Reconciling and Cleansing: An Approach to Inducing Domain Models

Sandeep Purao Veda C. Storey Arijit Sengupta Melody M. Moore

Department of CIS, J. Mack Robinson College of Business
Georgia State University, Atlanta, Georgia 30302-4015
Phone: 404 651 3859, Fax: 404 651 3842
Email: {spurao,vstorey, asengupt, melody}@gsu.edu

Abstract

Domain models, which provide templates for an application, can greatly improve the speed of application development and the quality of the design produced. Unfortunately, the construction of domain models is a time-consuming process requiring considerable experience and expertise. At the same time, there is a wealth of domain information embedded in existing legacy databases and code. Current reverse engineering techniques can extract some of this information. The domain model templates generated in this manner can, however, include considerable organization-specific noise. We present a methodology that reconciles and cleans the reverse-engineered application-specific models to create useful domain models. We illustrate the feasibility of the methodology by applying it in two different application domains and describe the research prototype being developed to implement the methodology.

Keywords: domain model, design heuristics, reverse engineering

1. Introduction

Conceptual information system design is difficult because it involves understanding portions of the real world that are important for an application domain and representing them in a model. This act, *domain modeling*, is independent of the different applications that may be built within a domain. Although many applications may have been created previously in a given domain, new application designs are often developed from scratch. A library of domain models, therefore, would serve as an extremely useful resource for designers by speeding up the time needed to design and create a new application [Karlsson et al 1995].

Constructing domain models is a complex process, requiring considerable developer expertise, time and effort [Reifer 1997]. Few domain models are available other than those proposed by industry initiatives such as SAP. Yet, much of the knowledge underlying domain models is locked in existing systems that many corporations possess. Extracting this intelligently can greatly enhance the process of domain model creation.

Research on database reverse engineering proposes techniques for extracting the conceptual schema from a relational database [Chiang et al., 1997]. However, the schemas so extracted tend to be at a low level of generality and can contain organization-specific noise. For example, a model might contain entities that capture two different states of the same concept such as a library book that is either on loan or being re-shelved. A significant challenge in generating domain models is to identify concepts that are generic and are not affected by organization-specific policies.

The objective of this research, therefore, is to: *develop a methodology to generate high-quality, reusable domain models from reverse engineered application models by retaining an appropriate level of detail and eliminating organization-specific noise.* The generated domain models can be useful in several ways such as for comparing an existing conceptual model to a domain model to assess the completeness of the conceptual model and for providing a starting point for automatically generating conceptual models.

2. Constructing Domain Models

Domain analysis techniques have been developed for domain-oriented design and reuse [Rugaber et al, 1997]. However, no approaches for automating the construction of domain models have been proposed yet. A domain model is a template of an application domain, expressed in the form of a conceptual model. Rich domain models are those that capture the essence of an application domain so they can serve as reasonable starting points for the creation of new systems, thus, greatly improving the speed and quality of the systems produced. An example of a domain model for a hotel application domain is shown in Figure 1.

It consists of the entities Guest, Preferred Guest, Manager, Room, Maid, Reservation, and External Partner (e.g., an airline or car rental agency) and the relationships among them. It also includes some notions of arrival and departure dates and times. From this, applications can be built for different requirements; for example, scheduling of maids, tracking frequent guests, and making reservations.

Another simple, yet powerful example of the use of domain models can be seen in the area of tools developed for electronic commerce applications that include capabilities such as 'secure transactions' or 'shopping basket' capabilities. By reusing the models built into these tools, a novice developer can quickly design and deploy an e-commerce application.

