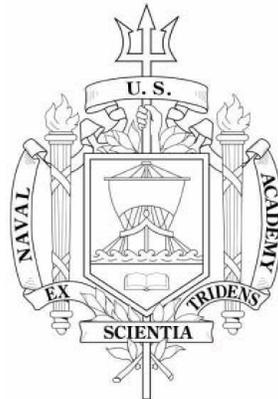


U.S. NAVAL ACADEMY
COMPUTER SCIENCE DEPARTMENT
TECHNICAL REPORT



Establishing Correspondence among Shared Information and Tasks

Childers, Candace M.

USNA-CS-TR-2005-06

June 7, 2005

Establishing Correspondence
Among Shared Information and Tasks

Candace Childers

29APR05

SI496 Spring 2005

Research Advisor: CAPT Young

MIDN Childers

CAPT Young

Professor Schultz

ABSTRACT

Creating interoperability among heterogeneous systems enhances our military's war-fighting capabilities. Differences in hardware, languages, and data models make interoperability hard to achieve. The Object-Oriented Method for Interoperability (OOMI) resolves modeling differences among systems through construction of a Federation Interoperability Object Model (FIOM) used to capture information and tasks shared among systems. The FIOM is constructed in either a bottom-up or top-down fashion using the OOMI Integrated Development Environment (OOMI IDE) and includes both component system and standard representations of the shared tasks and information.

When constructing a federation of interoperable systems, a correspondence must first be established among shared tasks and information before data modeling differences can be resolved. The OOMI IDE uses both semantic and syntactic correlation methodologies for establishing such correspondences. Syntactic correlation is performed using neural networks. Syntactic data concerning the structure and signature of shared information and tasks is used to create discriminator vectors for objects being compared. Neural Networks are used to compare these discriminator vectors to determine the degree of similarity among objects. A ranking of the scores returned from the neural network comparison is used to assist an interoperability engineer in identifying corresponding objects for which modeling differences can be resolved.

I. INTRODUCTION

A. NEED FOR SYSTEM INTEROPERABILITY

Computer technology is rapidly changing. New hardware and software are constantly being created in the fast paced computer world. The Department of Defense (DoD) is continually buying new computer equipment and systems in order to extend its capabilities. As technology has grown so has the awareness of how powerful computers that share functionality can be. Communications technology has driven this new recognition of power. Today's computers have the capability to send and receive data through networks. This new awareness is driving the need to find a way to create interoperability among systems with different hardware and software components that were not created with interoperability in mind. Interoperability is the ability for two systems to communicate, share information, and perform tasks concurrently.

B. TYPES OF INTEROPERABILITY

Achieving interoperability among independently developed systems is complicated due to the heterogeneities that exist among the systems. These heterogeneities can be classified into two main categories: differences in what is modeled, termed differences in view, and differences in how the information is modeled, termed differences in representation. [Young02].

Differences in what is modeled include heterogeneity of scope, level of abstraction, and temporal validity [Wied 93]. Heterogeneity of scope refers to the differences in the number and type of attributes and behaviors used to model an object. For example, System A, a traffic simulation system, may require a car's velocity, rate of acceleration/deceleration, and turn radius when modeling traffic flow. On the other hand, System B, a crash simulation system, may require a car's rate of acceleration, structure, and vehicle weight for modeling a vehicle's crash performance. If we want to share information and behaviors between the systems such as a traffic simulation where crash damage is accepted, then these differences must be resolved.

Level of abstraction refers to the degree or level to which a system aggregates component attributes in describing an object. For example, one model may refer to gross sales as the amount of sales in a year while another model may refer to gross sales as the amount of sales in a month. Level of Abstraction differences are difficult to recognize by just looking at the name or type of an attribute.

Heterogeneity of temporal validity refers to differences in the time frame used by two different models or differences of when data remains valid in different systems. Differences in temporal validity are common in military systems. In System A the deceleration may be measured in miles per hour; whereas, System B may require a more precise measure of deceleration in ft per minute. [Young02]

Heterogeneity of hardware and operating systems, structure, presentation and meaning are all differences in the way data is represented. Heterogeneity of hardware and operating Systems are a result of systems being developed at different times and with different technologies. Different hardware and operating systems can result in differences in the way data is stored, managed, and shared. Structure refers to the way the data of a model is arranged. Presentation refers to the units or domain of values used to represent data. Heterogeneity of presentation refers to differences in the units of measurement used in two attributes that measure the same condition. For example, System A may measure the speed of the car in miles per hour and System B may measure the speed in kilometers per hour. Meaning refers to the semantic definition of certain attributes or objects. Differences can arise when synonyms or homonyms are used for defining an object's attributes or behaviors. System A and B may have an attribute size but in System A size may refer to length of the car and in System B it may refer to the classification of the car (ie. Sedan, compact, etc.). [Young02]

These differences between systems create many obstacles that must be resolved before systems can interoperate. Differences in scope, level of abstraction, and temporal validity are the most difficult to reconcile. Information required by one system may not be available from another system that it wishes to interoperate with. A means must be found to provide such required information, or the desired interoperation cannot occur. Such means can include the use of default or derived information, or in worst case may require a modification of the source system to provide such information. Differences in

hardware and operating systems, structure, presentation, and meaning make it difficult to decide which objects in one system correspond to objects in another system. Once correspondences have been found, differences in hardware and operating systems, structure, presentation, and meaning must be resolved before data can be shared effectively. Two completely separate systems must be able to understand each other in order to communicate and to perform tasks cooperatively. System interoperability is the desired outcome; however, achieving true interoperability is not easy.

3. OBJECT ORIENTED METHOD FOR INTEROPERABILITY

Young's Object Oriented Method for Interoperability (OOMI) uses a combined model formulae, and ontology based approach to resolve many of the differences among systems so that they may interoperate with each other. The objective of the OOMI is to enable heterogeneous systems to exchange information and to utilize the tools and services of each other. The OOMI resolves differences among heterogeneous systems. By providing a common model of the information shared among systems in order to identify correspondences and resolve differences among shared information. This common model, the Federation Interoperability Object Model (FIOM), captures information to be shared among systems for each federation component. The FIOM provides a basis for resolving differences among components by providing a standard model for the shared information. A Federation Entity (FE) represents a real world entity whose information is shared among systems. The FE is an abstract view of the real world entity that hides details of how the data is modeled. Each FE is composed of several Federation Entity View's (FEV's) which are used to capture differences in view, ie. scope, level of abstraction, and temporal validity within the different systems. A Federation Class Representation (FCR) is then created to provide a standard representation of each FEV. The presence of a standard representation reduces the amount of translation that must be done in order for systems to interoperate. The FCR becomes the intermediate step in translation. Corresponding objects on each side of the translation are resolved to match the FCR and then resolved to match the object of the system that it is trying to interoperate with.

The FIOM is created prior to runtime through the use of the OOMI Integrated Development Environment (OOMI IDE). At runtime the OOMI translator resolves differences among heterogeneous systems using the FIOM created before runtime. Information to be shared by a system is first converted from its component system representation to the corresponding FCR by translations defined in the FIOM. The destination system uses translations in the FIOM to convert the FCR into an object that it can accept. [Young02] The OOMI IDE provides a GUI that allows the user to create an FIOM, correlate information to be shared among component systems, and create translations between the heterogeneous data representation.

4. CORRELATOR

The OOMI contains a component module correlator that is used to identify corresponding objects in two heterogeneous systems whose information and operations are desired to be shared. This is the first step in building an FIOM of information and operations to be shared by a federation component. Correlation is done by comparing Component Class Representations (CCR) with a number of previously created FCRs to find an FCR with corresponding view. A correlation between a CCR and a FCR must be found before translators for resolving differences in representation can be formed. The goal of the OOMI is to provide aid to the correlation process. Two types of correlation are used by the OOMI. First a semantic key word comparison is run and then a syntactic correlation is done through the use of neural networks. XML schemas are used to capture syntactic and semantic information for each FCR and CCR to aid the correlation process.

II. TYPES OF CORRELATION

Before heterogeneities between two systems can be resolved, correspondences must be made between the two systems. Finding two matching entities is the first step to resolving differences between two systems. There are two types of correlation techniques that can be used to find entities that represent the same object. The effectiveness of a correlator can be measured by its precision and its recall. [Young 02] Precision is a ratio of correct matches over the total number of matches returned. A high precision indicates that the correlator is able to reduce the work of the engineer by returning relevant matches without returning a lot of irrelevant matches. High Precision indicates that the search has been narrowed down but it does not guarantee that all of the relevant matches have been returned. Recall is the ratio of correct matches returned over the total number of correct matches that exist. A high recall indicates that the correlator is able to find the relevant matches, but it does not guarantee that the matches returned will be much less than the original bank of entities being compared. A high recall and precision are needed to ensure that the correct correspondences are being made and that the original problem of finding correspondences is being reduced.

A. Semantic Correlation

Semantic correlation is the process of using the behavior described by an attribute to find a corresponding behavior of the same type in a different entity. [LiCl 94] Semantic correlation is very difficult to implement because it is difficult to pull meaning out of data. However, semantic correlation provides the best correlation technique in terms of precision and recall. It provides the best result because ultimately what we are trying to find are entities in one system that behave similarly to an entity in another system. There are several techniques for implementing semantic correlation including keyword matching, specification matching, and the Semantic Web Approach.

Keyword matching is the easiest and most commonly implemented technique. When entities are created their attributes and behaviors are assigned keywords. In our example system of the car, the behavior of how fast the car is moving down the road is described by the keyword *speed*. The keywords of a component would be compared to a database list of keywords for matches. Once matches were found, an interoperability engineer would compare the results to ensure the two components corresponded. The keyword

approach is not very effective for several reasons. The first is that component attributes and behaviors are not always given names that reflect the actual behavior or characteristic being described. If the librarian choosing the keywords for the attributes and behaviors does not have a good understanding of the behavior the component is modeling, then keywords will most likely be chosen poorly. For example, if a person is tasked with coding a vehicle entity and they are given a data set that includes a number value of 14 feet they may choose a keyword of length when the data may actually be referring to width. When this component is run through the keyword correlator, then another attribute of speed will come up as a match when in fact the two attributes represent different behaviors. In a keyword matching correlator, problems arise when the database of keywords to compare to is too large or too small. If the keyword database is large then the number of returns will be higher; therefore, the precision ratio will be lower. If the database of keywords is small then the number of corresponding matches found will be smaller; therefore, decreasing the recall ratio of the search. Facet classification helps to increase the effectiveness of keyword searches. [Prieto-Diaz 90] Facet classification helps to increase the searching aspect of keyword matching by creating a classification scheme to choose keywords from. When a librarian chooses a keyword for an attribute they use the categories created by the facet classification system to better choose appropriate keywords. Even with the faceted approach keywords still fall short of achieving high levels of precision and recall. The keyword approach is best used in conjunction with other correlation techniques. The Data Element Tool Based Analysis (DELTA) correlator takes that approach. [BFHW 95] It uses a keyword search to reduce the size of the database being searched and then uses other methods to find correspondences.

