

TABLE OF CONTENTS

INTRODUCTION TO *BLANCHE*.....

THE MODEL.....

 ATTRIBUTES
 RELATIONS.....
 COGNITIVE ATTRIBUTES.....
 COGNITIVE RELATIONS
 VARIABLES.....

BUILDING MODELS.....

 VARIABLE EDIT WINDOW
 CREATING NEW VARIABLES.....
 MODIFYING A VARIABLE
 DELETING A VARIABLE.....
 SETTING THE NUMBER OF NODES
 SETTING THE NUMBER OF ITERATIONS.....

EQUATIONS.....

 SIGMAS
 FUNCTION LIST
 VARIABLE MODIFIERS.....
 NODE MODIFIERS.....
 RELATION MODIFIERS
 COGNITIVE FUNCTIONS.....
 TIME OFFSETS

RUNNING MODELS.....

 THE MENU BAR
 RUNNING A MODEL.....
 AVERAGE DATA
 SHARING DATA

VISUALIZING MODELS.....

 GRAPHING A MODEL.....
 USING THE NETWORK VISUALIZER.....

Introduction to *Blanche*

Blanche is a program designed to create and execute computational models of network behavior. It is intended to be used by researchers who wish to formulate a hypothesis of how a particular network (of people, organizations, or anything else) functions, and then evaluate the hypothesis by simulating the network and examining the results.

The Model

It is a collection of equations. Each equation describes how a particular factor changes over time. When the model is run, each equation is run at each iteration, so the network changes over time. The objects that make up the model are *nodes*, *attributes*, *relations*, *cognitive attributes*, and *cognitive relations*.

Attributes

An attribute is a numerical value that defines a property of a node. A node can have any number of attributes or none at all. Each attribute has an equation that describes how the value of the attribute changes over time. The value of the attribute in *Blanche* is considered a double-precision floating-point number. However, this value can be considered a binary value (0 or 1) or a categorical value, depending on how it is produced and used.

Relations

A Relation is a set of numerical values that define interactions between nodes. Relations are akin to N by N matrices but are lacking the diagonal.

Cognitive Attributes

Cognitive attributes represent everyone's view of everyone else's attributes. They look very similar to relation in that they are N by N matrices but the diagonal is present.

Cognitive Relations

Cognitive Relations represent everyone's view of every one else's relations. They are an N by N by $N-1$ array of matrices. This represents N people's view of the relations.

Variables

A variable is a name for either a network or an attribute. Both of them are defined by equations, and can be used in other equations. In this manual, the word "variable" is just shorthand for "an attribute or network".

Building Models

A model is specified by creating attributes, relations and cognitive structures that define it. Then, the user must define how each attribute and network changes over time by typing in equations.

The Variable Edit Window

Figure 1 shows the variable edit screen of *Blanche*. This window is used to edit all four variable types in *Blanche*. To access this menu pull down the “View” menu and select one of the four variable types, this will bring up a list of all the variables of that type. Then double click on an existing variable or select new.

The screenshot shows the "Edit Variable" dialog box. It features a title bar with a close button. The main area includes a "Variable Name:" text box, a "Level:" text box with the value "0", and a set of radio buttons for "Attribute", "Relation", "Cognitive Attribute", and "Cognitive Relation". There are also checkboxes for "Computational Variable" and "Pass Data". To the right are "Save" and "Cancel" buttons, and a "<<Prev" and "Next>>" button. Below this is a large "Equation" section with two sub-panels: "Model Variables" (an empty text area) and "Operations, Functions and Modifiers" (a list box containing "+", "-", "/", "*", "^", "<", "<="). Below the equation section is a large empty text area. At the bottom, there are fields for "SimVision Name:", "Data File:", and "Description:", along with "Browse..." and "Generate..." buttons.

Figure 1

Creating New Variables

Once at the Edit Variable window there are several fields that need to be completed.

