Automated Conceptual Abstraction of Large Semantic Diagrams

Christina Christodoulakis
University of Toronto
Toronto, Canada
christina@cs.toronto.edu

Daniel Levy
University of Toronto
Toronto, Canada
dlevy@cs.toronto.edu

ABSTRACT

The design and development of applications and systems is a multistep process involving developers and stakeholders. During the development lifecycle, numerous diagrams are often created in order to aid the developers and communicative process with stakeholders. Some diagrams become overly complex and are not comprehensible by stakeholders. To remedy this, developers can cluster elements from the diagram together and remove elements that only serve in the development of the system but don't lend to its high-level semantics. We propose a novel method for identifying clusters in relational diagrams using a highly tailored version of the min-cut algorithm in addition to WordNet which will define the density of a cluster. We discuss the benefits of using such a method and our vision for an IDE that can make use of abstraction to lessen the comprehensibility burden on software developers and stakeholders.

1. INTRODUCTION

Detailed diagrams such as ER and UML class diagrams are an essential part of the development process. They are used in several stages, during requirements engineering, software design and software implementation. They are an essential aid in communication between members of a software development team, between different development teams and between development teams and company executives. For large systems, they can become very complex, and therefore their usefulness decreases drastically, as they can become daunting or impossible to read, understand and manage. By abstracting the diagrams however, complex systems can be presented in an abstract and understandable way, as cognitive load is reduced [Moody, 2004]. In this paper, we propose a novel abstraction technique based on graph clustering and use of WordNet [Fellbaum, 1998].

The remainder of this paper is organized as follows. Section 2 outlines our vision of a tool wich would assist not only in reducing cognitive load on users when viewing large diagrams, but serve as a bridge between functional and structural requirements of a system under desing. In Section 3 we outline the relevant research areas to our vision. In Section 4 we describe a novel approach of abstracting large diagrams, and in Section 5 we describe our experiments using this approach. Finally, in Section 6 we outline the conclusions we reached regarding our approach.

2. VISION

The Software Development process consists of the following steps:

- Requirements Capturing and Engineering
- Software Design
- Implementation and Integration
- Testing
- Deployment
- Maintenance

There are multiple iterations of each step. During the design of a product, developers use requirements analysis to derive functional requirements before proceeding to structural requirements. Once structural requirements have been finalized then the class Diagram of the overall system is also finallized, and development of different modules of the system is delegated to groups of developers.

In the case of large systems under development, it would be of great assistance to a developer to be aware of the context in which he is developing. He/she does not always care about the entire system in detail, however there are times when knowledge or understanding of other modules is very beneficial. For this reason we propose a tool which would display the UML class diagram of the system under development, and allow automated abstraction of that diagram.

Given that developers do not always abstract diagrams in a similar way, the tool should provide an option to manually edit the abstracted diagram. This could be achieved by selecting a clustered node, expanding it, and dragging a class to another cluster. When reaching the cluster, the entity will be anchored to it automatically and the abstracted diagram edges updated.

In interest of visualization, each cluster could be assigned a random color, so that when unclustered, it is clear what

cluster the class belongs to. Users can manually change the color assigned to a cluster by selecting the clustered node and selecting another color, or change the cluster the entity is assigned to by changing the entity color (or by drag and drop, as mentioned above).

To tie the Requirements Analysis process with the Software Design process, we propose that the tool would display the UML class diagram of the system under development, along side an interface for browsing Use Cases documented for the system. The developer can view an abstraction of the diagram as a simple graph, with nodes connected via edges and tagged by the Entity class names in that cluster. The developer working on a specific Use Case can select the Use Case from the browsing interface, which would result in highlighting the nodes in the graph that have entities in them that overlap with the Use Cases in question. By selecting a node in the graph, the user can uncluster it and view the classes that have been clustered into it in full detail, with attributes and methods, and UML notation on the connecting edges (to show initial relationships). Another potential use of such a system is that a user selects nodes in a cluster and Use Cases that interact with the selected nodes are highlighted in the browsing window.

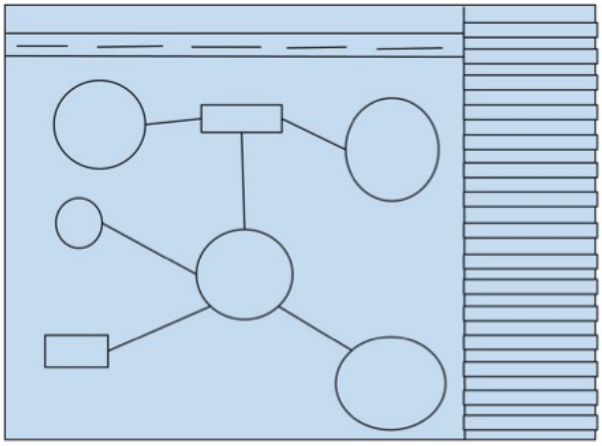

Figure 1: Interface layout of a system that would allow joint Functional and Structural Design browsing

