Expert Systems with Applications 39 (2012) 2893–2906

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A knowledge-based object modeling advisor for developing quality object models

Narasimha Bolloju [a,*], Vijayan Sugumaran [b,c]

[a] Department of Information Systems, City University of Hong Kong, Hong Kong
[b] School of Business Administration, Oakland University, Rochester, MI, USA
[c] Department of Service Systems Management and Engineering, Sogang Business School, Sogang University, Seoul 121-742, Republic of Korea

ARTICLE INFO

Keywords:
Class diagrams
Knowledge-based system
Model quality
Object modeling

ABSTRACT

Object models or class diagrams are widely used for capturing information system requirements in terms of classes with attributes and operations, and relationships among those classes. Although numerous guidelines are available for object modeling as part of requirements modeling, developing quality object models has always been considered a challenging task, especially for novice systems analysts in business environments. This paper presents an approach that can be used to support the development of quality object models. The approach is implemented as a knowledge-based system extension to an open source CASE tool to offer recommendations for improving the quality of object models. The knowledge component of this system incorporates an ontology of quality problems that is based on a conceptual model quality framework commonly found in object models, the findings of related empirical studies, and a set of analysis patterns. The results obtained from an empirical evaluation of the prototype demonstrate the utility of this system, especially with respect to recommendations related to the model completeness aspect of semantic quality.

1. Introduction

Systems analysts use conceptual modeling techniques during the analysis phase of information systems development to capture systems requirements. These requirements are represented using artifacts such as use case diagrams and descriptions, class diagrams, entity-relationship diagrams, and activity diagrams during the initial phases of the systems development life cycle. Many of these artifacts facilitate communication among stakeholders and system developers in defining, documenting and confirming system requirements. The cost and level of effort required to fix any errors in requirement specifications detected in the later phases of systems development can be extremely high (Boehm, 1981). Thus, the overall quality of conceptual models created in the early phases of systems development contributes significantly to the success of the system developed.

When developing conceptual models, novice or inexperienced systems analysts encounter difficulties in the areas of domain-specific knowledge, problem-structuring, and the cognitive process (Schenk, Vitalari, & Davis, 1998). A lack of understanding of domain knowledge among requirement engineers or systems analysts and miscommunication between users and technical personnel are two problems commonly associated with the early stages of systems development (de la Vara & Sánchez, 2008). The effectiveness of a conceptual model is affected by the complex relationships that exist between modeling constructs, task requirements, the analyst's modeling experience and cognitive abilities, and the interpreter's prior exposure to conceptual models (Wand & Weber, 2002). Although numerous guidelines on the development of quality conceptual models are available, novice analysts typically fail to benefit from such assistance due to their limited domain knowledge and modeling experience. Furthermore, the absence of standardized validation procedures makes conceptual modeling a complex task for novice analysts to perform efficiently and effectively (Shanks, Tansley, & Weber, 2003).

In recent years, Unified Modeling Language (UML) techniques (OMG, 2005) such as use case models and domain models (also known as object models) have been widely used for conceptual modeling in object-oriented systems development (Dobing & Parsons, 2006). However, researchers and practitioners criticize UML for its complexity and limited usability. For example, Siau and Cao (2001) evaluate the complexity of UML with respect to other modeling methods by using a set of structural metrics and found that UML is two to eleven times more complex than other modeling techniques, and that object models are the most complex of the UML artifacts to use. Though object models are considered extremely useful in supporting activities such as client verification,

* Corresponding author. Address: Tat Chee Avenue, Kowloon Tong, Hong Kong. Tel.: +852 34427545; fax: +852 34420370.
E-mail addresses: narsi.bolloju@cityu.edu.hk (N. Bolloju), sugumara@oakland.edu (V. Sugumaran).

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.08.151

clarifying technical understanding, programmer specifications, and maintenance documentation, even experienced analysts may not be using these models due to lack of understanding of this UML component (Dobing & Parsons, 2006). Novice analysts, in particular, have to overcome not only the steep learning curve associated with mastering object modeling techniques, but also the challenge of applying UML.

Considering the importance of object models (i.e., class diagrams) to systems development and the difficulties associated with the object modeling process, the need to support systems analysts should not be underestimated. Although typical computer-aided software engineering (CASE) tools include functionalities such as those enabling users to draw, document, and navigate among different models (Eriksson, Penker, Lyons, & Fado, 2004), they seldom offer capabilities that support the creation of high-quality object models. Recent years have seen several attempts to enhance the support provided in requirements modeling (e.g., Eshuis, 2006; Popescu, Rugaber, Medvidovic, & Berry, 2008; Shoval, Last, & Yampolsky, 2009). Novel extensions aimed at addressing this aspect of support are expected to not only contribute to the efficiency and effectiveness of the object modeling process, but also to help in shortening the learning process for novice analysts. Thus, the objectives of this study are to: (a) develop a knowledge-based approach that can be used to support the creation of quality UML object models; (b) implement this approach in a prototype as an extension to an existing CASE tool; and (c) demonstrate the usefulness of the prototype.

This paper presents the details of the proposed approach that exploits pattern knowledge for object model development, the implementation and evaluation of a knowledge-based system extension to an open source CASE tool. This extension will assist novice systems analysts in the important requirements capturing process task of developing quality object models. The applicable knowledge for this study is drawn from the conceptual model quality framework (Lindland, Sindre, & Solvberg, 1994) and collaboration patterns (Nicola, Mayfield, & Abney, 2001). The prototype system is evaluated by focusing on its utility of the recommendations provided. This study contributes to the extant literature by proposing a knowledge-based approach that combines a widely used conceptual model framework with a set of analysis patterns related to the business domain and by implementing it in an open source CASE tool to support the development of quality object models.

The rest of this paper is organized as follows. Section 2 reviews a widely used conceptual model quality framework and discusses three broad categories of approaches currently available for supporting the object modeling process. Section 3 presents an overview of the proposed approach and offers an example. Section 4 discusses the architecture and implementation of a knowledge-based system called the object modeling advisor (OMA) that is implemented as an extension to an open source CASE tool. Section 5 reports the results of an empirical evaluation of the extended functionality. After discussing the findings and limitations of this study and assessing its implications for research and practice in Section 6, the paper concludes by highlighting the contributions made to the existing literature.

2. Background

This section briefly reviews the conceptual model quality framework proposed by Lindland et al. (1994) and its suitability for the evaluation of object model quality. It also discusses the typical capabilities of current UML model checking tools and elaborates on three broad, but not distinct, categories of approaches available for supporting the object modeling process.

2.1. Quality of object models

The object modeling process involves the identification of classes, the attributes and operations of such classes, and the relationships among classes in the problem domain of interest. The resulting *object models* are graphically depicted as class diagrams. Object models have a relatively long history of use in both object-oriented programming and systems development (Siau, Erickson, & Lee, 2005) and are considered to be the most useful models in clarifying and understanding the static structure of a system among technical team members (Dobing & Parsons, 2006). However, the difficulties that arise in identifying required model elements (classes, attributes, operations, and relationships) often affect how closely the resulting model reflects the requirements of the system under consideration.

High quality object models not only adhere to commonly advocated notations and guidelines (e.g., the naming of classes, attributes, and associations; presentation layout for better readability), but also include required elements from the problem domain (e.g., no missing classes, attributes, or relationships; avoid using irrelevant classes, attributes, and relationships). Such quality object models can minimize communication problems and result in better understanding of model requirements, thereby reducing the effort required to accommodate such requirements at later stages of systems development.

Prior research related to conceptual model quality is relevant for studying the quality of object models. The quality of conceptual modeling can generally be divided into product and process quality (Poels, Nelson, Genero, & Piattini, 2003). Product quality is related to the resulting artifacts (*i.e.*, conceptual models), whereas process quality relates to the way in which the conceptual model is built. Among the studies addressing conceptual model quality surveyed by Wand and Weber (2002), frameworks for conceptual model quality provide a systematic structure for evaluation. Many quality criteria and frameworks have been proposed in the literature based on the desirable properties of data models (Batini, Ceri, & Navathe, 1992; Reingruber & Gregory, 1994). The framework proposed by Lindland et al. (1994) for evaluating the quality of conceptual models has gained wider acceptance due to its roots in the semiotic theory and applications, as reported in several empirical studies (Bolloju & Leung, 2006; Moody, Sindre, Brasethvik, & Solvberg, 2002; Moody, Sindre, & Brasethvik, 2003; Su & Ilebrekke, 2005). This quality framework addresses both process and product in assessing quality and is built on three linguistic concepts: syntax, semantics, and pragmatics. These concepts are applied to four aspects of modeling: language, domain, model, and audience participation.

According to this framework, the syntactic correctness of a model implies that all statements in the model accord with the syntax of the language. *Syntactic quality* captures how a given model adheres to language rules (i.e., syntax). Therefore, a smaller number of errors and deviations from the rules indicate better syntactic quality. *Semantic quality* is described in terms of validity and completeness goals. The validity goal specifies that all statements in the model are correct and relevant to the problem domain. The completeness goal specifies that the model contains all statements about the problem domain that are correct and relevant. However, it is possible that these two goals, unlike syntactic correctness, may not be achieved completely. *Pragmatic quality* addresses the comprehension aspect of the model from the stakeholders' perspective. It captures how the model has been selected from a large number of possible alternatives to express a single meaning and essentially deals with making the model easy to understand. The comprehension goal specifies that all audience members (or interpreters) completely understand the statements in the model that are relevant to them (i.e., the model projections).

Because the three quality categories of this framework view model quality from different perspectives, it is possible to develop systematic approaches used to assess and improve the quality of object models. Among the three categories, syntactic quality can be assessed by identifying violations of syntactic rules or conventions. Evaluating the semantic quality of a given model requires that the model be verified for completeness and validity. Pragmatic quality—which is more difficult to assess than the other two quality categories—requires significant support from different types of audiences for a given model. A number of tools exist for checking the quality of different types of models; the following section briefly discusses some of the popular UML model checking tools available.

