ABSTRACT

MANAGING TOXIC AND HAZARDOUS SUBSTANCES OF CONCERN IN MANUFACTURING

by Craig Jancerak

This thesis examined data management from an object relational database perspective. The database used was a product from Cornell University called PREDATOR. The context of the evaluation is the life-cycle of a manufactured product. The manufacturing life-cycle was chosen to correspond to an environmental life-cycle being done at NJIT. However, the goal was to use a generic model that could be applied to many situations.

The first phase developed the conceptual life-cycle model. At a high level, the model was not designed with a programming language in mind. Several questions needed to be answered and the a model answers the questions regardless of implementation.

Once the conceptual model was completed, the implementation phase began. Knowledge and past experience affected implementation of the life-cycle model. A totally object oriented approach was the first step. Class diagrams were first developed. Then a JAVA application was built against the class diagram.

Once the object oriented approach proved successful, the move to an object relational implementation began. As this morphing occurred, it became clear that, for this life-cycle model, the object relational approach is not appropriate. This is shown by demonstrating that the object model becomes a purely relational model.

MANAGING TOXIC AND HAZARDOUS SUBSTANCES OF CONCERN IN MANUFACTURING

by
Craig Jancerak

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

January 2001

APPROVAL PAGE
MANAGING TOXIC AND HAZARDOUS SUBSTANCES OF CONCERN IN MANUFACTURING

Craig Jancerak

Dr. Jason T.L. Wang, Dissertation Advisor Date
Director and Associate Professor,
Data Knowledge Engineering Lab

Dr. James Calvin, Committee Member Date
Assistant Professor and Co-Director,
Simulation and Modeling Laboratory

Dr. Franz J. Kurfess, Committee Member Date
Director, Software Engineering Laboratory
Associate Director, Electronic Enterprise Engineering

BIOGRAPHICAL SKETCH

Author:	Craig Jancerak
Degree:	Masters of Science
Date:	January 2001

Undergraduate Education:

- Bachelor of Science in Business Administration,
  Seton Hall University, South Orange, NJ 07079

Major:	Computer Science

ACKNOWLEDGMENT

I would like to thank Dr. Jason Wang for his guidance and for giving me the opportunity to write this thesis. In the time I worked with Dr. Wang on this thesis, he was very supportive of the work. Dr. Wang allowed me the freedom to explore this topic as I saw fit and gave me the guidance when I needed. I could not have asked for a better experience.

My thanks also goes to Dr. Mark Healey for making it possible to become closely involved with his project which is the basis for this thesis. Dr. Healey supported and championed my involvement. Without that involvement and up-close experience, this thesis would be much less complete.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Commercial data management technology is in a state of flux right now. Relational databases have been around for more than twenty years. The theories have been proven and put in practice. The technology is mature. And the knowledge to implement systems is widely known.

On the application development front, while not new, object oriented technologies are being to emerge. Some of the more well known and the ones I'm at some level familiar with are SmallTalk, C++, and Java. Yes, there are many others. And their numbers are growing rapidly. A testament to the technology. But object orientation is more than a technology, its a way of thinking.

The data management and application development worlds are beginning to converge. This is leading to a number of object oriented and object relational database systems. This thesis investigates object relational technology through the view of manufacturing. Specifically, a problem of tracking the life of toxic and hazardous materials through the process of building a product.

I begin the thesis examining the problem and discussing what some high level requirements have been posed (Section 2). The next section (Section 3) delves into a conceptual design framework. It discussed what a products life looks like then focuses on the manufacturing phase. I then outline (Section 4) the approach I took to try to implement the ideas from Section 3. This includes an in-depth look at object oriented design. Then I discuss issues with PREDATOR and the object relational approach. And finally wind up with a conclusion (Section 5).

I have taken the perspective in this thesis that the reader is knowledgeable in relational database systems and is familiar with PREDATOR.

CHAPTER 2

PROBLEM

What Is a THSC? As far as I am aware, there is no one 'list' of Toxic & Hazardous Substances of Concern (THSC). There are, however, several lists compiled by the EPA and other groups which focus on specific grouping of substances. Several of these lists are included in the appendix. As a result, a wide definition of THSC will be to include any substance used during, or for the purpose of, manufacturing a product. This can be a compound substance or a chemical component there of.

One of the purposes of identifying and tracking these substances is the need to do Activity Based Costing in the environmental arena. For our purposes, Activity Based Costing is the process of allocating costs associated with using a THSC back to the point at which the cost is generated. These costs may arise in the purchase, use, disposal, or cleanup of a substance. The result of such analysis will rank, based on real dollars, what the affect of using (or eliminating) a substance will be. While the end result will be a dollar figure, the quantity of a THSC will also be a consideration.

Of course, it would serve no purpose to track a THSC in isolation. Without taking into account the surrounding manufacturing environment, very little value can be gleaned from 'it cost $700 to use substance A'. However, if you know the who, what and where of the substance use, then you have valuable information.

As an example, you are a manufacturer of children's bicycles. These bicycles need to be painted and you have a choice of several 'brands' of paints to use. However, one of these paints may be inexpensive to purchase but difficult (and therefore costly) to clean and dispose of, while another is moderately expensive, easy to clean but can only be kept in small quantities requiring frequent replenishment. Its not very useful to know costs associated with this paint were some dollar value. But if you knew that cleanup cost were high and that the window frames you build with the same material didn't incur the same cost, then you have something.

While this scenario alone may be trivial, if you add in the steps of cutting the tubing for the frame, welding it together, then preparing it for painting, it becomes more complex. Its rolling this entire assembly together, looking at the use of THSC, presumably in several steps, and being able to quantify cost in the entirety of the products life that is the goal.

Some of the questions that would be important to answer include:

What are the quarterly costs for product X?
What are the top 5 most costly compounds in terms of disposal?
What part of product X is most expensive?
...

These types of questions are almost endless. It is my goal to introduce an object-relation solution to facilitate the answering of these questions.

CHAPTER 3

CONCEPTUAL DESIGN

Knowing the types of questions we'll need to answer, no data modeling can take place before a design is established. This section takes you through the original design consideration that were made.

Conceptually there are several parts of the equation to being able to answer questions about THSCs. Obviously, the first is what does a THSC look like? Next, because where dealing with a manufacturing environment, some notion of a products life must be created. Of course there is the obligatory inventory issues. And there are undoubtedly more, but these three take us far enough to be able to begin answering questions.

3.1 THSC Representation

Since we talking about THSCs, lets start there. One design consideration deals with the THSC. For illustrative purposes, lets consider adhesive an example of a THSC. The decision to be made here is to what level of abstraction is a THSC to be stored and manipulated. Not all of these nasty substances are basic chemical compositions like acetone. Our adhesive example can be decomposed into its basic composition as in Table 1.

Table 1. Components of an Adhesive

Component	CAS#	Pct
Acetone	67-64-1	15%
Butyl Acetate	123-86-4	20%
Ethyl Alcohol	64-17-5	25%
Isopropyl Alcohol	67-63-0	15%

The issue involved is how to record 'adhesive': as-is or as its separate chemical composition? This is not really an end-user issue as it is a system design issue. In so much as the design can be made to generate information on both the compound and its composition, if a design is chosen that favors the compound, access at the composition level may be unacceptable. The users need only know 'adhesive' but if Ethyl Alcohol is banned in the future, linkage between the two needs to be available.

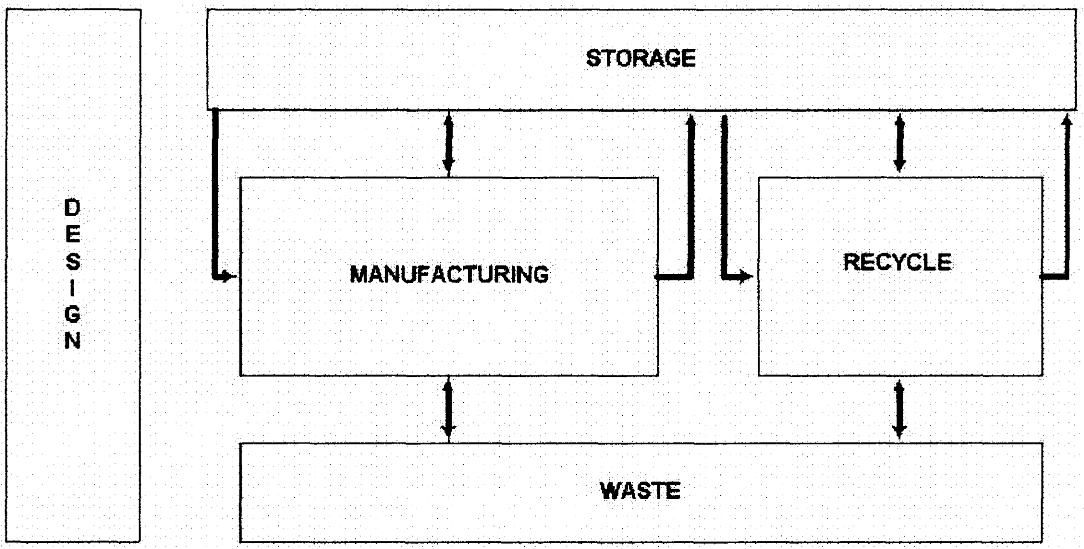
It seems prudent, at this point, to design to the compound (adhesive) level while ensuring that access to the components remains acceptable. Some of the data elements involved for a compound should probably include its name, manufacturer, and part number.

3.2 Life Cycle

Moving on from the THSC, the system needs to be able to know where in its life the part has progress to. I am referring here to the item being built, not the THSC. Or more specially, where a THSC might be used in the life of a product. This is usually represented by a Life Cycle. Leading us to the next decision to establish a life cycle that would be general yet representative of the problem domain. I will now describe the Life Cycle in some detail as it has an important place in keeping track of THSCs

From the outset, it was decided that any THSC can be present on any or all parts of a given life cycle. Figure 1 shows a block diagram of the simple manufacturing life cycle used. There are three fundamental stages that are represented by the boxes labeled Design, Manufacture, and Recycle. There is also a block for Storage as well as Waste. The lines interconnecting the blocks represent transportation.

Figure 1 Block Diagram of a Manufacturing Life Cycle



Reading time from left to right, the diagram simply states that after Design, materials are taken from storage and sent to Manufacturing. During Manufacturing, more materials may be added and Waste may be produced. After Manufacturing, materials are returned to storage. What is not depicted is the finished product itself. However, it is an important consideration because to completely account for a particular THSC, the quantity that left the manufacturing process on the product itself must be included. We now describe in detail the various stages in our life cycle.

3.2.1 Design

The first stage of the life cycle is 'Design'. For our purposes, no THSC are generated or present in this stage. It is included, however, because it's part of the life cycle and if the use of a THSC can be identified in the Design stage (the need to glue part A to part B), potential costs (activity based costing) of using alternate THSC can be analyzed.

3.2.2 Transportation

Connecting all the remaining stages of the Life Cycle together is 'Transportation'. At this time I am considering 'Transportation' as simply the physical movement of objects from one location to another. No THSC are generated or used here. However, there are potential costs that could be incurred. For instance, let's consider Trinitrotoluene or TNT. One should imagine that its much more expensive to move TNT than a simple paint. Therefore, our design should be able to account for these costs.