3. RELATED WORK

[Vanderveken et al., 2005] describes a process of deriving architectural descriptions from goal oriented requirement models. Understanding what modules of functional design are relevant to structural design is the focus of [Gonzalez-Perez, 2010] work. This relationship would be imperative to support a system such as the one described in Section 2. Enabling understanding via diagram abstraction has been a topic that has concerned many over the years, [Moody and Flitman, 1999, Villegas and Olivé, 2010, Rauh and Stickel, 1992, Akoka and Comyn-Wattiau, 1996, Francalanci and Pernici, 1994, Jaeschke et al., 1994, Teorey et al., 1989, Shoval et al., 2004] as is the visualization of large diagrams and browsing of large graphs [Gutwenger et al., 2003]. Relieving cognitive load when looking at diagrams is studied

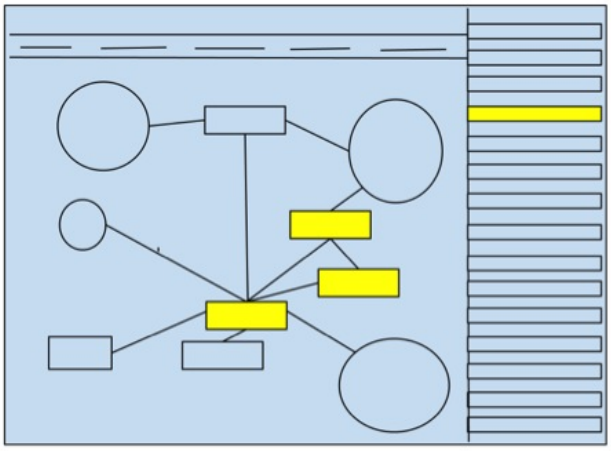

Figure 2: Fisheye view of the diagram, with a Use Case selected and the relevant entities highlighted in the diagram

by [Moody, 2009, Eades and Feng, 1997] and [Sarkar and Brown, 1992].

A number of algorithms that partition graphs using a technique called min-cut have been available as early as the 1970's. One particular example is Karger's Algorithm [Karger and Stein, 1993] . This algorithm works by randomly contracting nodes over an edge, converting the two nodes into a single node. This process is repeated until only two nodes remain. The edge between these two nodes is deemed a min-cut. This process is repeated a large number of times, computing a new min-cut each time. The min-cuts found by this algorithm have little meaning in a class diagram, and do not allow for finding min-cuts that span across multiple edges. Another algorithm which is discussed in [Boykov and Kolmogorov, 2004] using an *augmented path* technique, but is also irrelevant due to the algorithms heavy reliance on weight of edges.

4. A NOVEL ABSTRACTION TECHNIQUE

In this work we look at the clustering of a diagram. Research spanning many years has been done in regards to clustering ER diagrams, most of it focusing on the semantics of the diagram, (Abstraction, Aggregation, Composition, Cardinality of relations). In this work we do not try to continue that approach, rather we take a step back, and look at a diagram as a simple graph. We calculate candidate clusters of nodes in this graph, and then look at the node names within that candidate cluster to determine if the clustering density meets an acceptable threshold.

Very often related entities in an ER or UML diagram are named using words that belong under the same hierarchical tree. (Note here that the tree we use is WordNet, and that we expect a domain specific ontology would produce even better results than in Table 3). Often, an entity that is a part of another entity or very dependant on it uses the same name as that second entity, concatenated with another word, either through camelCase or under_scores.

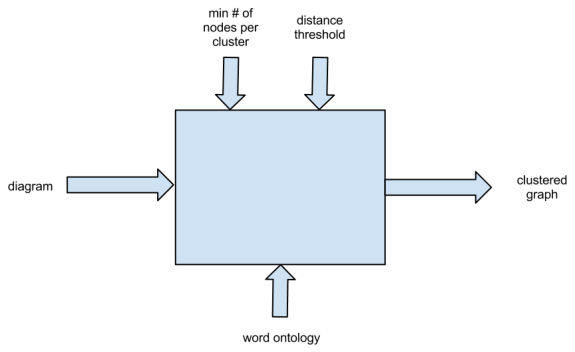

Figure 3: Architecture of our Framework

4.1 Min-Cut

We now discuss a novel method to identify min-cuts in an graph. Our first attempt at a solution to this problem was through the use of existing min-cut algorithms. The min-cut algorithm works by searching through a graph to find sets of edges that when removed from the graph, create disconnected partitions. Two partitions are disconnected if we can not reach one partition from the other through traversal of the partitions nodes. However, the existing algorithms that we reviewed in our related works research do not perform or partition well in terms of UML class diagrams and their semantics.

