Force-Directed Visualization for Conceptual Data Models

Andrew Battigaglia, Noah Sutter
Georgia Tech Research Institute

ABSTRACT

Conceptual data models are increasingly stored in an eXtensible Markup Language (XML) format because of its portability between different systems and the ability of databases to use this format for storing data. However, when attempting to capture business or design needs, an organized graphical format is preferred in order to facilitate communication to receive as much input as possible from users and subject-matter experts. Existing methods of achieving this conversion suffer from problems of not being specific enough to capture all of the needs of conceptual data modeling and not being able to handle a large number of relationships between entities. This paper describes an implementation for a modeling solution to clearly illustrate conceptual data models stored in XML formats in well organized and structured diagrams. A force layout with several different parameters is applied to the diagram to create both compact and easily traversable relationships between entities.

Keywords: Conceptual data model, data model, conceptual modeling, force-directed.

1. INTRODUCTION

A conceptual data model is a type of entity-relationship model used to document and illustrate the data of a particular domain. The primary building blocks of a conceptual data model are entities, relationships, and attributes [1]. An entity is a uniquely identifiable object from a certain domain, and is usually an object in the real world, such as a store, a customer, or an apple. A relationship explains how two or more entities are related. Examples of relationships include sells (between store and apple) and purchases (between customer and apple). Attributes are the relevant descriptors of an entity. An apple can include attributes about color, shape, size, or expiration date [1,2]. Figure 1 provides a typical demonstration of this example data model.

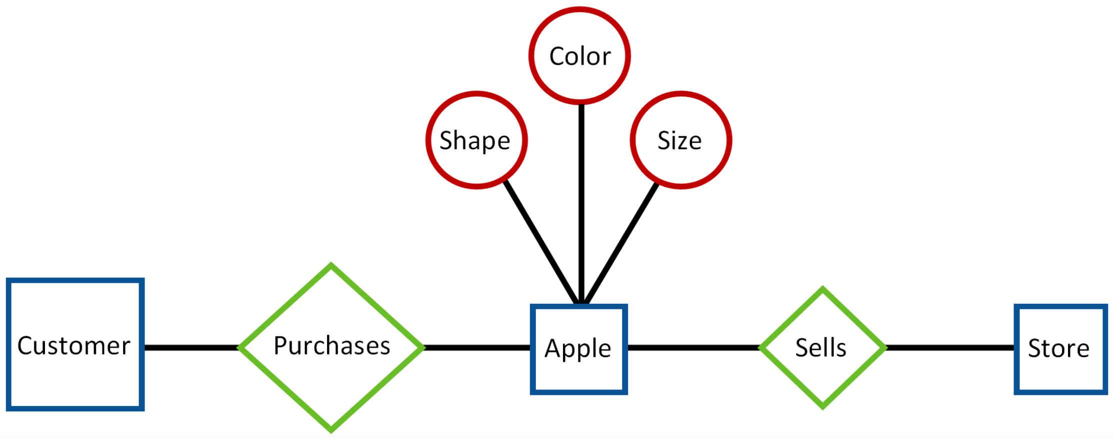


Figure 1. Conceptual data model for a grocery store.

These kinds of models are useful for communicating with stakeholders to capture important software and domain requirements from multiple perspectives. This range of perspectives often assists in the requirements determination process for software or information systems. Once completed, the conceptual data model can be used to implement a database or other software application. In order for a conceptual data model to achieve its goals of being a choice presentation and collaboration tool and remaining independent of the software tool which created it, the data must be represented in a clear way to avoid confusion amongst model users. There are many software tools that can be used to construct conceptual data models (e.g., Enterprise Architect, Rhapsody), and some do better than others at displaying these models.

2. MOTIVATION

Even when a particular tool can show data models in easily human readable presentation, there is no guarantee that the particular audience that needs to examine the data model will have access to the same tool used in the model construction. These tools usually do have the ability to import and export data into their own version of an XML format, but there is no mechanism to carry and convert diagrams during these procedures.

While storing conceptual data models in XML schema has many benefits, this format does not facilitate ease of communication to a greater, non-technical audience; likely clear understanding would only be understood by domain experts or experienced data modelers. When trying to develop software requirements, an entity-relationship diagram is a preferred format for a fundamental explanation of the important objects in a system and how they interact with each other [1].