Specification matching is another approach to semantic correlation. [ZW 95] Specification matching is an extension to signature matching, a syntactical approach proposed by Zaremski and Wing. Specification matching is an improvement on signature matching because it is based on a components behavior and types, and signature matching relies only on the data types of the parameters. Specification matching is useful in determining whether two components lack a semantic match to one another. Specification matching refers to two types of matching: function matching and module

matching. A post-condition and pre-condition specification is set for each function. The specifications for different functions are compared to determine if a semantic correspondence is present. A module is a set of related functions. In module matching the number of function specifications and each function specification in a module are compared to the number of function specifications and each function specification in another module to determine if a match exists. In a Specification Matching correlator the user has the ability to set the thresholds for a match. The user could choose to list all of the functions that have a partial match to a particular function or the user could choose to only allow exact matches to be displayed. Specification matching is a theory that has not been implemented into a working correlator. It is difficult to implement a correlator that can evaluate the pre and post conditions of a component to gather semantic information.

The newest semantic correlation approach is the use of the Semantic Web. [DeMe 00] The Semantic web uses Extensible Markup Language (XML) schemas and Resource Description Framework (RDF) schemes combined with an ontology based approach to extract semantic data about components. Components can be modeled by an XML or RDF document and the structure of the document can lead to clues about the meaning and behavior of the attributes and functions. Extracting semantic information is difficult when looking at XML schemas because the XML does not provide a common translation for the data in the document. In order for XML to be useful for correlation, both systems must know the details of the Document Type Definition (DTD) file. XML must be translated using XSL Transformations before information can be exchanged and this is based on both systems knowing the details of the DTD. XML is more useful for syntactic correlation because it lays out the types and domains of attributes in an easy to parse document. RDF is a much more useful tool for the Semantic Web approach to correlation. RDF provides built in terms to describe the hierarchy of classes which allows ontology represented languages to be described by RDF files. RDF provides a object-attribute structure which displays each object as an independent entity and provides a semantic layout of the data and relationships. RDF schemas have been enriched to allow them to describe more complicated models of data which have made them a more attractive data representation for the Semantic Web. XML and RDF files have semantic

meaning that is portrayed through their structure and the Semantic Web is a computer process that extracts this semantic meaning.

Semantic correlation requires the extraction of meaning from text and data which can be difficult. Although keyword search is the easiest type of semantic correlation to implement, the Semantic Web offers more opportunity for extracting valuable semantic information from a textual description of a component.

B. Syntactic Correlation

Syntactic correlation relies on the matching of components based on the structure of an object. For example, it compares an object's structure, data types, and an attribute's domain. If System A has an attribute speed that is an integer type and System B has an attribute weight that is an integer then they share the same type; therefore, they share a syntactic correlation. Syntactic correlation does not provide the high recall and precision provided by semantic correlation. Syntactic correlation fails to discover the true meaning of components and attributes; therefore, it has disadvantages in finding a true match. However, Syntactic correlation is much easier to implement, so it makes up most of the current correlators.

One approach to syntactic correlation, signature matching is a correlation technique based on comparing the signatures for a component's functions or comparing the component's interfaces. [ZW 93] Function matching and module matching are the two types of signature matching. In function matching the signature includes the input and output parameters of a particular function. A query is performed to find function signatures with the same types in order to find a correspondence. The user can set parameters on what constitutes a match. In some instances the user may want to see all the components that partially match an input instead of just seeing the exact matches. This allows the user to control the precision and recall of the correlation. Asking for exact matches will raise the precision of the query but lowering the threshold for a match will increase the recall for a query.

Another approach to signature matching is comparing the types and domains of different attributes of a component. A component can be modeled by an XML file and then the information needed can be parsed from that file. XML documents contain

information about type, attribute name, and the range and domain of the values allowed for that attribute. This information can be compared to the same information contained in another component's XML document representation. There are many ways to make the comparisons between the data, Semint is one approach that uses Neural Networks to identify the corresponding elements [WiCl 00]. Semint was developed to find corresponding components in heterogeneous databases. Corresponding components often have similar designs, constraints, and values for attributes. In the Semint process this information is extracted and put into a common form and then normalized. The normalized information is then used to train a Neural Network on what to look for in corresponding components. Once trained the Neural Network can compare any normalized information to the information it has already received and determine whether the information corresponds.

Syntactic correlation does not always produce corresponding elements, but when used in conjunction with semantic correlation it can greatly reduce the work of the interoperability engineer.

III. OOMI CORRELATOR

The OOMI correlator uses a two phase system for finding correspondences which seeks to increase precision and recall in each phase. [Young 02] The OOMI combines the keyword search method and the Neural Network syntactic approach to find correspondences among information shared by federation components. The OOMI IDE allows the user to control threshold values for the matches returned. If the user increases the threshold level then only closely matching or exact matches will be returned. The overall number of returns will be reduced which increases the precision of the correlation. If the user decreases the threshold then more matches will be returned but they will not all be close matches. This will increase the recall of the correlation.

A. Semantic Key Word Correlation

The first phase of the OOMI Correlator is a semantic keyword search. [Young 02] In the OOMI, information to be shared by a component is represented by XML schemas. The XML schema is parsed to retrieve keywords contained in the name and description of the corresponding object's attributes and operations. The first step of the keyword

correlator is to extract this keyword information from the (CCR) XML schema. The keyword information extracted from the (CCR) XML schema is compared to a database of keywords stored for each previously registered Federation Class Representation (FCR) schema. Once the comparison is made a list of matches within the threshold set by the interoperability engineer is returned. The interoperability engineer then selects the matches that he wants to put through the next phase of the correlator which is a syntactic correlator.

B. Syntactic Correlation By Neural Networks

Syntactic information can be extracted from XML schema documents in addition to the keyword semantic information. The syntactic information collected from the FCR schemas is used to train a Neural Network. Then the syntactic information obtained from the CCR schema is run through the Neural Network to check for a correspondence.

[Young 02]

The first step in the Neural Network approach to correlation is collecting syntactic information from the XML documents. The OOMI IDE looks for data element structure, data element type, frequency of occurrence, data size specifications, and data value constraints. All of this information is contained within the XML schema created to define the information shared among systems. Data element structure labels an attribute as *isComplex* if the attribute contains subtypes. Type refers to whether the attribute is a *string*, *int*, *Boolean*, etc. The frequency of occurrence determines how often an attribute can be found in a real world entity. Data size constraints place a range on the possible domain of a number or a limit on the length of a string. [Young 02] All of this information is used for comparing two components to find a correspondence.

When a new FCR is created the Java Document Object Model (JDOM) is used to parse the schema and extract the information that can be used to find a correlation with a CCR. [Shedd 02] The information gathered from the JDOM is used to create a discriminator vector for each attribute and operation for the FCR. The discriminator vectors will then be used to train a backpropagation neural network. The OOMI Correlator uses 28 different discriminator values to make up a discriminator vector for the backpropagation neural network. Table I lists the different discriminator values and

the input value to be placed in the vector for the information found from the XML document about each attribute or operation.

Table I. Discriminator Values [Shedd 02]

Number	Discriminator	Value to Vector
Structural Information		
1	propertyType	Operation or Attribute
2	isComplex	Describes whether an attribute is complex or atomic
3	numSubtypes	If attribute is complex, number of subtypes
4	numReqdSubtypes	If attribute is complex, number of required subtypes
5	numOptSubtypes	If attribute is complex, number or optional subtypes
6	numOperations	For complex attribute – total no. of operations defined for type
7	numParameters	For operation-number of parameters For complex attribute- sum of parameters for all operations
Type Specifications		
8	string type	If atomic attribute – < 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0 > If complex attribute or operation – Value normalized to [0.0, 1.0]
9	boolean type	If atomic attribute – < 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0]
10	float type	If atomic attribute – < 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0]
11	double type	If atomic attribute – < 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0]
12	bigDecimal type	If atomic attribute – < 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0]
13	int type	If atomic attribute – < 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0]
14	long type	If atomic attribute – < 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0]
15	short type	If atomic attribute – < 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0]
16	other type	If atomic attribute – < 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0]
Frequency of Occurrence		
17	minOccurs	Minimum number of times attribute must occur in class modeling real world identity
18	maxOccurs	Maximum number of times attribute may occur in class modeling real world identity
Data Size Specification		
19	minLength	For String Type – minimum length of string ¹
20	maxLength	For string Type – maximum length of string

21	totalDigits	Total number of digits in an attribute
22	fractionDigits	Number of digits in fraction part of attribute

Data Value Constraints

23	pattern	Restriction to values allowed for string and numeric types
24	numEnumerations	Number of enumeration values for attribute ¹
25	minExclusive	Lower open bound of interval defined for numeric attribute types
26	maxExclusive	Upper open bound of interval defined for numeric attribute types
27	minInclusive	Lower closed bound of interval defined for numeric attribute types
28	maxInclusive	Upper closed bound of interval defined for numeric attribute types

For attributes all of the discriminators are used except for the *numParameters* discriminator. For complex attributes the *numSubTypes* discriminator indicates how many elements the attribute is composed of. If an attribute was composed of 2 subtypes then the value of the *numSubType* discriminator would be 2 until normalized. Table 1 describes what the different discriminators represent. All input to the neural network must be between the value of 0.0 and 1.0. Each attribute's data type requires a vector of binary values instead of a single value in the range [0.0, 1.0]. If a data type is just given a single value then the neural network would consider some types to be closer to each other when considering a correspondence. [Shedd 02] For example, if a string was assigned the value 0.4, an integer was a value of 0.5, and a Boolean was assigned the value 0.6 then the neural network would consider an integer and a string a closer match than a string and a Boolean. To avoid an incorrect perception of closeness between data types each data type is given a unique sub vector. These sub vectors are listed in Table 1 under the Type Specifications section. After discriminator vectors are formed, they are stored with their respective CCR or FCR components.

Each FCR has its own trained neural network as one of its syntactic components. [Shedd 02] The first step in training the neural networks is to determine what output vectors should be displayed for each attribute in the FCR if a match is found. The length of the vector is equal to the number of attributes in the FCR.

If there are three attributes in a particular FCR then the vectors would look like this:

Attribute 1 <1.0, 0.0, 0.0>

Attribute 2 <0.0, 1.0, 0.0>

Attribute 3 <0.0, 0.0, 1.0>

The 1.0 is in a different place for each attribute so that each vector is unique which allows the interoperability engineer to see which attributes of a CCR match up to which attributes of a FCR when vector results are displayed.