3.2.3 Storage

The next stop in our Life Cycle is 'Storage'. Storage can manifest itself in many ways, including the physical location of an object. Our fundamental concern, however, is the logical representation of the items used to create the final product and the final product itself. Storage can also be thought of in terms of Inventory Control.

One can also see from Fig 1 that Storage exists before and persists after the remaining sections of the Life Cycle. This is a conscious design decision based on the assumption that there will be a quantity of inventory that is needed before Manufacturing begins and exists after Manufacturing is complete. This inventory will generate costs which could include warehousing, security, electricity, etc. that should be accounted for.

At any given point, this inventory will be represented as one of three different types. The first is 'Raw Materials'. These items take the form of adhesive, metals, paint; the build blocks which come from 'outside' of the manufacturing facility. For the most part, this is where the THSC will be in its industrial form. Consider this a refinement of a THSC. But THSCs are not the only type of Raw Material.

As these building blocks, or Raw Materials, are formed together the result becomes 'Work In Progress' (WIP), also referred to as a subassembly. It is often the case that several subassemblies and additional Raw Materials are combined to create yet more WIP (subassemblies). And, finally, there is the 'Finished Product'. It should be noted that the Finished Product is a special-case subassembly (i.e. the last one).

It is currently believed important to distinguish between items in each category but that the logical representation can be similar if not identical. Each of the items would need to minimally have a unique identification, or part number, quantity data, and physical location information. Later design consideration may call for more attributes and separation of identities.

3.2.4 Manufacturing

This leads us into 'Manufacturing'. It is here where the 'Raw Materials' and 'WIP' are taken as inputs to produce subassemblies (including the Finished Product). Other inputs and outputs of the Manufacturing stage not to be overlooked include 'Waste' (discussed shortly), labor, electricity, and other overhead that contribute to the cost of building a product.

It is here at the 'Manufacturing' stage that THSC get infinitely more difficult to track. Up to this point, a THSC generally existed on its own and could be easily located and accounted for. One of the stages in building a computer memory board is adhering the integrated circuit chip to the planar board. Simplistically there are 3 objects: the chip and board (subassemblies) and a quantity of soldier (Raw Material). After completing this subassembly you'll be left with a smaller quantity of soldier as Raw Material, some of the soldier will be on the subassembly itself, and some will leave as Waste. We're still only concerned with the soldier but it now exists in at least three places, all of which need to be identified and stored.

3.2.5 Recycling

Then, of course, in these times of 'greening' our environment, we cannot neglect recycling. In this part of the Life Cycle an old 'Finished Product' is disassembled. What can be reused is, and everything else is disposed of as Waste, almost a reverse manufacturing process. This process adds costs and therefore needs to be included in this analysis.

3.2.6 Waste

The final piece of the Life Cycle yet to be discussed is 'Waste'. As with any manufacturing process, there are materials left over that have no intrinsic value and are disposed of. In the case of THSC, disposal is not necessarily a simple or cheap activity. There are usually special handling requirements or fees imposed by government regulation as to how to dispose of THSC. Any attempt to activity base cost a THSC must include Waste. Our data management system must, therefore, be able to keep track of disposal costs and be able to identify precisely which process the waste was generated.

So we now have the THSC on one hand and the Life Cycle in the other. We've discussed the product Life Cycle with some of its intricacies and how to represent the THSC. Already we have seen some merging by acknowledging that a THSC can be a Raw Material in the Life Cycle. Yet, answers to questions posed earlier cannot yet be supplied.

To close the gap further, the last step of the design will focus on the manufacturing phase of the Life Cycle.

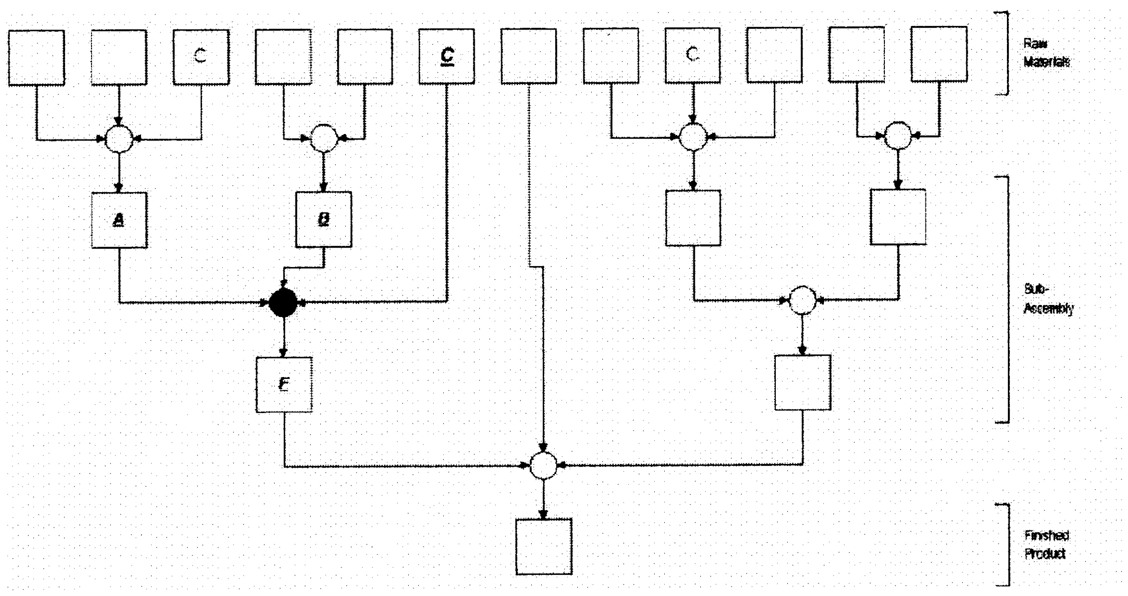
3.3 Work Order

While not unique, what the Work Order achieves is to completely detail the manufacture of a subassembly (and of course the final product by extension).

This 'Work Order' concept works as follows. For every subassembly (including the Final Product) there is a list of inputs. These inputs are raw materials and subassemblies. The output(s) would of course be the subassembly being built. Also, the Work Order entity must contain a Waste attribute. And since manufacturing may not take place in a single physical location, location should also be an attribute of the Work Order entity. The Work Order entity could/should also have the ability to record costs like labor, electricity, special handling, transportation, etc.

Using this Work Order idea, the manufacturing of a product can easily be represented as a tree. The leaves would represent Raw Material, each node a sub-assemble (WIP), and the root as being the Finished Product. This makes the intersection of branches the Work Order entity. Figure 2 shows a depiction of what a Work Order tree might look like.

Figure 2. Work Order Tree



In other words, to produce subassembly F, a Work Order W is generated. This Order specifies that to construct F, we need subassemblies A and B together with Raw Material C. Using a previous example, A and B can be the chip and board, respectively. Where C is the soldier. From this tree, it can also be discovered that to produce subassembly F, soldier (label C) is used in two places. Not shown, but an integral part of the Work Order would be quantities of soldier as input, waste, and that which is left on the subassembly along with attributed previously mentioned.

The grand scheme should now be coming into focus. At the highest level is the Life Cycle. Inside of that, in the Manufacturing phase, is the Work Order. And in the Work Order is a part which might be a THSC. There is now a framework to be able to answer some questions. I have a way to identify where a particular THSC (Raw Material) was used via the Work Order. This can also focus down to a particular product or subassembly.

One item obvious by its near absence is costs. I have yet to say where to store costs. I believe this is an implementation issue as there are many ways recording costs can be achieved. Certainly, though, enough infrastructure has been created to begin implementation.

CHAPTER 4

IMPLEMENTATION

This was the difficult part. The solution described in the previous section seemed like a good starting point and would be applicable to many manufacturing situations. You can see that it went from the very abstract view of the Life Cycle, then focused on the Manufacturing portion as the Work Order, finally to wind up at the finest definitions of THSC and cost (a dollar amount). Just figuring out a way to implement it was necessary.

My first attempts were from the relational perspective since that's where all of my practical experience comes from. It was relatively easy to develop a relational schema. But the relational design was not what I was interested in developing.

I then attempted to move from the relation schema I conceptualized to an object-relational design. Starting from the relational schema, it was not apparent to me what the 'objects' in the object relational schema would be. It was here that I began to wonder about the applicability of an object-relational approach. But more on that later.

4.1 Object Oriented

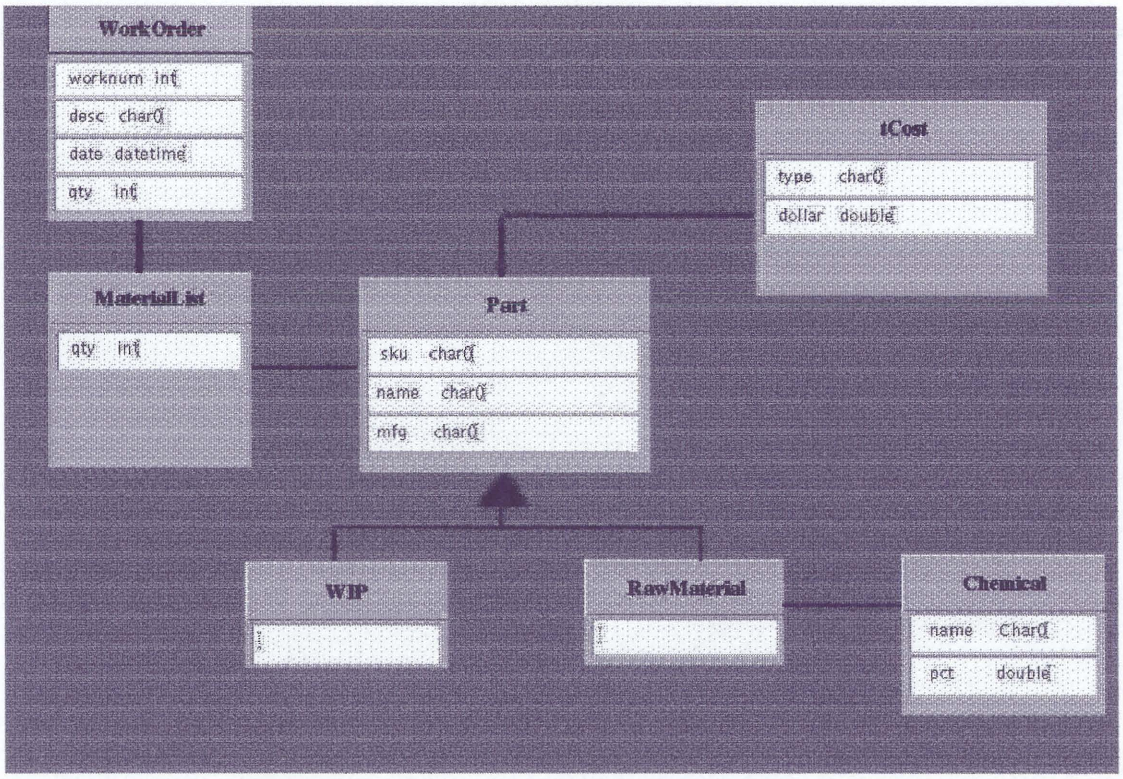
I went through several iterations of relational/object-relational schema's that didn't satisfactorily answer the questions in an efficient manner. At this point I stopped considering relational anything and began pursuing pure object orientation as a solution. However, I realized this meant learning, or more accurately trying to learn, object-oriented design and analysis. I also thought my lack of object oriented knowledge was hindering my ability in the object relational framework as well.