In order to enable this entity-relationship layout, graph and modeling tools like Graphviz and igraph were explored for their node graphing capabilities, but none supported direct interaction with XML data. In order for these tools to display the information of a conceptual data model stored in XML format, there needs to be a support mechanism to convert the relevant information into proper nodes. Once this translation occurs, the main issue with these tools involve edges crossing nodes which is detrimental to readability. The alternative to this problem is having the entities dispersed so that the entire diagram cannot fit well enough on a page and physically long paths must be traced, leading to poor readability. Refer to Figure 2 for an illustration of the typical output from one of Graphviz's commonly used engine algorithms, Neato [4].

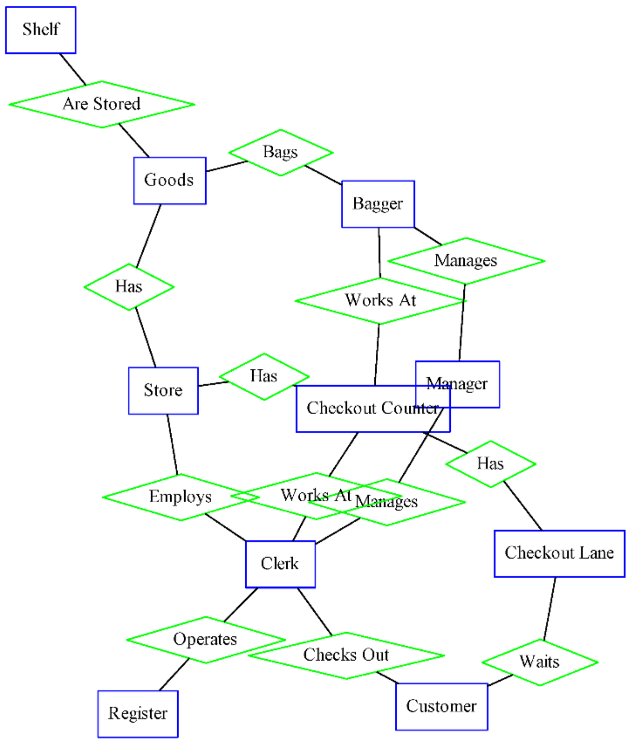


Figure 2. Existing tool visualization for sample conceptual data model [4].

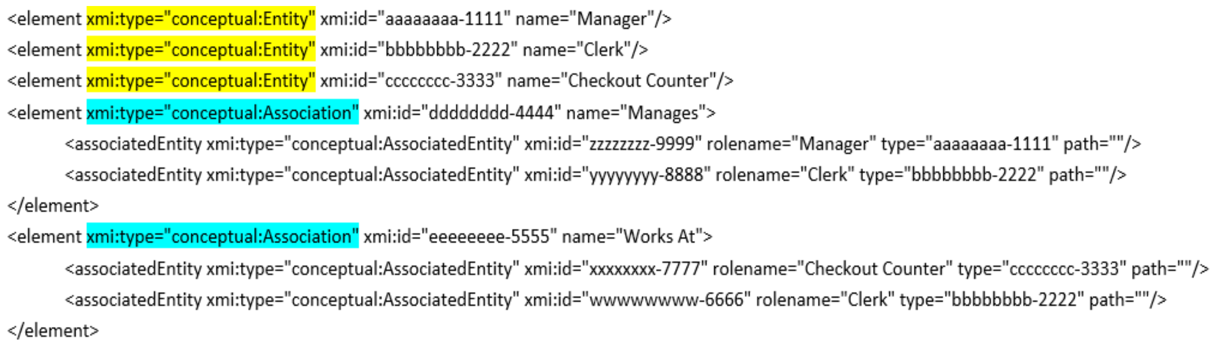
As can be seen in the figure above, these programs are not designed for capturing the needs of conceptual data models even when the XML data is retrieved for them. The graphing algorithms are designed with consideration mainly given to speed, not readability. Because the goal of an entity-relationship diagram is communication, largely ignoring readability does not allow for a satisfactory solution.

3. SOLUTION

A tool has been implemented that consumes a conceptual data model in an XML format and produces physics-based force-directed diagrams. The force-directed method treats edges like flexible springs that can attract connected elements in order to form a less cluttered picture [3]. Additionally, nodes can be given electrical charges, so they will either repel or attract each other in order to reduce disorder.