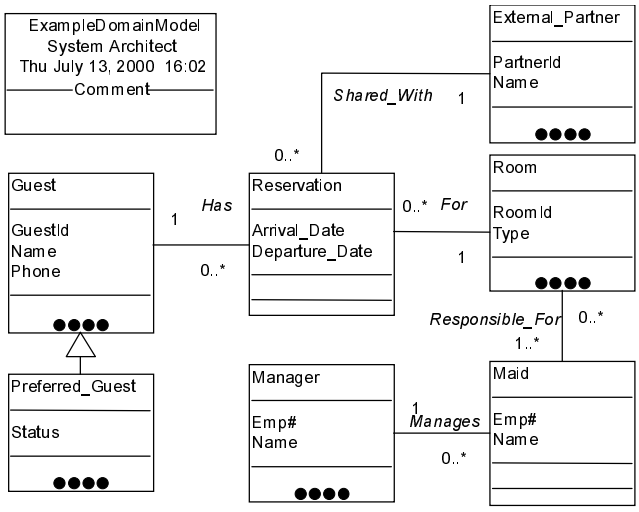

Figure 1: An Example Domain Model

The construction of domain models is a complex process because it involves discovering the few constructs that are general enough to be applicable to different applications while making the model semantically rich. A wealth of domain information, however, is embedded in existing legacy application designs, code and data.

One possible approach to creating domain models is to take the *union* of all the reverse engineered models. However, the resulting model would contain concepts that are particular to a given user view and not generic. Furthermore, many of the applications being reverse engineered may have had design problems that should be eliminated from the domain model. The *intersection* of all the reverse engineered models may be another approach. However, an intersection would be too restrictive because few constructs apply to all applications of a domain. In fact, user views could capture concepts that are very specific to the organization for which the view was created. A reasonable approach, then, would be to start with the intersection and expand it to a rich domain model, while eliminating process information.

3. A Methodology for Generating Domain Models

The methodology we propose exploits available information from existing applications such as data instances, database schema, data dictionary and application code. Existing reverse engineering techniques are used to generate a conceptual model of the application. We focus on generating a domain model by reconciling these reverse engineered, application-specific conceptual models, aided by the use of data instances and data dictionary. The use of application code is being incorporated as the approach evolves. Figure 2 shows the methodology.

The methodology consists of three major steps: 1) creating a seed model, 2) growing the seed model and 3) pruning the model. The seed model is created by taking the intersection of the application-specific models. Then, reconciliation heuristics are applied to augment the seed model. Finally, the model is pruned by applying process noise elimination heuristics. Each step is outlined with the help of an example based on three real-world databases that were developed for reviewing and assigning conference papers.

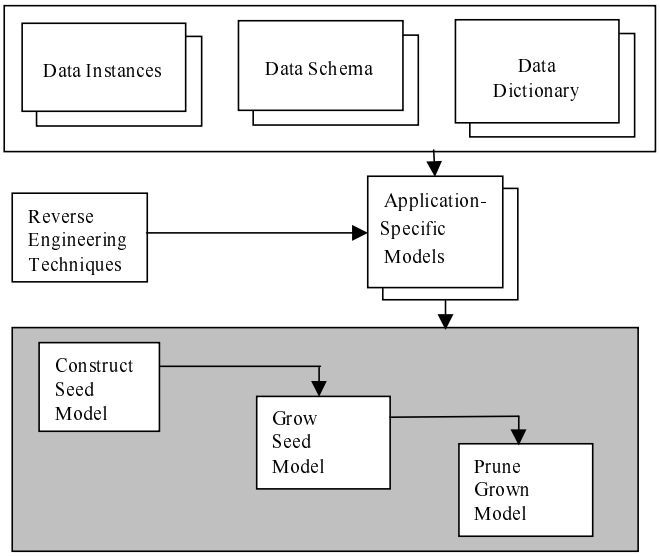

Figure 2: The Methodology

Step 1: Construct a Seed Model We begin with an intersection of the reverse engineered conceptual models. Starting with such a seed model ensures that no (or minimal) process noise is retained. The seed model is not expected to be semantically rich since it includes concepts that appear identically in every model. In other words, semantically equivalent concepts may have been represented using different constructs. As a result, a small seed model is created. Figure 3 shows the seed model.