To remedy my short comings, I turned to several sources. These sources included, among others, *Design Patterns* by Gamma et. al.[10], articles in JavaWorld [13] electronic magazine, the Gamalan web pages, and many other object oriented and Java publications.

4.1.1 Class Diagram

Learning and truly understand object oriented design and analysis is not a one semester endeavor. While I am more comfortable, much more experience is needed. But the result of what I did learn can be seen in Fig 3.

Figure 3 Object Oriented class diagram



Borrowing from a design in UML Distilled by Martin Fowler, the figure is a simple class diagram showing the classes and their relation to each other. It is meant to mimic the

Life Cycle/Work Order/THSC concept presented earlier. While the Life Cycle is not directly evident, this diagram can be construed as the manufacturing phase. Additionally, since the Part class and its subclasses don't change during the Life Cycle, these could easily be used with additional classes representing the other parts of the Life Cycle (i.e. Inventory, Recycling).

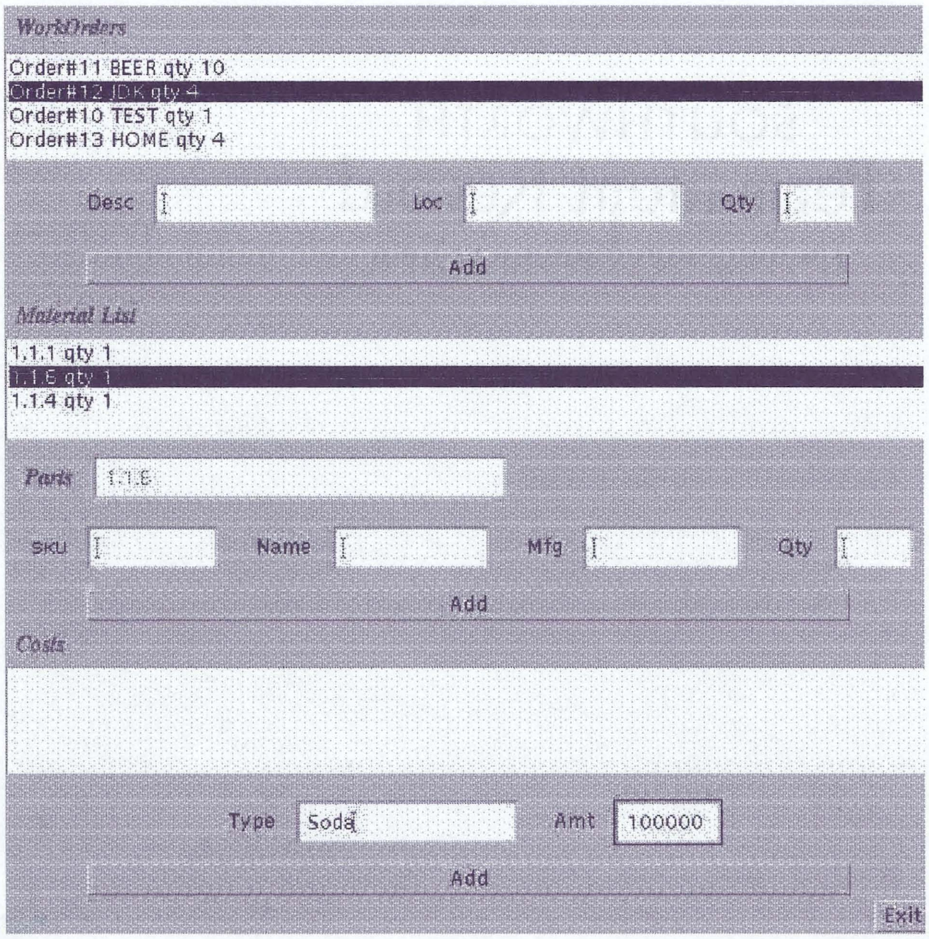
In its current form, you wind up with 7 classes. There are 2 interesting pieces of the diagram that need to be pointed out. The first is the MaterialList class. For a inexperienced object oriented designer, this class was somewhat perplexing. Since the WorkOrder contained many lineItems (MaterialList class), and the MaterialList contained only one Part, couldn't the WorkOrder contain Part(s)? As it works out, the MaterialList class is a refinement that helps better define what Part is in a WorkOrder. It can allow for an ordering of Parts in a WorkOrder or can deal with inventory issues that do not belong to a Part.

The other point of interest is the Part class itself. This should be, if I understand object oriented design and analysis, an abstract class. From this, the diagram shows that the WIP and RawMaterial classes inherit from Part. This means that WIP and RawMaterial have the same basic methods and attributes. So now, if I create my MaterialList class with an instance of Part, it can hold either a Compound or a Component. And if further specialization is needed, say for instance a distinction between a Raw Material that is a THSC and one that's not, all that needs to happen is for the new class to inherit Part, and it too can be used in a MaterialList.

4.1.2 Java Application

With some additional design specifications considered and further analysis, the model could be improved. But, now I've got an object oriented model to work with. As proof of concept, I created a Java application (Fig 4) from the model. The applications top level is the WorkOrder. Viewing down the figure is an obvious visual representation of the remaining classes. That was the goal and its exciting to see a design working like expected.

Figure 4 Screen capture of Java application



I can now demonstrate the concepts I described earlier. It is possible to navigate from a WorkOrder entry down to a Part, attach a cost, and move to another Part. It is this navigation that will make a full implementation of the THSC concept workable.

What you see in Figure 4 is a nice visual picture. But it does not let you see how the classes actually work. I'll look at the Part and Cost classes in more detail. First the Cost class. If the class diagram were a tree, the Cost class would be a leaf-node. Several other classes us it, but it uses no other classes.

The specification for the Cost class is that it holds a 'type' of cost. This could be something like Manufacturing, Inventory, Acquisition, ... And it also contains a dollar amount. My Java Cost class also had a phase and unit attribute which I'm not sure are necessary. Like its attributes, Cost has simple methods including getCost(), getType(), setCost(), and setType(). These allow access to the private attributes _type and _dollars.

The Part class get a little more interesting. Its attributes include stock keeping number (Sku), a name, a manufacturer, and a hash set. The hash set is used to hold and manipulate Cost objects. Part also has some basic access methods like getSku(), getMfg(), and getName(). But this is where it gets interesting. The Part class also has a methods called getCosts() and getTotalCost().

If you were to instantiate a Part object called _part, a statement like _part.getCosts(); would return all Cost *objects* associated with that part. If the method took and argument, say type, it would return only Cost objects of that specific type. These are not copies of the Cost objects, they are reference to the actual objects. So any manipulation affect the real objects. And getTotalCost() is similar. But instead if returning objects, it returns a dollar value representing the sum of each individual Costs dollar attribute.

Why is this important? Because it is the essence of the navigation through the class diagram presented earlier. It doesn't matter how 'far' away from the Cost object you get, it always accessible (if designed right). Say you wanted to find all the Waste costs for the soldier Part in a WorkOrder. You would issue a statement like:

```
_workorder.getParts("soldier").getCosts("Waste");
```

Its statements similar to these that my GUI uses to display data. This is why you can enter a cost for a Part, move to another Part in a different WorkOrder, and then return to the original Part and have the cost still there.

4.1.3 Object Persistence

In writing the code for these class, I choose to implement both WorkOrder and Part as composite classes. Hence the hash set I described for the Part class. I also touched on this previously when I mentioned WorkOrder contains MaterialList. From an object oriented standpoint, this is an acceptable way of implementing the relationship between WorkOrder and MaterialList. But the data in this Java application is not persistent. Or, there is no database behind it to store the objects created. To address this, you could transfer the class attributes to a relational database (not interested), create mapping to an object-relational database like PREDATOR (discussed shortly), or store the classes in an object database.

For storing classes in an object database, the ODMG has created a C++ standard for defining how to create the objects that will be stored. While my application is in Java, its easy to see how this C++ standard can be applicable in Java.* In the next couple of paragraphs I attempt to describe how this is done. To help me, I'll be referencing the book "C++ Object Databases. Programming with the ODMG Standard" by David Jordan.

(*Note, since all database class must inherit from d_Object in the Standard, a direct implementation of the Standard in Java impacts Java single inheritance model)

According to that standard, "When an application uses the ODMG interface, a class must be derived (directly or indirectly) from the ODMG d_Object base class so that its instances can be stored in the database." If I were to convert my WorkOrder class to C++, the class definition might look something like the following:

```
class WorkOrder : public d_Object {
     private char   *_name;
     private d_Ref<Product>  _prod;
     private d_Rel_Set<MaterialList, _id>   _materials;
     ...
}
```

What makes any instance of this object persistent is the new operator used. Three overloaded new() operators are supplied. Using the new() method as follows

```
WorkOrder wo = new(&db, "WorkOrder") WorkOrder();
```

would accomplish this. Here &db is a reference to a connection to an object oriented database.

The C++ definition of the WorkOrder class also introduces 2 ODMG types used to deal with using persisted objects: d_Ref<> and d_Rel_Set<>. d_Ref<Product>, in the case of the WorkOrder, indicates a reference to a Product object that is stored in the database. And d_Rel_Set<MaterialList> is a set representation of "a relationship between two classes." Its the second type, d_Rel_Set, that allows us to manage the composite classes. Reference to MaterialList objects are where they belong, in the WorkOrder class.

For my Part class, the hash set in the original code would be replaced with d_Rel_Set<Cost, id>. And since both the Part and Cost classes would have subclassed off of d_Object, I now have my information stored in a object database.

This is a fantastic standard from a new object oriented developers view. I purchased this book what must be six months ago, or more. As I started reading the book I realized my grasp of object orientated development and analysis was weak. It wasn't until I recently returned to it, after studying object orientation and writing the Java app, that I began to realized how powerful object orientation can be.

4.1.4 Inheritance

Getting back to the Java application, what I have not done is implement the inheritance outlined for the Part class. This leaves a hole in that there is no class for the THSC and no way to visualize it. However, in this case I don't see that as significant. MaterialList already contains Part. If I were to abstract it and create THSC, WIP, RawMaterial, etc., these new classes could be saved to 'MaterialLists' Part attribute without change to the MaterialList class itself. My goal was to only demonstrate the possibility, not create a full blown application.