2.2. UML model checking tools

Most of the existing UML model checking tools primarily rely on syntactic checking. Some of these tools are briefly described below. The model consistency checker (MCC) (Simmonds & Bastarrica, 2005) is a visual tool implemented as a plug-in for the Poseidon CASE tool. It provides the interface between Poseidon and the RACER description logic engine (Haarslev & Müller, 2001) and performs consistency checking at the UML model level. Any model loaded into Poseidon is also loaded into RACER using the fact extractor component. Intra-model consistency analysis is carried out by selecting consistency problems to be detected from a pre-established list of implemented selection predicates.

vUML is a tool used for verifying UML models, especially concurrent and distributed models containing active objects (Paltor & Lilius, 1999). Maintaining consistency between multiple UML diagrams that represent the static and dynamic aspects of a system is a non-trivial task, particularly when models are frequently changed (Kotb & Katayama, 2005). Although existing tools perform reasonably well in checking individual models such as in verifying UML activity diagrams (Eshuis & Wieringa, 2004), they have proven to be cumbersome for consistency checking and providing quality and timely feedback (Egyed, 2006). Lange (2006) advocates the use of metrics and visualization techniques to improve the quality of UML models during development. Egyed (2004) discusses a tool called UML/Analyzer that is integrated into IBM Rational Rose and automatically determines the most appropriate consistency checking rules to execute during model changes to detect inconsistencies and provide instant feedback to the designer. While most of the above tools focus on syntactic checking, Campbell, Cheng, McUmber, and Stirewalt (2002) describe an advanced tool that is capable of performing structural and behavioral analysis to detect errors and present them in terms of the original UML diagrams. This tool performs intra- and inter-diagram consistency checks.

The extant literature on model design verification includes studies that focus on the verification of UML models and the use of design patterns to improve the efficiency of automated verification techniques. For example, Mehlitz and Penix (2003) discuss a system called "design for verification (D4V)" that utilizes design patterns for component verification and selects a set of components that can be used to generate a system that meets specified attribute goals. They contend that this approach greatly improves the efficiency of automated verification through the use of domain-specific patterns and guidelines.

Design patterns are knowledge elements that can be created using a common language and then shared within a community for different purposes. Dietrich and Elgar (2007) present an approach to the formal definition of design patterns that uses the web ontology language (OWL) and related concepts such as pattern participant, pattern refinement, and pattern instance. They also discuss a prototype that can detect patterns in Java code using a set of predefined pattern definitions. To some extent, our work is similar to theirs in the sense that both approaches employ predefined patterns to detect pattern instances in software using rules. However, the rules used by Dietrich and Elgar are tightly coupled with the design patterns in a particular domain, whereas under our approach, the rules are loosely coupled with analysis patterns, are scalable, and are better at handling variations.

2.3. Supporting the development of object models

Prior to identifying or developing approaches to evaluating the quality of object models according to the three quality categories discussed in section 2.1, it is important to review existing approaches used to support object modeling. These approaches can be categorized into three groups based on their focus: (a) generation-focused approaches; (b) guidance-focused approaches; and (c) critique-focused approaches.

Generation-focused approaches cover a variety of mechanisms that attempt to create object models directly from textual descriptions of system requirements or through a question-and-answer session. Techniques such as natural language processing and expert systems are employed in this process. Wohed (2000) details a conceptual modeling prototype that gathers user responses to a sequence of six questions in incrementally building a conceptual model for the reservation domain. Natural language processing is used to translate requirements into conceptual models (Kaindl, 2004; Overmyer, Lavoie, & Rambow, 2001; Purao, 1998). A web-based prototype called APSARA creates object-oriented designs from simple requirement descriptions written in natural language by using heuristics and a patterns database (Purao, 1998). RETH—another example of a requirement engineering tool—uses natural language processing to generate associations and specialization relations between objects based on natural language definitions of classes (Kaindl, 2004). Natural language processing is also employed to extract words in a textual document and identify their part-of-speech and base forms to support analysts in tagging such words as classes, operations, or attributes before generating a corresponding object model (Overmyer et al., 2001). Purao, Storey, and Han (2003) present details of the development and empirical evaluation of an intelligent assistant that generates initial conceptual designs for later refinement. This intelligent assistant incorporates learning mechanisms to enhance analysis pattern reuse.

Guidance-focused approaches include mechanisms such as practitioner-oriented recommendations for developing quality models, analysis patterns and frameworks for specific domains, and wizards. Practitioner recommendations are often given as guidelines for various tasks such as identifying classes and relationships, naming and presenting conventions, and the use of analysis patterns (Batra, 2005; Bolloju, 2004) and frameworks. Some of these approaches employ reusable frameworks constructed by domain engineers to customize the model and suit the target environment (Hakala et al., 2001; Morisio, Travassos, & Stark, 2000; Viljamaa, 2001). Antony and Batra (2002) describe a rule-based expert system called CODASYS that guides database designers using question dialogs and warning messages to reduce errors by restricting the search space. Sugumaran and Storey (2002) and Sugumaran and Storey (2006) use domain ontologies to guide novice and experienced modelers in creating complete and consistent database designs.

Critique-focused approaches emphasize the provision of advice either during or after the development of conceptual models. For example, ArgoUML (Robbins & Redmiles, 1998, 2000) incorporates a set of critiquing features that address the cognitive needs of software designers by using agents that continuously check the existing model for potential problems and advise improvements. The critiquing approach used by Robbins and Redmiles (1998) also

includes correctness, completeness, consistency, and optimization recommendations that are generated through rules-based analysis. Both our approach and that of Robbins et al. present the user with a list of recommendations that can then be applied to the model. However, one distinguishing aspect of our approach is that the use of an ontology and rule base enables the recommendations generated to be meaningfully associated with each other. By assigning attributes or probabilities gathered through usage data to such relationships, more complex or even predictive recommendations can be made. In this sense, our approach is different from a critic, which is an independent entity that has no effect on the behavior of other critics.

Each of the above approaches has certain strengths and limitations. Although generation-focused approaches may appear attractive due to their automation, they depend on the quality of inputs such as textual descriptions of the requirements used and do not scale up very well to real-life systems. While guidance-focused approaches provide exhaustive lists of *dos* and *don'ts*, few of them are built into CASE tools; thus, to benefit from such approaches, novice analysts require some degree of support in dealing with the large number of guidelines. Critique-focused approaches, which are normally expected to identify errors in a given model and to provide recommendations, mostly offer broad sets of recommendations in the form of lengthy checklists that are often drawn from guidance-based approaches. They do, however, offer learning opportunities for inexperienced analysts.

3. Overview of the approach

As discussed in Section 2, supporting systems analysts in developing quality object models requires an innovative combination of different approaches both to benefit from their strengths and to address the weaknesses of individual approaches. This section presents an overview of our approach to supporting high-quality model development that makes use of the guidance-focused and critique-focused approaches discussed in the previous section. Our approach is implemented in a prototype system called object modeling advisor (OMA) and employs analysis patterns that are commonly found in business domains to provide recommendations on improving model quality. We describe the modeling process and provide an example to illustrate the nature of the support provided. Although our system offers recommendations that fall within both the syntactic and the semantic quality categories, this paper focuses on semantic quality, which makes use of collaboration patterns (see Appendix A).

Nicola et al. (2001) have developed a set of 12 domain-independent collaboration patterns (e.g., role-transaction, group-member, place-transaction) that frequently occur in business object models. These patterns define typical relationships between classes belonging to the four types of pattern players: *people*, *places*, *things*, and *events*. Generalization-specialization relationships are excluded from this set because no collaboration exists among sub-classes and super-classes and it is possible that classes representing any of the four types of pattern players can be specialized independently. The process of applying these patterns typically involves the selection of a number of key classes (e.g., a "store" class for the pattern player type *place*) from the problem domain before connecting these classes with other relevant classes (e.g., a "sale" class for the *transaction* type connected to the "store" class). Collaboration patterns can also be used to enhance the semantic quality of object models by identifying missing classes and validating their relationships (Bolloju & Leung, 2006).

Although a small set of 12 collaboration patterns may appear to be insufficient to cover all possible modeling situations, these patterns were in fact abstracted from over 30 or more complex analysis patterns (Coad, North, & Mayfield, 1997). In addition, by analyzing several object models, we have also observed that about 90% of relationship pairs (excluding the generalization type) can be captured by these patterns. Furthermore, the simplicity of these patterns makes their application much easier than the application of a large number of complex patterns.

3.1. Procedure

The overall procedure for developing quality object models is shown in Fig. 1. It consists of the following steps: (a) develop the initial object model; (b) analyze the model; (c) review the recommendations; and (d) revise the object model. A systems analyst first develops an initial object model using the CASE tool and assigns collaboration patterns to each relationship. The knowledge-based model verification functionality is then invoked to diagnose the developed object model and provide recommendations. The analyst can revise the object model based on the recommendations and invoke the knowledge-based system functionality again after assigning collaboration pattern specifications for any new relationships introduced. This process can be repeated until all the applicable recommendations have been incorporated into the object model. Thus, it is possible to build an object model incrementally, to improve its quality with each iteration, and to understand the rationale behind each change.