3.1 Identification of conceptual data model elements in XML format

To appropriately transform the contents of an XML conceptual data model into a more readable diagram, the entities, relationships, and attributes must first be identified as their respective concepts. As previously stated, conceptual data models stored in these files will appear different based on which tool was used for their creation. For a given format, this identification process may vary because stereotypes will be slightly different. This tool requires that the user identify one instance of an entity, relationship, and attribute. This selection is used to automatically parse the file and determine the relevant information. Figure 3 provides a sample XML file with entity and relationship stereotypes identified in yellow and blue, respectively.



```
<element xmi:type="conceptual:Entity" xmi:id="aaaaaaaa-1111" name="Manager"/>
<element xmi:type="conceptual:Entity" xmi:id="bbbbbbbb-2222" name="Clerk"/>
<element xmi:type="conceptual:Entity" xmi:id="cccccccc-3333" name="Checkout Counter"/>
<element xmi:type="conceptual:Association" xmi:id="dddddddd-4444" name="Manages">
        <associatedEntity xmi:type="conceptual:AssociatedEntity" xmi:id="zzzzzzzz-9999" rolename="Manager" type="aaaaaaaa-1111" path=""/>
        <associatedEntity xmi:type="conceptual:AssociatedEntity" xmi:id="yyyyyyyy-8888" rolename="Clerk" type="bbbbbbbb-2222" path=""/>
</element>
<element xmi:type="conceptual:Association" xmi:id="eeeeeeee-5555" name="Works At">
        <associatedEntity xmi:type="conceptual:AssociatedEntity" xmi:id="xxxxxxxx-7777" rolename="Checkout Counter" type="cccccccc-3333" path=""/>
        <associatedEntity xmi:type="conceptual:AssociatedEntity" xmi:id="wwwwwwww-6666" rolename="Clerk" type="bbbbbbbb-2222" path=""/>
</element>
```

Figure 3. Tool identification of entity and relationship stereotypes.

3.2 Translation of XML format into nodes and edges

Once the identification process is complete, entities, relationships, and attributes are transformed into nodes for the layout. These nodes can be customized for color and shape; this example specifically employs the typical conceptual data model shapes for each concept, and a color scheme is selected for easy distinction between nodes of different types. Once the nodes are properly displayed, the edges, or lines, between the nodes are drawn.

3.3 Applying a force-directed layout to the nodes and edges

After the nodes and edges are identified and displayed, forces must be applied. In order to accomplish this step, a medium and an engine are selected. For this tool's implementation, a web browser (e.g., Chrome) was chosen as the medium. The widespread distribution of web browsers means that this implementation, written in JavaScript with the D3.js JavaScript library, can be opened and run by most computers. D3 was selected for this tool because it has a force-directed layout that allows nodes and edges as an input, it can resolve the layout efficiently, and it allows for the specification of electrical, gravitational, frictional, and restorative forces on nodes.

For this implementation, negative electrical charges were assigned to all nodes to sort them into a stable layout. Electrical forces are approximated using the Barnes-Hut approximation which works by computing a quadtree where each node contains a position and charge. When a node has children, or references to other nodes, it is known as an internal node; the position and charge it contains are the total charge and center of the charge of its children. If the center of charge of an internal node is far enough away from the node to which forces are being applied, then the internal node's charge and center of charge are used to approximate the forces its children would exert. An internal node is determined to be far enough away for approximation if the equation satisfies the formula (1).

$$s/d < \theta \qquad (1)$$

Where s is the width of the area represented by an internal node, and d is the distance between the internal node and the node to which forces are being applied. This implementation uses a θ value of 0.8 instead of the typical 0.5 because distant nodes have negligible impact on the final layout of this type of diagram.

The remaining forces are set to create an environment that fosters the creation of well-organized diagrams. A gravitational force is initialized to keep the nodes on screen. Restorative forces, implemented in D3 as weak geometric constraints, or ideal node distances, are set to small values to create final node positions based on relationships.

Forces and constraints are applied together using Verlet integration. This method only tracks positions of the nodes, which means velocity is implicit in Verlet Integration (2).

$$\boldsymbol{x}' = 2\boldsymbol{x} - p\boldsymbol{x} + \boldsymbol{a} \cdot \Delta t^2 \tag{2}$$

Where *x* is the current position, *px* is the previous position, *a* is acceleration, and *t* is time. Verlet integration allows for the use of constraints and forces at the same time because velocity is not used. The forces and constraints stay in effect until a low energy configuration is found, and the layout becomes stable.