- *Enter the name of the variable.* The name of the variable should ideally be short yet distinguishable, in order to make equations in which it is used more understandable. The variable name can have letters and numbers, and should start with a letter. Variable names are case-sensitive, so you can have, for example, a variable x and a variable X .
- *Choose the type of the variable.* This is done by default however you can at any time change the type of any variable.
- *Set the level of the variable.* The level determines when the variable gets executed within an iteration. It is not necessary to enter a level; if you leave it blank *Blanche* will automatically set a level for the variable.
- *Type in an equation that defines the behavior of the variable.* This is done by typing in the edit window that has an equal sign to the left of it. Text can be typed, or inserted from the **Variables** and **Functions** list boxes by a double-click. The **Variables** window lists the current variables available, and those variables that are of a lower level, and thus can be used at the current time iteration. For example, if there is already a variable x defined, and it is at a lower level, then both “ x ” and “ x_t ” will show up in the variables window. The **Functions** window lists available functions. A description of how to write equations is described in the next section.
- *Select a data file.* This can be done by either typing in a path and file name directly into the space provided, or clicking on the **Browse** button. When the **Browse** button is clicked, a dialog box is opened that enables the user to select a file (this is a standard Windows file dialog box, it should be familiar to anyone who has used Windows before). Data files are assumed to have a “dat” extension. A data file defines the values of the variable at and before the first iteration. For example, if a variable uses another variable at time $t-1$, at the first iteration, the value of the variable in the equation is defined by the data file.
- *Enter a description.* The description of the variable is up to the user to enter, or not enter, any sort of comment that can explain the variable or the equation. There are no rules, and this description is not used. It is merely a space for the user to write whatever he or she thinks would be helpful.

When you are finished filling out the above information, press **OK** to add the variable to the model, and **CANCEL** to close the window without making any changes to the model. Only the name above needs to be filled out, the rest of the information can be filled out at any time by modifying the variable. An example dialog box filled out is shown in Figure 2.