Instead of using a pre-existing min-cut algorithm, we developed a novel min-cut algorithm that identifies min-cuts that lead to candidate clusters in a class diagram (and other diagrams with similar structures and semantics, for example ERDs). The algorithm is described below:

Input: graph
Output: candidate min-cuts
1 **foreach** *node in the graph* ***n*** **do**
2 Find all possible combinations of the edges of **n** ;
3 **foreach** *each combination* ***c*** **do**
4 Remove the edges in **c** from **n**; **if** *all the nodes connected to edges in* ***c*** *excluding* ***n*** **then**
5 **if** *node* ***n*** *is NOT reachable from any node in* ***c*** **then**
6 **if** *are at least 2 entities in one of the partitions* **then**
7 return this node as this node is a candidate min-cut;
8 **end**
9 **end**
10 **end**
11 Restore the removed edges.;
12 **end**
13 **end**

Algorithm 1: ComputeMincuts() Computes mincuts in a given graph

Line 2: The meaning of combination is to combine the edges in all possible ways, assuming every possibility is equally likely. For example, if node n has edges 1, 2 and 3, one possible combination is the edges [2 3]. The full set of combinations are [1],[2],[3],[1 2], [1 3], [2 3] ,[1 2 3]. See Figure 4 for an example.

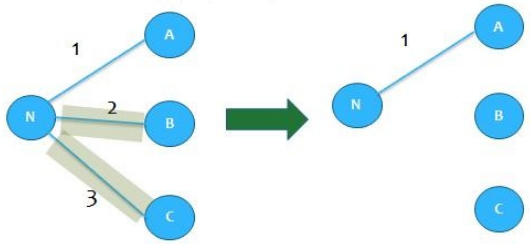

Figure 4: Removing a candidate min-cut

Line 4: This step is performed to find *only* a minimal set of min-cuts, i.e we wish to ignore min-cuts composed of all min-cuts. We essentially want the min-cut to strictly create two partitions. This step is best explained as an example. In Figure 5, we see four nodes, A, B,C and D. One candidate min-cut consists of the combination of edges [1 3]. In this scenario, we have 3 distinct partitions, C, A, D, B instead of the desired value of two.

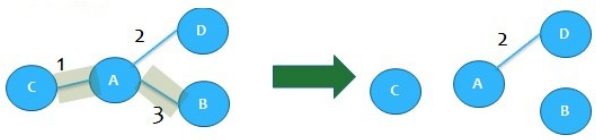

Figure 5: rejected min-cut that creates three partitions

Instead, we consider the min-cut of 1 which results in two partitions as seen in Figure 6 below. The min-cuts 2, 3 would have also worked.

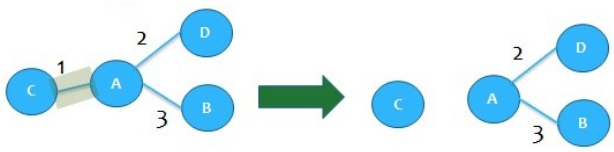

Figure 6: accepted mincut

4.2 Discovering Candidate Clusters

Now that we have a set of candidate min-cuts, we must cut the graph at specific min-cuts to form clusters. Some of our discovered min-cuts will be *applied* to the graph, and others will be discarded. The general idea here is to apply the outermost min-cuts, i.e remove these edges from the graph, and check if the cut partitions the graph into a cluster using the cluster's WordNet distance metric. If a cluster is discovered, we remove the edges connecting the cluster from the graph.

If not, we attempt to combine this mincut with its neighboring mincuts. If we still cannot form a cluster, this min-cut is ignored. The algorithm essentially begins processing from the outer points of the graph and works towards the centre.

The algorithm is described below:

Input: graph, candidate min-cuts
Output: clusters
1 Store list of min-cuts in **cuts**;
2 **while** ***cuts*** *is not empty* **do**
3 **foreach** *min-cut* ***c*** *in* ***cuts*** **do**
4 Temporarily remove the edges in **c** from the graph, separating the graph into two partitions;
5 **foreach** *Partition* ***p*** **do**
6 Traverse through the nodes and attempt to locate a **cut** in **cuts**;
7 **if** *a* ***cut*** *was reachable* **then**
8 Label **p** as NOT A CLUSTER;
9 **end**
10 **else if** ***p*** *does not meet WordNet distance threshold* **then**
11 Label **p** as NOT A CLUSTER;
12 Mark **p**for potential combination with other clusters;
13 **end**
14 **else**
15 Label **p** as CLUSTER;
16 Remove the **cut** from the list of cuts;
17 **end**
18 **end**
19 **end**
20 **if** *No clusters were found in this iteration;*
21 **then**
22 Recompute the list of **cuts**;
23 **end**
24 **end**
25 **foreach** *Cluster found* **do**
26 Repeat from step 2;
27 **end**
Algorithm 2: IDENTIFYCLUSTERS() Identify clusters using mincuts

Line 5 This step forces the algorithm to first discover nodes on the outskirts of the graph.

Line 7 This min-cut will be returned to later, once we have processed the min-cuts leading out of this partition.

Line 16 We do not restore the removed edges from the min-cut here. This allows us to cluster the inner edges during the following iterations.