Next a three-layer backpropagation neural network is created. The three layers are input layers, hidden layer, and an output layer of nodes. [Shedd 02] The number of input nodes is equal to 28 because 28 discriminators will be entered into the neural network in vector form. The output layer nodes ensures that there is a node for each attribute of the FCR so the number of output layer nodes is equal to the length of the particular vectors for the FCR attributes (shown above). The number of hidden nodes is the floor of the average number of input and output nodes. For example, if an FCR has 4 attributes and the discriminators of the CCR vectors are of length 28 then there will be:

Input Layer: Nodes = Number of Discriminators = 28

Output Layer: Nodes = Length of FCR Attribute vectors = 4

Hidden Layer: Nodes = Floor[(Input + Output)/2] = 16

Once the network has been created it must be trained to find correspondences between the FCR it represents and CCRs that are put into the trained neural network as input. Training is accomplished through a learning algorithm. To train the network each FCR attribute vector is forward propagated through the network individually. For each vector an error is calculated. The error represents the difference between the actual output vector and the desired output vector that was determined above. The error is used to adjust the weight matrices of the network and then the process is repeated until the error is within the tolerance value that has been set by the interoperability engineer. This process is done for each FCR attribute vector. Once the network is trained it is saved with the FCR as a syntactical component.

Once the interoperability engineer has run the CCR through the semantic keyword correlator and come up with a list of possible matches he can use the syntactic correlator to try and find a one-to-one correspondence with an existing FCR. The interoperability engineer must first set the threshold setting on the syntactic correlator. The threshold setting determines at what point a match will be accepted or rejected. If the threshold is high then only close matches will be displayed; however, if the threshold is lowered then matches that are not quite as close will be displayed. Next the interoperability engineer must select which FCR component neural networks the CCR discriminator vector will be run through for correspondence. The semantic correlator gives the engineer an idea of which FCR's may be a match; therefore, helping him with his decision of which FCR's to run through the correlator for comparison.

The next step is to take the discriminator vectors stored with the CCR and perform a syntactic evaluation against each FCR neural network component. Once all of the evaluations have been run then a list of scores is displayed in the OOMI IDE for the interoperability engineer. All of the scores that fall above the set threshold setting are displayed with the higher scores displayed first. The output of the syntactical correlator is an output vector with the same number of elements as the FCR attribute output vector. The closer the numbers in the actual output vector are to one the closer they are to a match for the attribute corresponding to that spot in the output vector. For example, if the trained FCR attribute output vectors are considered and a neural network for an FCR produces an output of $\langle 0.00560, 0.8999, 0.2309 \rangle$ for a given CCR, then the input CCR discriminator vector most closely corresponds to the second FCR attribute because 0.8999 is the highest returned value.

The interoperability engineer can adjust the threshold settings if he does not immediately find a correlation or if too many matches are returned. This allows the interoperability engineer to adjust the precision and recall of the correlator. The OOMI correlator is not fully computer automated. It requires an interoperability engineer to review the results and adjust the settings.

III. INTEGRATING THE CORRELATOR

A. Semantic Correlator Integration

LT Shedd created a stand alone correlator and LT Lawler began the integration of that correlator into the OOMI. However, the integration into the OOMI was never completed. The goal of this project was to finish the integration of the semantic and syntactic correlators and to analyze their effectiveness. A step by step debugging and fixing methodology was used to complete this goal.

The first problem we encountered was a simple math error. After each semantic or syntactic score is calculated it must be compared to the threshold value that is set by the user in the Graphical User Interface (GUI). The threshold value is expressed as a integer multiple of ten ranging from zero to 100. The calculated scores of the correlation are expressed as values from 0.00 to 1.00. The calculated score was being directly compared to the threshold value without being converted to the higher range which resulted in the semantic score never being larger than the threshold; therefore, no results were returned in the filtered array and displayed. The solution to this problem was to multiply the semantic score by 100 in the comparison loop.

The next problem was a much larger problem that stemmed from the reasoning behind the correlation algorithm. LT Shedd stored the semantic, syntactic, and combined scores in different arrays and then passed all three arrays to the correlator panel for display. This posed three difficulties that needed to be overcome. The first difficulty arose when trying to correlate the semantic and syntactic results. The results for each FCR are displayed in a row in the table. The semantic and syntactic scores for a particular FCR are displayed next to each other. Since the two scores were stored in different arrays that were sorted in a different order, a way was needed to display the correct scores with each FCR Name. The next problem was a null pointer exception. The semantic results are calculated and displayed in the table before the syntactic scores. This means that when the semantic scores are passed to the correlator panel the syntactic score array is null. The table does not know what to print as the syntactic scores. The third problem was an efficiency problem. The correlator is set up so that the user can run the correlation several times with different threshold values. Every time the user chooses a

new threshold value the scores are recomputed for each CCR. This poses a problem of redundancy and increases the runtime of the correlator especially in the case of the syntactic correlator where neural networks are created for each comparison. This problem arose from the lack of a global storage structure. Figure 3.1 describes LT Shedd's correlation algorithm.

The best solution to the problem was to create a global vector to store the results the in as they were computed and to create new methods filtering, display, and sorting methods. The approach to computing the scores did not change at all. The new method diagram for the updated algorithm is shown in figure 3.2. The first step was to create a data structure that could hold all the scores for a particular FCR in one instance. This class was called TableData. A TableData element has the following data members: FCR_Name, semanticScore, syntacticScore, combinedScore; and methods to set and get each of the data members. A TableData element can hold all of the needed information for a FCR in one place. This solves the problem of having the scores stored separately and needing to correlate the results before displaying them. The next step was to create a global vector to hold all of the TableData elements as they were created. This removes the need to recompute scores each time the correlation is performed. By making the vector global we allow the semantic and syntactic processes to access and add scores to the vector. Several methods were removed and added to the ComponentModelCorrelator.java and CorrelatorPanel.java source files in order to implement this new algorithm. The methods are described in section C and highlighted in the source code found in appendix A.