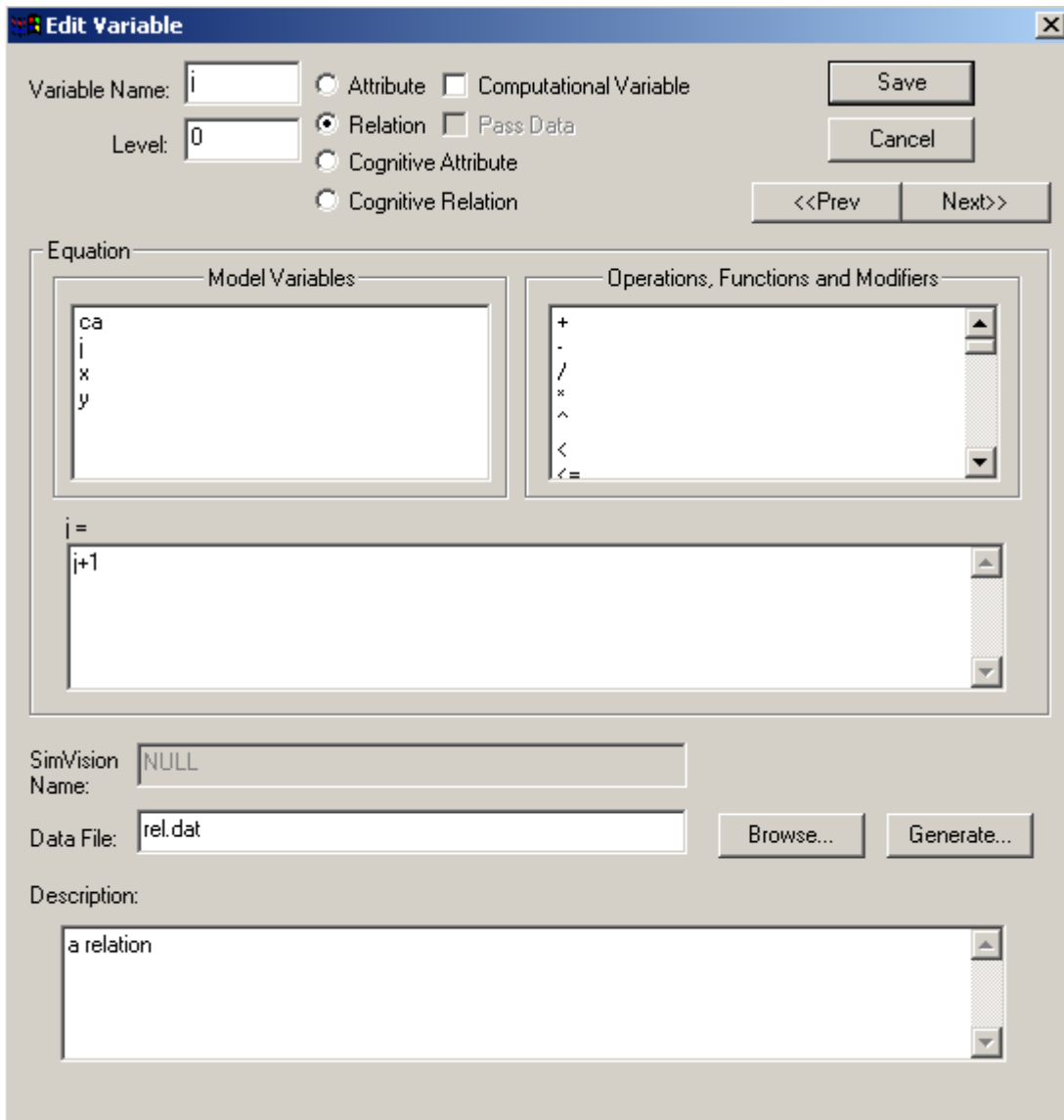


Figure 2

Modifying a Variable

Sometimes a variable needs to be changed, or certain parts of a variable need to be filled out if they were not specified before. To modify a variable, first click on **View** button to select the type of variable desired, then double click on the variable to modify the variable, or highlight the variable desired and click on the **Modify** button.

Once a variable is being modified, the dialog will come up as in Figure 2. Any of the fields can be changed now. When you are done making changes, press **OK** to keep the changes, or **Cancel** to cancel them.

Deleting a Variable

A variable can be deleted from a model if it is no longer needed. After deleting a variable, though, it is important to delete all the references to that variable in other variable's equations. To delete a

variable, first navigate to the variable list of the type desired under the **View** menu and highlight the variable then click the delete button.

Setting the Number of Nodes

The number of nodes is set in the **Model Parameters** window, which can be accessed under the **Simulation** menu. This can be set before or after the variables are created. However once the model has been initialized (run for at least 1 iteration), it will need to be reset before changes to the number of nodes can be made.

Setting the Number of Iterations

The number of iterations is set in the Model Parameters window, which can be accessed under the **Simulation** menu. This can be set before the model is started. Once started, the model needs to be reset before changes are allowed to the number of iterations.

Equations

Sigmas

Some obvious and subtle difficulties arise from translating an equation from standard mathematical symbolic notation into a purely textual format. For example, there are many ways to translate the following equation into text,

$$y = \left(\sum_{j=1}^N 3x_j + 2 \right) - \sin(x_t)$$

In the way *Blanche* interprets equations, only one set of rules applies. The example equation would be translated as “SIGMA (3 * x_j + 2) - SIN(x_t)”. Although there is no single standard way to translate from symbolic notation to text, *Blanche’s* way of interpreting these equations is very close to many accepted methods, without needlessly complicating some matters. Here are some basic rules:

- Functions use parenthesis for their arguments, and commas to separate arguments from each other. If there is only one argument, no comma is necessary. (e.g. LOG(y)). For a random variable with a normal distribution with mean 0 and standard deviation 0.1, we must use a comma to separate both arguments: “NRAND(0,0.1)”.
- The summation symbol \sum is represented by “SIGMA”. Since we are dealing with networks, *Blanche* only needs one sigma operation - the summations across all the person’s links to other people. For example, if we want to sum up everyone else’s x score in the network, we represent this with, in an attribute’s equations “SIGMA (x_j)”, or in a network’s equations, “SIGMA (x_k)”.
- Underscores are used to denote subscripts. For example, x_j translates to “x_j”.
- To denote that a variable belongs to another person we use the subscript “_j”. “j” always represents the other person. If the variable is an attribute, then the subscript can only be used inside of a SIGMA. The reason for this is that there is no single “other person”. There are multitudes, so the expression “_j” is ambiguous. We can only use it when summing across all the links, so the expression “_j” takes on the value of each successive person in the network. In a link, however, there is always a definite other person: the person the link points to. So, “_j” may be used at any time.
- The subscript “_ij” is used to refer to relation and cognitive attribute variables. If we have an equation from a variable I , and we want to simply increment it by one each iteration, the equation would be “I_ij + 1”. We can also refer to another link variable that points to the same person that the current link variable points to. For example, “I_ij + L_ij”, would add the values of I and L , that both originate from person i and point to the same person j .