Line 20 Recomputing the list of min-cuts will provide a new list as the graph has been modified with the removal of certain min-cuts (from approved clusters).

4.3 Evaluating Cluster Density

For every candidate cluster of entities, we calculate the average minimum pairwise distance given by WordNet RiTa [Howe,], keeping in mind that entity names should be nouns. We want each node name to be similar to at least another node name in the candidate cluster.

We start by computing the pairwise distance of node names in the candidate cluster. We break each node name into terms (ex: madeTransactions becomes made, Transactions, or flight_attendant becomes flight, attendant), and consider pairwise distance to be the minimum pairwise distance on terms in each name. We compute the distance via the *getDistance()* method of the RiTa WordNet API.

This function returns the min distance between any two senses for the 2 words in the WordNet tree (result normalized to 0-1) with specified pos, or 1.0 if either is not found.

The algorithm used by RiTa procedes as follows:

Input: word1, word2
Output: shortest distance between senses of two words in WordNet
1 **if** *locate node cp, the common parent of the two lemmas, if one exists, by checking each sense of each lemma* **then**
2 calculate minDistToCommonParent, the shortest path from either lemma to cp;
3 calculate distFromCommonParentToRoot, the length of the path from cp to the root of ontology;
4 **return** *(minDistToCommonParent / (distFromCommonParentToRoot + minDistToCommonParent))*;
5 **end**
6 **else**
7 **return** *1.0*;
8 **end**
Algorithm 3: GETDISTANCE() shortest distance between senses of two words in WordNet (normalized)

We then evaluate each pairwise distance computed and sum over the minimum distance each node has to any other node, averaging over all nodes, to calculate the average distance in the cluster.

At this point we have a distance measure assigned to the cluster. We compare this distance measure to a distance threshold set by the user, to accept the candidate cluster or not. In future work, we plan to automatically compute this threshhold value from aggregate data of the diagram.

This can be applied to any diagram that is in a graph structure. For graphs such as UML class diagrams or ER diagrams that contain more information, preprocessing techniques can be applied first that take into account relationships such as abstraction, composition and aggregation, and cardinality. After this preprocessing, our techniques can be applied on the graph to provide another layer of abstraction.

5. EXPERIMENTATION AND RESULTS

We ran our algorithm on two diagrams to examine its performance, capabilities and accuracy. It is important to note that these diagrams were created before the algorithm and that the diagrams were not referred to while creating the algorithm, hence the algorithm is in no way tailored to the diagrams.

The first diagram shown in Figure 7 represents an online

information system for medical students. The diagram was produced in January 2012, and during this time, another version of this diagram containing a more *simplified view* of the system was also produced (Figure 8). The latter is essentially a manually created abstraction of the original diagram. Ideally, our automated abstraction should create a diagram that is similar to the diagram depicted in Figure 8. After running our algorithm on this diagram we discovered 4 clusters, as seen in Figure 9. The clusteres are labeled numerically. Table 1 describes the resulting clusteres along with their distance measures.

An interesting point to note is the behaviour of the Image entity. The min-cut to this cluster were the edges Specimen-Slide, and Specimen-Image. Two partitions are created, however Image is not part of the partition that Slide is in (unreachable from Slide), and is instead reachable from the partition that Specimen is in. Hence, Image is not clustered in cluster 3.

Table 1: Distance measures from RiTa

Cluster	Description	Distance Measure
1	Person	0.11
2	Specimen Media	0.03
3	Slide (cells on stained glass)	0.15
4	Student exercise/test	0

The clusters form an accurate representation and coincide with our expectations from the *simplified view* diagram, with the exception that cluster 4 could have been split into an additional cluster (although this could be debated depending on the person that partitions the graph) and User Group would be better represented in cluster 1.

The second diagram that we tested our algorithm on is a representation an airport management system and can be viewed in Figure 10. Unlike the diagram above, we did not have a simplified view for this diagram. Instead, we asked two users (experienced in UML class diagrams), who had never seen the diagram to attempt to manually cluster it. The results are shown in Figure 13 and Figure 12 (ideally this task would have been performed by the software engineers who originally created this diagram). Our algorithm found 5 clusters in this diagram which can be seen in Figure **??**. The clusters are labeled numerically. Table 2 describes the resulting clusteres along with their distance measures.

Table 2: Distance measures from RiTa

Cluster	Description	Distance Measure
1	Flight Technical Servicing	0.36
2	Staff	0
3	Flight meta data	0.375
4	Scheduling	0
5	Transaction and money handling	0.4

Once again, the clusters form an accurate representation of the graph. We expect that the entity Aircraft would be better represented outside of cluster 1. Our results match up very well with the manual clustering performed by the users with minor differences of adding/removing an entity or two between clusters. The biggest difference is that the first user in Figure 13 clustered Flight Schedule and Flight Done together with cluster 3, whereas the second user in Figure 12 clustered these nodes in a separate cluster as our automated system has done.