The identification of semantic quality problems constitutes a major component of the model analysis step. This process includes an analysis of the given object model that involves the use of applicable collaboration patterns and the identification of underlying semantic completeness such as by pinpointing missing classes and associations that could later be used to generate suitable recommendations. As part of this analysis, different roles played by a given class in existing relationships are used to determine either missing patterns (i.e., to identify missing classes and relationships) or to validate multiplicities and aggregation types for the existing patterns. In addition, for the existing classes, the applicable pattern player types are used to identify missing attributes and operations by comparing them with the expected sets of attributes and

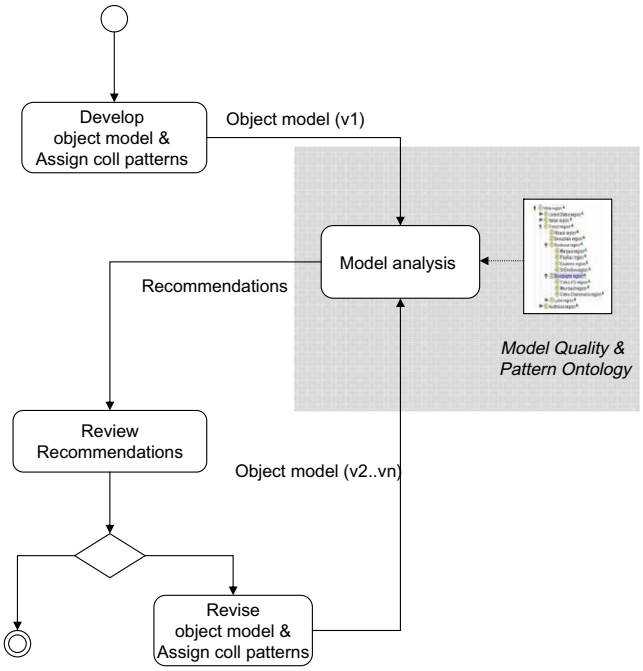

Fig. 1. Supporting high-quality model development.

operations. A formal representation of the patterns and object models and the algorithm for identifying semantic quality problems as part the model analysis step are presented in Appendix B.

3.2. An example

A segment of a video rental system object model (Fig. 2) is used to illustrate the different types of recommendations provided by the knowledge-based system extension. This object model segment represents various elements that capture a typical video rental problem domain. In the current version of the system, the user assigns one of the 12 collaboration patterns (see Appendix A) to each relationship. In this example, the three relationships are assigned the patterns (a) Role [R] – CompositeTransaction [CT]; (b) CompositeTransaction [CT] – LineItem [LI]; and (c) SpecificItem [SI] – LineItem [LI]. An instance of a composite transaction (e.g., "Sale" object) is composed of several line items (e.g., "SaleLineItem" objects for each item sold). The collaboration pattern player tags (e.g., +[R], +[CT], +[LI]) are shown at either end of the lines representing the relationships in Fig. 2.

The system reports 21 recommendations for improving the syntactic and semantic quality of the video rental object model. Part of this report is shown in Fig. 4, in which each recommendation consists of the type (an indication of the severity of the problem through an error, warning, or advice label), a diagnostic message, cases pointing to the location of the problem, and a suggestion with some representative examples. Seven out of the 21 recommendations are used to improve the object model (the revised model is shown in Fig. 3). The recommendations, some of which are not shown in Fig. 4, are used to revise the original model and include suggestions to: (a) change the multiplicity of unnamed associations; (b) change the multiplicity of aggregations ending from 0..* to 1..*; (c) use a better name for the association "have"; and (d) replace Video with Video and VideoCopy. Another recommendation directs the user to consider the possibility of using a particular pattern (Item-SpecificItem) to model the Video class as Video and VideoCopy classes. This recommendation also suggests the likely association between Item and SpecificItem.

A revised version of the object model incorporating some of the recommendations is depicted in Fig. 3. It can be observed that this object model exhibits better quality due to the inclusion of some missing elements (classes and associations) suggested by the system.

4. OMA design and implementation

This section discusses the design and implementation of the OMA prototype. As illustrated in Section 3, this knowledge-based system provides facilities for the specification of collaboration pattern tags for each relationship in an object model under development and for performing diagnosis to offer recommendations. This section presents the details of the tools selected for building this functionality, the system architecture, and the knowledge base representation.

4.1. Selection of development tools

The prototype system has been implemented using a number of open source software frameworks. ArgoUML (2006) is selected as the CASE tool for extension because it has several advantages over other open source tools such as StarUML, BOUML, and UMLet. Because ArgoUML is compliant with the UML 1.4 specification and is written in Java, it can easily be integrated with any external Java library. In addition, there is an active development community available to offer technical support in dealing with any implementation difficulties. To support its integration with external components, ArgoUML provides a set of application programming interfaces to extend its functionality. New functionality can be added as an extension module loaded when ArgoUML starts without making changes to the original code.

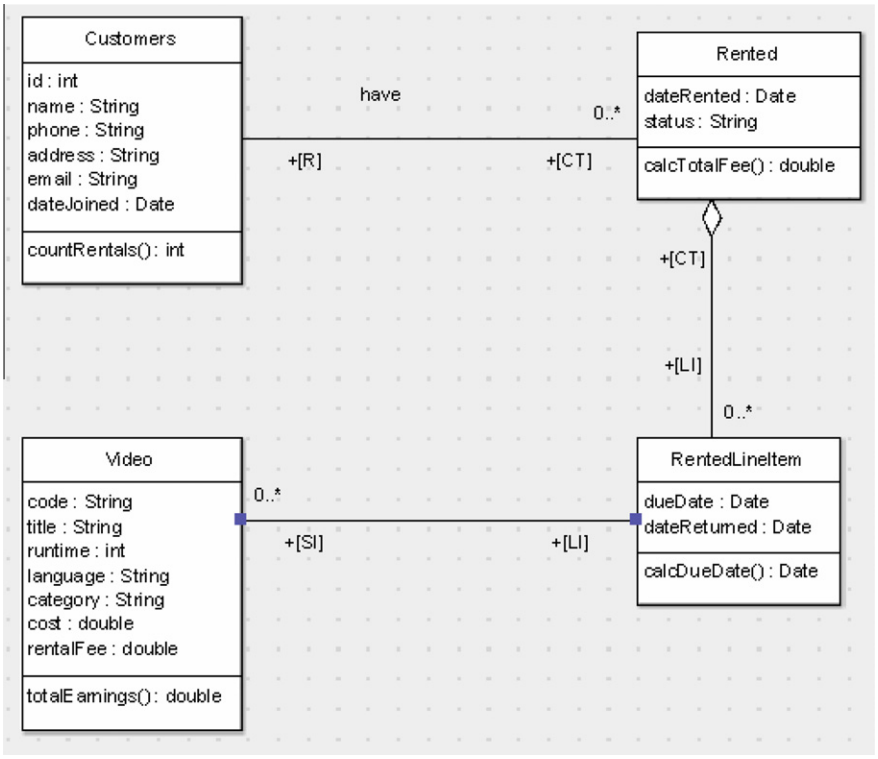

Fig. 2. Object model of video rental system.

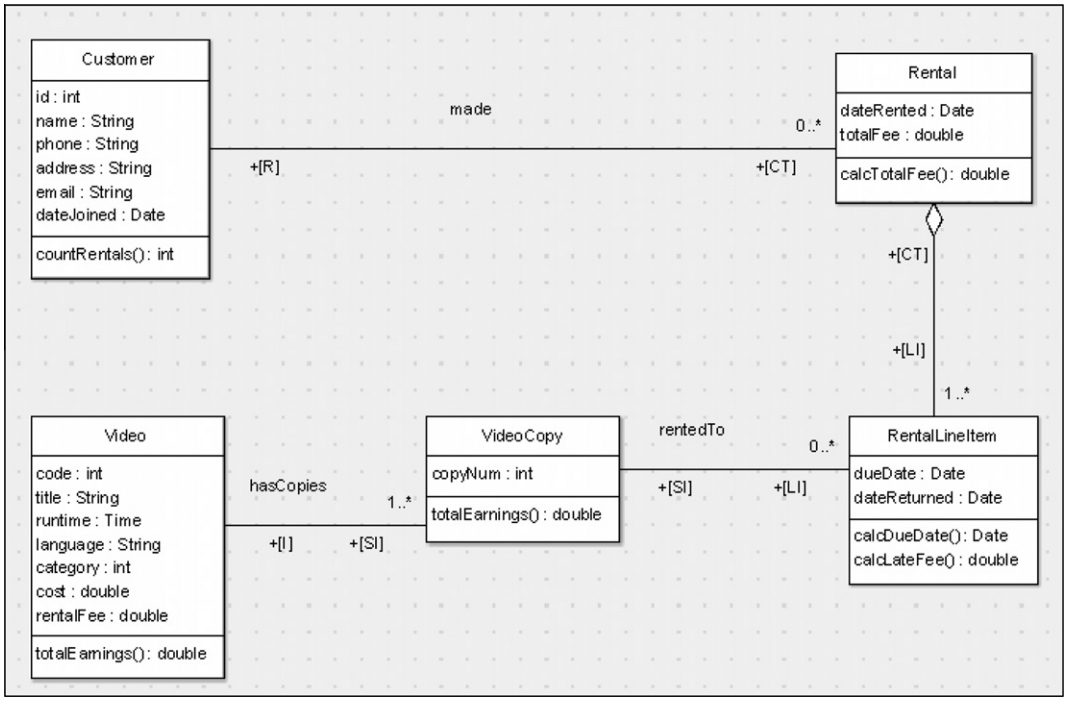

Fig. 3. Revised object model of video rental system.

