

**An Object-Oriented Approach to Modeling Enterprise Processes as Directed Graphs
Using SCL Lists and SAS v6.11 Widgets**

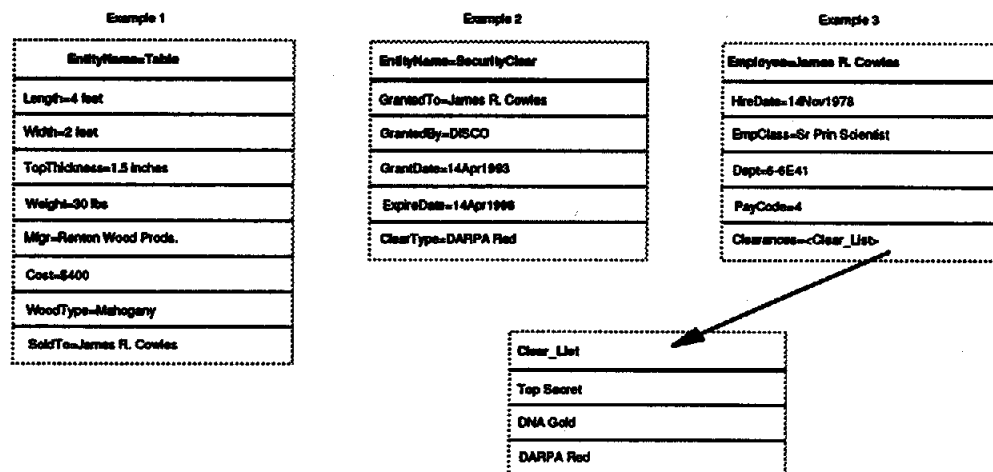
by

James R. Cowles

In order for an enterprise to remain viable in an increasingly competitive world market, an uncompromising commitment to quality is necessary, irrespective of the nature of the product. Perhaps more so than at any time in the past, the very survival of a business depends on its ability to achieve in practice the kind of quality that its management preaches in theory: quality that is achieved repeatedly, reliably, and quantifiably. But as American business, especially the manufacturing sector, has discovered in recent years, this goal of consistent and repeatable quality in any product cannot be achieved through traditional methods of end-item inspection. For by the time a manufacturing problem has been detected through end-item inspection, the mistake has already been made and quality has already been compromised. Instead of quality control methods that attempt to inspect quality into an already finished end-item, consistent and repeatable quality can only be achieved through intimate and quantifiable knowledge of the manufacturing process itself that allows the enterprise to build quality into the end item as far "upstream" as possible in the manufacturing process. This detailed knowledge of the manufacturing process presupposes the ability for an enterprise to model these processes in such a way that relevant information about each process is captured, and that is easily adaptable to the changing demands of the business and technological environments. This paper outlines a general method of using SCL lists to model business processes and of representing these processes graphically using widget classes available with SAS v6.11.

We begin by giving a more or less rigorous, but still somewhat heuristic, definition of what we mean by the term "entity" and when we say that one entity "influences" another. An *entity* E may be thought of as a collection of predicates or attributes (these terms will be used synonymously), say, $\{P_1, P_2, \dots, P_N\}$.

An Entity Is A Collection Of Quantifiable Attributes



The foregoing examples illustrate a couple of important points with regard to defining entities as collections of attributes contained in SCL lists. Example 1 is straightforward: the entity "Table" comprises eight attributes named "Length," "Width," "TopThickness," "Weight," etc. having the values, respectively, "4 feet," "2 feet," "1.5 inches," etc. Comparing examples 2 and 3 illustrates a practice that will be crucial in the subsequent descrip-

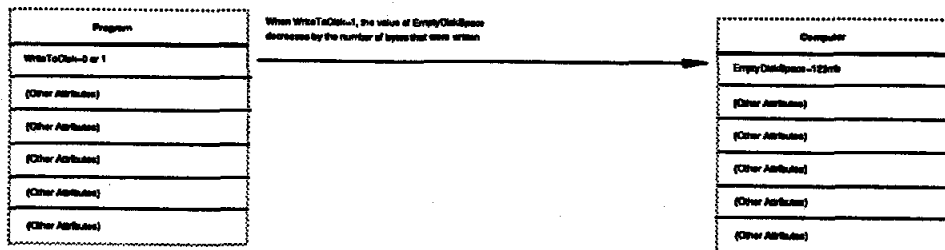
tion of the use of sublists within an entity's SCL list description. In example 2, the entity called "SecurityClear" can accommodate only a single (text) value for the attribute named "ClearType." However, if a person has several security clearances, it would be easy to accommodate this case by defining the attribute named "ClearType," not as a simple text variable, but as an SCL sublist called "ClearList," which would contain a pointer to still another SCL list that contains the person's security clearances.