Shown in Table 3 is an example of pairwise distance measures for entities in the proposed cluster. For the distance measures in Table 3, we assigned the distance metric of 0.4047619 to the candidate cluster. The distance measure 0.4047619 is enough to accept the candidate cluster given that our tolerance threshold is 0.5. We noticed however that some of the distances calculated were not optimal (getDistance(madeTransactions, transactions) should have been 0 ideally), and propose a solution to this in Conclusions.

6. CONCLUSION

This paper describes our first attempt at a novel graph clustering algorithm for abstracting relational diagrams. We use a highly tailored version of the popular min-cut algorithm to identify candidate clusters in diagrams and WordNet to evaluate similarity of concepts in names of nodes clustered. While much more development is required for the algorithm to be robust, accurate and optimized, we have shown that our algorithm produces positive results and demonstrates that word hierarchies have relevance to relational diagram abstraction. In the following sections we describe improvements to be done in future work.

6.1 Calculating Distance

Semantic similarity or distance measures of words in a thesaurus of words is still a hot area in research. We used the getDistance() function provided by the RiTa API but agree that the results are not always intuitive. In future work we will explore different distance computing algorithms.

6.2 Automated Distance Threshold

Instead of having the user set a distance threshold, the threshold can be automatically computed. The problem now is determining whether that distance shows high or low similarity given the context defined by all entity names used in the diagram. For example: Are chair and table similar or not? True, they will have a distance metric. How does that compare to the largest pairwise distance of words used in the diagram? If the diagram touches on a very broad ontology, then chair and table are extremely similar, and therefore make should ideally be clustered together. If all words in the diagram are furniture related, the greatest pairwise distance will still be small, making chair and table much less similar and much less probable to be clustered.

6.3 Creating Subgraphs

In our current algorithm, the system finds candidate clusters and gives them an inter cluster distance measure. If the distance measure doesnt exceed a threshold, we accept that cluster as a reasonable one. Consider now the case where within a proposed cluster, all node names are similar but one. Currently, the whole candidate cluster will be discarded. There should be an extension in the algorithm that allows for creation of subclusters.

Table 3: Distance measures from RiTa

Entity names	waiting List	transactions	madeTransaction	cashier	customer	reservation
waiting List	-	0.625	0.2857143	0.8	0.85714287	0.375
transactions	0.625	-	0.75	0.875	0.875	0.625
madeTransaction	0.2857143	0.75	-	0.8333333	0.85714287	0.375
cashier	0.8	0.875	2356	-	0.42857143	0.625
customer	0.85714287	0.875	0.85714287	0.42857143	-	0.7
reservation	0.375	0.625	0.375	0.625	0.7	-

6.4 Strictness of Abstraction

Depending on the purpose of the diagram, the user might need to see a different level of abstraction. Different purposes could be to see what packages should be created during development, to view a high level overview of a system to present to a boardroom meeting, or just for the sake of understanding the diagram, in which increasing levels of abstraction would also help. Depending on the level of abstraction the user wants, the user should be able to specify the minimum number of nodes that should be placed in a cluster.

6.5 Advanced Min-Cut Discovery/Combination

Our current technique for discovering min-cuts in a graph, while producing accurate results, represents only a first attempt at a solution. One fairly large issue that our approach has is its difficulty in identifying clusters where a single min-cut is split across multiple nodes. Currently, we approach this problem by recursively executing the min-cut algorithm, each time with various cuts removed which produces a similar result, however a more accurate and optimized approach would be one that could locate all candidate min-cuts during the first execution. Additionally, we require a more advanced technique for the combination of min-cuts to form clusters. As mentioned in this paper, we currently only try to merge cuts that are adjacent, however this approach will produce inaccurate results when the clusters relating to three or more min-cuts need to be merged.

One technique we are considering is: For a node **n** identify the edge combinations as performed in the current approach. If no disconnection is found between **n** and it's neighboring nodes, recursively search through the other edges of **n** and attempt to locate paths to the disconnected edges. If patterns of common nodes appear high up in the path hierarchy (close to node **n**), the edge related to these nodes can be appended to the current combination to form a mincut, given that they form a disconnection.

6.6 Advanced testing

We would like to perform additional testing once we have resolved a number of issues described in this section. Specifically, we would like to test our algorithm on much larger diagrams, and bring in a larger set of users to form a detailed comparison between our automated clustering and manual clustering performed by the users.

7. REFERENCES

[Akoka and Comyn-Wattiau, 1996] Akoka, J. and Comyn-Wattiau, I. (1996). Entity-relationship and object-oriented model automatic clustering. *Data & Knowledge Engineering*, 20(2):87–117.

[Boykov and Kolmogorov, 2004] Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1124–1137.

[Eades and Feng, 1997] Eades, P. and Feng, Q. (1997). Multilevel visualization of clustered graphs. In *Graph drawing*, pages 101–112. Springer.