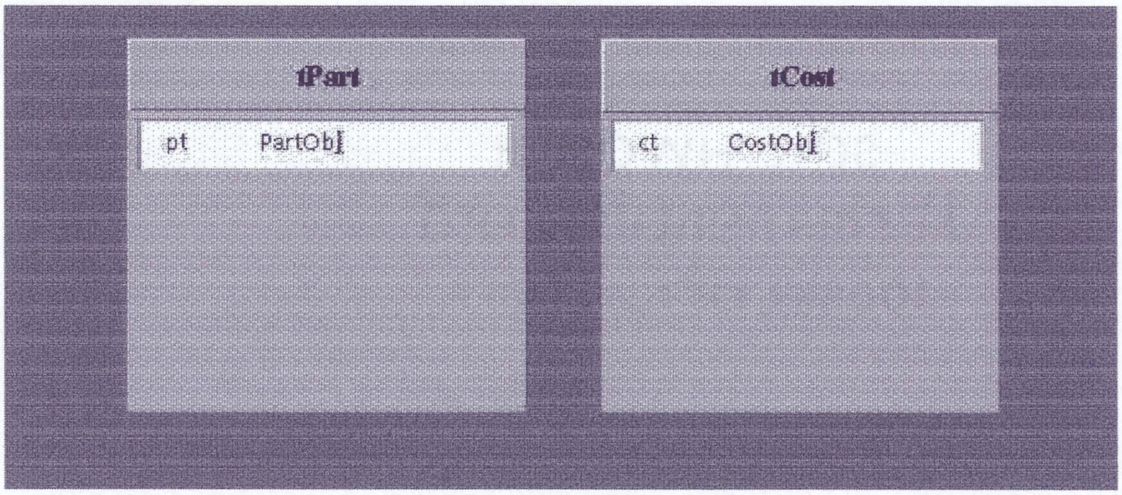
4.2 Object Relational

Where does this leave us. I didn't create a relational model because that is not what I'm studying. And, I believe we can agree that it would be trivial to create. I have, however, created an object-oriented model and take a good deal of time to show it to be workable. The last step in my process, and the goal when I started this project, was to move from the two opposite ends, relational and object-oriented, to the middle ground of object-relational using PREDATOR.

Unfortunately, as a middle ground, object-relational is just that. Its sort-of relational and sort-of object oriented and, for this design, its not a solution I would choose. This happens for a couple of reasons I'll discuss.

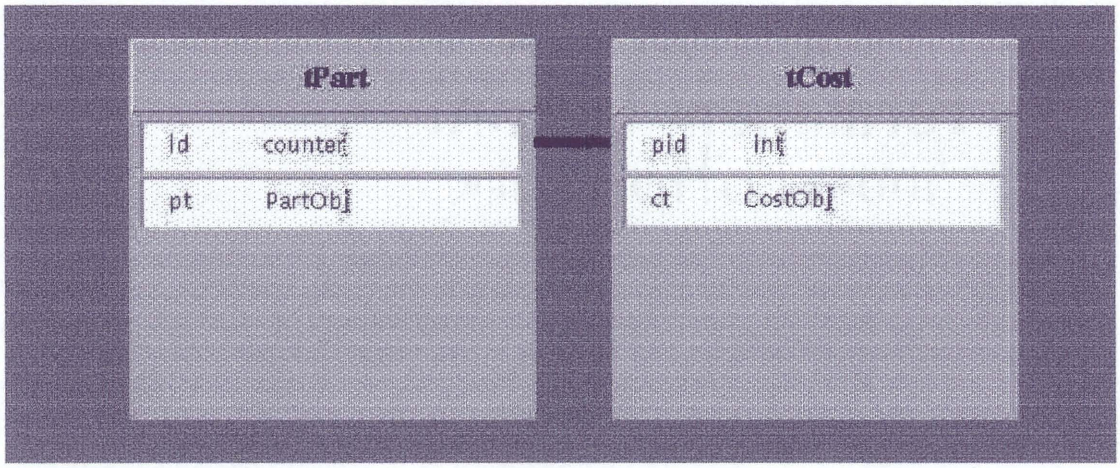
Again it comes back to the model. This time, instead of moving from the relational model, I moved from the object model. The initial vision was to take each class in the object model and make it a PREDATOR EADT. These EADTs needed to be stored, so for these EADTs, I'd create a relational table for each one. But then I ended up with tables that looked like Figure 5.

Figure 5 Part Table
w/ one field 'pt' of type PartObject Cost table & one field 'ct' of type CostObject



Now that each class is converted to a data type and put in their own table, I needed a key field to relate them on. Easy enough, just add a field to the table that's a counter. And to complete the relationship, a foreign key field will be needed in some tables. The result is shown in Figure 6.

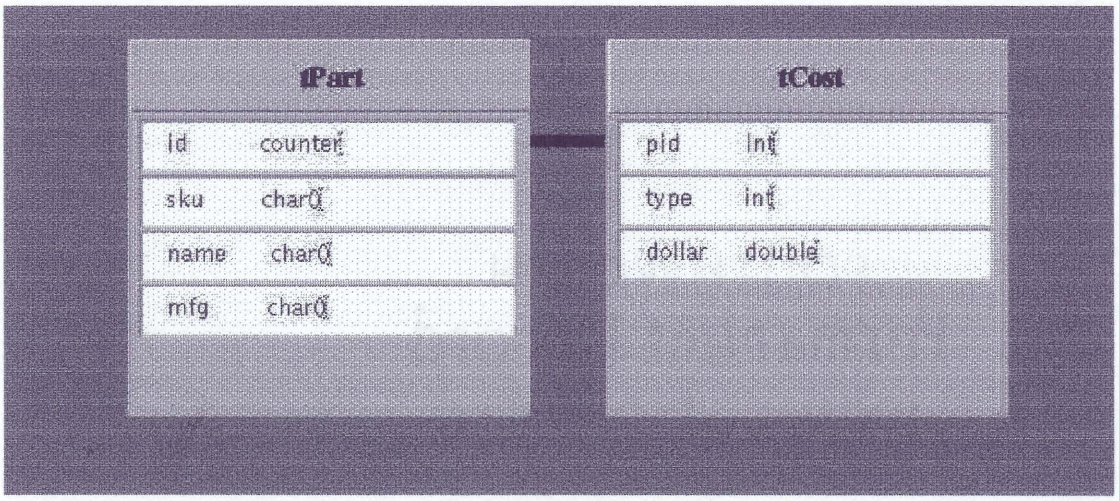
Figure 6 Relational Schema with EADTs



I should note here that I implemented the foreign keys as an attribute of the EADT. This is what is reflected in the code in the appendix. But you will see in a moment why this is not a significant issue.

It didn't take long, in fact I only implemented 2 EADTs in PREDATOR before it became completely obvious that we are really dealing with a relational model (Fig 7). The EADTs in this case turned out to be nothing more than a composite key. A little normalization, and its all cleaned up. At this point, implementing additional EADTs becomes nothing more than a programming exercise. I would benefit more by using my time to read object oriented books.

Figure 7 Relational Schema with EADT Attributes Converted to Fields



This is not the only issue I have with object-relational databases and the THSC problem. In the object oriented application, when I designed the Part class it included a hash set data structure (I could have used and array). This hash set kept in it one or more actual Cost objects: a one-to-many relationship. So, one Part object held references to its related Cost objects. Which also held true when I introduced the ODMG d_Rel_Set class and made Part and Cost persistent. This is the opposite result of what normalization on a relational databases does. There, it is the Costs responsibility to know its Part. I don't see any straight forward logical way to keep Parts composition intact in PREDATOR.

Also, I have a hard time seeing how PREDATOR can handle inheritance. The WIP and RawMaterial subclassing off of Part work well in the object oriented design. Even persistent in the ODMG standard, using the following statement satisfies inheritance:

```
Part _wip = new(&dw, "WIP") WIP();
```

But do I need to create a separate EADT and relational entity for each one in PREDATOR? If so, they are no longer related. Will I be able to store a WIP in the Part field? And if I can, how will I know its a WIP as opposed to a THSC? I leave these as questions because I don't know how to solve these problems.

I also have another thought. I'm not sure the magnitude, but consider this. Say you discover that PREDATOR is a good solution for image type data. Doing manipulations and calculations. PREDATOR implements the EADTs as integral to the database engine. What happens to performance if you have several users performing CPU or I/O intensive operations on these image types?

CHAPTER 5

CONCLUSION

I am both disappointed and excited about what I've learned. The disappointment is the understanding that, for this topic (THSC), PREDATOR's object-relational EADTs are not a good solution. Could they be made to work, absolutely. But it is my belief that its just as important to know when a technology doesn't apply as when it does.

This does not mean PREDATOR is not a good fit for another topic. A lot of individuals smarter than I feel that object relational databases do offer good solutions and that PREDATOR is a excellent research tool. The NSF awards attest to that. Even so, I stand by my conclusion.

The excitement, on the other hand, is the growing awareness of, and skill in, designing object oriented solutions. This was definitely a surprise, but looking back it seems terribly logical. Even though PREDATOR had relational foundations, the concept as I took it of EADTs was object oriented. To make a true evaluation, I needed to know both relational and object oriented design. Relational I know/knew. I now know object oriented design as well. Although no where near as well as relational.

My recommendations to come out of this work would be to keep looking for subject matter that is more tailored to PREDATOR if you wish to pursue that technology. However, to be intellectually honest, you need a solid grasp of object oriented design. object orientation is a way of thinking, not just an implementation. It has dramatically changed the way I view data.

I spent the better part of 3 months trying to learn to think object oriented. Had I not spent the time, I would not have developed the object oriented model presented here. And it follows that I wouldn't have come to the same conclusion.

In fact, if given the opportunity, and gazing into a crystal ball, I would concentrate on a purely object oriented database solution. I've now had a taste of object oriented and I'm convinced its the future course.

My other recommendation, this to the group in CEES working on the THSC problem, I believe the concept I outlined can lead you being able to answers the questions of your users. And when you go looking for an implementation, use a relational database.

APPENDIX A

THSC LISTS

A partial collection of hazardous substance lists.

1. The EPA list of toxic pollutants and hazardous substance prepared under sections 307 and 311 of the Federal Clean Water Act
2. The EPA list of hazardous air pollutants prepared under section 112 of the Federal Clean Air Act
3. The EPA list of restricted use pesticides found at 40 CFR 162.30 (relating to optional procedures for classification of pesticide uses by regulation).
4. The EPA Carcinogen Assessment Group's List of Carcinogens
5. The OSHA list of toxic and hazardous substances found in 29 CFR 1910, Subpart Z
6. The International Agency for Research on Cancer sublist, entitled Substances Found to Have at Least Sufficient Evidence of Carcinogenicity in Animals
7. The National Toxicology Program's list of substances published in the latest Annual Report on Carcinogens
8. The National Fire Protection Association list found in Hazardous Chemicals Data (NFPA 49)
9. The National Fire Protection Association list found in Fire Hazard Properties of Flammable Liquids, Gases, Volatile Solids (NFPA 325M), but only those substances found on sublists for health items, categories 2, 3 and 4; sublists for reactivity items, categories 3 and 4; sublists for flammability, categories 3 and 4
10. The American Conference of Governmental Industrial Hygienists' list found in Threshold Limit Value for Chemical Substances and Physical Agents in the Workplace.
11. The National Cancer Institute sublist, entitled Carcinogens Bioassays With at Least Evidence Suggestive of Carcinogenic Effect, but including only those substances which satisfy criteria of the National Toxicology Program indicating significant carcinogenic effect.