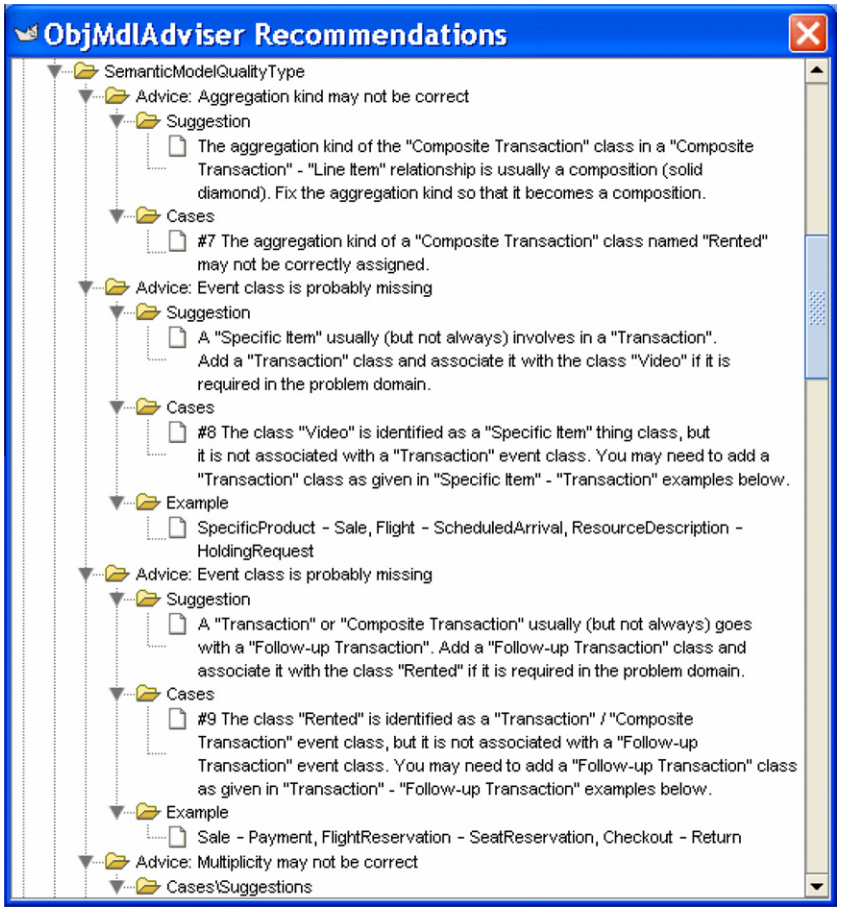

Fig. 4. Sample recommendations for semantic quality improvement of the model in Fig. 2.

Various other open source tools have also been used in implementing our prototype. Protégé (Knublauch, Fergerson, Noy, & Musen, 2004) is used as the ontology editor to manually construct the ontology and save it in OWL DL format (McGuinness & van Harmelen, 2004). Jess (Friedman-Hill, 2003) supports the development of the rule-based system. JessTab (Eriksson, 2004) integrates Protégé and Jess so that the Jess rules can work on the ontology. In addition, other tools such as WordNet (Miller & Hristea, 2006) and the Stanford log-linear part-of-speech tagger (SLLPOST, 2006) have been used to support the parsing and analysis of identifiers used for naming classes, attributes, operations, and associations.

4.2. System architecture

Fig. 5 depicts the architecture of the OMA system. It primarily consists of the following six components (shown as shaded boxes): (a) a collaboration pattern module; (b) an adapter; (c) an ontology; (d) a rule base; (e) collaboration pattern extended knowledge; and (f) a utilities wrapper. The *collaboration pattern module* plug-in enables collaboration pattern tags to be assigned to various relationships. These tags are stored in the ArgoUML data structures that correspond to the object model under development and must therefore be specified only once. The *adapter* component facilitates the interaction between ArgoUML and the OMA core. This adapter translates object models in ArgoUML data structures into ontology instances and controls the interaction between ArgoUML and other system components. The knowledge base is expressed as a combination of an ontology and a set of rules (described in Section 4.3). Performing reasoning on the current state of the ontology is the responsibility of the rule-based system which, in turn, uses the services provided by various third-party libraries accessed through the utilities wrapper component.

4.3. The knowledge base

The knowledge base of OMA consists of the *ontology* and *rule base* components and is built according to guidelines taken from several sources (Ambler, 2005; Coad et al., 1997; Larman, 2001; Nicola et al., 2001). These guidelines are categorized and collated into syntactic, semantic, and pragmatic categories. A subset of the guidelines, excluding those belonging to the pragmatic category based on implementation feasibility, has been selected. The rules for assessing semantic quality, however, are mostly drawn from the collaboration patterns described in Section 3.1.

The support provided to enhance semantic quality addresses both the validity and completeness aspects of object models. The completeness aspect of semantic quality requires that elements from the domain of interest be captured in the object model. Domain-independent collaboration patterns (see Appendix A) have been found to be extremely effective in identifying errors such as missing classes, missing relationships, and wrong association multiplicities (Bolloju, 2004). For example, if a class of the *transaction* type in a given object model is not associated with any class of

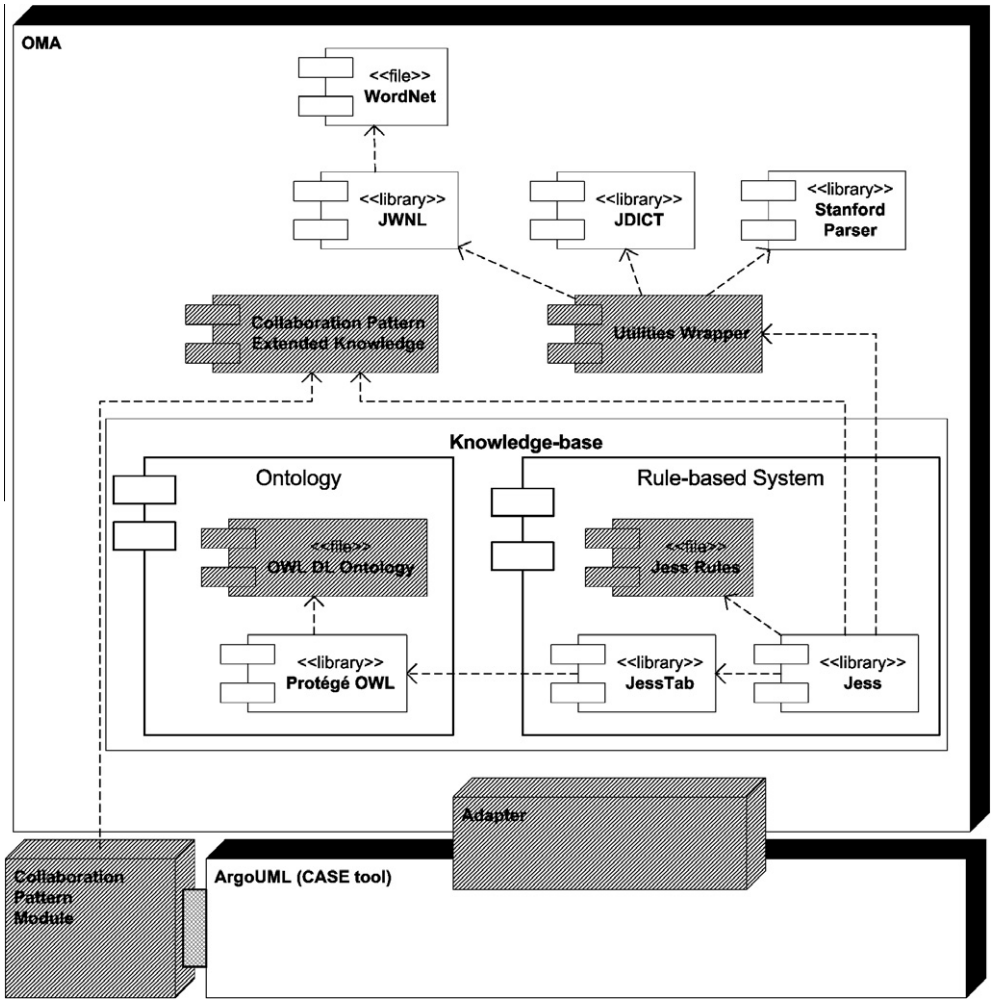

Fig. 5. OMA architecture.

the *role* type, this results in a warning message indicating a missing role and/or an association with an existing role class.

The validity aspect of semantic quality indicates that elements such as classes, relationships, attributes, and operations present in the model are, in fact, drawn from the problem domain. Collaboration patterns can be helpful in finding potentially incorrect multiplicities and aggregation by analyzing the difference between the expected multiplicity and the actual multiplicity used in a given collaboration.

The recommendations provided by OMA are classified into three types: errors, warnings, and advice. Most recommendations for syntactic quality improvements are categorized as errors (e.g., missing association names) or warnings (e.g., not using a singular noun for a class name). Most recommendations for semantic quality improvements fall into the advice category (e.g., missing attributes in a class).

Protégé is used to create the ontology representing syntactic and semantic quality problems and the elements of class diagrams are imported from ArgoUML. Fig. 6 depicts a segment of the ontology. Nodes in the ontology corresponding to different types of model elements (e.g., class name) are extended by other types of nodes as specializations representing associated quality problems (e.g., improper class name). Different types of model elements are extended to represent elements with quality problems (e.g., improper class names). The ontology also stores knowledge about the pattern players, multiplicities, and aggregation types (composition or aggregation) of each collaboration pattern. Elements exhibiting quality deficiencies are linked to a corresponding diagnosis to describe the problem that is then linked to an associated suggestion.

The rule base component contains a collection of Jess rules that specify how inferencing is carried out using instances in the ontology corresponding to the active object model. This component completes the inferencing process in three stages, with a separate set of rules being applied in each stage: (1) update the properties of each instance in the ontology; (2) classify instances with quality deficiencies; and (3) create diagnosis reports for each instance. Collaboration patterns corresponding to each relationship are created and used to find any missing collaborations (i.e., the corresponding generic class and a relationship) for a given class in the object model, as well as to find any invalid relationship multiplicities.

The *Collaboration Patterns Extended Knowledge* component includes: (a) for each pattern player type, a database of frequently used attributes and operations enabling the rule-based system to determine whether a class is missing critical attributes or operations; and (b) for each collaboration pattern, a database of relevant class names found in example cases enabling the system to suggest the most applicable collaboration patterns.