[Fellbaum, 1998] Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. Bradford Books.

[Francalanci and Pernici, 1994] Francalanci, C. and Pernici, B. (1994). Abstraction levels for entity-relationship schemas. *EntityRelationship ApproachER'94 Business Modelling and Re-Engineering*, pages 456–473.

[Gonzalez-Perez, 2010] Gonzalez-Perez, C. (2010). Filling the voids: From requirements to deployment with open/metis @ONLINE.

[Gutwenger et al., 2003] Gutwenger, C., Jünger, M., Klein, K., Kupke, J., Leipert, S., and Mutzel, P. (2003). A new approach for visualizing uml class diagrams. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 179–188. ACM.

[Howe,] Howe, D. C. *The RiTa Wordnet API*.

[Jaeschke et al., 1994] Jaeschke, P., Oberweis, A., and Stucky, W. (1994). Extending er model clustering by relationship clustering. *Entity Relationship Approach ER'93*, pages 451–462.

[Karger and Stein, 1993] Karger, D. R. and Stein, C. (1993). An ø(n2) algorithm for minimum cuts. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 757–765, New York, NY, USA. ACM.

[Moody, 2004] Moody, D. (2004). Cognitive load effects on end user understanding of conceptual models: An experimental analysis. In *Advances in Databases and Information Systems*, pages 129–143. Springer.

[Moody, 2009] Moody, D. (2009). The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *Software Engineering, IEEE Transactions on*, 35(6):756–779.

[Moody and Flitman, 1999] Moody, D. and Flitman, A. (1999). A methodology for clustering entity relationship models - a human information processing approach. *Conceptual Modeling - ER'99*, pages 77–77.

[Rauh and Stickel, 1992] Rauh, O. and Stickel, E. (1992). Entity tree clustering - a method for simplifying ER designs. *Entity-Relationship Approach- ER'92*, pages

62–78.

[Sarkar and Brown, 1992] Sarkar, M. and Brown, M. (1992). Graphical fisheye views of graphs. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 83–91. ACM.

[Shoval et al., 2004] Shoval, P., Danoch, R., and Balabam, M. (2004). Hierarchical entity-relationship diagrams: the model, method of creation and experimental evaluation. *Requirements Engineering*, 9(4):217–228.

[Teorey et al., 1989] Teorey, T., Wei, G., Bolton, D., and Koenig, J. (1989). Er model clustering as an aid for user communication and documentation in database design. *Communications of the ACM*, 32(8):975–987.

[Vanderveken et al., 2005] Vanderveken, D., Van Lamsweerde, A., Perry, D., and Ponsard, C. (2005). Deriving architectural descriptions from goaloriented requirements models.

[Villegas and Olivé, 2010] Villegas, A. and Olivé, A. (2010). A method for filtering large conceptual schemas. *Conceptual Modeling–ER 2010*, pages 247–260.