- Inside a variable the subscript “ i ” refers to the originator of the link, “ j ” refers to the receiver of the link, and “ k ” is used inside of a SIGMA to represent another person. So, if you want, inside a link L , to divide a link variable by the sum of i ’s links to everyone else, you can say “ $L_{ij} / \text{SIGMA } (L_{ik})$ ”. “ k ” becomes in turn every node that is not “ i ” or “ j ”.
- The subscript “ $_{ji}$ ” is only usable in a link variable or a SIGMA. This subscript denotes the person j ’s link to the person whose equation is being run. This is the reverse of the normal “ $_{ij}$ ” link. For example, if there is a link variable called L whose equation is “ $L=L_{ji}$ ”, each person’s value for each link would be switched with the other person’s link to them.
- To denote that a variable should be used at the current time iteration, we use the subscript “ $_t$ ”. Otherwise, the variable is assumed to be at iteration $t-1$. Note that when you use “ $_t$ ”, be careful that whatever variable you are using is calculated at a lower level. Otherwise, by the time the equation is calculated, the variable at the current time step is not yet calculated, so the result will be an error. It is also possible to use equations from several iterations ago, for example “ $_t-3$ ”. Be aware that every iteration before the first one will have the same data (the initial data specified by the data file).
- When combining time and person subscripts, time comes last, and a subscript is necessary for both. So, x at the current time, belonging to person j would be written as “ x_{jt} ”

To sum up the relationship between subscript, attributes, networks, and sigma’s:

- The subscript i always refers to the person whose attributes or links are being calculated. In an attribute, i refers to the owner of the attribute. In a link, i is the originator of the link. An attribute reference, in either a link or a network calculation, does not need “ $_i$ ” to denote that the owner’s attribute is being referenced. That is assumed. Therefore, in referencing an attribute x , “ x_i ” is equivalent to “ x ”. However, the i is always needed in denoting a link, “ l_{ij} ” is not equivalent to “ l_j ”, which would cause a syntax error.
- The subscript j in an attribute can only be used inside a SIGMA expression. Inside that SIGMA expression it refers to each other node successively. In a link equation, j refers to the receiver of the link.
- The subscript k can only be used in a link equation, inside a SIGMA expression. Inside that SIGMA expression it refers to each other node successively.

Function List:

- ABS - Returns the absolute value of its argument.
- AND - The logical “and” symbol. It can be used anywhere, but is usually used in IF statements.
- URAND(min,max) - A continuous distribution of random numbers between min and max. The “min” and “max” identifiers are only used to identify the parameters involved. They should, of course, be substituted with numbers or variables.

- ELSE - The counterpart to IF. In an IF statement, if the IF statement failed, the variables assumes whatever is after the ELSE statement.
- IF - The IF statement is used to do different calculations based on whether a statement evaluates to be true. For example, the equation for a variable “x” might be “IF (x > 3) THEN y / 2 ELSE y / 3”, which would mean, if x at $t-1$ is greater than three, $x = y / 2$, otherwise $x = y / 3$.
- LOG - Basic e logarithm function.
- NRAND(mean,stddev) - A normal distribution with a specified mean and standard deviation. Replace “mean” and “stddev” with the correct values.
- OR - The logical OR function, usually used in IF functions.
- SIGMA - Evaluates the argument that follows, for each step, any variable with a “j” subscript in turn becoming the “j” variable of each successive person in the network. Then, each step is added together to produce the final result for the SIGMA expression.
- SIN - the standard sin function, which computes in radians.

There are also several built in modifiers. A modifier can give a statistic about a variable or another interpretation of it. There are three different types of modifiers: those that give statistics about a variable, those that give statistics about a person’s centrality in a network, and those that give another interpretation of a link in a network. The standard syntax for these modifiers includes the variable name, a period, and then the modifier in all lower case.

Variable Modifiers:

- mean : Usage: v .mean, where v is any attribute or network variable. This will return the mean of the variable v ’s values.
- stddev : Usage: v .stddev where v is any attribute or network variable. This will return the standard deviation of the variable v ’s values.
- density : Usage: l .density where l is a network variable only. This will return the density of network l .
- betweenness : Usage: l .betweenness where l is a network variable only. This will return the network betweenness centrality of l .
- closeness: Usage: l .closeness where l is a network variable only. This will return the network closeness of l .
- min: Usage: v .min where v is an attribute or network variable. This returns the minimum value (signed, not absolute) of v .