The knowledge base is represented as a combination of ontology and rules to improve the flexibility and performance of the implementation. Managing the domain knowledge as an ontology increases its readability and maintainability as it is easier to visualize relationships between concepts than in other formalisms such as rule-based representations. This also improves the reusability of the knowledge component in other applications, allowing that component to be shared among people and computers. Finally, the architecture of the implemented system allows the OMA core to be seamlessly integrated with different open-source CASE tool environments by developing adapters without affecting other components.

5. Evaluation

The effectiveness of OMA should ideally be assessed by analyzing the quality improvements made to object models created by novice analysts working on real-world projects based on recommendations provided by the system. Due to the challenges and difficulties commonly encountered in setting up field experiments for such an evaluation, one alternative is to conduct a laboratory experiment. Although a controlled experiment may in this case address the issue of rigor, typical object models created and revised by novice analysts in a two to three hour timeframe can be very simple and the utility of the results obtained may be limited. OMA is evaluated to: (a) assess the performance of the prototype system using several object models as test scenarios; (b) collect feedback from two independent and experienced systems analysts on the utility of the recommendations; and (c) study the quality of the revised models generated by novice systems analysts by implementing the recommendations provided by our tool.

5.1. Evaluation using the test scenarios

For this evaluation, we used 24 object models from the semester-long project work reports submitted by small teams of undergraduate information systems students from a systems analysis course. These object models cover 8 different application areas such as banking, insurance, the travel industry, logistics, and airlines. Although students learning object modeling can make many mistakes, the object models used in this evaluation can be considered to be reasonable representations of the work performed by novice systems analysts because of the duration and complexity of their projects and the opportunities the teams had to make revisions before making their final project submissions. Table 1 summarizes the characteristics of the 24 object models used in this evaluation. These characteristics indicate the size and complexity of the problem domains considered in the student projects.

A graduate student and a research assistant who have some training in object-oriented systems analysis performed the task of assigning collaboration patterns to various relationships in these object models. This task of assigning patterns is found to be relatively straightforward, even for novice analysts, because only 12 simple patterns need to be considered in assigning such patterns to various relationships in the object model. Only a very small percentage (less than 5%) of the relationships were not assigned to any of the 12 collaboration patterns. Furthermore, a study on the effect of collaboration patterns on the quality of object models revealed that the systems analysis undergraduate course students were able to assign patterns after just 20 min of training and that only 7 of the 12 patterns examined were frequently used (Bolloju, 2004).

Table 2 presents the counts of various types of recommendations provided by OMA. Although these counts amount to an average of about one hundred recommendations for a typical class diagram with 25 classes, the system groups together multiple occurrences of a particular type of recommendation (e.g., missing association name) before presenting them in a pop-up window. This type of presentation reduces a large number of recommendations to a list of a manageable size. In addition, this form of presentation is expected to contribute to learning effectiveness because the modeler is given an opportunity to go over multiple occurrences of a particular type of recommendation. The categorization of the recommendations as errors, warnings, and advice also contributed to reducing the cognitive load of the user in understanding and interpreting the recommendations.

About 9% of the recommendations made on semantic quality improvement are related to the validity aspect of semantic quality, with most suggesting incompatible association multiplicities. Among the remaining recommendations, missing classes, attributes, and operations account for approximately 36%, 24%, and 31%, respectively. Due to being based on the use of collaboration patterns, each recommendation related to a missing class also identifies the type of association required to link the missing class to the existing class for which the recommendation is made. Although it is not necessary that all classes suggested be applicable

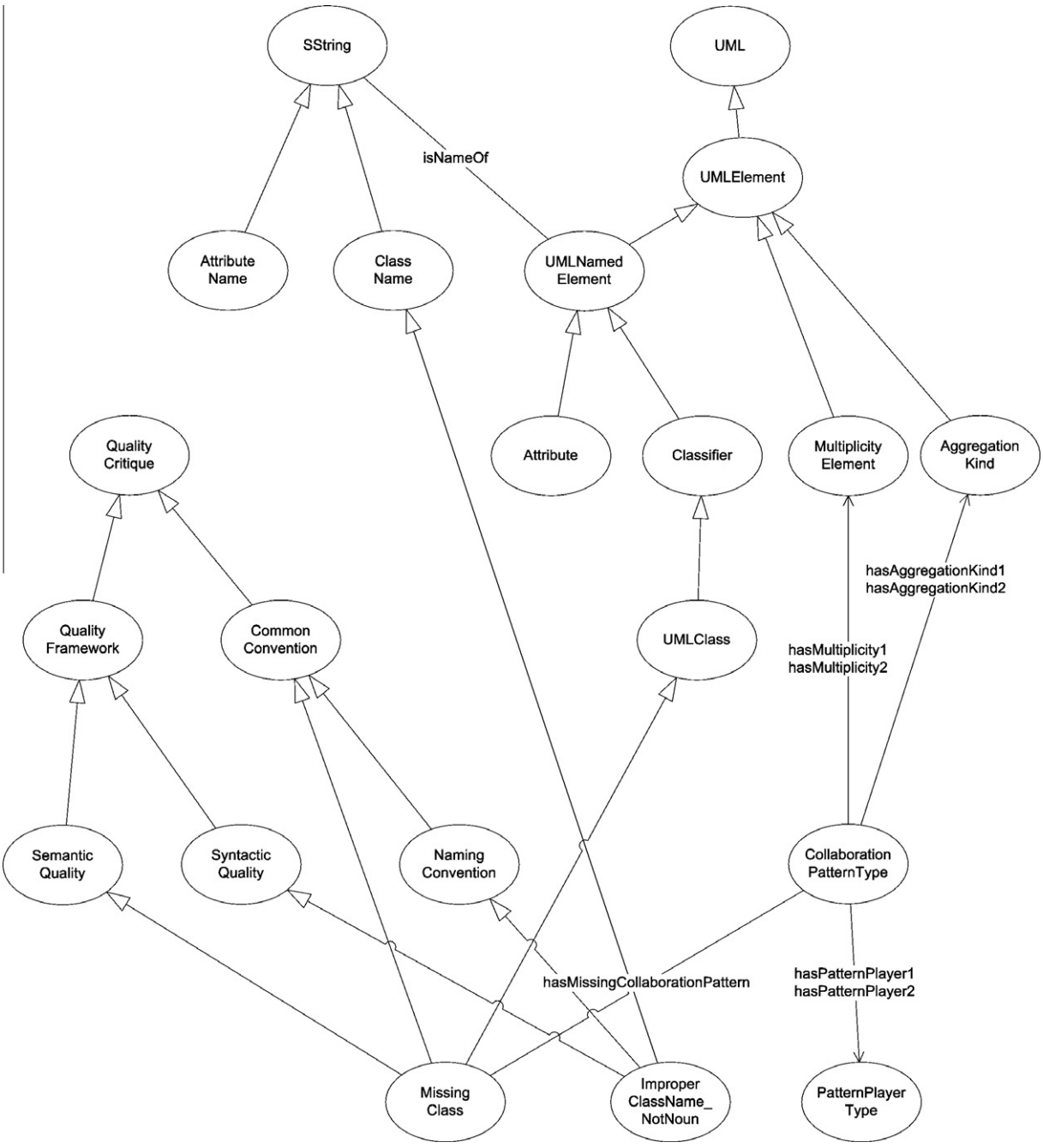

Fig. 6. Ontology segment.

Table 1
Characteristics of object models used for evaluation.

UML element	Average	Minimum	Maximum
Classes	21	14	28
Attributes	71	40	120
Operations	25	0	52
Specializations	6	2	15
Associations	18	9	26
Aggregations	3	0	8

to a given problem domain, these recommendations provide an opportunity for the modeler to consider such classes and to ensure the completeness of the object model. Recommendations on missing attributes and missing operations in a given class also provide examples of attributes and operations relevant to that class.

5.2. Assessment of recommendations by experienced systems analysts

To further understand the extent of the support provided by OMA, the recommendations it makes are assessed by two experienced systems analysts. These analysts were neither aware of the systems being evaluated nor associated with the authors in any other research work. It is assumed that experienced analysts are more capable of interpreting the value of recommendations than are novice analysts. Three representative object models from three different application areas were randomly selected from those used in the test scenario evaluation.

Most of the recommendations made in the semantic category are due to the use of collaboration patterns in OMA in identifying missing classes (71), missing attributes (66) missing operations (64), and invalid relationship multiplicities (11). Thus, the

Table 2
Recommendations provided.

	Syntactic	Semantic	Total	Average
Errors	884	0	884	36.8
Warnings	103	5	108	4.5
Advice	13	1442	1455	60.6
Total	1000	1447	2447	
Average	41.7	60.3	102.0	

recommendations provided by OMA contribute mostly towards ensuring the completeness of the object model.

The analysts were instructed to assess the utility of each recommendation on a scale of 1 (lowest) −7 (highest). Multiple occurrences of similar recommendations are presented as one item for this assessment. The analysts were also given the corresponding business descriptions and use case descriptions of the three projects to assist them in conducting their assessment. The two analysts reviewed each recommendation with respect to its applicability to the object model concerned and its contribution in improving the object model. The mean scores obtained through subjective assessment of different types of recommendations indicates that the recommendations made by OMA are valuable (5.04(1.53) and 4.96(1.11) mean and standard deviations of 78 recommendations assessed by two experienced analysts respectively).

5.3. Evaluation of OMA with novice systems analysts

To demonstrate the usefulness of OMA, we conducted further evaluation of the system with novice systems analysts who are the typical users of this tool. The OMA system offers recommendations that can be used to improve the quality of objects models that already exist or at the time of their creation. For this validation exercise, two novice analysts enhanced a set of four existing object models by understanding and implementing the changes recommended by the system. An independent expert assessed the quality of the original and revised object models. By comparing the overall quality of the original model and the revised models, quality improvements were identified and conclusions were drawn on specific quality improvements.