3.4 Post-processing of edges

Once the force-directed layout has settled, the final positions of each node are recorded. At this stage, there is one main problem with the diagram that needs to be resolved. This problem is that edges appear at odd angles, which creates a visually unappealing and confusing layout. The main goal of an entity relationship diagram for a conceptual data model is to communicate technical ideas in a way that non-technical stakeholders can understand so they can contribute to the data model. In order to successfully accomplish this goal, the entity-relationship diagram must be clear and readable. One of the main contributors to readability is lines or edges that are easy to follow [5].

Lines at odd angles are a detriment to one's ability to follow a line; there is no way to know for certain which way a line may turn. This variety of angled lines hampers the predictability of the diagram and increases the time required to understand it. In order to avoid the inherent lack of readability that comes with lines that flow in every direction, the lines in this implementation were made orthogonal. Predictable, right angles allow readers to anticipate the flow of the lines as they can only flow vertically, horizontally, or some combination of the two.

In order to make the lines orthogonal, this tool inserts a node in the middle of each edge, which effectively creates two edges where there was only one previously. This added node's horizontal position is set to that of one of the original edge's end nodes, while the new node's vertical position is set to that of the other end node. By this method, the new node is aligned vertically with one of the original nodes and aligned horizontally with the other. The tool determines which node to align with vertically, and which one to align with horizontally by testing which orientation will create fewer intersections between edges.

Once the step is complete, another major problem appears. Lines can end up laying directly over top of one another. This issue weakens readability to the point where some relationships can no longer be discerned because the start and end points are indistinguishable. Refer to Figure 4 for an example of this issue.

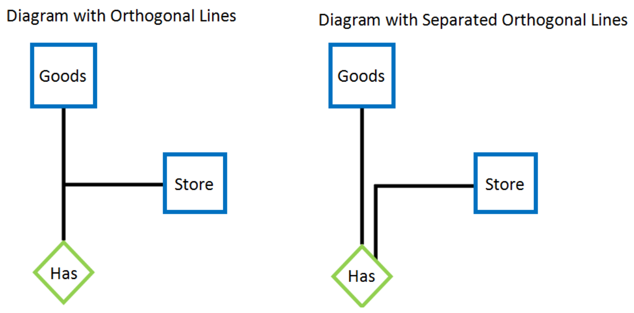


Diagram with Orthogonal Lines
Goods
Store
Has
Diagram with Separated Orthogonal Lines
Goods
Store
Has

Figure 4. Initial lack of clarity of orthogonal lines.

In order to fix the issue of overlaid edges, the tool determines how many lines leave a node in each cardinal direction and shifts edges where multiple leave from the same direction. The implementation separates the lines along the length or height of the node. Length and height are determined by the shape of the node which, by default, is a square large enough to hold the element's name. An algorithm then arranges the lines so that none cross even where the right angle occurs with the new node. This is accomplished by comparing the lengths of the edges and their secondary directions to find the ideal layout. Making lines orthogonal and separating overlaid edges increases the readability of the diagram dramatically. The post-processing of edges helps make this implementation produce conceptual data model diagrams that are comprehensible.

4. VERIFICATION

Verification of the tool was performed by inspecting the output from Graphviz in Figure 2 and comparing it to the tool that uses the proposed solution. Figure 5 depicts the output from the same conceptual data model as the one used in the problem space.