APPENDIX

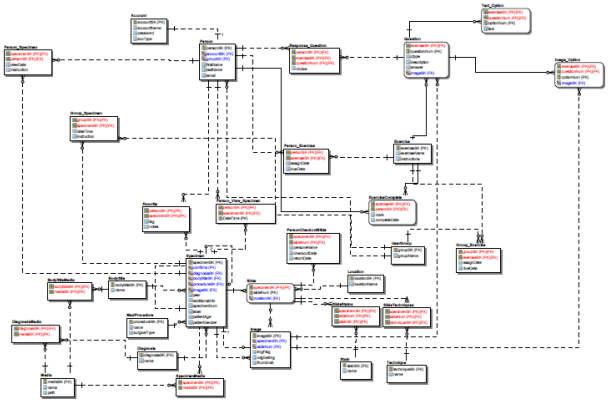

Figure 7: Medical information system

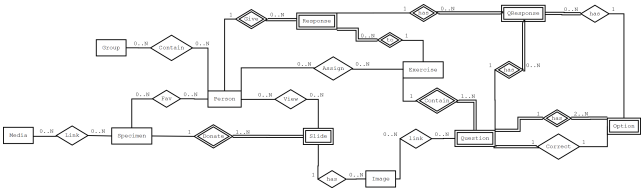

Figure 8: Simplified medical information system

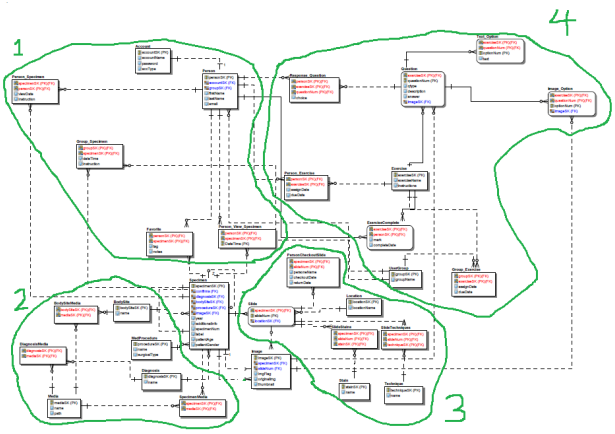

Figure 9: Automated abstraction of medical information system

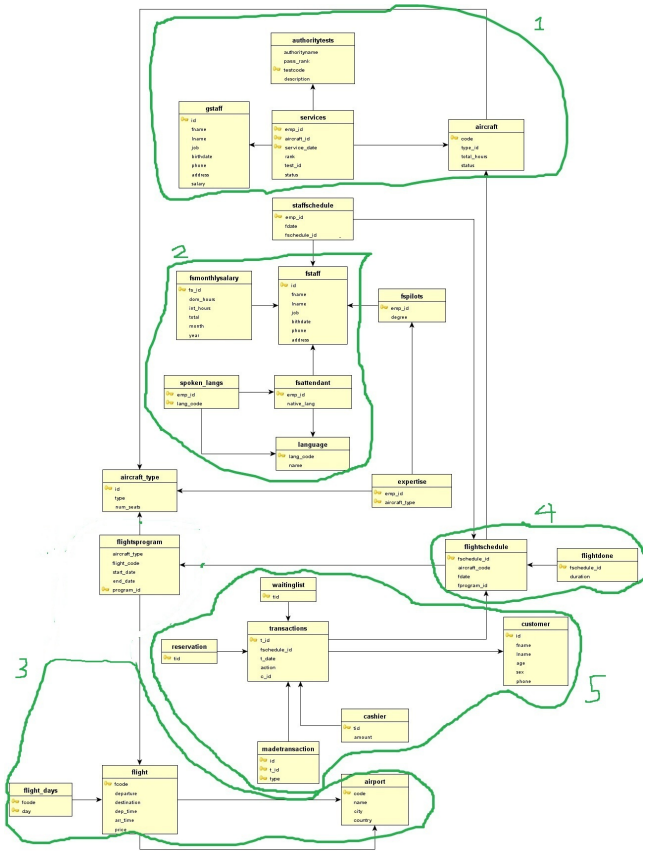