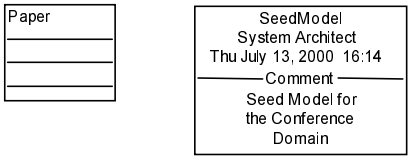

Figure 3: Seed Model

Step 2: Grow the Seed Model A number of heuristics are applied to discover concepts common in multiple application-specific models. These concepts may represent entities, relationships, or attributes in the application–specific models. With each, the seed model is augmented. We use several NLP-based and data instance values-based heuristics for this step. The grown model includes concepts that can be meaningful in different applications in this domain. Figure 4 shows a grown model for the conference papers review domain.

Step 3: Prune the Grown Model The grown model is populated with the union of attributes and behaviors of the entities from the application-specific models. Since these can contain process noise and superfluous details, another set of heuristics is applied to prune the model. The resulting model, shown in Figure 5, contains important concepts in the application domain but not unnecessary process noise or superfluous detail. To avoid clutter, only a few attributes are shown. The figure indicates heuristics used in italics. These are briefly explained in the next section as part of the solution architecture implementing the methodology.

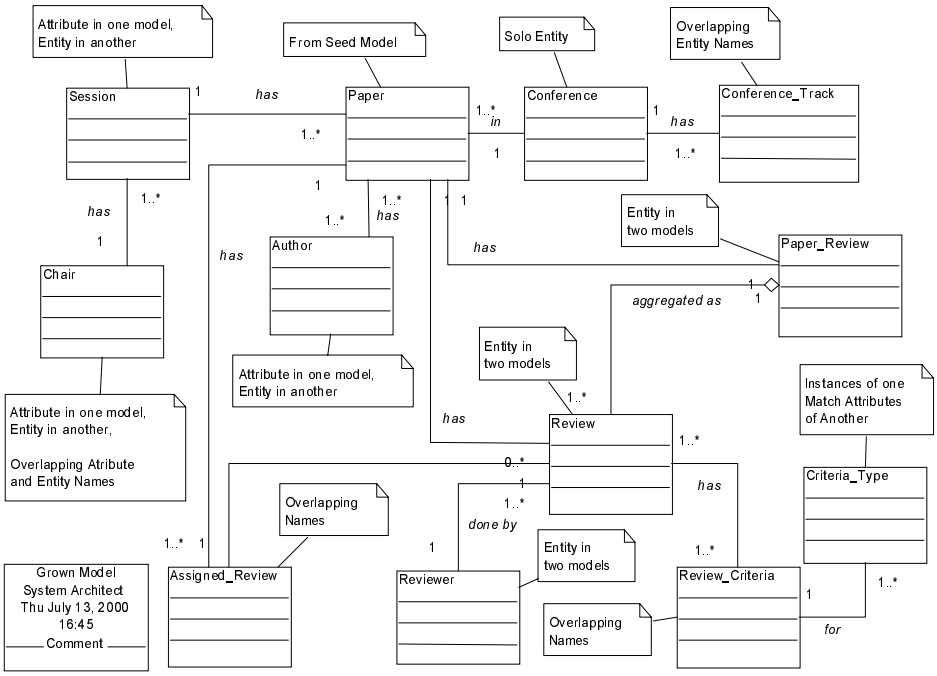

Figure 4: Grown Model

4. Heuristics

Several heuristics underlie the methodology described above. These have been incorporated into a flexible solution architecture that is under implementation. The heuristics exploit natural language constructs and data instance values to carry out the three steps in the methodology outlined above. The heuristics identify real-world concepts that are represented by different constructs and selects one to be used in the domain model. For example, after ascertaining that 'Author' and 'Writer' refer to the same real-world concept, 'Author' may appear in the domain model as an entity construct. They are aided by meta-heuristics, which dictate the sequence and conditions for firing each heuristic. Figure 6 shows an overview of the heuristics. A total of sixteen heuristics have been identified, with some heuristics (e.g. 3 and 3A) capturing different ways of achieving a similar goal. Due to constraints on space, we show and explain below sample heuristics from the set above to provide the readers a flavor of the different kinds of heuristics employed.