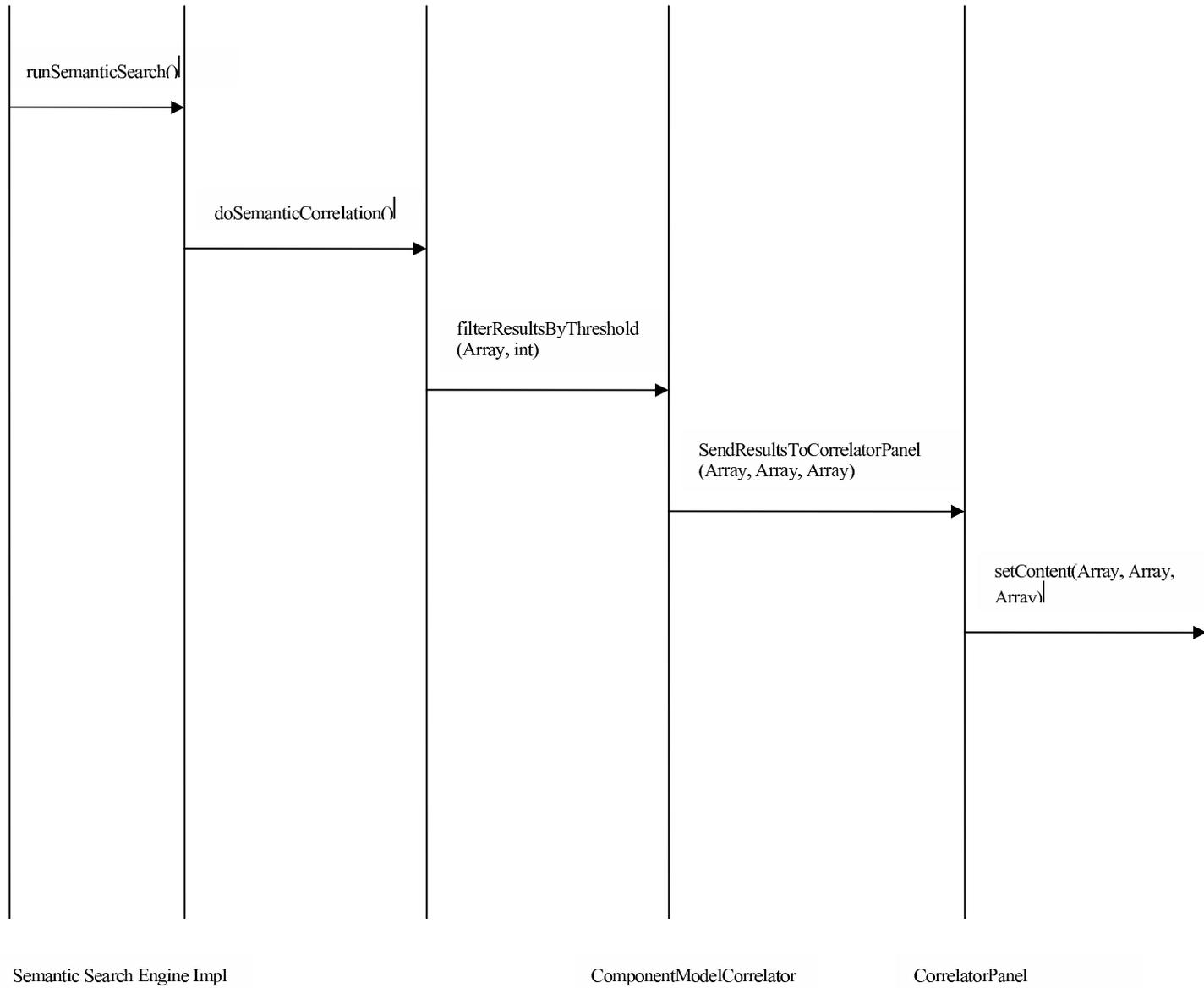


Figure 3.1- Original Correlator Code Diagram

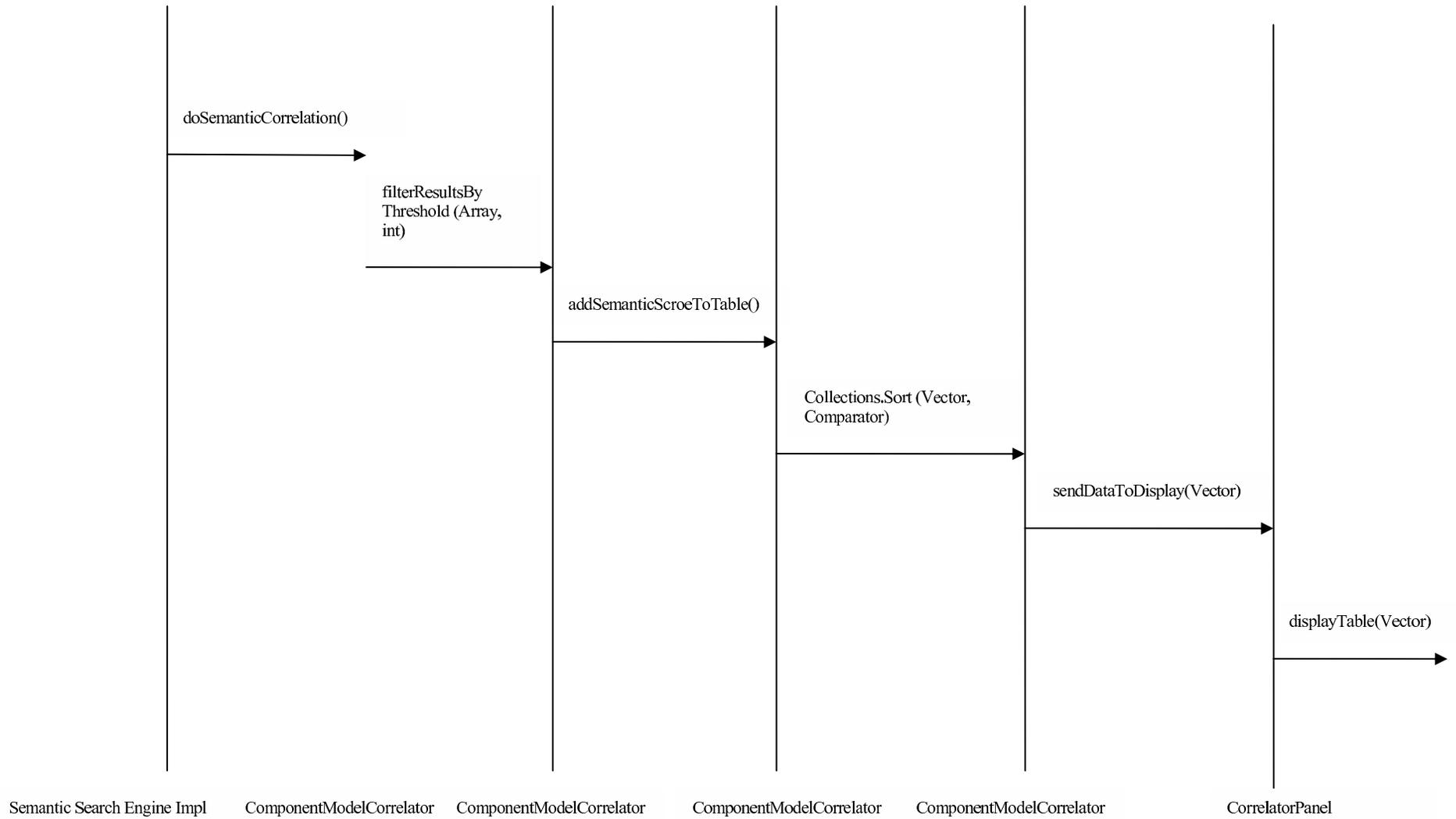


Figure 3.2 – Modified Correlator Code Diagram

B. Syntactic Correlator Integration

The changes made to the syntactic correlator were similar to the changes made to the semantic correlator. When the syntactic scores are calculated they are added to the global vector `TableDataVector`. This is very important in the syntactic correlation. The overhead for computing syntactic scores is much higher than calculating semantic scores because of the overhead associated with creating a neural network. The neural network must be recreated every time a correlation is done unless the score is already stored somewhere. In the new algorithm each time the compute score method is called, the program first checks to see if the score is already stored in `TableDataVector` and if it is then the score is not recomputed. This decreases the runtime of the program.

Before syntactic scores are computed the `syntacticScore` data member of the `TableData` element is set to zero as a default. Once syntactic scores are computed they must be stored in the correct `TableData` element inside the `TableDataVector` vector. The semantic and syntactic scores associated with each `TableData` element should correspond to the same `FCR_Name`. This is accomplished through a method that compares the `FCR_Name` associated with the syntactic score to the `FCR_Names` of the `TableData` elements already stored. The index number of the corresponding element is returned and the syntactic score is stored in that element. Since all of the results are stored in a global vector, only that vector needs to be passed to the `correlatorPanel` for display.

The syntactic correlator integration is not complete. Work is being continued to resolve conflicts in already completed code.

C. New Classes and Methods

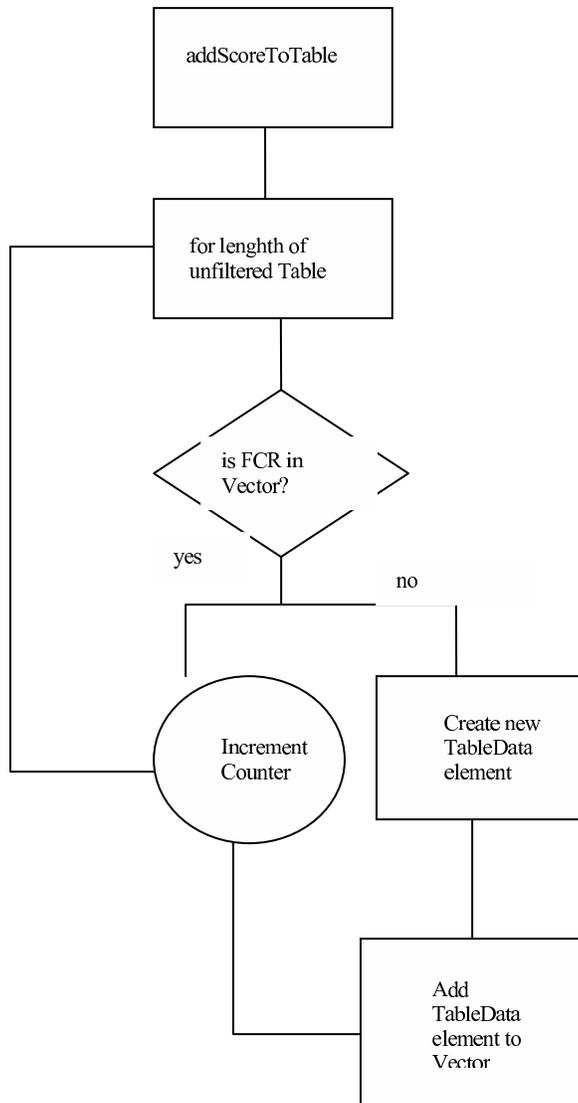
The following section explains the function of the new methods that were added to enhance the performance of the Semantic and Syntactic Correlator. Refer to Appendix A for source code.

Class TableData – The class TableData is a data structure that allows the program to store all of the scores for correlation in one element. The class contains data members FCR_Name, SemanticScore, SyntacticScore, and CombinedScore. The method also contains methods for setting and retrieving the data members.

Vector TableDataVector – This vector is a global vector of TableData elements that is used to store all the TableData elements as they are created. The global attribute of this vector allows it to be accessed during the semantic correlation as well as the syntactic correlation.

Class SemanticComparator/Syntactic Comparator – The comparator classes are used for sorting the TableData elements before they are displayed. The results must be displayed descending order by either semantic or syntactic score.

addSemanticScoreToTable / addSyntacticScoreToTable – This method takes the unfilteredResults Array and puts the scores into the global TableDataVector. For each FCR_Name that is not already in the vector a new TableData element is created with the corresponding score and put into the vector.



filterSemanticResultsByThreshold / filterSyntacticResultsByThreshold – This method takes the vector of unfilterResults and adds the elements with scores above the threshold to the vector filteredResults. This is the vector of results that will be displayed. The function *filterSyntacticResultsByThreshold* also calls a function to ensure the syntactic scores are added to the correct element of the global vector.

Collection.sort – This function uses the comparators to sort the results in descending order by score.

sendDataToDisplay – This method takes the filteredResults vector and sends it to the correlator panel for display in the results table.

setContent – This method receives the filteredVector and displays the results in a JTable.

V. POSSIBLE IMPROVEMENTS FOR OOMI CORRELATOR

The two step OOMI correlator can be improved by implementing semantic information into the Neural Network portion of the correlator. The semantic web idea presented earlier suggests a way that semantic information can be extracted from a file representation of a component, if the components for the correlator were represented using an RDF file instead of an XML file. The structure of RDF files represents the data of a component in a way that allows a parser to extract semantic information in addition to syntactic information. Once this information is extracted it can be used to increase the size and accuracy of the discriminator vector. More fields can be added to the discriminator vector which will allow the neural network to take semantic information into consideration when performing a correlation evaluation.

IV. FUTURE RESEARCH

As future research, I will look into increasing the effectiveness of the semantic correlator through the use of a newer concept known as the semantic web. The semantic web uses Resource Descriptive Framework (RDF) files instead of XML files to parse semantic information from a component's structure. RDF files will allow more semantic information beyond just keywords to be used in the correlation process.