Four representative projects of similar complexity (hotel, insurance, logistics, and travel information systems) were selected from the initial set of 24 projects for this validation exercise. The two novice analysts (NSA1 and NSA2) who participated in this experiment had completed a course on systems analysis using UML. They were neither associated with the selected projects (which are taken from a previous cohort) nor involved in developing the OMA system. The analysts were trained, in separate 40-minute sessions, on the use of our system, and collaboration patterns. They did not know each other and never communicated with each other during the evaluation period.

The novice analysts first read the project report and then used OMA to complete two iterations of pattern assignment, obtaining recommendations, interpreting the recommendations received, and making changes to the object model. For each project, the analysts were asked to keep track of the time needed to complete each task. For reading the project report, the analysts took an average of about 50 min to become familiar with each project report which included the business description, a use case diagram, several use case descriptions, and an object model.

For pattern assignment, the analysts required an average of about 30 min for both the iterations to each object model. The time required by OMA to analyze the object model and generate recommendations was relatively short – averaging about 2 min. For each relationship present in each object model, the analysts first decided whether or not a pattern can be assigned. Their decisions in this regard matched for 96.2% (75 out of 78) of the relationships found in the four object models. Among the relationships to which patterns were assigned, about 78% of the instances of pattern assignment were an exact match. Most of the differences in pattern assignment are due to subjectivity in interpreting classes of *item* type or *specific item* and in interpreting classes of *transaction* or *composite transaction*. Thus, it is evident from these results that there is not much subjectivity in pattern assignment.

The average time required for interpreting recommendations provided by the system and incorporating applicable changes to the object models is about 70 min. A detailed analysis of the recommendations received and the changes made to the object models by the analysts provides interesting results. Table 3 summarizes the key results obtained from this evaluation exercise. With OMA, the two analysts interpreted and acted on over 90% of the recommendations in exactly the same way (i.e., they decide either to implement the recommendation or not to implement it). Some of the recommendations did result in multiple changes. The differences in the recommendations received by the analysts were due to minor differences in pattern assignment.

An experienced systems analyst familiar with the conceptual model quality framework of Lindland et al. (1994) reviewed all the object models (one original version and four revised versions for each project) and made a subjective assessment of each model in terms of its syntactic and semantic quality. This expert evaluator was not involved in the study and had no relationship with the authors. The evaluator was instructed to consider the syntactic and semantic aspects of each object model in assessing its quality and to assign a score for each dimension using a 5-point scale (1 – low, 3 – medium, 5 – high). The quality scores for each dimension were aggregated for the models revised using OMA across all four projects. The resulting scores indicated an average improvement of 3.00(1.07) and 2.63(1.69) for syntactic and semantic quality of the object models.

The results of this evaluation exercise reconfirm the effectiveness of OMA in assisting novice systems analysts to improve the quality of object models. Considering that the novice analysts were not involved in working on the original object models developed by teams of students who also prepared the problem descriptions and system requirements, we believe that the quality improvements achieved could be much higher were these development teams to use OMA recommendations on draft versions of the object models. Furthermore, novice analysts with better problem domain knowledge may be able to interpret and implement even more recommendations.

6. Discussion and implications

6.1. Discussion

The results obtained from evaluating the OMA prototype illustrate its effectiveness in addressing certain difficulties commonly encountered by novice systems analysts in object modeling, especially in ensuring semantic completeness and validity. The two experienced systems analysts also found many of the recommendations provided by the system to be valuable. They observed that the recommendations provided by our system provided incremental value over those provided by ArgoUML, especially those related to semantic quality. The effectiveness of semantic recommendations, as interpreted and applied by novice systems analysts in several instances and as confirmed by the expert analyst (Section 5.3), also highlights the usability and utility of the recommendations provided by OMA in comparison with those provided by ArgoUML.

The recommendations made in relation to semantic quality confirm the role collaboration patterns play in identifying missing

Table 3
Types of revisions made to object models.

		NSA1	NSA2
Average # of recommendations received	Syntactic	20.5	23.75
	Semantic	8.25	7.5
Average # of recommendations applied	Syntactic	27.75	26.0
	Semantic	10	14.5
Types of revisions frequently made	Syntactic	Revised attribute names, class names, and association names; added association names	
	Semantic	Added new attributes, associations, and classes; removed associations, attributes, and classes	

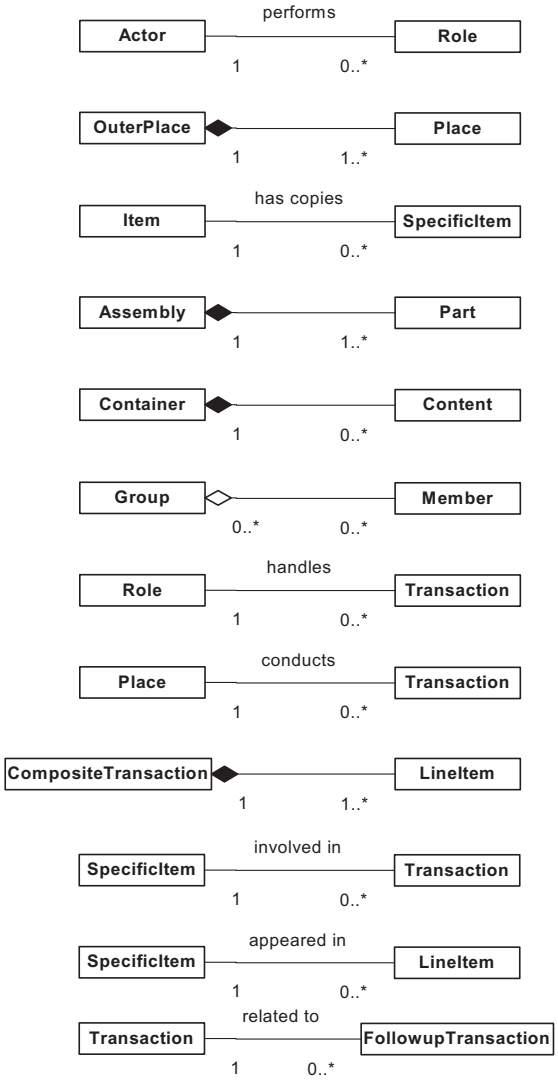

Fig. A.1. Collaboration patterns.

elements and invalid relationship multiplicities. While experienced systems analysts might routinely verify the need for various classes and relationships relevant to a given problem domain, novice analysts need support in this process due to their limited experience with different problem domains and their lack of modeling expertise. Novice analysts encounter difficulties in identifying relationships in data models due to the need to select useful relationships from among a large number of possible associations (Batra, 2005). The findings also confirm the utility of using patterns to identify new classes and associations and to validate association multiplicities (Bolloju, 2004). Although it is likely that many of the recommendations provided by OMA in this category may not be applicable to a given problem domain, they guide novice analysts in capturing as many requirements as possible. Thus, incorporating pattern-based support into CASE tools is valuable in helping novice analysts to develop more complete and valid object models.

6.2. Limitations

One limitation of OMA is the manual assignment of collaboration pattern player tags to various relationships in object models. Although the current version offers support to users by placing the most relevant patterns at the top of the list based on the patterns already assigned, this assignment process requires familiarity with collaboration patterns and results in some subjectivity in pattern assignment. The absence of domain-specific knowledge in the diagnosis and recommendation process is another limitation that results in a large number of recommendations.

Using student projects as test scenarios might have resulted in more positive findings on the recommendations made. Because the object models were drawn from teams of students working on semester-long projects in which they had opportunities to make revisions, the test scenarios may be considered to be a reasonable representation of models created by typical novice systems analysts. The incompleteness of the knowledge base used for diagnosis and recommendations is a limitation that is difficult to eliminate. However, significant effort has been made to gather as much knowledge related to the syntactic and semantic quality of the object models as possible (including knowledge related to entity-relationship modeling). The exclusion of pragmatic quality—one of the three components of Lindland et al.'s conceptual model quality framework—is a limitation that can be addressed by including domain-specific knowledge. Finally, the recommendations made by the OMA were assessed by only two experienced systems analysts who assessed three representative object models. Further research should employ a greater number of analysts and object models to provide more robust evidence confirming our findings.

The current version of our tool does not provide ontology and rule base evolution mechanisms; however, this aspect is beyond the scope of the current paper. Business domain-specific patterns are maintained in an XML document that can be expanded by introducing additional patterns. As part of our future work, we will develop a standard workflow and provide the necessary interface for managing the ontology and rule base. Similarly, our tool does not automate the process of applying recommendations. Automatically applying the recommendations to modify the model might be a challenging task, even when domain knowledge is introduced. Only some error types can be easily fixed. This feature is being addressed in a new tool under development (with linguistic analysis).

6.3. Implications for research and practice

We have identified several opportunities to advance the research presented in this paper. First, there is scope to include recommendations on how to improve the pragmatic quality of object models by considering different audiences such as end users and

systems developers. In dealing with pragmatic quality, considering even one type of audience can be quite challenging. Domain-specific knowledge (Sugumaran & Storey, 2006) could be employed for this purpose. This would enhance the overall quality of the recommendations made because domain knowledge could be used to prune the list of recommendations and to offer more specific recommendations. Second, laboratory or field experiments could be conducted to evaluate the effectiveness of similar extensions in supporting both the modeling process and the learning process of novice analysts. Employing novice and expert analysts in these experiments would lead to better understanding of the extent to which support systems are useful. Such experiments would be helpful in identifying differences in the types of errors committed and the patterns employed by experts in object modeling. Finally, the scope of the knowledge-based support systems can be expanded to other types of UML artifacts such as use case diagrams, activity diagrams, and sequence diagrams.

The practical implications of this study relate to various groups of stakeholders involved in information systems development. Project teams could employ similar tools or techniques to establish standard notations and guidelines for the development of quality object models. This type of system could also be used to train new recruits in object modeling. Finally, CASE tool developers could use our proposed system to provide intelligent support for novice systems analysts and adapt the level of support as analysts gain experience.