APPENDIX B

SOURCE CODE

Due to the nature and format of this document, a complete listing of all source code is not included. However, several pages are included at the back for *every* source file. If the source is less than three pages, it is included in its entirity. Otherwise, the souce code is terminated with an elipsis (...) after three pages.

The full source is available on request by emailing jancera@ibm.net

REFERENCES

1. M. Fowler, *UML Distilled. Applying The Standard Object Modeling Language*, Addison Wesley Longman, Reading, MA, 1997

2. D. Jordan, *C++ Object Databases. Programming with the ODMG Standard*, Addison Wesley Longman, Reading, MA, 1998

3. NJIT CEES home, http://www.njit.edu/CEES

4. NJIT NJPIES project, http://njpies.njit.edu/

5. P. Seshadri, *PREDATOR*, Computer Science Department, CORNELL UNIVERSITY, http://simon.cs.cornell.edu/Info/Projects/PREDATOR/

6. P. Seshadri, *PREDATOR Design Document*, Computer Science Department, CORNELL UNIVERSITY, http://simon.cs.cornell.edu/Info/Projects/PREDATOR/designdoc.html

7. National Science Foundation, PREDATORdocument, http://www.nsf.gov/cgi-bin/showaward?award=9702149

8. Shore Storage Manager, Computer Science Department, University of Wisconsin, http://www.cs.wisc.edu/shore/

9. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, Inc, Englewood Cliffs, NJ, 1991

10. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley Longman, Reading, MA, 1995

11. M. Stonebraker and P. Brown, *Object-Relational DBMSs: Tracking the Next Great Wave*, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1999

12. C.J. Berg, *Advanced Java: Development for Enterprise Applications*, Prentice Hall PTR, Upper Saddle River, NJ, 1998

13. *JavaWorld*, http://www.javaworld.com

Part.C

```
/***************
 *  Part EADT
 *  Class definition code
 *  adapted by Craig Jancerak
 *  November 1998
 *  New Jersey Institute of Technology
 **************/

#include <stdio.h>
#include <string.h>
#include "part.h"
#include "funcexpr.h"
#include "aggrexpr.h"
#include "JavaUDF.h"

//
// Part Number ADT
//

XxxBasicType XxxPartADT::PartTypeId;

XxxPartADT::XxxPartADT(XxxBasicType Id)
 : XxxSmallFixedADT(Id, "part", sizeof(PartStruct))
{
  PartTypeId = Id;
  /*********
     Register methods for Part
     each function name ( eg sku() ) has an entry
  *********/
  NumMethods = 4;
  MethArray = new FuncMethodInfo* [NumMethods];
  MethArray[0] = new FuncMethodInfo("sku",XXX_STRING, 0,0,
                           (ExecuteFunc)(PlanInfo::execSku),0);
  MethArray[1] = new FuncMethodInfo("name",XXX_STRING, 0,0,
                           (ExecuteFunc)(PlanInfo::execName),0);
  MethArray[2] = new FuncMethodInfo("mfg",XXX_STRING, 0,0,
                           (ExecuteFunc)(PlanInfo::execMfg),0);
  MethArray[3] = new FuncMethodInfo("subclass",XXX_INT, 0,0,
                           (ExecuteFunc)(PlanInfo::execSubclass),0);

}

// constructor
XxxPartADT::~XxxPartADT()
{
}


XxxErrCode XxxPartADT::methOptimize(XxxFuncParseInfo* FuncParse,
                              XxxValueExprPlan* OwnerPlan,
                              XxxValExprPlanList* ArgPlans,
                              XxxFuncPlanInfo*& FPlan) const
{

  if(!MethArray[FuncParse->MethodIndex]->OptFunc) {
```

```
  if(!MethArray[FuncParse->MethodIndex]->OptFunc) {
    FPlan = new PlanInfo(FuncParse, (ArgPlans?ArgPlans->Count():0));
    return XXX_OK;
  }
  else
    return XxxADTClass::methOptimize(FuncParse, OwnerPlan,
                                     ArgPlans, FPlan);

}


/*---------------------------------------------------------------------------
----
  function to specify how the attribute of Part are written
  to outstream (screen, disk, ...)
  */
XxxErrCode XxxPartADT::WriteText(ostream& OutStream,
                                 const XxxADTValue* Object,
                                 const XxxADTMetaInfo* MetaInfo) const
{
 PartStruct *CVal = (PartStruct *)Object;
 OutStream << CVal->_sku << " "
           << CVal->_name << " "
           << CVal->_mfg << " "
           << CVal->_subclass;
 return XXX_OK;
}
/*---------------------------------------------------------------------------
----
  function to specify how the attribute of Part are read
  from instream (screen, disk, ...).  Also used where
  insert statements are issued.
  */
XxxErrCode XxxPartADT::ReadText(istream& InStream,
                                XxxADTValue* ObjectRef,
                                const XxxADTMetaInfo* MetaInfo) const
{

  PartStruct *CVal = (PartStruct *)ObjectRef;
  ASSERT(CVal);
  CVal->Init();   //  ???????
  /*
   */
  InStream >> CVal->_sku
           >> CVal->_name
           >> CVal->_mfg
           >> CVal->_subclass;
  if (InStream.bad()) {
    XXX_NOTE_ERROR(XXX_BADPARSE,
               "The proper syntax for a part number is: sku name mfg 1");
    return XXX_NOT_OK;
  }
  return XXX_OK;
}
```

```
/*-----------------------------------------------------------------------------
----
  method that defines how two instances of class are equal
  */
XxxBool  XxxPartADT::Equals(const XxxADTValue* Obj1,
                          const XxxADTMetaInfo* MInfo1,
                          const XxxADTValue* Obj2,
                          const XxxADTMetaInfo* MInfo2) const
{
 PartStruct *CVal1 = (PartStruct *)Obj1;
 PartStruct *CVal2 = (PartStruct *)Obj2;

 ASSERT((CVal1 && CVal2));
 if (CVal1->_sku == CVal2->_sku)
   return XXX_TRUE;
 else
   return XXX_FALSE;
}

//----------------------------------------------------------------------------
---
//----------------------------------------------------------------------------
---
//OWNER: Fabian Camargo DATE: 1/29/97
//PURPOSE: Returns a bounding box, which can then be used for an rtree
search
//NOTES: This is not really useful in itself, but it was used to test and
// debug the rtree code.

XxxBool
XxxPartADT::GetBoundingBox(const XxxADTValue*& Object,
                         XxxBoundingBox& Box)
{
      // code removed. cgj
}

/*-----------------------------------------------------------------------------
-----
  casting functions
  determines whether Part can be convert into or from
  another class
  */
XxxBool XxxPartADT::CanCastTo(XxxBasicType TargetType,
                          const XxxADTMetaInfo* SourceMetaInfo,
                          const XxxADTMetaInfo* TargetMetaInfo)
{
    return XXX_FALSE;
}

XxxErrCode XxxPartADT::CastTo(XxxBasicType TargetType,
                          const XxxADTMetaInfo* SourceMetaInfo,
                          const XxxADTMetaInfo* TargetMetaInfo,
                          XxxADTValue* OldObject,
                          XxxADTValue* NewObject)
{
}
...
```

Part.H

```
/***************
 *  Part EADT
 *  Class Header code
 *  adapted by Craig Jancerak
 *  November 1998
 *  New Jersey Institute of Technology
 ****************/

#ifndef XXX_PART_H
   #define XXX_PART_H
   #include "genadt.h"
   #include "optrules.h"
   #include "BoundingBox.h"
   #include "adtfunc.h"

/**
 // this type represents part numbers
 */
class XxxPartADT : public XxxSmallFixedADT {
public:
   static XxxBasicType PartTypeId;
protected:
   class PartStruct : public XxxADTValue {public:
      int   _subclass;
      char  _sku[30];
      char  _name[30];
      char  _mfg[30];
   };

  //// plan structure for methods on part numbers
   class PlanInfo : public XxxFuncPlanInfo {
   friend class XxxPartADT;
   protected :
      XxxErrCode execSku(XxxValueEnv* Env, XxxADTValue *ReturnValue) const;
      XxxErrCode execName(XxxValueEnv* Env, XxxADTValue *ReturnValue)
const;
      XxxErrCode execMfg(XxxValueEnv* Env, XxxADTValue *ReturnValue) const;
      XxxErrCode execSubclass(XxxValueEnv* Env, XxxADTValue *ReturnValue)
const;

   public :
      PlanInfo(const XxxFuncParseInfo* FuncParse, int argCount )
      :  XxxFuncPlanInfo(FuncParse, argCount){}
   };


   virtual XxxErrCode methOptimize(XxxFuncParseInfo* FuncParse,
                            XxxValueExprPlan* OwnerPlan,
                            XxxValExprPlanList* ArgPlans,
                            XxxFuncPlanInfo*& FPlan) const;


public:
    XxxPartADT(XxxBasicType Id);
    virtual ~XxxPartADT();

    virtual XxxErrCode WriteText(ostream& OutStream,
                   const XxxADTValue* Object,
                   const XxxADTMetaInfo* MetaInfo) const;
    virtual XxxErrCode ReadText(istream& InStream,
                   XxxADTValue* ObjectRef,
                   const XxxADTMetaInfo* MetaInfo) const;
    virtual XxxBool  Equals(const XxxADTValue* Obj1,
                   const XxxADTMetaInfo* MInfo1,
                   const XxxADTValue* Obj2,
                   const XxxADTMetaInfo* MInfo2) const;


   /**
    * Determines if this ADT can cast an object of another type into
    * its own type.
    * @param SourceType The type to cast from
    * @param SourceMetaInfo The meta info of the object to be casted
    * @param TargetMetaInfo The meta info of the new object to be created
    */
    virtual XxxBool CanCastFrom(XxxBasicType SourceType,
                    const XxxADTMetaInfo* SourceMetaInfo,
                    const XxxADTMetaInfo* TargetMetaInfo);
   /**
    * Executes a cast from another type into this type.
    * @param CastType The type id of the old object
    * @param SourceMetaInfo The meta info of the object to be casted
    * @param TargetMetaInfo The meta info of the new object to be created
    * @param OldObject The object to be casted
    * @param NewObject The new object to be created
    */
    virtual XxxErrCode CastFrom(XxxBasicType SourceType,
                    const XxxADTMetaInfo* SourceMetaInfo,
                    const XxxADTMetaInfo* TargetMetaInfo,
                    XxxADTValue* OldObject,
                    XxxADTValue* NewObject);
   /**
    * Determines if this ADT can cast an object of its type into
    * another type.
    * @param TargetType The type to cast to
    * @param SourceMetaInfo The meta info of the object to be casted
    * @param TargetMetaInfo The meta info of the new object to be created
    */
    virtual XxxBool CanCastTo(XxxBasicType TargetType,
                     const XxxADTMetaInfo* SourceMetaInfo,
                     const XxxADTMetaInfo* TargetMetaInfo);
   /**
    * Executes a cast from this type into another type.
    * @param CastType The type id of the new object
    * @param SourceMetaInfo The meta info of the object to be casted
    * @param TargetMetaInfo The meta info of the new object to be created
    * @param OldObject The object to be casted
    * @param NewObject The new object to be created
...
```