- max: Usage v .max where v is an attribute or network variable. This returns the maximum value (signed, not absolute) or v .
- pL: p^* modifier. Usage: l pL where l is a network variable only. This will return the change statistics for the Choice parameter of l when a tie is forced to be present vs. when a tie is forced to be absent.
- PLw(group): p^* modifier. Usage: l pLw where l is a network variable only. This will return the change statistics for the Choice within blocks parameter of l when a tie is forced to be present vs. when a tie is forced to be absent. To use this modifier, an attribute variable of group affiliations of all nodes should also be specified.
- pM: p^* modifier. Usage: l pM where l is a network variable only. This will return the change statistics for the Mutuality parameter of l when a tie is forced to be present vs. when a tie is forced to be absent.
- pMw: p^* modifier. Usage: l pMw where l is a network variable only. This will return the change statistics for the Mutuality within blocks parameter of l when a tie is forced to be present vs. when a tie is forced to be absent. To use this modifier, an attribute variable of group affiliations of all nodes should also be specified.
- pTt: p^* modifier. Usage: l pTt where l is a network variable only. This will return the change statistics for the Transitivity parameter of l when a tie is forced to be present vs. when a tie is forced to be absent.
- pTc: p^* modifier. Usage: l pTc where l is a network variable only. This will return the change statistics for the Cyclicity parameter of l when a tie is forced to be present vs. when a tie is forced to be absent.
- pTwoIn: p^* modifier. Usage: l pTwoIn where l is a network variable only. This will return the change statistics for the In 2-star parameter of l when a tie is forced to be present vs. when a tie is forced to be absent.
- pTwoOut: p^* modifier. Usage: l pTwoOut where l is a network variable only. This will return the change statistics for the Out 2-star parameter of l when a tie is forced to be present vs. when a tie is forced to be absent.
- pTwoMixed: p^* modifier. Usage: l pTwoMixed where l is a network variable only. This will return the change statistics for the Mixed 2-star parameter of l when a tie is forced to be present vs. when a tie is forced to be absent.

Node Modifiers

- indegree: Usage: l _s.indegree where l is a network variable only, and s is the identifier i , j , or k . This will return the indegree of the node s in the network l .
- outdegree: Usage: l _s.outdegree where l is a network variable only, and s is the identifier i , j , or k . This will return the outdegree of the node s in the network l .

- betweenness: Usage: $L_{.s}$.betweenness where L is a network variable only, and s is the identifier $i, j,$ or k . This will return the betweenness of the node s in the network L . This is different than the variable-level network betweenness because it measures the betweenness of a node and not the whole network. If a node identifier is given, this betweenness will be used, otherwise the network level betweenness will be used.
- closeness: Usage: $L_{.s}$.closeness where L is a network variable only, and s is the identifier $i, j,$ or k . This will return the closeness of the node s in the network L . This is different than the variable-level network closeness for the same reason that the two betweenness modifiers differ, as described above.

Relation Modifiers

- dichot: Usage: $L_{.st}$.dichot where L is a network variable only, s is the originator of the link, and t is the recipient of the link, where s and t could be $i, j,$ or k . This returns the link from s to t of the network L , dichotomized by the mean (by default). This then gives a one or zero representation of the link.
- structequiv: Usage: $L_{.st}$.structequiv where L is a network variable only, s is the originator of the link, and t is the recipient of the link, where s or t could be $i, j,$ or k . This returns how structurally equivalent (using the distance formula) nodes s and t are, which is a measure of how much other nodes in the network have similar ties to s and t . This gives a number where the lower the number is, the more structurally equivalent s and t are, with zero being perfect equivalence. There is no set maximum value for valued matrixes.
- maxpath: Usage: $L_{.st}$.maxpath where L is a network variable only, s is the originator of the link, and t is the recipient of the link, where s or t could be $i, j,$ or k . This returns the amount of communication from s to t through the shortest distance (but maximum strength) path.
- rownorm: Usage: $L_{.st}$.rownorm where L is a network variable only, s is the originator of the link, and t is the recipient of the link, where s or t could be $i, j,$ or k . This takes the value of the link from s to t , and divides it by the summed values of all of s 's links. In communication terms, this returns the fraction of time s spends talking to t out of all the nodes s talks to.

Cognitive Functions

- Average: Usage: Average(cognitive attribute/relation). Sums across i and divides by the number of nodes. For an attribute node $i = \text{SUM}(\text{child node } a_i, 0 \leq a \leq G) / G$. For a relation node $ij = \text{SUM}(\text{child node } a_{ij}, 0 \leq a \leq G) / G$.
- Consensus: Usage: Consensus(cognitive attribute/relation, threshold) where the threshold is either a constant or of the same type as the parent. It sums across i excluding the self-report and dichotomizes based on the threshold. For an attribute node $i = 1$ if $\text{SUM}(\text{child node } a_i, 0 \leq a \leq G \ \&\& \ a \neq i) > \text{threshold}$ and 0 otherwise. For a relations node $ij = 1$ if $\text{SUM}(\text{child node } a_{ij}, 0 \leq a \leq G \ \&\& \ a \neq i) > \text{threshold}$ and 0 otherwise.
- Locally Aggregate Structure: Usage: LAS(cognitive relation, Type). Where the type is:
 - ROL. This uses the i th row of the i th view to form a composite.