7. Conclusion

This paper discusses the development and evaluation of a knowledge-based extension to a CASE tool used to support novice systems analysts in building object models with better syntactic and semantic quality. This novel extension – exploiting pattern knowledge for object modeling – analyzes a given object model and offers recommendations based on semantic quality deficiencies identified in that model. The nature of the support provided in object modeling, as observed in the evaluation, demonstrates the benefits of employing such extensions to CASE tools.

This study makes two major contributions to the extant literature. First, it describes a knowledge-based system that offers intelligent support to novice systems analysts in addressing the difficulties associated with requirements capturing using object models. Second, it provides for a better understanding of the role of collaboration patterns in enhancing semantic quality. Based on the evaluation of the recommendations provided on how to enhance the semantic quality of object models, it can be argued that this extension to the capabilities of CASE tools is valuable to the systems requirements capturing process. We are pursuing further research to address some of the limitations of the current version of our system by eliminating less applicable recommendations, making use of the domain-specific knowledge available in use case descriptions or narratives, applying natural language processing techniques to automate the assignment of patterns, and providing more specific recommendations.

The research presented in this paper deals with the important issue of how to assist novice systems analysts in developing quality object models, an area that is not adequately addressed by either the research community or CASE tool developers. We find that there is a growing level of concern in this area and note that related research undertaken in recent years has been aimed at addressing the challenges faced in requirements specification such as checking UML activity diagrams (Eshuis & Wieringa, 2004), the preferred order for creating analysis artifacts (Shoval et al., 2009), and turning requirements engineering from a craft into a discipline (van Lamsweerde, 2008).

The support provided by most of the existing tools described in Section 2.2 is limited to the syntactic validation of domain models. In contrast, our approach supports both the syntactic and semantic validation of object models. In this respect, the building and evaluation of a knowledge-based system extension to a CASE tool highlights that using CASE tools to provide guidance from the quality perspective can contribute to the effectiveness and efficiency of the requirements modeling process. Furthermore, the use of such systems by novice systems analysts is expected to build their expertise more quickly. By supporting the modeling process with the objective of enhancing the overall quality of artifacts representing information system requirements, it is possible to minimize the effort otherwise required to accommodate system requirements in the later phases of systems development. It also makes the system implemented more successful as it is likely to include most of the required features and functionalities.

Acknowledgments

The work described in this paper was substantially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 110207). The work of the second author has been partly supported by Sogang Business School's World Class University Program (R31–20002) funded by Korea Research Foundation.

Appendix A. Collaboration patterns

Collaboration patterns (Nicola et al., 2001) involve pairs of classes belonging to the four types of pattern players: *people*, *places*, *things*, and *events*. The set of 12 primary patterns (Fig. A.1) can be effectively employed to identify relationships (other than generalization-specialization) during the development of object models.

Appendix B. Algorithm for identifying semantic quality problems

Let P be the set of generic collaboration patterns where each pattern is represented by a 5-tuple $\langle \mathbf{t}_1, \mathbf{m}_1, \mathbf{rt}, \mathbf{t}_2, \mathbf{m}_2 \rangle$ where

- $\mathbf{t}_1$ and $\mathbf{t}_2$ are the pattern player roles belonging to T, a set of pattern player types (*role*, *transaction*, etc.);
- **rt** is the relationship type (*generalization, aggregation, association*); and
- $\mathbf{m}_1$ and $\mathbf{m}_2$ are multiplicities corresponding to the roles $\mathbf{t}_1$ and $\mathbf{t}_2$, respectively.

In addition, for each pattern player type (**t**), the expected attributes (**A**) and operations (**O**) are represented as TAO, a 3-tuple $\langle \mathbf{t}, \mathbf{A}, \mathbf{O} \rangle$.

Let OM = (C,R) denote an object model where C is a set of classes and R is a set of relationships among the classes. Each member of C is a 3-tuple $\langle \mathbf{cn}, \mathbf{A}, \mathbf{O} \rangle$ representing a class name **cn**, a set of attributes **A**, and a set of operations **O**. Each member of R is an 8-tuple $\langle \mathbf{rt}, \mathbf{rn}, \mathbf{c}_1, \mathbf{m}_1, \mathbf{t}_1, \mathbf{c}_2, \mathbf{m}_2, \mathbf{t}_2 \rangle$ representing a relationship of type **rt** (generalization, aggregation, or association) between classes $\mathbf{c}_1$ and $\mathbf{c}_2$ (not necessarily distinct) with name **rn** (optional) and multiplicities $\mathbf{m}_1$ and $\mathbf{m}_2$ on either end, and $\mathbf{t}_1$ and $\mathbf{t}_2$ are the respective pattern player types of the classes on either end of the relationship.

Let SQP = (MC, MA, MO, IR) denote semantic quality problems identified through analyzing a given object model, where.

- MC is a set of 3-tuples $\langle \mathbf{c}, \mathbf{t}, \mathbf{p} \rangle$ representing a missing class of pattern player type **t** via pattern **p** that should be related to an existing class **c**;
- MA is a set of 2-tuples $\langle \mathbf{c}, \mathbf{t} \rangle$ representing missing attributes of class **c** for pattern player type **t**;

- MO is a set of 2-tuples $\langle \mathbf{c}, \mathbf{t} \rangle$ representing missing operations of class **c** for pattern player type **t**; and
- IR is a set of 3-tuples $\langle \mathbf{r}, \mathbf{cp}, \mathbf{pl} \rangle$ representing a quality problem label **pl** with relationship **r** related to pattern **cp**.

The functions used in the following algorithm are formally described as:

- getMissingCollaborationPatterns (c:class) returns a set of collaboration patterns $\{p | p \in P_c \wedge p \notin P_r\}$ where
  - P_c=$\{p | p \in P \wedge (p \cdot t_1 = t), \forall t \in$ getPatternPlayerTypes $(c)\}$ is a set of all collaboration patterns applicable to class c in which the pattern player types from the set returned by getPatternPlayerTypes (c) participate;
- P_r=$\{p | p \in P \wedge ((c = r \cdot c_1 \wedge p \cdot t_1 = r \cdot t_1 \wedge p \cdot t_2 = r \cdot t_2) \vee (c = r \cdot c_2 \wedge p \cdot t_2 = r \cdot t_1 \wedge p \cdot t_1 = r \cdot t_2)), \forall r \in \mathrm{R}\}$ is a set of all collaboration patterns from all the relationships that involve class c, such that the pattern player $p.t_1$ is always the pattern player of c in a relationship.
- getPatternPlayerTypes (c:class) returns $\{r \cdot t_1 | r \in R \wedge r \cdot c_1 = c\} \cup \{r \cdot t_2 | r \in R \wedge r \cdot c_2 = c\}$, which is a set of pattern player types for a given class **c** by collecting pattern player types from all the existing relationships in which **c** participates;
- getCollaborationPatternTypeByPlayer1(t:patternType) returns $\{p | p \in P \wedge (p \cdot t_1 = t \vee p \cdot t_2 = t)\}$, which is a set of all patterns derived from collaboration patterns P involving t on either side of the pattern (e.g., for the Transaction type, possible patterns include Role, Place, FollowupTransaction, etc.);
- matches (r, t, cp) instantiates a matching collaboration pattern $cp \in P$ such that $\mathbf{cp} \in P \wedge (cp \cdot t_1 = t \vee cp \cdot t_2 = t)$;
- multiplicitiesSubsumed (r:relationship,cp:collaborationPattern) returns true when the range of $r \cdot m1$ falls within $cp \cdot m_1$ and $r \cdot m_2$ falls within $cp \cdot m_2$ (e.g., the range 1...1 falls within 0...1);
- countSimilarPatternPlayerAttributes (t:patternType,A:attributeSet) returns the number of attribute names in A that match any of the attributes in $\{e \cdot A | e \in \mathrm{TAO} \wedge e \cdot t = t\}$;
- countSimilarPatternPlayerOperations (t:patternType,O:operationSet) returns the number of operation names in O that match any of the operations in $\{e \cdot O | e \in \mathrm{TAO} \wedge e \cdot t = t\}$.

```
findSemanticQualityProblems (
        (C,R):object model;
        (MC, MA, MO, IR):semantic quality problems)
begin
// find recommendations for each class in the object model
for each class c ∈ C do
begin
    // find missing relationships and classes for c
    for each p_m ∈ getMissingCollaborationPatterns (c) do
      MC ← MC ∪ {⟨c, p_m · t_2, p_m⟩};
      // missing class via relationship
    end for each p_m
    // get applicable patterns for class c
    for each t ∈ getPatternPlayerTypes (c) do
      //find any missing attributes for c based on role t
      if countSimilarPatternPlayerAttributes (t, c · A) < α_a
  then
      MA ← MA ∪ {⟨c,t⟩};// missing attributes for t
    end if
    //find any missing operations for c based on role t
    if countSimilarPatternPlayerOperations (t, c · O) < α_o then
      MO ← MO ∪ {⟨c, o>};// missing operations for t
    end if
    for each cp ∈ getCollaborationPatternTypeByPlayer1 (t)
    do
      if ∃ r ∈ R ∧ matches (r, t, cp) then
        begin
        // a pattern involving c exists in the object model
        if r.rt ≠ generalization then
          // it is an association type or an aggregation type
          begin
            // validate association multiplicity
          if ¬ multiplicitiesSubsumed (r, cp)
          then IR ← IR ∪ {⟨r, cp, invalidMult⟩};
          // invalid multiplicities
          if r · rt = AGGR ∧ conflicts (r, cp)
          // validate aggregation kind
          then IR ← IR ∪ {⟨r, cp, invalidAggr⟩};
        end;
        end
      end for each cp
    end for each t
  end for each c
end findSemanticQualityProblems
```

References

Ambler, S. W. (2005). *The elements of UML 2. 0 style*. Cambridge Univ Pr.