APPENDIX A. CORRELATOR SOURCE CODE

A. ComponentModelCorrelator.java

```
/**
 * *****
 * Filename:.....ComponentModelCorrelator.java
 * Parent Project:.....SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
 * Master of Computer Science Thesis Project
 * Original Compiler:....Sun JDK 1.3.1_04
 * Author:.....LT Steve Shedd, USN and LT George Lawler, USN
 * Company:.....Naval Postgraduate School, Monterey, California
 * Date:.....September 2002
 * Notes:.....This class was originally created by LT George Lawler
 * who is currently the master developer for the overall OOMI IDE user interface.
 * *****
 */

package mil.navy.nps.cs.babel.Correlator;

import javax.swing.JComponent;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;

import java.util.List;
import java.util.Iterator;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Vector;
import java.util.*;

import mil.navy.nps.cs.babel.Correlator.CMCorrelatorAdaptor;

import org.jdom.Document;

import mil.navy.nps.cs.babel.event.CorrelatorPanelEvent;
import mil.navy.nps.cs.babel.event.CorrelatorPanelEventListener;
import javax.swing.event.EventListenerList;
import mil.navy.nps.cs.babel.event.CurrentSelectionChangeListener;
import mil.navy.nps.cs.babel.event.CurrentSelectionChangeEvent;
import mil.navy.nps.cs.babel.connectors.OOMIDisplayInterface;

import mil.navy.nps.cs.babel.constructionManager.FIOMConstructionManager;
import mil.navy.nps.cs.babel.connectors.FIOMDBDirectInterface;
import mil.navy.nps.cs.babel.data.FIOMDatabase;
import mil.navy.nps.cs.babel.oomi.fiom.FCR;

import mil.navy.nps.cs.babel.Correlator.semanticComponentGenerator.*;
import mil.navy.nps.cs.babel.Correlator.semanticSearchEngine.*;
import mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator.*;
import mil.navy.nps.cs.babel.Correlator.syntacticSearchEngine.*;

import mwa.ai.neural.*;

/**
 * *****
 * <br>
 * The <i>ComponentModelCorrelator</i> is the overall executive of the Component
 * Model Correlator module of the OOMI IDE. This class coordinates the following:
 * <br><br>
 * <ul>
 * <li> Generation of semantic correlation components
 * <li> Generation of syntactic correlation components
 * <li> Semantic correlation
 * <li> Syntactic correlation
 * </ul>
 * <br>
 *
 *
 */
```

```

*
* @author LT Steve Shedd, LT George Lawler
* @version 1.0
*
***** */

public class ComponentModelCorrelator extends CMCorrelatorAdaptor implements
CorrelatorPanelEventListener, CurrentSelectionChangeListener
{

    private CorrelatorPanel cmCorrelatorPanel = null;

    protected static FIOMDBDirectInterface handleForFIOMDB = null;

    private EventListenerList listenerList = new EventListenerList();

    private OOMIDisplayInterface handleForGUI = null;

    // Results data structures
    private ArrayList semanticResultsFCRList = new ArrayList();

    private Object[] [] filteredSemanticResults = null;

    private Object[] [] filteredSyntacticResults = null;

    private Object[] [] filteredCombinedResults = null;

    private String newFCR=null;

    //new definitions March 05 by MIDN 1/C Childers
    Vector tableDataVector= new Vector();
    Vector filteredTableData Vector=new Vector();

    /** *****
    * <br>
    * The TableData Class is the data structure that holds all of the score information
    * <br><br>
    * <ul>
    * <li> Holds Semantic Scores
    * <li> Holds Syntactic Scores
    * <li> Holds Combined Scores
    * </ul>
    * <br>
    *
    * @author MIDN 1/C Candace Childers, CAPT Young
    * @version 1.0
    *
    ***** */

    public class TableData{
        private String fcrName;
        private Float semanticScore;
        private Float syntacticScore;
        private Float combinedScore;

        private TableData()
        {
            fcrName=null;
            semanticScore= new Float(0);
            syntacticScore=new Float(0);
            combinedScore=new Float(0);
        }

        public void setFCR_Name(String name)
        {
            fcrName=name;
        }
    }
}

```

```

public String getFCR_Name()
{
    return fcrName;
}

public void setSemanticScore(Float semScore)
{
    semanticScore=semScore;
}

public Float getSemanticScore()
{
    return semanticScore;
}
public void setSyntacticScore(Float synScore)
{
    syntacticScore=synScore;
}

public Float getSyntacticScore()
{
    return syntacticScore;
}

public void setCombinedScore(Float combScore)
{
    combinedScore=combScore;
}

public Float getCombinedScore()
{
    return combinedScore;
}
}

/** *****
 * <br>
 * The TableData Comparator Class is used to sort the vector of TableData elements
 * <br><br>
 * <ul>
 *
 *
 *
 * @author MIDN I/C Candace Childers, CAPT Young
 * @version 1.0
 *
 * ***** */
private class SemanticComparator implements Comparator
{
    int semanticCompare;
    TableData semanticScore1, semanticScore2;

    public int compare(Object object1, Object object2)
    {
        semanticScore1=(TableData) object1;
        semanticScore2=(TableData) object2;

        semanticCompare=(semanticScore2.getSemanticScore()).compareTo(semanticScore1.getSemanticScore());
        return semanticCompare;
    }
}

private class SyntacticComparator implements Comparator
{
    int syntacticCompare;
    TableData syntacticScore1, syntacticScore2;

    public int compare(Object object1, Object object2)
    {
        syntacticScore1=(TableData) object1;

```

```

    syntacticScore2=(TableData) object2;

    syntacticCompare=(syntacticScore2.getSyntacticScore()).compareTo(syntacticScore1.getSyntacticScore());
    return syntacticCompare;
}
}

// Correlator Properties and Preferences
private static int semanticThreshold = 70;

private int syntacticThreshold = 70;

private int combinedThreshold = 70;

private float nueralLearningRate = 0.25f;

private int neuralMaxEpochs = 100000;

private double neuralOutputErrorTolerance = 0.1;

public ComponentModelCorrelator( OOMIDisplayInterface paramGUI )
{

    // Reference to the OOMI IDE display interface which can be used to send
    // status messages to the GUI.
    handleForGUI = paramGUI;

    //Singleton on the data base, only one FIOMDatabase
    handleForFIOMDB = FIOMDatabase.getDatabase();
    handleForFIOMDB.addCurrentSelectionChangeListener( this );

    cmCorrelatorPanel = new CorrelatorPanel( this );

} //end constructor

//these functions allow you to get and set the value of semanticThreshold

public static int getSemanticThreshold()
{
    return semanticThreshold;
}

public static void setSemanticThreshold(int newValue)
{
    semanticThreshold=newValue;
}

/** *****
 *
 *
 * @author LT George Lawler, USN
 *
 * ***** */
public void panelEventOccured( CorrelatorPanelEvent event )
{
    switch ( event.getEventID())
    {
        case CorrelatorPanelEvent.PREFERENCES_BUTTON_PRESSED :
            break;
        case CorrelatorPanelEvent.KEYWORD_BUTTON_PRESSED :
            doSemanticCorrelation();
            break;
        case CorrelatorPanelEvent.NEURAL_NET_BUTTON_PRESSED :
            doSyntacticCorrelation();
            break;
        case CorrelatorPanelEvent.COMBINED_BUTTON_PRESSED :
            break;
    }
}

```

```

case CorrelatorPanelEvent.SHOW_FEV_BUTTON_PRESSED :
    // Show selected FEV to the Register Tab Panel.
    break;
case CorrelatorPanelEvent.CCR_SELECTED :
    this.handleForFIOMDB.setCurrentCCR(
        (String)((CorrelatorPanel)event.getSource()).ccrSelected.getSelectedItemId());
    break;
default :
    System.out.println( "Default evoked" );
    break;
}
} //end panelEventOccured
}

/** *****
 * This method forces the correlator panel to get the latest
 * Unregisterd CCR list.
 * ***** */
public void updateCCRList()
{
    this.cmCorrelatorPanel.setCCRList( this.handleForFIOMDB.getCCRList());
}

/** Just returns the correlator panel that is held in this class */
public JComponent getCorrelatorPanel()
{
    return ( ( JComponent )this.cmCorrelatorPanel );
} //end getCorrelatorPanel

/** *****
 * <br>
 * Creates the semantic components for a CCR needed for the semantic correlation.
 *
 * @param ccrJdomDoc - a JDOM Document representation of the CCRs
 * XML Schema file.
 * @return None
 *
 * @author LT Steve Shedd
 * @version 1.0, September 2002
 *
 * ***** */
public void generateCCR.SemanticComponents( Document ccrJdomDoc )
{
    // Create a temporary data structure
    String[] keywords = null;

    // Get an instance of a keyword generator
    KeywordGenerator semanticGen = new KeywordGeneratorImpl();

    try
    {
        // Get the keywords for the CCR
        keywords = semanticGen.generateKeywords( ccrJdomDoc );
    }
    catch ( Exception e )
    {
        e.printStackTrace();
    }

    // Associate the keywords to the CCRSemantics component
    this.handleForFIOMDB.getCurrentCCR().getCCRSemantics().setKeywordList( keywords );
}

```

```

// Send a status message back to the OOMI IDE
this.handleForGUI.setStatusBarText( "CCR Semantic Components Generated" );
}          //end generateCCRSemanticComponents

/** *****
 * <br>
 * Creates the syntactic components for a CCR needed for the
 * syntactic correlation.
 *
 * @param ccrJdomDoc - a JDOM Document representation of the CCRs
 * XML Schema file.
 * @return None
 *
 * @author LT Steve Shedd
 * @version 1.0, September 2002
 *
 ***** */
public void generateCCRSyntacticComponents( Document ccrJdomDoc )
{

// Create an instance of the Discriminator Generator
DiscriminatorGenerator gen = new DiscriminatorGeneratorImpl();

// Create a list for the discriminator vectors
List discr = null;

try
{
// run the generator
discr = gen.generateDiscriminatorVectors( ccrJdomDoc );
}
catch ( Exception e )
{
e.printStackTrace();
}

// Associate the discriminator vectors with the CCRSyntax component
handleForFIOMDB.getCurrentCCR().getCCRSyntax().setDiscriminatorVectors( discr );

// Send a status message back to the GUI
this.handleForGUI.setStatusBarText( "CCR Syntactic Components Generated" );

}          //end generateCCRSyntacticComponents

/** *****
 * <br>
 * Creates the semantic components for an FCR needed for the
 * semantic correlation.
 *
 * @param fcrJdomDoc - a JDOM Document representation of the FCRs
 * XML Schema file.
 * @return None
 *
 * @author LT Steve Shedd
 * @version 1.0, September 2002
 *
 ***** */
public void generateFCRSemanticComponents( Document fcrJdomDoc )
{

// Create a temporary data structure
String[] keywords = null;

// Get an instance of a keyword generator
KeywordGenerator semanticGen = new KeywordGeneratorImpl();

```

```

try
{
    // Get the keywords for the FCR
    keywords = semanticGen.generateKeywords( fcrJdomDoc );
}
catch ( Exception e )
{
    e.printStackTrace();
}

// Associate the keywords to the FCRSemantics component
this.handleForFIOMDB.getCurrentFCR().getFCRSemantics().setKeywordList( keywords );

// Send a status message back to the OOMI IDE
this.handleForGUI.setStatusBarText( "FCR Semantic Components Generated" );

} //end generateFCRSemanticComponents

/** *****
 * <br>
 * Creates the syntactic components for an FCR needed for the
 * syntactic correlation.
 *
 * @param fcrJdomDoc - a JDOM Document representation of the FCRs
 * XML Schema file.
 * @return None
 *
 * @author LT Steve Shedd
 * @version 1.0, September 2002
 *
 ***** */
public void generateFCRSyntacticComponents( Document fcrJdomDoc )
{

    // Create an instance of the Discriminator Generator
    DiscriminatorGenerator gen = new DiscriminatorGeneratorImpl();

    // Create a list for the discriminator vectors
    List discr = null;

    try
    {
        // run the discriminator generator
        discr = gen.generateDiscriminatorVectors( fcrJdomDoc );
    }
    catch ( Exception e )
    {
        e.printStackTrace();
    }

    // Associate the discriminator vectors with the CCRSyntax component
    handleForFIOMDB.getCurrentCCR().getCCRSyntax().setDiscriminatorVectors( discr );

    // Create an instance of the neural net generator
    NeuralNetGenerator nnGen = new NeuralNetGeneratorImpl();

    // Create and train a neural network for the FCR
    Neural nn = nnGen.generateNeuralNet( discr );

    // Set the path for the neural network file
    String path = "X:OOMI_IDE\\root\\etc\\babel\\generated\\neural\\";
    path = path + handleForFIOMDB.getCurrentFCR().getJavaClassName();

    // Create and save a formatted neural network file
    nnGen.makeFile( nn, path );
}