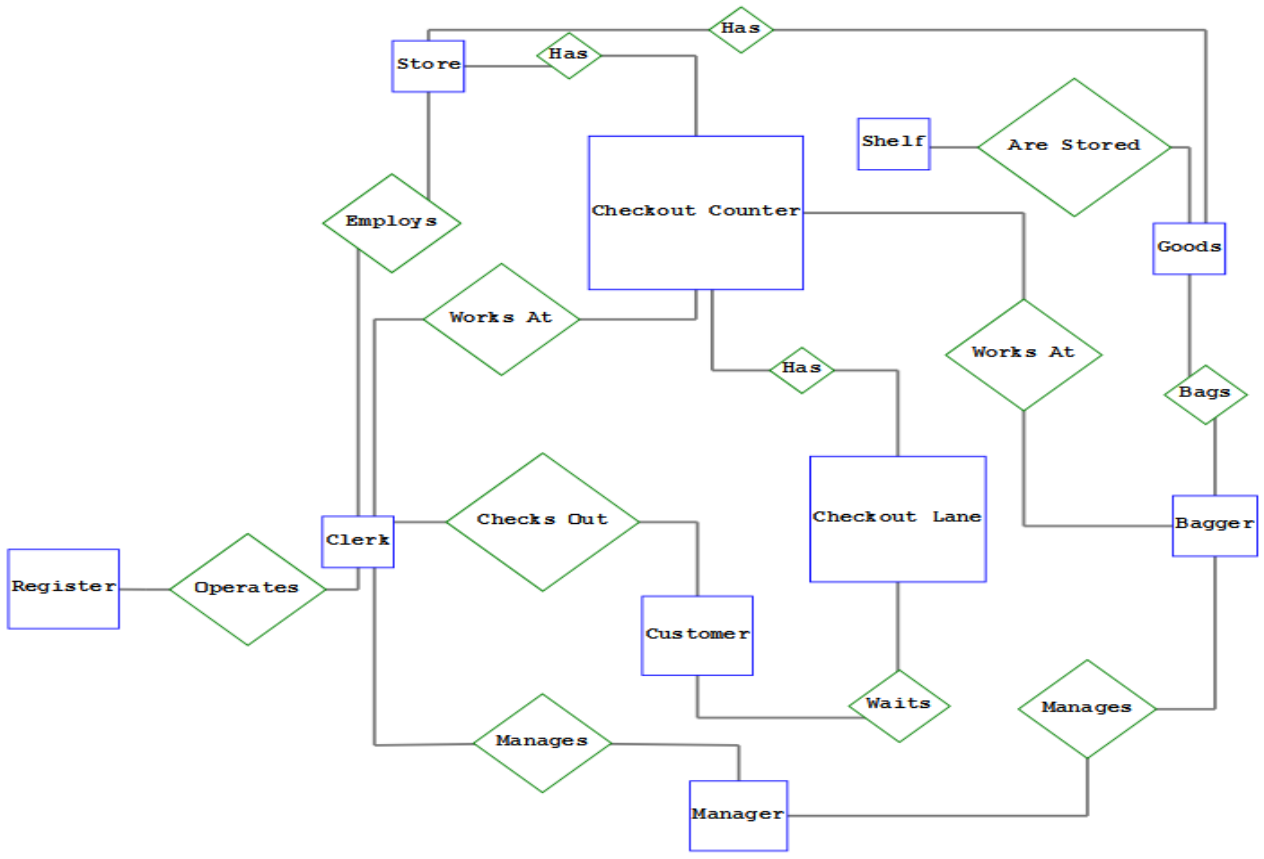


Figure 5. Implemented tool output for force-directed visualization of a sample conceptual data model.

Noticeable differences are observed immediately in the manner that none of the nodes or edges have any overlap. The lack of overlap and the spacing of the elements are defined by the electrical charges applied to the nodes and the gravity tying every node to the center of the layout. Because there is no overlap, this visualization allows the reader to clearly see full names of both relationships and entities, determine which relationships are between which entities, and discover when entities participate in multiple relationships.

5. CONCLUSION

This paper demonstrates a tool that solves the problem of how to appropriately visualize conceptual data models that are stored in an XML format without having access to the software originally used to create the data model. Previously existing methods are insuficient not only for appropriately converting the format to nodes and edges, but also for displaying data models that are easily readable. Future work will involve developing a method that can allow the output to be more interactive, allowing conceptual data model users to move nodes and edges for more custom views of the diagram.

REFERENCES

[1] P.P.-S Chen,"The Entity-Relationship Model-Toward a Unified View of Data,"ACM Transactions on Database Systems 1(1), 9-36, (1976).

[2] B.Walek and C. Klimes,"Fuzzy tool for conceptual modeling under uncertainty,"Proc SPIE 8349, (2012).

[3] D. Holten and J. van Wijk,"Force-Directed Edge Bundling for Graph Visualization,"Proc IEEE-VGTC Symposium on Visualization 28(3), 983-990, (2009).

[4] Graphviz, https://media.readthedocs.org/pdf/graphviz/latest/graphviz.pdf (2016).

[5] Henk Koning, Claire Dormann, Hans van Vliet,"Practical guidelines for the readability of IT-architecture diagrams,"Proceedings of the 20th annual international conference on Computer documentation, p.90-99, (2002).