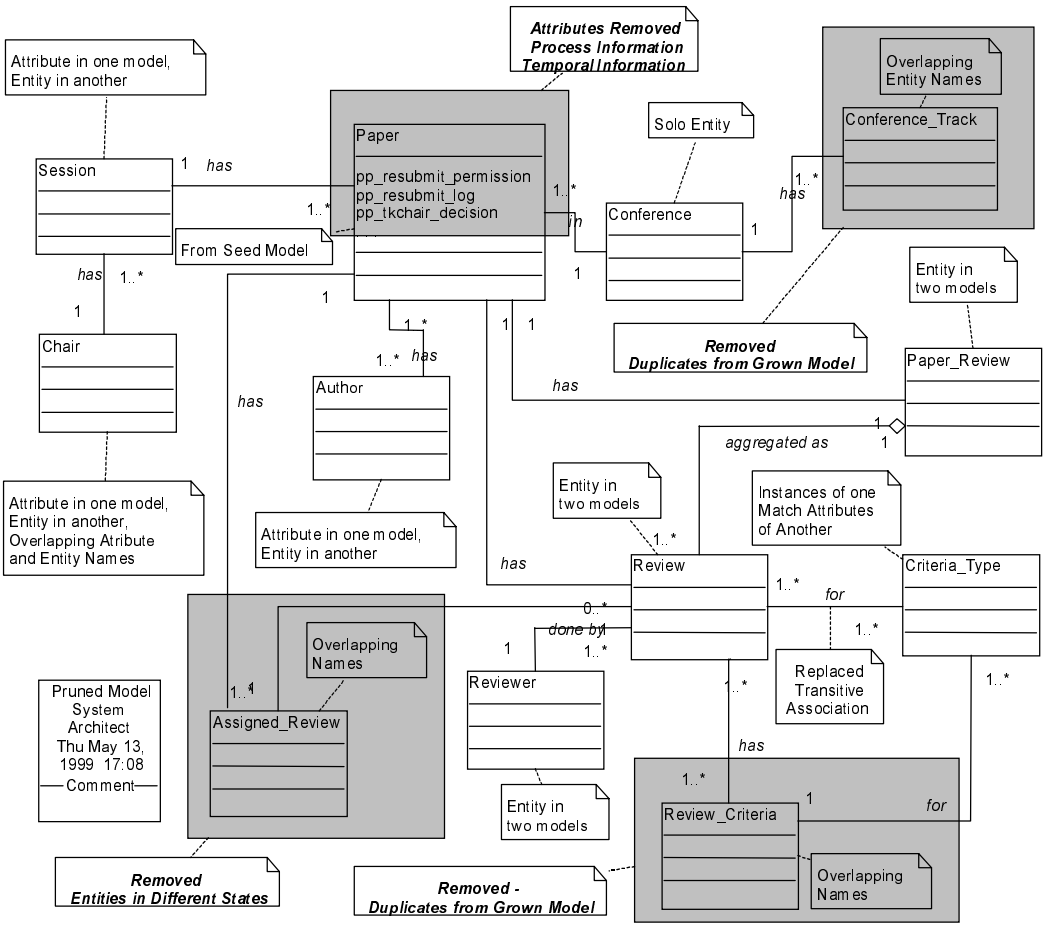

Figure 5: Pruned Model for Conference Review Domain

Sample Heuristics for Step 2: Growing the Model

Heuristic 7. Derived Aggregates Perform a string search to identify cases where one entity name is a subset of another (e.g., Review and PaperReview). Then search for attributes that represent an aggregate (e.g., min/max/avg/total/sum/count/overall). This is useful because in some databases both the individual and aggregate concepts are stored.

Application 1: Review : […] Application 2: PaperReview: […averageoverall, minimumofoverall, maximumofoverall …]
Result: Place both in the Domain Model, and create a relationship between the two.