Figure 11: Automated abstraction of an airline management system

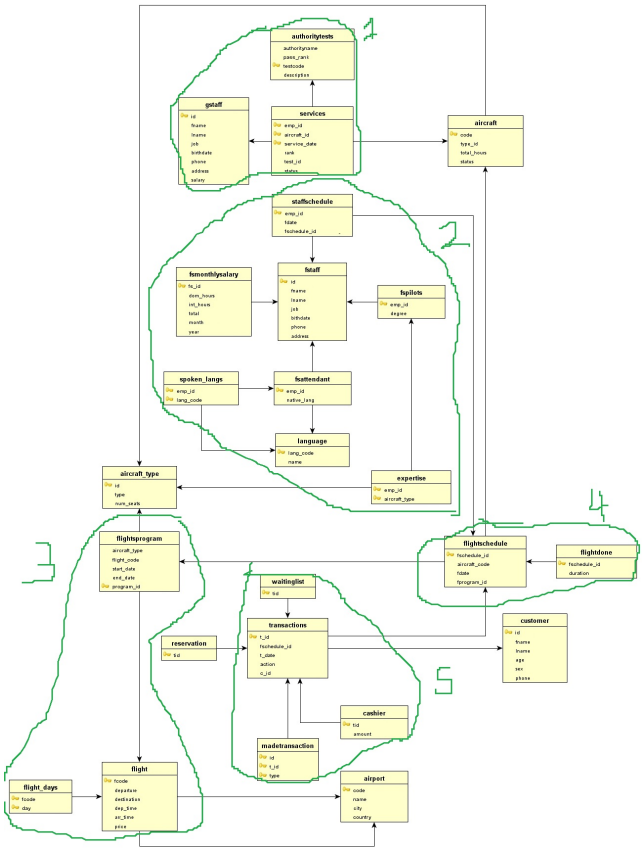

Figure 12: Abstraction solution performed by User 1

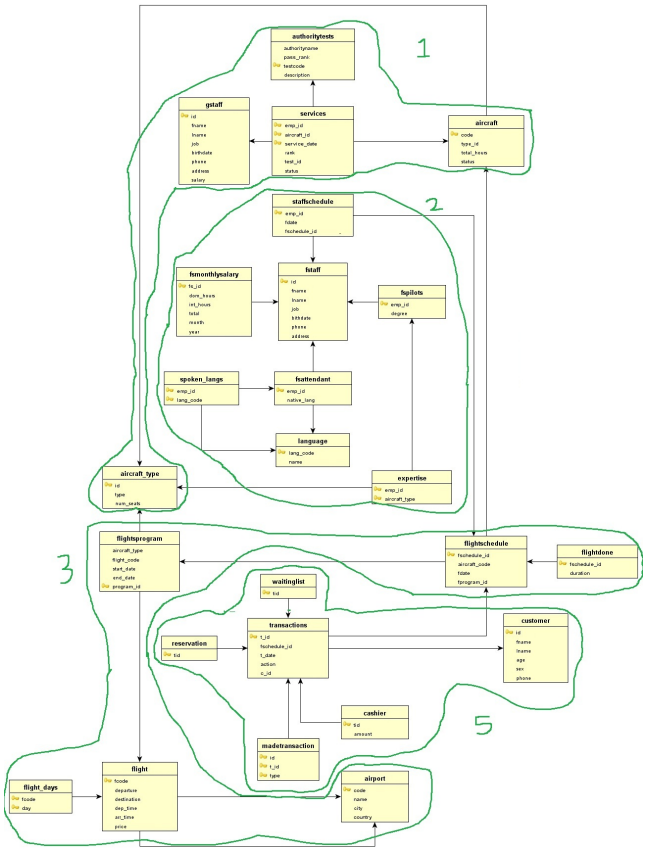

Figure 13: Abstraction solution performed by User 2

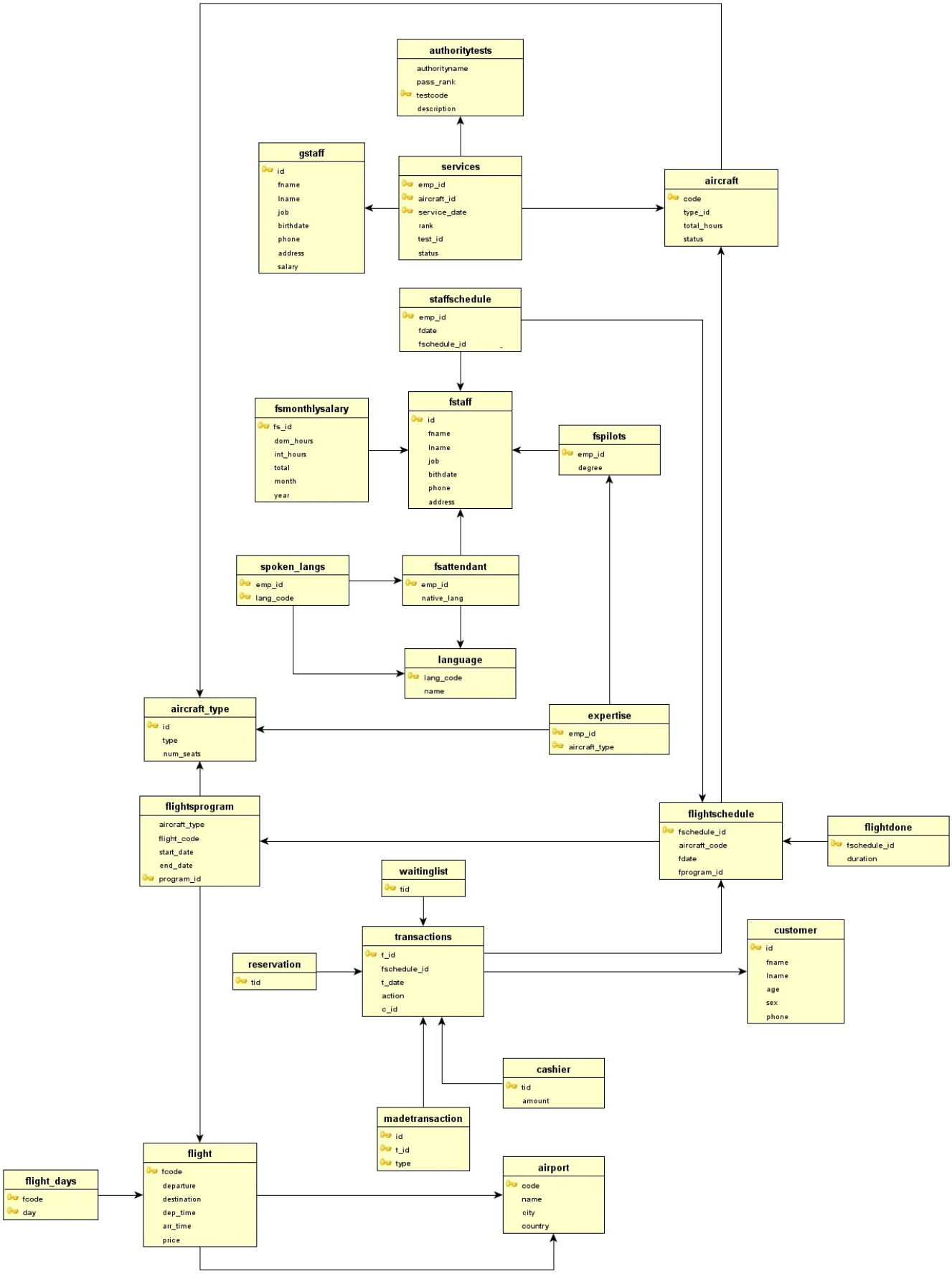

Figure 10: Diagram of an airline management system