Antony, S. R., & Batra, D. (2002). CODASYS: A consulting tool for novice database designers. *SIGMIS Database, 33*(3), 54–68. doi:10.1145/569905.569911.

ArgoUML. (2006). ArgoUML. ArgoUML.

Batini, C., Ceri, S., & Navathe, S. B. (1992). Conceptual Database Design: An Entity-Relationship Approach. 1992. Benjamin Cummings.

Batra, D. (2005). Conceptual data modeling patterns: Representation and validation. *Journal of Database Management, 16*(2), 84–106.

Boehm, B. W. (1981). *Software engineering economics*. Prentice-Hall.

Bolloju, N. (2004). Improving the quality of business object models using collaboration patterns. *Communications of the ACM, 47*(7), 81–86. doi:10.1145/1005817.1005827.

Bolloju, N., & Leung, F. S. K. (2006). Assisting novice analysts in developing quality conceptual models with UML. *Communications of the ACM, 49*(7), 108–112.

Campbell, L. A., Cheng, B. H., McUmber, W. E., & Stirewalt, R. E. (2002). Automatically detecting and visualising errors in UML diagrams. *Requirements Engineering, 7*(4), 264–287.

Coad, P., North, D., & Mayfield, M. (1997). *Object models: Strategies, patterns, and applications*. Yourdon Press.

de la Vara, J. L., & Sánchez, J. (2008). Improving requirements analysis through business process modelling: A participative approach. In *Business information systems: Proceedings of the 11th international conference, Bis 2008, Innsbruck, Austria, May 5–7, 2008* (pp. 165). Springer.

Dietrich, J., & Elgar, C. (2007). Towards a web of patterns. *Web Semantics: Science, Services and Agents on the World Wide Web, 5*(2), 108–116.

Dobing, B., & Parsons, J. (2006). How UML is used. *Communications of the ACM, 49*(5), 109–113.

Egyed, A. (2004). Consistent adaptation and evolution of class diagrams during refinement. *Fundamental Approaches to Software Engineering*, 37–53.

Egyed, A. (2006). Instant consistency checking for the UML. In *Proceedings of the 28th international conference on Software engineering* (p. 390).

Eriksson, H. (2004). JessTab Manual—Integration of Protégé and Jess. Linköping University. <http://www.herzberg.ca.sandia.gov/jess>.

Eriksson, H., Penker, M., Lyons, B., & Fado, D. (2004). *UML 2 toolkit*. Wiley.

Eshuis, R. (2006). Symbolic model checking of UML activity diagrams. *ACM Transactions on Software Engineering and Methodology, 15*(1), 1–38.

Eshuis, R., & Wieringa, R. (2004). Tool support for verifying UML activity diagrams. *IEEE Transactions on Software Engineering, 30*(7), 437–447.

Friedman-Hill, E. (2003). Jess in action: rule-based systems in Java. Manning.

Haarslev, V., & Müller, R. (2001). RACER system description. *Automated Reasoning*, 701–705.

Hakala, M., Hautamaki, J., Koskimies, K., Paakki, J., Viljamaa, A., & Viljamaa, J. (2001). Annotating reusable software architectures with specialization patterns. In *Proceedings of the working IEEE/IFIP conference on software architecture* (pp. 171). Washington, DC, USA: IEEE Computer Society<http://www.portal.acm.org/citation.cfm?id=837415>.

Kaindl, H. (2004). Active tool support for requirements engineering through RETH. In *Proceedings of the 12th IEEE international requirements engineering conference, 2004* (pp. 362–363).

Knublauch, H., Fergerson, R. W., Noy, N. F., & Musen, M. A. (2004). The Protégé OWL plugin: An open development environment for semantic web applications. *The Semantic Web-ISWC, 2004*, 229–243.

Kotb, Y., & Katayama, T. (2005). Consistency checking of UML model diagrams using the xml semantics approach. In *Special interest tracks and posters of the 14th international conference on World Wide Web* (p. 983).

Lange, C.F. (2006). Improving the quality of UML models in practice. In *Proceedings of the 28th international conference on software engineering* (p. 996).

Larman, C. (2001). *Applying UML and patterns: An introduction to object-oriented analysis and design and the unified process*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Lindland, O. I., Sindre, G., & Solvberg, A. (1994). Understanding quality in conceptual modeling. *IEEE Software, 11*(2), 42–49.

McGuinness, D. L., & van Harmelen, F. (2004). OWL Web Ontology Language Overview. W3C Recommendation, February 2004.

Mehlitz, P. C., & Penix, J. (2003). Design for verification using design patterns to build reliable systems. In *Proceedings of 6th workshop on component-based software engineering*.

Miller, G. A., & Hristea, F. (2006). WordNet nouns: Classes and instances. *Computational Linguistics, 32*(1), 1–3.

Moody, D. L., Sindre, G., Brasethvik, T., & Solvberg, A. (2002). Evaluating the quality of process models: Empirical analysis of a quality framework. In *21st International conference on conceptual modeling (ER'2002), Tampere, Finland, October* (pp. 7–11).

Moody, D. L., Sindre, G., Brasethvik, T., et al. (2003). Evaluating the quality of information models: Empirical testing of a conceptual model quality framework. In *Proceedings of conceptual modeling ER 2002*, LNCS Volume 2503. (pp. 380–396).

Morisio, M., Travassos, G. H., & Stark, M. E. (2000). Extending UML to support domain analysis. In *Proceedings of the 15th IEEE international conference on automated software engineering* (pp. 321). Grenoble, France: IEEE Computer Society<http://www.portal.acm.org/citation.cfm?id=786950>.

Nicola, J., Mayfield, M., & Abney, M. (2001). *Streamlined object modeling: patterns, rules, and implementation*. Prentice Hall.

OMG. (2005). Superstructure, version 2.0. Specification, OMG, 2005. Object Managment Group.

Overmyer, S. P., Lavoie, B., & Rambow, O. (2001). Conceptual modeling through linguistic analysis using LIDA. In *icse* (p. 0401).

Paltor, I., & Lilius, J. (1999). vUML: A Tool for Verifying UML Models. In *14th IEEE international conference on automated software engineering* (pp. 255–258).

Poels, G., Nelson, J., Genero, M., & Piattini, M. (2003). Quality in conceptual modeling-new research directions. *Advanced Conceptual Modeling Techniques*, 243–250.

Popescu, D., Rugaber, S., Medvidovic, N., & Berry, D. M. (2008). Reducing ambiguities in requirements specifications via automatically created object-oriented models. *Lecture Notes In Computer Science*, 103–124.

Purao, S. (1998). APSARA: A tool to automate system design via intelligent pattern retrieval and synthesis. *ACM SIGMIS Database, 29*(4), 45–57.

Purao, S., Storey, V. C., & Han, T. (2003). Improving analysis pattern reuse in conceptual design: Augmenting automated processes with supervised learning. *Information Systems Research, 14*(3), 269–290.

Reingruber, M. C., & Gregory, W. W. (1994). *The data modeling handbook: A best-practice approach to building quality data models*. New York, NY, USA: John Wiley & Sons, Inc.

Robbins, J. E., & Redmiles, D. F. (1998). Software architecture critics in the argo design environment. *Knowledge-Based Systems, 11*(1), 47–60.

Robbins, J. E., & Redmiles, D. F. (2000). Cognitive support, UML adherence, and XMI interchange in Argo/UML* 1. *Information and Software Technology, 42*(2), 79–89.

Schenk, K. D., Vitalari, N. P., & Davis, K. S. (1998). Differences between novice and expert systems analysts: What do we know and what do we do? *Journal of Management Information Systems, 15*(1), 50.

Shanks, G., Tansley, E., & Weber, R. (2003). Using ontology to validate conceptual models. *Communications of the ACM, 46*(10), 89.

Shoval, P., Last, M., & Yampolsky, A. (2009). Data modeling and functional modeling: Examining the preferred order of. *Innovations in Information Systems Modeling: Methods and Best Practices*, 122.

Siau, K., & Cao, Q. (2001). Unified modeling language: A complexity analysis. *Journal of Database Management, 12*(1), 26–34.

Siau, K., Erickson, J., & Lee, L. Y. (2005). Theoretical vs. practical complexity: The case of UML. *Journal of Database Management, 16*(3), 40–57.

Simmonds, J., & Bastarrica, M. C. (2005). A tool for automatic UML model consistency checking. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering* (p. 432).

SLLPOST. (2006). Stanford log-linear part-of-speech tagger. stanford log-linear part-of-speech tagger. The Stanford Natural Language Processing Group.

Sugumaran, V., & Storey, V. C. (2002). Ontologies for conceptual modeling: their creation, use, and management. *Data & Knowledge Engineering, 42*(3), 251–271.

Sugumaran, V., & Storey, V. C. (2006). The role of domain ontologies in database design: An ontology management and conceptual modeling environment. *ACM Transactions on Database Systems, 31*(3), 1064–1094. doi:10.1145/1166074.1166083.

Su, X., & Ilebrekke, L. (2005). Using a semiotic framework for a comparative study of ontology languages and tools. *Information Modeling Methods and Methodologies*, 278–299.

van Lamsweerde, A. (2008). Requirements engineering: From craft to discipline. In *Proceedings of the 16th ACM SIGSOFT international symposium on foundations of software engineering* (pp. 238–249). New York, NY, USA: ACM.

Viljamaa, A. (2001). Pattern-based framework annotation and adaptation – A systematic approach. Helsinki, Finland: Department of Computer Science, University of Helsinki.

Wand, Y., & Weber, R. (2002). Research commentary: Information systems and conceptual modeling – A research agenda. *Information Systems Research, 13*(4), 363–376.

Wohed, P. (2000). Tool support for reuse of analysis patterns: A case study. In *Proceedings of the 19th international conference on conceptual modeling* (pp. 196–209). Salt Lake City, Utah, USA: Springer-Verlag<http://www.portal.acm.org/citation.cfm?id=1765132>.