The value of p_i , $V(p_i)$, is such that when substituted into any sentence, the sentence "Attribute p_i of entity E has the value $V(p_i)$ " is true or false. So for example, if E is a C program, then one attribute might be "Compile_Status" which could assume the values such as "Not Yet Compiled," or "Compiled Successfully," or "Compiled With Warnings," or "Fatal Compilation Error." We shall denote an entity E having properties $\{p_1, p_2, \dots, p_N\}$ as $E(p_1, p_2, \dots, p_N)$. An entity $D(p_1, p_2, \dots, p_M)$ is said to *influence* another entity $E(q_1, q_2, \dots, q_N)$ if and only if there exists a function R (called the *influence relation*) which maps the values of the D predicates to the values of the E predicates such that $q_j = R(p_i)$, for some $i=1, 2, \dots, M$ and for some $j=1, 2, \dots, N$. Intuitively, this means that entity D influences entity E if and only if at least one attribute of E has its value determined by at least one attribute of D. In this case, the notational convention shall be: $D(p_1, p_2, \dots, p_M) \rightarrow E(q_1, q_2, \dots, q_N)$, or more often just $D \rightarrow E$. So both conceptually and notationally, this idea of one process influencing another lends itself very well to being represented as a directed graph. Two examples are given below.

Definition Of Influence Relations

Entity A is said to *influence* entity B if and only if the value of at least one B-attribute is a function of the value of at least one A attribute. For example:

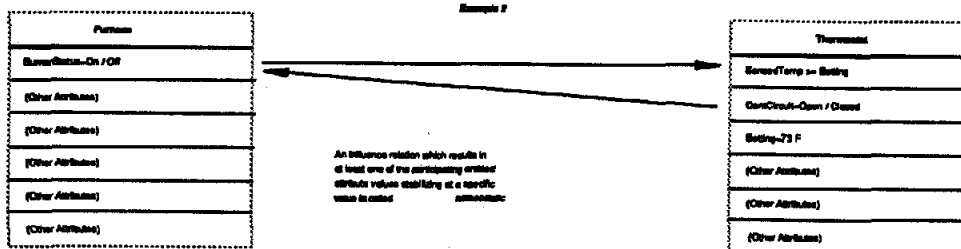
Example 1



Definition Of Influence Relations

The influence relation between two entities is said to be *reciprocal* if A influences B and vice versa. For example:

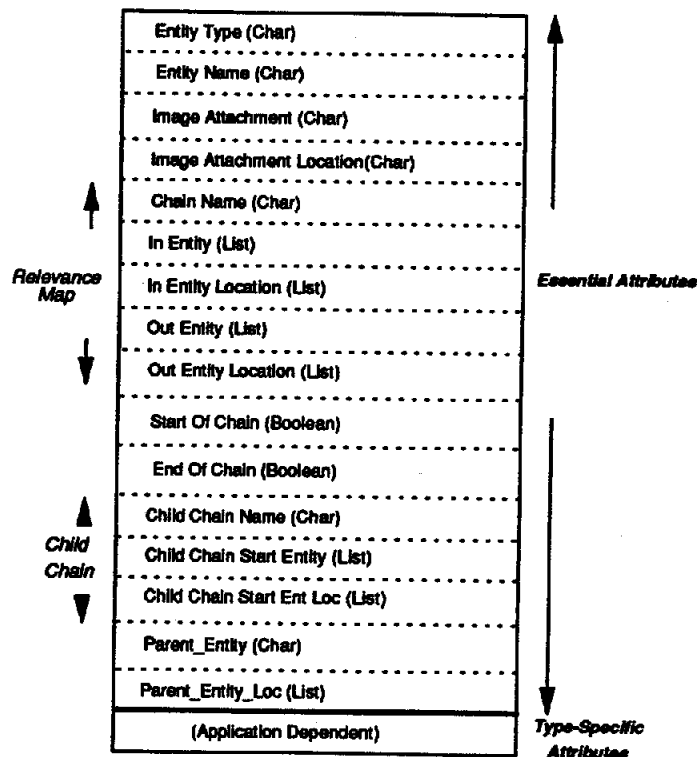
Example 2



Vis a vis the preceding definition of the term "influence relation" and the accompanying example diagrams, it should be noted that the function defining the influence relation is just that: a function. No assumptions are made about the nature of this function. Both its domain and range may be either numeric or character. Its values may be deterministic, or the function may describe some statistical or stochastic process.