Heuristic 9. Attributes in one Entity same as Instances in Another Search for cases where the attributes of one entity have the same names as values of an attribute in another entity, indicating the same real world concept is being modeled in different ways.

Application 1: Review: [… originality, tech-quality, significance, clarity, relevance, …] Application 2: Criteria_Type: […, cty_desc, …]
Result: Since the values for the attribute "cty_desc" include 'originality,' 'tech-quality,' 'significance,' etc., then *Criteria_Type* should be an entity in the domain model.

Sample Heuristics for Step 3: Pruning the Model

1. Temporal Information Temporal information pertains to the creation or duration of information (i.e., start and end time). It excludes permanent data such as birthdates. Search the attributes for any that have a data type of date, time or timestamp.

Application 1: Paper: [Paperid, title, authors, session, status, evaluation] Application 2: Paper: […, pp_submit-date, pp_resubmit-date, pp_withdrawal-date, … pp_track-chair-date, pp_ae-date]

Result: eliminate the following attributes from consideration for the domain model: pp_submit-date, pp_resubmit-date, pp_withdrawal-date, track-chair-date, pp_ae-date, etc.

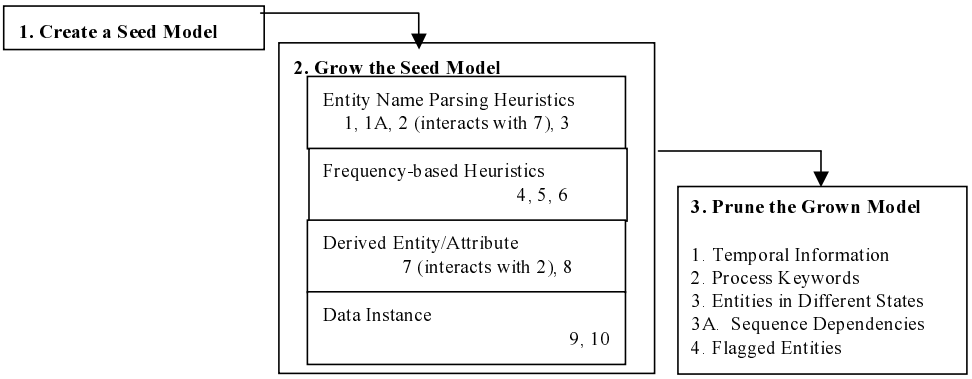

Figure 6: Heuristics

2. Process Keywords This extends the above because it includes parsing for process keywords such as "log" and "decision."

Application 1: Paper: [Paperid, title, authors, session, status, evaluation] Application 2: Paper: […, pp_resubmit-log, pp_ae-decision, pp_trackchair-decision, pp_cmt-decision]
Result: eliminate the attributes pp_resubmit-log, pp_ae-decision, pp_trackchair-decision, pp_cmt-decision from consideration in the domain model.

3A. Sequence Dependencies Sequence dependencies are discovered by investigating the timestamps on data instances in different entities.

Application 1: AssignedReview: […. Paper-id, …] Review: […. Paper-id, …]

Assigned Review

38	Fred
42	Sam
67	Mary
87	Beth

Review

38	Fred
67	Mary
87	Beth

The corresponding timestamps in AssignedReview occur prior to those in Review

Result: Retain one of the Entities in the Domain Model. Represent the other as a different state using a state transition diagram.

4. Flagged Entities Search for flagged entities. These are entities that contain a state such as a boolean flag and indicate a probable process component.

Application 1: Paper: […, pp_resubmit_permission, …] Data type = Char Flag Application 1: Award_Recommend: […, awcm_recommended,] Data type = Int Flag Application 1: Paper_Review: […, pprvw_yn_score, ….] Data type = Char Flag Application 1: Award: […, award_order, ….] Data type = Int Flag
Result: Remove these attributes from the domain model.