```

```

// Associate the syntactic components with the FCRSyntax component
handleForFIOMDB.getCurrentFCR().getFCRSyntax().setDiscriminatorVectors( discr );
handleForFIOMDB.getCurrentFCR().getFCRSyntax().setNeuralNetwork( nn );
handleForFIOMDB.getCurrentFCR().getFCRSyntax().setNNfilePath( path );

// Send a status message to the OOMI IDE
this.handleForGUI.setStatusBarText( "FCR Syntactic Components Generated" );
} //end generateFCRSyntacticComponents

/** *****
 * <br>
 * Implementation of the semantic correlation phase of the Component Model
 * Correlator. This
 * method performs the semantic correlation between a CCR and the
 * list of FEV FCRs in
 * the FIOM. It then sends the results of the correlation to the
 * results component of
 * the Correlator Panel in the OOMI IDE.
 *
 * @param None
 * @return None
 *
 * @author LT Steve Shedd
 * @version 1.0, September 2002
 *
 ***** */
public void doSemanticCorrelation()
{

    this.filteredSyntacticResults = null;

    // Create an instance of the semantic search engine
    SemanticSearchEngine semanticSearch = new SemanticSearchEngineImpl();

    // Create a reference for the unfiltered search results
    Object[] [] unfilteredResults = null;

    // Get a list of all FCRs in the FIOM
    List localFIOMFCRList = this.handleForFIOMDB.getFCRList();

    // Check to see if there are FCRs in the FIOM
    if( localFIOMFCRList != null )
    {
        // Perform the semantic correlation. This will produce a sorted object
        // array of FCR names and Scores for every FCR in the FIOM
        unfilteredResults = semanticSearch.runSemanticSearch( localFIOMFCRList,
            handleForFIOMDB.getCurrentCCR());
    }
    else
    {
        this.handleForGUI.setStatusBarText( "No FEVs in FIOM" );
    }

    //Call function to add all scores to the vector unfilteredResults
    addSemanticScoreToTable(unfilteredResults);

    filteredTableDataVector.removeAllElements();

    //add all FEV's above threshold to filteredTableDataVector
    filteredTableDataVector=filterSemanticResultsByThreshold(tableDataVector, this.semanticThreshold);

    //The following code saves the FCR's in the filteredTableDataVector to a list of FCR's that can be passed to the syntactic
    //correlator

    Iterator fcrIter = localFIOMFCRList.iterator();

```

```

while ( fcrIter.hasNext() )
{
    FCR testerFCR = ( FCR )fcrIter.next();
    for ( int i = 0; i < filteredTableDataVector.size(); i++)
    {
        if(((testerFCR.getFCRName()).equals(((TableData)filteredTableDataVector.elementAt(i)).getFCR_Name()))
        {
            this.semanticResultsFCRList.add(testerFCR);
        }
    }
}

//sort the FEV's in filteredTableDataVector
Collections.sort(filteredTableDataVector, new SemanticComparator());

//send the sorted and filtered Vector to the display table
sendDataToDisplay(filteredTableDataVector);
}          // End doSemanticCorrelation

//The following function adds the SemanticScore to the global vector tableDataVector if the score is above the threshold set
//by the user in the GUI

private void addSemanticScoreToTable(Object [][] unfilteredSemanticResults)
{
    for(int i=0; i<unfilteredSemanticResults.length;i++)
    {
        //if FCR name not in table
        if(findFCRName(tableDataVector, (String)unfilteredSemanticResults[i][0])==false)
        {
            //add new FCR to Table
            TableData newFCR=new TableData();
            newFCR.setFCR_Name((String)unfilteredSemanticResults[i][0]);
            newFCR.setSemanticScore(((Float)unfilteredSemanticResults[i][1]));
            tableDataVector.addElement(newFCR);
        }
    }
}

//The following function checks to see if an FCR has already been added to TableDataVector so no duplicates are added
private boolean findFCRName(Vector table, String FCR)
{
    //for each object in table
    for(int i=0; i<table.size();i++)
    {
        //see if FCR already added to table
        if((((TableData)table.elementAt(i)).getFCR_Name()).equals(FCR))
        {
            return true;
        }
    }
    return false;
}

```

```

/** *****

```

```

* <br>
* Implementation of the syntactic correlation phase of the Component Model
* Correlator. This
* method performs the syntactic correlation between a CCR and the

```

```

* list of FEV FCRs in
* the FIOM. It then sends the results of the correlation to the
* results component of
* the Correlator Panel in the OOMI IDE.
*
* @param None
* @return None
*
* @author LT Steve Shedd
* @version 1.0, September 2002
*
***** */
public void doSyntacticCorrelation()
{

    // Create an instance of the syntactic search engine
    SyntacticSearchEngine syntacticSearch = new SyntacticSearchEngineImpl();

    // Create a reference for the unfiltered search results
    Object[] [] unfilteredResults = null;

    // Check to see if the semantic correlation returned some results
    if ( semanticResultsFCRList != null )
    {
        // Perform the semantic correlation. This will produce a sorted object
        // array of FCR names and Scores for every FCR in list populated from
        // the semantic correlation results
        unfilteredResults = syntacticSearch.runSyntacticSearch( semanticResultsFCRList,
            handleForFIOMDB.getCurrentCCR());
    }
    else
    {

        // If there are no results from the semantic correlator, do the syntactic
        // correlation with the FCR list from the entire FIOM.

        List localFIOMFCRList = this.handleForFIOMDB.getFCRList();

        if ( localFIOMFCRList != null )
        {
            unfilteredResults = syntacticSearch.runSyntacticSearch( localFIOMFCRList,
                handleForFIOMDB.getCurrentCCR());
        }
        else
        {
            this.handleForGUI.setStatusBarText( "No FEVs in FIOM" );
        }
    }

    // The combined results equals the average of the semantic and syntactic scores.
    for ( int i = 0; i < this.filteredSemanticResults.length; i++ )
    {
        float sem = ( ( Float )this.filteredSemanticResults[ i ][ 1 ] ).floatValue();
        float syn = ( ( Float )this.filteredSyntacticResults[ i ][ 1 ] ).floatValue();
        float avg = ( sem + syn ) / 2;

        this.filteredCombinedResults[ i ][ 1 ] = new Float( avg );
    }

    addSyntacticScoreToTable(unfilteredResults);

    filteredTableDataVector.removeAllElements();

    //add all FEV's above threshold to filteredTableDataVector
    filteredTableDataVector=filterSyntacticResultsByThreshold(tableDataVector, this.syntacticThreshold);

```

```

//sort the FEV's in filteredTableDataVector
Collections.sort(filteredTableDataVector, new SyntacticComparator());

//send the sorted and filtered Vector to the display table
sendDataToDisplay(filteredTableDataVector);

} // End doSyntacticCorrelation

//This function adds the syntactic score to the global vector TableDataVector if the score is above the threshold. The function
//ensures that the score is stored in the correct corresponding TableData Element of the vector.

private void addSyntacticScoreToTable(Object [][] unfilteredSyntacticResults)
{
    for(int i=0; i<unfilteredSyntacticResults.length;i++)
    {
        //if already in table
        int element=findFCRName_Syntactic(tableDataVector, (String)unfilteredSyntacticResults[i][0]);
        if(element!=(tableDataVector.size()+1))
        {
            //add syntactic score to existing element in vector
            ((TableData)tableDataVector.elementAt(element)).setSyntacticScore(((Float)unfilteredSyntacticResults[i][1]));
        }
        else
        {
            //add new FCR to Table
            TableData newFCR=new TableData();
            newFCR.setFCR_Name((String)unfilteredSyntacticResults[i][0]);
            newFCR.setSyntacticScore(((Float)unfilteredSyntacticResults[i][1]));
            tableDataVector.addElement(newFCR);
        }
    }
} //end addSyntacticScoreToTable

//This function returns the index value of the corresponding TableData Element in the vector that has the same name as the
//name passed in as //a parameter.
private int findFCRName_Syntactic(Vector table, String FCR)
{
    //for each object in table
    for(int i=0; i<table.size();i++)
    {
        //see if FCR already added to table
        if((((TableData)table.elementAt(i)).getFCR_Name()).equals(FCR))
        {
            return i;
        }
    }
    return (table.size()+1);
} //end findFCRName_Syntactic

/** *****
 * Utility function to filter a list of FEV/Score pairs by
 * threshold value setting.
 * This function returns a new double dimension array. The array must
 * have an FEV name
 * in index 0 and an FEV correlation score as index 1
 *
 * @param input - double dimension array containing FEV/Score pairs.
 * @param threshold - The threshold setting for the particular correlation
 * @return A new double dimension array containing the FEV/Score pairs
 * above the threshold
 *
 * @author LT Steve Shedd
 *
 * ***** */

//new SemanticfilterResults method...puts results above threshold into vector

private Vector filterSemanticResultsByThreshold(Vector unfilteredTable, int threshold)
{

```

```

//create a new structure for filtered output
Vector filteredTable = new Vector();

//cycle through the unfilteredTable and copy elements above threshold
for(int i=0;i<unfilteredTable.size();i++)
{
    if((((TableData)unfilteredTable.elementAt(i)).getSemanticScore().floatValue()*100)>=threshold)
    {
        filteredTable.add(unfilteredTable.elementAt(i));
    }
    else
        break;
}
return filteredTable;
}

private Vector filterSyntacticResultsByThreshold(Vector unfilteredTable, int threshold)
{
    //create a new structure for filtered output
    Vector filteredTable = new Vector();

    //cycle through the unfilteredTable and copy elements above threshold
    for(int i=0;i<unfilteredTable.size();i++)
    {
        if((((TableData)unfilteredTable.elementAt(i)).getSyntacticScore().floatValue()*100)>=threshold)
        {
            filteredTable.add(unfilteredTable.elementAt(i));
        }
        else
            break;
    }
    return filteredTable;
} //end filterSyntacticResultsByThreshold

/** *****
 * <br>
 * Formats the FEV/Score pairs produced by the semantic and syntactic
 * correlators for the
 * Correlator Panel. The Correlator Panel uses a JTable to display the
 * correlation results.
 * The defaultTableModel for a JTable can be used to convert a double
 * dimension array into
 * a JTable. This method merges the three input double dimension arrays
 * into a single double
 * dimension array. The JTable in the correlator panel is then populated
 * with the results.
 * @param semanticResults - the filtered results of the semantic correlation
 * @param syntacticResults - the filtered results of the syntactic correlation.
 * @param combinedResults - the filtered combined correlation results.
 *
 * @author LT Steve Shedd
 *
 ***** */
private void sendDataToDisplay(Vector display)
{
    this.cmCorrelatorPanel.setContent(display);
}

// The correlator panel works with the CCR list, and so it must
// know when the current CCR is changed, this listener method
// will be called when ever the CurrentCCR is changed in the
// FIOMDatabase class.
public void selectionChanged( CurrentSelectionChangeEvent event )
{