Partmethods.C

```
/***************
 *  Part EADT
 *  implementation code for EADT methods
 *  adapted by Craig Jancerak
 *  November 1998
 *  New Jersey Institute of Technology
 **************/

#include "part.h"
#include "aggrexpr.h"

// .sku() method
XxxErrCode XxxPartADT::PlanInfo::
execSku(XxxValueEnv* Env, XxxADTValue *ReturnValue) const{
  PartStruct* CV = ( PartStruct* )(Env->OwnerValPtr);
  ASSERT((Env->NumArgs == 0));
  strcpy( ((XxxCoreADTClass::StrValue *)ReturnValue)->Value, CV->_sku );
  return XXX_OK;
}

// .name() method
XxxErrCode XxxPartADT::PlanInfo::
execName(XxxValueEnv* Env, XxxADTValue *ReturnValue) const{
  PartStruct* CV = ( PartStruct* )(Env->OwnerValPtr);
  ASSERT((Env->NumArgs == 0));
  strcpy( ((XxxCoreADTClass::StrValue *)ReturnValue)->Value, CV->_name );
  return XXX_OK;
}

// .mfg() method
XxxErrCode XxxPartADT::PlanInfo::
execMfg(XxxValueEnv* Env, XxxADTValue *ReturnValue) const{
  PartStruct* CV = ( PartStruct* )(Env->OwnerValPtr);
  ASSERT((Env->NumArgs == 0));
  strcpy( ((XxxCoreADTClass::StrValue *)ReturnValue)->Value, CV->_mfg );
  return XXX_OK;
}

// .subclass method
XxxErrCode XxxPartADT::PlanInfo::
execSubclass(XxxValueEnv* Env, XxxADTValue *ReturnValue) const{
  PartStruct* CV = ( PartStruct* )(Env->OwnerValPtr);
  ASSERT((Env->NumArgs == 0));
  ((XxxCoreADTClass::IntValue *)ReturnValue)->Value = CV->_subclass;
  return XXX_OK;
}
```

Cost.C

```
/***************
 *  Cost EADT
 *  Class definition code
 *  adapted by Craig Jancerak
 *  November 1998
 *  New Jersey Institute of Technology
 **************/

#include <stdio.h>
#include <string.h>
#include "cost.h"
#include "funcexpr.h"
#include "aggrexpr.h"
#include "JavaUDF.h"

//
// Cost Number ADT
//

XxxBasicType XxxCostADT::CostTypeId;

XxxCostADT::XxxCostADT(XxxBasicType Id)
 : XxxSmallFixedADT(Id, "cost", sizeof(CostStruct))
{
  CostTypeId = Id;
  // Register methods for Cost
  NumMethods = 6;
  MethArray = new FuncMethodInfo* [NumMethods];
  MethArray[0] = new FuncMethodInfo("type",XXX_STRING, 0,0,
                            (ExecuteFunc)(PlanInfo::execType),0);
  MethArray[1] = new FuncMethodInfo("phase",XXX_STRING, 0,0,
                            (ExecuteFunc)(PlanInfo::execPhase),0);
  MethArray[2] = new FuncMethodInfo("dollars",XXX_DOUBLE, 0,0,
                            (ExecuteFunc)(PlanInfo::execDollars),0);
  MethArray[3] = new FuncMethodInfo("partID",XXX_INT, 0,0,
                            (ExecuteFunc)(PlanInfo::execPartID),0);
  MethArray[4] = new FuncMethodInfo("increase",XXX_DOUBLE, 0,0,

(ExecuteFunc)(PlanInfo::execIncrease),1,XXX_DOUBLE);
  MethArray[5] = new FuncMethodInfo("decrease",XXX_DOUBLE, 0,0,

(ExecuteFunc)(PlanInfo::execDecrease),1,XXX_DOUBLE);
  MethArray[6] = new FuncMethodInfo("reset",XXX_DOUBLE, 0,0,

(ExecuteFunc)(PlanInfo::execReset),1,XXX_DOUBLE);

}

XxxCostADT::~XxxCostADT()
{
}

XxxErrCode XxxCostADT::methOptimize(XxxFuncParseInfo* FuncParse,
                              XxxValueExprPlan* OwnerPlan,
                              XxxValExprPlanList* ArgPlans,
                              XxxFuncPlanInfo*& FPlan) const
{

  if(!MethArray[FuncParse->MethodIndex]->OptFunc) {
    FPlan = new PlanInfo(FuncParse, (ArgPlans?ArgPlans->Count():0));
    return XXX_OK;
  }
  else
    return XxxADTClass::methOptimize(FuncParse, OwnerPlan,
                            ArgPlans, FPlan);

}


XxxErrCode XxxCostADT::WriteText(ostream& OutStream,
                           const XxxADTValue* Object,
                           const XxxADTMetaInfo* MetaInfo) const
{
 CostStruct *CVal = (CostStruct *)Object;
 OutStream << CVal->_type << " "
          << CVal->_phase << " "
          << CVal->_dollars << " "
          << CVal->_partOID;
 return XXX_OK;
}


XxxErrCode XxxCostADT::ReadText(istream& InStream,
                          XxxADTValue* ObjectRef,
                          const XxxADTMetaInfo* MetaInfo) const
{

  CostStruct *CVal = (CostStruct *)ObjectRef;
  ASSERT(CVal);
  CVal->Init();   //  ??????? cgj
  InStream >> CVal->_type
           >> CVal->_phase
           >> CVal->_dollars
           >> CVal->_partOID;
  if (InStream.bad()) {
    XXX_NOTE_ERROR(XXX_BADPARSE,
              "The proper syntax for a cost number is: type phase dollar
partId");
    return XXX_NOT_OK;
  }
  return XXX_OK;
}

//-----------------------------------------------------------------------
----
XxxBool  XxxCostADT::Equals(const XxxADTValue* Obj1,
                         const XxxADTMetaInfo* MInfo1,
                         const XxxADTValue* Obj2,
                         const XxxADTMetaInfo* MInfo2) const
```

```
{
 CostStruct *CVal1 = (CostStruct *)Obj1;
 CostStruct *CVal2 = (CostStruct *)Obj2;

 ASSERT((CVal1 && CVal2));
 if ((CVal1->_partOID == CVal2->_partOID) && (CVal1->_type ==
CVal2->_type))
   return XXX_TRUE;
 else
   return XXX_FALSE;
}

//------------------------------------------------------------------------
----
//------------------------------------------------------------------------
----
//OWNER: Fabian Camargo DATE: 1/29/97
//PURPOSE: Returns a bounding box, which can then be used for an rtree
search
//NOTES: This is not really useful in itself, but it was used to test and
// debug the rtree code.

XxxBool
XxxCostADT::GetBoundingBox(const XxxADTValue*& Object,
                        XxxBoundingBox& Box)
{
/*
      removed by cgj
*/
}

/*-----------------------------------
 * casting functions
 */
XxxBool XxxCostADT::CanCastTo(XxxBasicType TargetType,
                         const XxxADTMetaInfo* SourceMetaInfo,
                         const XxxADTMetaInfo* TargetMetaInfo)
{
    return XXX_FALSE;
}

XxxErrCode XxxCostADT::CastTo(XxxBasicType TargetType,
                         const XxxADTMetaInfo* SourceMetaInfo,
                         const XxxADTMetaInfo* TargetMetaInfo,
                         XxxADTValue* OldObject,
                         XxxADTValue* NewObject)
{
}

XxxBool XxxCostADT::CanCastFrom(XxxBasicType SourceType,
                           const XxxADTMetaInfo* SourceMetaInfo,
                           const XxxADTMetaInfo* TargetMetaInfo)
{
    return XXX_FALSE;
}
...
```

Cost.H

```
/***************
 *  Cost EADT
 *  Class Header code
 *  adapted by Craig Jancerak
 *  November 1998
 *  New Jersey Institute of Technology
 ****************/

#ifndef XXX_COST_H
   #define XXX_COST_H
   #include "genadt.h"
   #include "optrules.h"
   #include "BoundingBox.h"
   #include "adtfunc.h"

/**
 // this type represents cost numbers
 */
class XxxCostADT : public XxxSmallFixedADT {
public:
   static XxxBasicType CostTypeId;
protected:
   class CostStruct : public XxxADTValue {public:
      char     _type[30];
      char     _phase[30];
      double   _dollars;
      int      _partOID;
   };

  //// plan structure for methods on cost numbers
   class PlanInfo : public XxxFuncPlanInfo {
   friend class XxxCostADT;
   protected :
      XxxErrCode execType(XxxValueEnv* Env, XxxADTValue *ReturnValue)
const;
      XxxErrCode execPhase(XxxValueEnv* Env, XxxADTValue *ReturnValue)
const;
      XxxErrCode execDollars(XxxValueEnv* Env, XxxADTValue *ReturnValue)
const;
      XxxErrCode execPartID(XxxValueEnv* Env, XxxADTValue *ReturnValue)
const;
      XxxErrCode execIncrease(XxxValueEnv* Env, XxxADTValue *ReturnValue)
const;
      XxxErrCode execDecrease(XxxValueEnv* Env, XxxADTValue *ReturnValue)
const;
      XxxErrCode execReset(XxxValueEnv* Env, XxxADTValue *ReturnValue)
const;

   public :
      PlanInfo(const XxxFuncParseInfo* FuncParse, int argCount )
      :  XxxFuncPlanInfo(FuncParse, argCount){}
   };
   virtual XxxErrCode methOptimize(XxxFuncParseInfo* FuncParse,
                           XxxValueExprPlan* OwnerPlan,
                           XxxValExprPlanList* ArgPlans,
                           XxxFuncPlanInfo*& FPlan) const;


public:
   XxxCostADT(XxxBasicType Id);
   virtual ~XxxCostADT();

   virtual XxxErrCode WriteText(ostream& OutStream,
                   const XxxADTValue* Object,
                   const XxxADTMetaInfo* MetaInfo) const;
   virtual XxxErrCode ReadText(istream& InStream,
                   XxxADTValue* ObjectRef,
                   const XxxADTMetaInfo* MetaInfo) const;
   virtual XxxBool  Equals(const XxxADTValue* Obj1,
                   const XxxADTMetaInfo* MInfo1,
                   const XxxADTValue* Obj2,
                   const XxxADTMetaInfo* MInfo2) const;


  /**
   * Determines if this ADT can cast an object of another type into
   * its own type.
   * @param SourceType The type to cast from
   * @param SourceMetaInfo The meta info of the object to be casted
   * @param TargetMetaInfo The meta info of the new object to be created
   */
   virtual XxxBool CanCastFrom(XxxBasicType SourceType,
                    const XxxADTMetaInfo* SourceMetaInfo,
                    const XxxADTMetaInfo* TargetMetaInfo);
  /**
   * Executes a cast from another type into this type.
   * @param CastType The type id of the old object
   * @param SourceMetaInfo The meta info of the object to be casted
   * @param TargetMetaInfo The meta info of the new object to be created
   * @param OldObject The object to be casted
   * @param NewObject The new object to be created
   */
   virtual XxxErrCode CastFrom(XxxBasicType SourceType,
                    const XxxADTMetaInfo* SourceMetaInfo,
                    const XxxADTMetaInfo* TargetMetaInfo,
                    XxxADTValue* OldObject,
                    XxxADTValue* NewObject);
  /**
   * Determines if this ADT can cast an object of its type into
   * another type.
   * @param TargetType The type to cast to
   * @param SourceMetaInfo The meta info of the object to be casted
   * @param TargetMetaInfo The meta info of the new object to be created
   */
   virtual XxxBool CanCastTo(XxxBasicType TargetType,
                     const XxxADTMetaInfo* SourceMetaInfo,
...
```

