No. 10, 1986, pp. 1431-1444.

[2] 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.

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

[4] B. Henderson-Sellers and J. M. Edwards, "Analysis and Design" in *Methodologies and Notation, Tutorial TOOLS*, Paris, 1991.

[5] 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.

[6] S. P. Lee and C. Rolland, "Integration of the Tool (AWB) Supporting the O* Method in the PCTE-Based Software Engineering Environment" in *Object Technologies for Advanced Software in the Proceedings of the First JSSST Int. Symposium*, Springer-Verlag, Kanazawa, Japan, Nov. 4-6, 1993.

[7] J. A. Gulla, O. I. Lindland and G. Willumsen, "PPP: A Integrated CASE Environment" in *CAISE'91*, 1991, pp. 194-221.

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

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

[10] J. M. Smith and D. C. P. Smith, "Database Abstractions: Aggregation and Generalization" in *ACM Transactions on Database Systems*, Vol. 2, No. 2, 1977.

[11] 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.

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

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

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

[15] 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.

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

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

[18] D. T. Ross and K. G. Schoman, "Structured Analysis for Requirement Definition" in *IEEE Transactions on Software Engineering*, Vol. 3, No. 1, Jan. 1977.

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

[20] P. P. Chen, "The Entity-Relationship Model: Towards a Unified View of Data" in *ACM Transactions on Database Systems*, Vol. 1, No. 1, March, 1976.

[21] G. M. A. Verheijen and J. Van Bekkum, "NIAM: an Information Analysis Method" in *IFIP TC8 International Conference on Comparative Review of Information Systems Design Methodologies*, North-Holland, 1982.

[22] M. L. Brodie and E. Silva, "Active and Passive Component Modelling: ACM/PCM" in *IFIP TC8 International Conference on Comparative Review of Information Systems Design Methodologies*, North-Holland, 1982.

[23] M. R. Gustafsson, T. Karlsson and J. A. Bubenko, "A Declarative Approach to Conceptual Information Systems Modeling" in *IFIP TC8 International Conference on Comparative Review of Information Systems Design Methodologies*, North-Holland, 1982.

[24] TEMPORA, "ESPRIT II Project" in *Concepts Manual*, 1989.

[25] A. Sernadas, J. Fiedero, C. Sernadas and H. D. Ehrich, *The Basic Building Block of Information Systems, Information Systems Concepts: An In-depth Analysis*. Edited by E. Falkenberg, P. Lindgreen, North Holland, 1989.

[26] M. Fowler, "The Use of Object-Oriented Analysis in Medical Informatics for Large Integrated Systems" in *TOOLS*, Prentice Hall, Paris, 1991.

[27] C. Cauvet and F. Semmak "Abstraction Forms in Object-Oriented Conceptual Modelling: Localization, Aggregation and Generalization Extensions" in *Advanced Information Systems Engineering, CAISE*, Utrecht, Netherlands, June 1994.

[28] R. Jungclaus, G. Saake, T. Hartmann and C. Sernadas, "Object-Oriented Specification of Information Systems: The TROLL Language" in *Technische Universität Braunschweig*, Dec. 1991.

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

[30] Object Database Management Group, *Object Databases - The ODMG-93 Standard (Version 0.5)*. Edited by R. Cattell, SunSoft, 1993.

[31] F. Bancilhon, S. Cluet and C. Delobel, "A Query Language for an Object-Oriented Database System" in *Proceedings of the Second Workshop on Database Programming Languages*, Salishan, Oregon, USA, June 1989.

[32] F. Dignum, T. Kemme, W. Kreuzen, H. Weigand and R. P. van de Riet, "Constraint Modelling using a Conceptual Prototyping Language" in *DKE 2*, 1987, pp. 213-254.

[33] M. Shaw, "Toward Higher-Level Abstractions for Software Systems" in *Data & Knowledge Engineering*, Dec. 1989.

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

[35] 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.

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

[37] 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.

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

[39] S. P. Lee, J. Brunet and C. Rolland, "Abstraction in an Object-Oriented Method" in *Malaysian Journal of Computer Science*, Vol. 10, No. 1, June 1997.

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

[41] 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.

[42] C. McClure, "The CASE Experience" in *BYTE*, April 1989.

[43] J. Brunet, S. P. Lee and J-L. Barbe, "Analyst Workbench Tutorial" in *Esprit project 5311 (Business Class)*, Feb. 1992.

[44] M. Heym and H. Osterle, "Computer-aided methodology engineering" in *Information and Software Technology*, Vol. 35, No. 6/7, June 1993.

[45] Rank Zerox, "GraphTalk - Meta-modeling (Version 2.4)" in *Reference Manual*, 1992.

[46] MetaCase Consulting, *MetaEdit Personnal 1.2: User Manual*. Micro Works Finland, 1991.

[47] G. Boudier, F. Gallo, R. Minot and I. Thomas, "An Overview of PCTE and PCTE+" in *Proceedings of 3rd ACM Symposium on Software Environments*, Nov., 1988.

[48] V. J. Mercurio, B. F. Meyers, A. M. Nisbet and G. Radin, "AD/Cycle Strategy and Architecture" in *IBM Systems Journal*, Vol. 29, No. 2, 1990.

[49] K. Kronlöf, *Method Integration: Concepts and Case Studies*. Wiley Professional Computing, 1993.

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.