```

```

        if ( event.getSelectionThatChanged() ==
            CurrentSelectionChangeEvent.CCR_SELECTION_CHANGED )
        {
            //Update the CCR list in the correlator panel
            this.updateCCRList();
        }
        else
        {
            // no action for Current FCR change event
        }
    } //end selectionChanged
} //end ComponentModelCorrelator

```

B. CorrelatorPanel.java

```

/*****
// Filename:.....CorrelatorPanel.java
// Parent Project:.....OOMI IDE
// Original Compiler:....Sun JDK 1.3.1_04
// Author:.....LT George Lawler, USN
// Company:.....Naval Postgraduate School, Monterey, California
// Date:.....September 2002
// Notes:.....This class was originally created by LT George Lawler
// during the development of the OOMI IDE prior to the completion of the features
// of the Component Model Correlator.
*****/

package mil.navy.nps.cs.babel.Correlator;

import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.*;
import java.util.Vector;

import mil.navy.nps.cs.babel.event.CorrelatorPanelEvent;
import mil.navy.nps.cs.babel.event.CorrelatorPanelEventListener;
import javax.swing.event.EventListenerList;

import javax.swing.JTable;
import javax.swing.table.TableModel;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableModel;
//import javax.swing.table.TableColumn;

/** *****
 * Title: Babel Correlator Panel
 * Description: Front end for Register Tab Panel connection to the Correlator.
 * Also provides area for display of correlator results.
 *
 * Pass in the MainFrame parametere during construction so that
 * this panel can access the controlChannel to pass events to the
 * back end.
 *
 * @author LT George Lawler
 *****/
public class CorrelatorPanel extends JPanel
{
    private JPanel contentPane = null;

    /** *****
 * The table used to display the results of the component model
 * correlation phases.
 *****/

```

```

JTable table = null;

/** *****
 * The title for the first column in the table displayed on this panel.
 * This title is built into a TableModel that is used to construct the
 * table.
 * ***** */
public static String COLUMN_ZERO_TITLE = new String( "FEV" );

/** *****
 * The title for the second column in the table displayed on this panel.
 * This title is built into a TableModel that is used to construct the
 * table.
 * ***** */
public static String COLUMN_ONE_TITLE = new String( "Keyword Score" );

/** *****
 * The title for the third column in the table displayed on this panel.
 * This title is built into a TableModel that is used to construct the
 * table.
 * ***** */
public static String COLUMN_TWO_TITLE = new String( "Syntactic Score" );

/** *****
 * The title for the fourth column in the table displayed on this panel.
 * This title is built into a TableModel that is used to construct the
 * table.
 * ***** */
public static String COLUMN_THREE_TITLE = new String( "Combined Score" );

// ** This next section is indented to show which parts are 'contained'
//    in which of the overall panel's sub containers.

//This control area panel holds all the widgets
JPanel controlArea = new JPanel();

JPanel ccrSelectionSubPanel = new JPanel();

JLabel ccrSelectionLabel = new JLabel();

JComboBox ccrSelected = new JComboBox();

JButton preferencesButton = new JButton();

JPanel thresholdSubPanel = new JPanel();

JLabel buttonLabel = new JLabel();

JButton filterOnKeyword = new JButton();

JButton filterOnNeuralNet = new JButton();

JButton filterOnCombined = new JButton();

JLabel comboBoxLabel = new JLabel();

JComboBox keywordThreshold = null;    //new JComboBox();

JComboBox neuralNerThreshold = null;  //new JComboBox();

JComboBox combinedScoreThreshold = null;    //new JComboBox();

JPanel actionSubPanel = new JPanel();

JButton showSelectedFE = new JButton();

```

```

private Insets buttonInsets = new Insets( 2, //top
    2, //left
    2, //bottom
    2 ); //right

//This data area panel holds the table that shows the results
JScrollPane dataArea = null;

//This is how many choices to put in the Threshold drop down arrays
private static int NUMBER_OF_VALUES = 10;

//This is the distance between thresholds. Increment is
// multiplied by the index plus one (plus one so the first
// value isn't zero). NUMBER_OF_VALUES times the INCREMENT
// is the maximum value in the table.
private static int INCREMENT = 10;

// so that the value shown is 70
private static int DEFAULT_THRESHOLD_INDEX = 6;

//this Object array holds strings that represent the
// integer value for the search threshold
private Object[] thresholdValues = null;

private EventListenerList listenerList = new EventListenerList();

/** *****
 * Constructor calls the private init method to set up the
 * panel, and catches any exceptions encountered during panel
 * setup.
 * @param MainFrame reference used to gain access to the OOMICControlInterface
 *
 * Pass in to this object a CCR List and an FCR list so that the panel
 * will come up with a few things in it for the demo.
 *
 * Hold off on constructing this panel until the FIOM is created, so that
 * there will be values available to pass as parameters.
 * ***** */
public CorrelatorPanel( CorrelatorPanelEventListener paramEventListener )
{
    try
    {
        init();
        this.contentPane = this;
        this.addCorrelatorPanelEventListener( paramEventListener );
    }
    catch ( Exception e )
    {
        e.printStackTrace();
    }
    //end CorrelatorPanel constructor
}

public void addCorrelatorPanelEventListener( CorrelatorPanelEventListener
listener )
{
    this.listenerList.add( CorrelatorPanelEventListener.class, listener );
}

public void removeCorrelatorPanelEventListener( CorrelatorPanelEventListener
listener )

```

```

{
    this.listenerList.remove( CorrelatorPanelEventListener.class, listener );
}

public void fireCorrelatorPanelEvent( int eventID, int threshold,
String selectedName )
{
    //Guaranteed to return a non-null array
    Object[] listeners = this.listenerList.getListenerList();
    // Process the listeners last to first, notifying
    // those that are interested in this event
    for ( int i = listeners.length - 2; i >= 0; i -= 2 )
    {
        CorrelatorPanelEvent event = null;
        if ( listeners[i] == CorrelatorPanelEventListener.class )
        {
            // Lazily create the event:
            if ( event == null )
            {
                event = new CorrelatorPanelEvent( this,
                    eventID,
                    threshold,
                    selectedName );
            }
            (( CorrelatorPanelEventListener )listeners[i+1]).panelEventOccured(event);
        }
        // end of for loop to notify listeners
    }
    //end fireCorrelatorPanelEvent()
}

```

```

private void init()
{
    this.setLayout( new BorderLayout() );

    this.controlArea.setLayout( new FlowLayout() );

    this.initCCRSelector();

    this.preferencesButton.setText( "Preferences" );
    this.preferencesButton.setMargin( buttonInsets );
    this.preferencesButton.setEnabled( true );
    this.preferencesButton.addActionListener(
        new ActionListener()
        {
            public void actionPerformed((ActionEvent e)
            {
                preferencesButton_actionPerformed( e );
            }
        }
    );

    this.filterOnCombined.setText( "Combined Score" );
    this.filterOnCombined.setMargin( buttonInsets );
    this.filterOnCombined.setEnabled( true );
    this.filterOnCombined.addActionListener(
        new ActionListener()
        {
            public void actionPerformed( ActionEvent e )
            {
                filterUsingCombined_actionPerformed( e );
            }
        }
    );

    this.filterOnKeyword.setText( "Keywords" );
    this.filterOnKeyword.setMargin( buttonInsets );
    this.filterOnKeyword.setEnabled( true );
    this.filterOnKeyword.addActionListener(

```

```

new ActionListener()
{
    public void actionPerformed( ActionEvent e )
    {
        filterUsingKeywords_actionPerformed( e );
    }
} );

this.filterOnNeuralNet.setText( "Neural Net" );
this.filterOnNeuralNet.setMargin( buttonInsets );
this.filterOnNeuralNet.setEnabled( true );
this.filterOnNeuralNet.addActionListener(
new ActionListener()
{
    public void actionPerformed( ActionEvent e )
    {
        filterUsingNeuralNet_actionPerformed( e );
    }
} );

//get an array of proper length for Threshold values
this.thresholdValues = new Object[NUMBER_OF_VALUES];

//fill the array with values based on INCREMENT
for ( int i = 0; i < NUMBER_OF_VALUES; i++ )
{
    //I add one to i, so that the first element is not zero.
    this.thresholdValues[i] = String.valueOf( ( i + 1 ) * INCREMENT );
}

//then just build the combo boxes out of the array from above
this.keywordThreshold = new JComboBox( this.thresholdValues );
this.keywordThreshold.setSelectedIndex( DEFAULT_THRESHOLD_INDEX );

this.combinedScoreThreshold = new JComboBox( this.thresholdValues );
this.combinedScoreThreshold.setSelectedIndex( DEFAULT_THRESHOLD_INDEX );

this.neuralNetThreshold = new JComboBox( this.thresholdValues );
this.neuralNetThreshold.setSelectedIndex( DEFAULT_THRESHOLD_INDEX );

this.buttonLabel.setText( new String( "Filter On:" ) );
this.buttonLabel.setHorizontalAlignment( JLabel.RIGHT );
this.comboBoxLabel.setText( new String( "Threshold Values:" ) );
this.comboBoxLabel.setHorizontalAlignment( JLabel.RIGHT );

//combo box action listener for keyword
this.keywordThreshold.addActionListener(
new ActionListener()
{public void actionPerformed (ActionEvent e)
{
    System.out.println("action listener works");
    setKeywordThreshold();//some parameter from combo box);
}}
);

//combo box action listener for combinedScore
this.combinedScoreThreshold.addActionListener(
new ActionListener()
{public void actionPerformed (ActionEvent e)
{
    System.out.println("action listener works");
    //setCombinedScoreThreshold();//some parameter from combo box);
}}
);

this.neuralNetThreshold.addActionListener(
new ActionListener()
{public void actionPerformed (ActionEvent e)