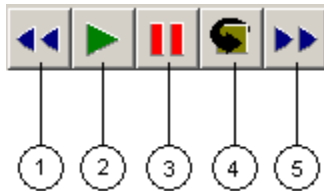
- COL. This uses the i th column of the i th view to form a composite.
- MEAN. This is the average of the two views of the relation. Node $ij = \text{child nodes } (i_{ij} + j_{ij}) / 2$.
- UNION. This will return a 1 if either node i_{ij} or j_{ij} reports a positive relation and zero otherwise.
- UNIONMAX. This will return the most positive of child nodes i_{ij} and j_{ij} or if both are negative the most negative.
- INTERSECTION. This will be 1 if both child nodes i_{ij} and j_{ij} are positive.
- INTERSECTIONEXACT. This will return 1 only if child nodes i_{ij} and j_{ij} are exactly the same.
- INTERSECTIONMIN. This will return the minimum between child nodes i_{ij} and j_{ij} assuming both are positive and zero otherwise.

Time Offsets

Time offsets allow *Blanche* to access previous iteration of anytime of data. Time offsets can be used with any type of variable in any combination with the aforementioned functions and modifiers. The proper syntax for using a time offset is $_(t\#)$, the parentheses and dash must be present. In combination with other modifiers it should be of the form: $\text{name_subscripts_}(t\#)\text{.modifier}$. If the time offset is negative (e.g. using $(t-10)$) it is negative for iterations 0-9) □ *Blanche* will use the initial data provided for that variable from the data file if present or failing that zero.

Running Models

The Menu Bar



1. **Step Back**, this steps the simulation back a certain number of iterations, the exact number is determined by the step size that has been set. The default is one.
2. **Run**, this runs the simulation until it reaches the last iteration, this includes all runs if you have selected multiple runs.
3. **Pause**, if a simulation is running it will stop it.
4. **Reset**, this allows all aspects of the model to be changed and resets all variables to their original data sets.
5. **Step Forward**, this steps the simulation forward a certain number of iterations, the exact number is determined by the step size that has been set. The default is one.

Running a Model

There are several ways to run a model in *Blanche*, clicking on the “Run” or “Step Forward” button will start the model running. Using the “Jump” command under the “Simulation” Menu will also run the model up to the iteration specified.

Average Data

Blanche allows average data to be stored over multiple runs of a model. To enable this feature the box labeled “Store Average Values” must be checked in the Model Parameters dialog, which is accessible under the “Simulation” menu. Once this feature is enabled average value will be displayed only when the “View Avg. Values” option is selected in the “View” Menu. The word “Average” will be displayed towards the lower right hand side of the *Blanche* window when average values are being displayed. When the average values are being displayed, *Blanche* treats this data as the default data set so that visualizing, saving, or loading data will deal with just the average data.

Sharing Data

Blanche allows data to be passed between other simulation programs and *Blanche* itself. To enable this feature the “Enable Docking” button must be checked in the Model Parameters dialog box. This will activate the “pass data” and “SimVision Name:” boxes in the Variable editing window. Checking the “pass data” box will cause that variable to be passed to the docking program. Specifying a name in the “SimVision Name” box will cause *Blanche* to try to retrieve a variable of that name from the program it is docked with. While docked, *Blanche* and the other program take

turns running, so while the other program is running *Blanche* must be idle (or locked). To do this a file called blanchelock.txt is created. While this file exists, *Blanche* will not simulate anything. This file and all others associated with docking are created in the directory specified in the Model Parameters dialog box.

Visualizing Models

Graphing a Model

Blanche is able to graph many aspects of a model with respect to time in order to visually demonstrate the evolution of the network over time. Selecting “Graph” under the “View” menu will access the graph menu. In the graph menu a whole set of variables can be selected or only individual item and/or network metrics. Clicking “OK” will generate a graph of all the items on the right hand side of the window.

Using the Network Visualizer

If a static graph is insufficient, *Blanche* also offers a dynamic Visualizer that spatially represents links between nodes. To access the Network Visualizer select “Visualize” from the “View” menu. This will open a dialog box allowing the selection of the attributes and relations to be visualized. Clicking on “OK” will open an external Java window that will step through the model showing the changes in the model at each iteration.