5. Architecture and Implementation

We implemented the above heuristics and methodology in a research prototype using Java™. The architecture of the research prototype is shown in Figure 7. The inputs to the methodology consist of graphs

of application-specific models, which contain two types of nodes, corresponding to entities and attributes. Edges include edges between two entities and those between entities and attributes. In addition, we used a secondary B-tree index covering the entire schema vocabulary, for the purpose of efficient traversal and search through the graphs. Table 1 describes the different stages of the algorithm used (with n being the total number of nodes and e being the number of edges). The overall complexity of the algorithm is within a poly-log bound.

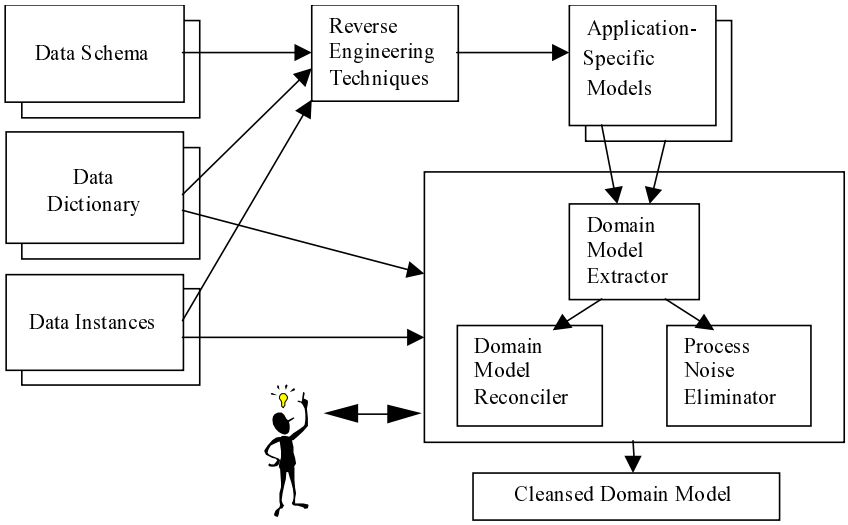

Figure 7: The Solution Architecture

Three conference review systems databases have served as the testbed during the creation of a domain model. To further assess the effectiveness of the methodology, it was applied to a set of conceptual models for on-line electronic-commerce sales systems. The application-specific models were created by student teams from existing systems. Both the seed model and the grown and pruned model showed highly encouraging results (eliminated here due to space considerations). Many of the heuristics were applied, and a reasonable model was generated following the methodology, even though all of the heuristics were not needed. The results from these experiments matched closely with the theoretical presentation, and demonstrated the areas where the heuristics could be improved by the incorporation of more intelligent methods such as the use of natural language processing techniques and dictionaries and thesaurus.

7. Conclusions

We have presented a methodology for generating domain models from reverse engineered application-specific models. We have implemented a major portion of this methodology to experimentally show its effectiveness. Two sets of heuristics form the basis of the methodology. The methods were initially tested against three real-world databases in the domain of conference paper reviewing. It was then applied to another set of reverse-engineered models of on-line electronic commerce systems. Both experiments have given us highly encouraging results that demonstrate the effectiveness of the method. Future research is needed complete the implementation and carry out more testing. Planned enhancements to the methodology and solution architecture include addition of natural language processing capabilities and integration of an on-line thesaurus or dictionary for greater control on the detection on similar names.

References

1. Chiang, R.H.L., Barron, T.M., and Storey, V.C., "A Framework for the Design and Evaluation of Reverse Engineering Methods for Relational Databases," *Data and Knowledge Engineering*, Vol.21, No.2, 1997, pp.57-77.
2. Karlsson, E. et al 1995. Editor. *Software Reuse. A Holistic Approach.* John Wiley & Sons, Inc.
3. Reifer, D. *Practical Software Reuse.* John F. Wiley and Sons, August, 1997.
4. Rugaber S. et al. "A Case Study of Domain-Based Program Understanding," *5th International Workshop on Program Comprehension,* May 1997.