```

```

{
    System.out.println("action listener works");
    //setNueralNetThreshold();//some parameter from combo box;
}}
);

// set up the threshold buttons and combo boxes
this.thresholdSubPanel.setLayout( new GridLayout( 2, 4 ) );
this.thresholdSubPanel.add( this.buttonLabel );
this.thresholdSubPanel.add( this.filterOnKeyword );
this.thresholdSubPanel.add( this.filterOnNeuralNet );
this.thresholdSubPanel.add( this.filterOnCombined );
this.thresholdSubPanel.add( this.comboBoxLabel );
this.thresholdSubPanel.add( this.keywordThreshold );
this.thresholdSubPanel.add( this.neuralNetThreshold );
this.thresholdSubPanel.add( this.combinedScoreThreshold );

this.showSelectedFE.setText( new String( "Show FE" ) );
this.showSelectedFE.setMargin( buttonInsets );
this.showSelectedFE.setEnabled( true );
this.showSelectedFE.addActionListener(
    new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            showSelectedFE_actionPerformed( e );
        }
    }
);

this.actionSubPanel.add( this.showSelectedFE );

//put all the sub comonents into the controlArea
this.controlArea.add( ccrSelectionSubPanel );
this.controlArea.add( this.preferencesButton );
this.controlArea.add( this.thresholdSubPanel );
this.controlArea.add( this.actionSubPanel );

//This sets up a table with 5 rows with the column headers found at
// the start of this class.
TableModel dataModel = new AbstractTableModel()
{
    public int getColumnCount() { return 4; }

    //This method is used to construct the table, and so
    // by returning the desired heading label, the table is
    // constructed each time to look as we desire.
    public String getColumnName( int columnIndex )
    {
        String columnName = null;
        switch ( columnIndex )
        {
            case( 0 ) :
                columnName = CorrelatorPanel.COLUMN_ZERO_TITLE;
                break;
            case( 1 ) :
                columnName = CorrelatorPanel.COLUMN_ONE_TITLE;
                break;
            case( 2 ) :
                columnName = CorrelatorPanel.COLUMN_TWO_TITLE;
                break;
            case( 3 ) :
                columnName = CorrelatorPanel.COLUMN_THREE_TITLE;
                break;
            //The following code was commented out because we decided we only needed four columns for the table
            // case( 4 ) :
            //     columnName = CorrelatorPanel.COLUMN_FOUR_TITLE;
            //     break;
            // case( 5 ) :

```

```

        // columnName = CorrelatorPanel.COLUMN_FIVE_TITLE;
        // break;
        default:
            columnName = new String( "error" );
        }
        return ( columnName );
    }

    public Class getColumnClass( int column )
    {
        Class classValue = null;
        switch ( column )
        {

            default:
                classValue = ( Class ) java.lang.Object.class;
            }
        return ( classValue );
    }

    public int getRowCount() { return 5; }

    //eventually this will need to be written to link the table that we
    // view, which is being defined here, to arrays of information that
    // are passed into the CorrelatorPanel from the correlator.
    // So that as the table is constructed, this statement can
    // return that which is to be displayed.
    // right now I'm trying to display a blank table with
    // checkboxes in the third and fifth columns.
    public Object getValueAt( int row, int col )
    {
        Object cellValue = null;
        switch ( col )
        {
            //case(2) : case(5) : cellValue = new JCheckBox();
            // break;
            default:
                cellValue = new String( "" );
            }
        return ( cellValue );
    }
};

this.table = new JTable( dataModel );

this.add(table, BorderLayout.CENTER);
dataArea = new JScrollPane( table );

//put the control area and the data area in the main panel
//this.add(this.controlArea, BorderLayout.NORTH);
//this.add(this.dataArea, BorderLayout.CENTER);
this.refreshCorrelatorPanel();

} //end init

public void initCCRSelector()
{
    this.ccrSelected.addActionListener(
        new ActionListener()
        {
            public void actionPerformed( ActionEvent e )
            {
                changeToSelectedCCR( e );
            }
        }
    );
}

this.ccrSelectionSubPanel = new JPanel();

```

```

this.ccrSelectionSubPanel.setLayout( new GridLayout( 2, 1 ) );
this.ccrSelectionLabel.setText( "Component Class Selected:" );
this.ccrSelectionSubPanel.add( this.ccrSelectionLabel );
this.ccrSelectionSubPanel.add( this.ccrSelected );

}

/** *****
 * This function accepts a Vector of TableData elements which are going to be displayed.
 *
 *
 *
 * Modified on 05March2005 by MIDN 1/C Candace Childers
 *
 * ***** */
public void setContent( Vector tableContent )
{

String[] columnNames = { CorrelatorPanel.COLUMN_ZERO_TITLE,
                        CorrelatorPanel.COLUMN_ONE_TITLE,
                        CorrelatorPanel.COLUMN_TWO_TITLE,
                        CorrelatorPanel.COLUMN_THREE_TITLE,};

TableModel newModel=new DefaultTableModel(columnNames,tableContent.size());

int row = 0;

for (int i = 0; i < tableContent.size(); i++) {

newModel.setValueAt( ( (ComponentModelCorrelator.TableData)
tableContent.get(i)).getFCR_Name(), row, 0);
newModel.setValueAt( ( (ComponentModelCorrelator.TableData)
tableContent.elementAt(i)).getSemanticScore(),
row, 1);
newModel.setValueAt( ( (ComponentModelCorrelator.TableData)
tableContent.elementAt(i)).getSyntacticScore(),
row, 2);
newModel.setValueAt( ( (ComponentModelCorrelator.TableData)
tableContent.elementAt(i)).getCombinedScore(),
row, 3);

row++;
}

this.table = null;

this.table = new JTable( newModel );
this.dataArea = new JScrollPane( table );

System.out.println("table row count: " + this.table.getRowCount() );
this.refreshCorrelatorPanel();

} //end setContent

public void setCCRList( Object[] paramList )
{

this.ccrSelected.removeAllItems(); // = new JComboBox();

for ( int i = 0; i < paramList.length; i++ )
{
this.ccrSelected.addItem( paramList[i] );
}
}

```

```

//this.initCCRSelector();
//System.out.println(this.ccrSelected.getItemCount());

refreshCorrelatorPanel();
} //end setCCRList

public void refreshCorrelatorPanel()
{
    this.removeAll();

    this.add( this.controlArea, BorderLayout.NORTH );
    this.add( this.dataArea, BorderLayout.CENTER );
}

// ***** Button Handlers *****

private void preferencesButton_actionPerformed( ActionEvent e )
{
    fireCorrelatorPanelEvent( CorrelatorPanelEvent.PREFERENCES_BUTTON_PRESSED,
CorrelatorPanelEvent.NULL_INT,
CorrelatorPanelEvent.NULL_STRING );
} //end preferencesButton_actionPerformed

private void filterUsingKeywords_actionPerformed( ActionEvent e )
{
    int threshold = Integer.parseInt(
( this.keywordThreshold.getSelectedItem() ).toString() );
    fireCorrelatorPanelEvent( CorrelatorPanelEvent.KEYWORD_BUTTON_PRESSED,
threshold,
CorrelatorPanelEvent.NULL_STRING );
} //end filterUsingKeywords_actionPerformed

private void filterUsingNeuralNet_actionPerformed( ActionEvent e )
{
    int threshold = Integer.parseInt( ( this.neuralNetThreshold.getSelectedItem()
).toString() );
    fireCorrelatorPanelEvent( CorrelatorPanelEvent.NEURAL_NET_BUTTON_PRESSED,
threshold,
CorrelatorPanelEvent.NULL_STRING );
} //end filterUsingNeuralNet_actionPerformed

private void filterUsingCombined_actionPerformed( ActionEvent e )
{
    int threshold = Integer.parseInt( ( this.combinedScoreThreshold.getSelectedItem()
).toString() );
    fireCorrelatorPanelEvent( CorrelatorPanelEvent.COMBINED_BUTTON_PRESSED,
threshold,
CorrelatorPanelEvent.NULL_STRING );
} //end filterUsingCombined_actionPerformed

private void showSelectedFE_actionPerformed( ActionEvent e )
{
    try
    {
        //this.parentFrame.controlChannel.correlatorShowSelectedFE();
    }
    catch ( java.lang.UnsupportedOperationException uoe )

```

```

    {
        //this.parentFrame.notImplementedDialog( uoe );
    }
    //end showSelectedFE_actionPerformed

private void changeToSelectedCCR( ActionEvent e )
{
    fireCorrelatorPanelEvent( CorrelatorPanelEvent.CCR_SELECTED,
        CorrelatorPanelEvent.NULL_INT,
        ( ( String )this.ccrSelected.getSelectedItem() ) );
}
//the following methods set the threshold integers to the value stored in the combo box
private void setKeywordThreshold()

{
    ComponentModelCorrelator.setSemanticThreshold(Integer.parseInt(
        ( this.keywordThreshold.getSelectedItem() ).toString() ));

    System.out.println("semantic threshold: " + ComponentModelCorrelator.getSemanticThreshold());
}

private void setCombinedScoreThreshold()

{
    //combinedThreshold=combinedScoreThreshold.getSelectedItem();
    System.out.println(combinedScoreThreshold.getSelectedItem());
}

private void setNeuralNetThreshold()

{
    //syntacticThreshold=keywordThreshold.getSelectedItem();
    System.out.println(neuralNetThreshold.getSelectedItem());
}

public JPanel getCorrelatorPanel()

{
    return ( this.contentPane );
}

//end CorrelatorPanel

//EOF CorrelatorPanel.java

```

REFERENCES

- [BFHW 95] Benkley, S., and others, *Data Element Tool-Based Analysis (DELTA)*, MITRE Corporation report MTR95B0000147, December 1995.
- [DeMe 00] Decker, S., and others, *The Semantic Web: The Roles of XML and RDF*, IEEE Internet Computing, Sept-Oct 2000.
- [LiCl 94] Wen-Syan, L., Clifton, C. *Semantic Integration in Heterogeneous Databases Using Neural Networks*, Proceedings of the 20th VLDB Conference, 1994.
- [Prieto-Diaz 90] Prieto-Diaz, R., "Implementing Faceted Classification for Software Reuse", *Communications of the ACM*, Volume 34, No. 5, May 1990.
- [Shedd 02] Shedd, S., *Semantic and Syntactic Object Correlation in the Object-Oriented Method for Interoperability*, Master's thesis, Naval Postgraduate School, Sep 2002.
- [WiCl 00] Wen-Syan, L., Clifton, C. *SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks*, C&C Research Laboratories, 2000.
- [Wied 93] Wiederhold, G., "Intelligent Integration of Information", *ACM-SIGMOD 93*, Washington, DC, May 1993, pp. 434-437.
- [Young 02] Young, P. *Heterogeneous Software System Interoperability Through Computer-Aided Resolution of Modeling Differences*, PhD dissertation, Naval Postgraduate School, Jun 2002.
- [ZW 95] Zaremski, A.M., Wing, J.W., "Specification Matching of Software Components", *SIGSOFT '95*, ACM 0-89791-716-2/95/0010, pp.6-17.
- [ZW 93] Zaremski, A.M., Wing, J.W., "Signature Matching: A Key to Reuse", *SIGSOFT '93*, ACM 0-8971-625-5/93/0012, pp. 182-190