Costmethods.C

```
/***************
 *  Cost EADT
 *  implementation code for EADT methods
 *  adapted by Craig Jancerak
 *  November 1998
 *  New Jersey Institute of Technology
 **************/


#include "cost.h"
#include "aggrexpr.h"

// .type()
XxxErrCode XxxCostADT::PlanInfo::
execType(XxxValueEnv* Env, XxxADTValue *ReturnValue) const{
  CostStruct* CV = ( CostStruct* )(Env->OwnerValPtr);
  ASSERT((Env->NumArgs == 0));
  strcpy( ((XxxCoreADTClass::StrValue *)ReturnValue)->Value, CV->_type );
  return XXX_OK;
}

// .phase()
XxxErrCode XxxCostADT::PlanInfo::
execPhase(XxxValueEnv* Env, XxxADTValue *ReturnValue) const{
  CostStruct* CV = ( CostStruct* )(Env->OwnerValPtr);
  ASSERT((Env->NumArgs == 0));
  strcpy( ((XxxCoreADTClass::StrValue *)ReturnValue)->Value, CV->_phase );
  return XXX_OK;
}

// .dollars()
XxxErrCode XxxCostADT::PlanInfo::
execDollars(XxxValueEnv* Env, XxxADTValue *ReturnValue) const{
  CostStruct* CV = ( CostStruct* )(Env->OwnerValPtr);
  ASSERT((Env->NumArgs == 0));
  ((XxxCoreADTClass::DblValue *)ReturnValue)->Value = CV->_dollars;
  return XXX_OK;
}

// .partid()
XxxErrCode XxxCostADT::PlanInfo::
execPartID(XxxValueEnv* Env, XxxADTValue *ReturnValue) const{
  CostStruct* CV = ( CostStruct* )(Env->OwnerValPtr);
  ASSERT((Env->NumArgs == 0));
  ((XxxCoreADTClass::IntValue *)ReturnValue)->Value = CV->_partOID;
  return XXX_OK;
}

// .increase(x)
XxxErrCode XxxCostADT::PlanInfo::
execIncrease(XxxValueEnv* Env, XxxADTValue *ReturnValue) const{
  CostStruct* CV = ( CostStruct* )(Env->OwnerValPtr);
  ASSERT((Env->NumArgs == 1));
  double* ArgCV = ((double*)(Env->ArgValsPtrs[0]));
  ((CostStruct *)ReturnValue)->_dollars = CV->_dollars + *ArgCV;
  return XXX_OK;
}

// .decrease(x)
XxxErrCode XxxCostADT::PlanInfo::
execDecrease(XxxValueEnv* Env, XxxADTValue *ReturnValue) const{
  CostStruct* CV = ( CostStruct* )(Env->OwnerValPtr);
  ASSERT((Env->NumArgs == 1));
  double* ArgCV = ((double*)(Env->ArgValsPtrs[0]));
  ((CostStruct *)ReturnValue)->_dollars = CV->_dollars - *ArgCV;
  return XXX_OK;
}

// .reset(x)
XxxErrCode XxxCostADT::PlanInfo::
execReset(XxxValueEnv* Env, XxxADTValue *ReturnValue) const{
  CostStruct* CV = ( CostStruct* )(Env->OwnerValPtr);
  ASSERT((Env->NumArgs == 1));
  double* ArgCV = ((double*)(Env->ArgValsPtrs[0]));
  ((CostStruct *)ReturnValue)->_dollars = *ArgCV;
  return XXX_OK;
}
```

Main.java

```
import Thesis.THSC.*;

//import com.sun.java.util.collections.*;
import COM.objectspace.jgl.*;
import java.io.*;
import java.util.*;
import java.awt.event.*;

/**
 * This class was generated by a SmartGuide.
 *
 */
public class Main  {
      private     Visual          _visual = null;      // this the just
the gui
      private     WorkOrder   _wo = null;
      private     ThscPart          _part = null;
      private     ThscCost          _cost = null;
      private     HashSet           _orders = null;

      /**
       * inner class used by the _visual objects Work Order Add button.
       * When that button is pushed the Desc, Loc, and Qty text fields
       * are read and a new WorkOrder object is created and added to
       * to the _orders HashSet.  An item is also added to the
       * Work Order List component.
       */
      class AddWOListener implements ActionListener {
            public void actionPerformed( ActionEvent evt ) {
                  _visual.ivjAddWO.setEnabled( false );
                  Stringdesc = _visual.ivjWODesc.getText();
                  Stringloc = _visual.ivjWOLoc.getText();
                  int         qty;

                  if ( _visual.ivjWOQty.getText().length() > 0) {
                        qty = new Integer( _visual.ivjWOQty.getText()
).intValue();
                  } else {
                        qty = 0;
                  }

                  if ( desc.length()>0 & loc.length()>0 & qty>0 ) {
                        _wo = new WorkOrder( desc, loc, qty );
                        _orders.add( _wo );
                        _visual.ivjWorkOrderList.add( _wo.toString() );
                        _visual.ivjWODesc.setText( null );
                        _visual.ivjWOLoc.setText( null );
                        _visual.ivjWOQty.setText( null );
                  }

                  System.out.println( _orders.size() );
                  _visual.ivjAddWO.setEnabled( true );
            }
	}

	/**
	 * inner class used by the _visual objects Materials Add button.
	 */
	class AddPartListener implements ActionListener {
		public void actionPerformed( ActionEvent evt ) {
			_visual.ivjAddPart.setEnabled( false );
			Stringsku = _visual.ivjPartSku.getText();
			Stringname = _visual.ivjPartName.getText();
			Stringmfg = _visual.ivjPartMfg.getText();
			int		qty;

			if ( _visual.ivjPartQty.getText().length() > 0) {
				qty = new Integer( _visual.ivjPartQty.getText()
).intValue();
			} else {
				qty = 0;
			}

			if ( sku.length()>0 & name.length()>0 & mfg.length()>0 &
qty>0 ) {
				_part = new ThscPart( sku, name, mfg );
				String m = _wo.addMaterials( _part, qty );
				_visual.ivjMaterialListList.add( m );
				_visual.ivjPartSku.setText( null );
				_visual.ivjPartName.setText( null );
				_visual.ivjPartMfg.setText( null );
				_visual.ivjPartQty.setText( null );
			}

			System.out.println( _wo.getMaterialCount() );
			_visual.ivjAddPart.setEnabled( true );

		}
	}

	/**
	 * inner class used by the _visual objects Cost Add button.
	 */
	class AddCostListener implements ActionListener {
		public void actionPerformed( ActionEvent evt ) {
			_visual.ivjAddCost.setEnabled( false );
			Stringtype = _visual.ivjCostType.getText();
			doubleamt;

			if ( _visual.ivjCostAmt.getText().length() > 0) {
				amt = new Double( _visual.ivjCostAmt.getText()
).doubleValue();
			} else {
				amt = 0;
			}

			if ( type.length()>0 & amt>0 ) {
				_cost = new ThscCost( type, amt );
				_part.addCost( _cost );
```

```
                              _visual.ivjCostList.add( _cost.toString() );
                              _visual.ivjCostType.setText( null );
                              _visual.ivjCostAmt.setText( null );
                        }

                        System.out.println( _wo.getMaterialCount() );
                        _visual.ivjAddCost.setEnabled( true );

                  }
            }

            /**
             * inner class that is used by the _visual objects Work Order List
component.
             * When an item in the Work Order list is selected, the WorkOrder
object (_wo) is
             * interogated to see if it has any associated Materials.  If it
does, the
             * Materials List gets populated.
             */
            class WorkOrderListListener implements ItemListener {
                  public void itemStateChanged ( ItemEvent evt ) {
                        int adv = 1;
                        String iteration = null;
                        String selected =
_visual.ivjWorkOrderList.getSelectedItem();
                        WorkOrder wo = null;
                        HashSetIterator i = _orders.begin();

                        System.out.println( "\nLooking for: " + selected );
                        System.out.println( "iterating " + _orders.size() + "
time(s)" );

                        //find the actual workorder object
                        while ( i.hasMoreElements() ) {
                              iteration = i.get().toString();
                              System.out.println( "\tchecking item " + iteration
);

                              if ( selected.equals( iteration ) ) {
                                    System.out.println( "\t--- FOUND ---" );
                                    _wo = (WorkOrder)i.get();
                                    adv = i.distance( _orders.end() );
                              }
                              i.advance( adv );
                        }

                        _visual.ivjMaterialListList.removeAll();
                        _visual.ivjCostList.removeAll();
                        _visual.ivjPartsText.setText( null );
                        _visual.ivjNewMaterialsPanel.setEnabled( true );
                        _visual.ivjAddPart.setEnabled( true );

                        i = _wo.getMaterials(); // getMaterials is a func that
rtns an iterator of materials contained in a workorder

...
```

MaterialList.java

```
package Thesis.THSC;
import com.sun.java.util.collections.*;
/**
 * This class was generated by a SmartGuide.
*/
public class MaterialList {
      private int _qty;
      private ThscPart _part;
      private WorkOrder _wo;
/**
 * This method was created by a SmartGuide.
 */
public MaterialList ( WorkOrder wo, ThscPart part, int qty ) {
      _qty = qty;
      _part = part;
      _wo = wo;
}
/**
 * This method was created by a SmartGuide.
 * @return Thesis.THSC.ThscPart
 */
public ThscPart getPart( ) {
      return _part;
}
/**
 * This method was created by a SmartGuide.
 * @return int
 */
public int getQty( ) {
      return _qty;
}
/**
 * This method was created by a SmartGuide.
 * @return Thesis.THSC.ThscPart
 * @param part Thesis.THSC.ThscPart
 */
public ThscPart setPart( ThscPart part ) {
      _part = part;
      return _part;
}
/**
 * This method was created by a SmartGuide.
 * @param newqty int
 */
public void setQty( int newqty ) {
      _qty = newqty;
      return;
}
/**
 * Returns a String that represents the value of this object.
 */
public String toString() {return ( _part.toString() + " qty " + _qty ); }
}
```

WorkOrder.java

```
package Thesis.THSC;
import java.util.*;
//import com.sun.java.util.collections.*;
import COM.objectspace.jgl.*;
/**
 * This class was generated by a SmartGuide.
 *
 */
public class WorkOrder {
      private static int_staticnumber = 9;
      private int             _worknumber;
      private String          _desc = null;
      private Date      _dateissued = null ;
      private String          _location = null ;
      private int             _qty = 0;

      private HashSet   _materials = null;
/**
 * This method was created by a SmartGuide.
 */
public WorkOrder ( String desc, String loc, int qty ) {
      _worknumber = ++_staticnumber;
      _desc = desc;
      _dateissued = new Date();
      _location = loc;
      _qty = qty;

      _materials = new HashSet( true );
}
/**
 * This method was created by a SmartGuide.
 * @return String
 * @param part Thesis.THSC.ThscPart
 * @param qty int
 */
public String addMaterials( ThscPart part, int qty ) {
      MaterialList ml = new MaterialList( this, part, qty );
      if ( _materials.add( ml ) == null )
            return ml.toString();
      return "";
}
/**
 * This method was created by a SmartGuide.
 * @return java.lang.String
 */
public String equals( ) {
      return ( "Order#" + _worknumber + " " + _desc );
}
/**
 * This method was created by a SmartGuide.
 * @return java.lang.String
 */
public String getDesc( ) {
      return _desc;
}
/**
 * This method was created by a SmartGuide.
 * @return java.util.Date
 */
public Date getIssued( ) {
      return _dateissued;
}
/**
 * This method was created by a SmartGuide.
 * @return int
 */
public int getMaterialCount( ) {
      return _materials.size( );
}
/**
 * This method was created by a SmartGuide.
 * @return COM.objectspace.jgl.HashSetIterator
 */
public HashSetIterator getMaterials( ) {
      return _materials.begin();
}
/**
 * This method was created by a SmartGuide.
 * @return
 */
public int getOrderNumber( ) {
      return _worknumber;
}
/**
 * This method was created by a SmartGuide.
 * @return int
 */
public int getQty( ) {
      return _qty;
}
/**
 * This method was created by a SmartGuide.
 * @return int * @param newqty int
 */
public int setQty( int newqty ) {
      _qty = newqty;
      return _qty;
}
/**
 * This method was created by a SmartGuide.
 * @return java.lang.String
 */
public String toString( ) {
      //return ( _dateissued + " - Order# " +
Integer.toString(_worknumber) + _desc + " for " + Integer.toString(_qty) );
      return ( "Order#" + _worknumber + " " + _desc + " qty " + _qty ); }
}
```

Visual.java

```
package Thesis.THSC;
/**
 * This class was generated by a SmartGuide.
 */
public class Visual extends java.awt.Frame implements
java.awt.event.ActionListener, java.awt.event.WindowListener {
      public java.awt.Button ivjAddCost = null;
      public java.awt.Button ivjAddPart = null;
      public java.awt.Button ivjAddWO = null;
      public java.awt.TextField ivjCostAmt = null;
      public java.awt.List ivjCostList = null;
      public java.awt.TextField ivjCostType = null;
      public java.awt.Button ivjEXIT = null;
      public java.awt.Label ivjlCosts = null;
      public java.awt.Label ivjlMaterialList = null;
      public java.awt.Label ivjlParts = null;
      public java.awt.Label ivjlWOdesc = null;
      public java.awt.Label ivjlWOdesc1 = null;
      public java.awt.Label ivjlWOdesc2 = null;
      public java.awt.Label ivjlWOdesc21 = null;
      public java.awt.Label ivjlWOdesc211 = null;
      public java.awt.Label ivjlWOloc = null;
      public java.awt.Label ivjlWOloc1 = null;
      public java.awt.Label ivjlWOloc2 = null;
      public java.awt.Label ivjlWOqty = null;
      public java.awt.Label ivjlWorkOrder = null;
      public java.awt.List ivjMaterialListList = null;
      public java.awt.Panel ivjNewCostPanel = null;
      public java.awt.Panel ivjNewMaterialsPanel = null;
      public java.awt.FlowLayout ivjNewMaterialsPanelFlowLayout = null;
      public java.awt.Panel ivjNewWOPanel = null;
      public java.awt.Panel ivjPanel3 = null;
      public java.awt.Panel ivjPanel4 = null;
      public java.awt.Panel ivjPanel5 = null;
      public java.awt.Panel ivjPanel6 = null;
      public java.awt.Panel ivjPanel7 = null;
      public java.awt.Panel ivjPanel71 = null;
      public java.awt.Panel ivjPanel72 = null;
      public java.awt.Panel ivjPanel73 = null;
      public java.awt.Panel ivjPanel8 = null;
      public java.awt.Panel ivjPanel81 = null;
      public java.awt.TextField ivjPartMfg = null;
      public java.awt.TextField ivjPartName = null;
      public java.awt.TextField ivjPartQty = null;
      public java.awt.TextField ivjPartSku = null;
      public java.awt.TextField ivjPartsText = null;
      public java.awt.TextField ivjWODesc = null;
      public java.awt.TextField ivjWOLoc = null;
      public java.awt.TextField ivjWOQty = null;
      public java.awt.List ivjWorkOrderList = null;
/**
 * Constructor
 */
```

```
/* WARNING: THIS METHOD WILL BE REGENERATED.
 */
public Visual() {
      super();
      initialize(); }
/**
 * Visual constructor comment.
 * @param title java.lang.String
 */
public Visual(String title) {
      super(title);
}
/**
 * Method to handle events for the ActionListener interface.
 * @param e java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
public void actionPerformed(java.awt.event.ActionEvent e) {
      // user code begin {1}
      // user code end
      if ((e.getSource() == getEXIT()) ) {
            conn7(e);
      }
      // user code begin {2}
      // user code end
}
/**
 * conn0:  (Visual.window.windowClosing(java.awt.event.WindowEvent) -->
Visual.dispose())
 * @param arg1 java.awt.event.WindowEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void conn0(java.awt.event.WindowEvent arg1) {
      try {
            // user code begin {1}
            // user code end
            this.dispose();
            // user code begin {2}
            // user code end
      } catch (java.lang.Throwable ivjExc) {
            // user code begin {3}
            // user code end
            handleException(ivjExc);
      }
}
/** * conn7:  (EXIT.action.actionPerformed(java.awt.event.ActionEvent) -->
Visual.dispose())
 * @param arg1 java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void conn7(java.awt.event.ActionEvent arg1) {
      try {
            // user code begin {1}
            // user code end
            this.dispose();
            // user code begin {2}
```

```
				// user code end
			} catch (java.lang.Throwable ivjExc) {
				// user code begin {3}
				// user code end
				handleException(ivjExc);
			}
}
/**
 * Return the AddCost property value.
 * @return java.awt.Button
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private java.awt.Button getAddCost() {
	if (ivjAddCost == null) {
		try {
			ivjAddCost = new java.awt.Button();
			ivjAddCost.setName("AddCost");
			ivjAddCost.setLabel("Add");
			// user code begin {1}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {2}
			// user code end
			handleException(ivjExc);
		}
	};
	return ivjAddCost;
}

/**
 * Return the AddPart property value.
 * @return java.awt.Button
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private java.awt.Button getAddPart() {
	if (ivjAddPart == null) {
		try {
			ivjAddPart = new java.awt.Button();
			ivjAddPart.setName("AddPart");
			ivjAddPart.setLabel("Add");
			// user code begin {1}
			// user code end
		} catch (java.lang.Throwable ivjExc) {
			// user code begin {2}
			// user code end
			handleException(ivjExc);
		}
	};
	return ivjAddPart;
}
/**
 * Return the AddWO property value.
 * @return java.awt.Button
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
...
```

ThscCost.java

```
package Thesis.THSC;
/**
 * This class was generated by a SmartGuide.
 */
public class ThscCost {
      private String _type;
      private String _phase;
      private double _perUnitCost = 0.0;
      private String _units;
/**
 * This method was created by a SmartGuide.
 * @param type java.lang.String
 * @param totalCost float
 */
public ThscCost (String type, double totCost ) {
      _type = type;
      _perUnitCost = totCost;
}
/**
 * This method was created by a SmartGuide.
 * @param type java.lang.String
 * @param phase java.lang.String
 * @param totCost float
 */
public ThscCost ( String type, String phase, float totCost) {
      _type = type;
      _phase = phase;
      _perUnitCost = totCost;
}
/**
 * This method was created by a SmartGuide.
 * @return java.lang.String
 */
public String equals ( ) {
      return ( _type + _phase + _perUnitCost );
}
/**
 * This method was created by a SmartGuide.
 * @return double
 */
public double getCost ( ) {
      return _perUnitCost;
}
/**
 * This method was created by a SmartGuide.
 */
public String getPhase( ) {
      return _phase;
}
/**
 * This method was created by a SmartGuide.
 * @return java.lang.String
 */
public String getType( ) {
      return _type;
}
/**
 * This method was created by a SmartGuide.
 * @return java.lang.String
 */
public String toString ( ) {
      return ( _type + _phase + _perUnitCost );
}
/**
 * This method was created by a SmartGuide.
 * @param newcost double
 */
public void updateCost(double newcost) {
      _perUnitCost = newcost;
      return;
}
}
```

ThscPart.java

```
package Thesis.THSC;
import COM.objectspace.jgl.*;
/**
 * This class was generated by a SmartGuide.
 *
 */
 public class ThscPart {
      public String _SKU;
      private String _name;
      private String _mfg;
      private HashSet _costSet;
/**
 * This method was created by a SmartGuide.
 */
public ThscPart ( String sku, String name, String mfg ) {
      _SKU = sku;
      _name = name;
      _mfg = mfg;
      _costSet = new HashSet( false );
}
/**
 * This method was created by a SmartGuide.
 * @return Thesis.THSC.ThscCost
 * @param cost Thesis.THSC.ThscCost
 * returns cost if successful, null otherwise
 */
public void addCost( ThscCost cost ) {
      _costSet.add( cost );
      return;
}
/**
 * Returns a String that represents the value of this object.
 */
public String equals() {
      return ( _SKU + _name );
}
/**
 * This method was created by a SmartGuide.
 * @return COM.objectspace.jgl.HashSetIterator
 */
public HashSetIterator getCosts() {
      return _costSet.begin();
}
/**
 * This method was created by a SmartGuide.
 * @return com.sun.java.util.collections.ListIterator
 */
public double getTotalCost() {
      HashSetIterator itr = _costSet.begin();
      double totcost = 0.0;
      ThscCost cost;

      while ( itr.hasMoreElements() ) {
            cost = (ThscCost)itr.get();
            totcost += cost.getCost();
            itr.advance( 1 );
      }

      return totcost;
}
/**
 * Returns a String that represents the value of this object.
 */
public String toString() {
      return ( _SKU + _name );
}
/**
 * This method was created by a SmartGuide.
 * @return Thesis.THSC.ThscCost
 * @param cost Thesis.THSC.ThscCost
 */
public ThscCost updateCost( ThscCost cost ) {
      _costSet.add( cost );

      return cost; } }
```