Given an entity $E(p_1, p_2, \dots, p_N)$, it is important to note that, while in some important sense all attributes p_i may be created equal, some are more equal than others. In particular, and within the context of the process modeling paradigm outlined in this paper, the set of predicates $\{p_1, p_2, \dots, p_N\}$ may be decomposed into two distinct and disjoint subsets: essential attributes and type-specific attributes. *Essential attributes* are those that are shared by every entity in the business enterprise. In other words, essential attributes are the instance variables of the entity class. *Type-specific attributes* are instance variables specific to subclasses of the entity class.

Entity Structure, Instance Variables, and Instance Variable Types



For example, considered as entities in an enterprise process model, the entities DRILL_PRESS and DATA_BASE, as subclasses of the ENTITY class, would share a common set of essential-attribute instance variables, but would certainly have different sets of instance variables representing the type-specific attributes pertaining to their particular subclasses. Of particular interest in developing the present SAS-based process modeling paradigm are the following sixteen essential predicates: Entity_Type, Entity_Name, Image_Attachment, Image_Attachment_Loc, Chain_Name, In_Entity, In_Entity_Loc, Out_Entity, Out_Entity_Loc, Start_Of_Chain, End_Of_Chain, Child_Chain_Name, Child_Chain_Start_Entity, Child_Chain_Start_Ent_Loc, Parent_Entity, and

Parent_Entity_Loc. The attributes Chain_Name, In_Entity, In_Entity_Loc, Out_Entity, and Out_Entity_Loc constitute the relevance map of <Entity_Name> with respect to chain <Chain_Name>; Child_Chain_Name, Child_Chain_Start_Entity, and Child_Chain_Start_Entity_Loc, the (possibly empty) sublist Child_Chain. (Please note that for the sake of clarity, I am deliberately violating the eight-character length limit for SCL list names. Actual list names would be appropriately chosen by the developer.) The function of these essential attributes will become evident as we define what we mean by a process chain.

Two or more entities D and E are said to constitute a *process chain* (synonymously: *P-chain* or simply *chain*) if and only if one entity in the chain influences the other, that is, if and only if $D \rightarrow E$ or $E \rightarrow D$. Now suppose we have the following set of entities with the corresponding influence relations, as above, indicated by the arrows:

$$C: D(p_1, p_2, \dots, p_L) \rightarrow E(q_1, q_2, \dots, q_M) \rightarrow F(r_1, r_2, \dots, r_N)$$

Furthermore, let us call this chain C. Now consider the second entity in the chain, $E(q_1, q_2, \dots, q_M)$. Given that we are considering entity E and not entities D or F, we say that entity E is the *focus entity* with respect to entities D and E in chain C. Also, with respect to chain C, we call entity D the *in-entity* with respect to E in chain C, and entity F the *out-entity* with respect to entity E in chain C. That is to say, in chain C, the values of the D attributes influence the values of the E attributes, and that the values of the E attributes influence the values of the F attributes. For a given focus entity, and with respect to a given chain in which it participates, the in-entity is the entity which influences the values of the focus entity's attributes; the out-entity, the entity whose attributes' values are influenced by the attribute values of the focus entity. Of course the previous sentence also applies *mutatis mutandis* if, as will often be the case, a given focus entity may have more than one in-entity and/or more than one out-entity.

It should be noted that in the previous paragraph, two phrases that frequently recurred were "in chain C" and "with respect to chain C." These qualifications are critical because entities D, E, and F may participate in several different chains, and their influence relations may be different for each chain. For example, in addition to chain C defined above as:

$$C: D(p_1, p_2, \dots, p_L) \rightarrow E(q_1, q_2, \dots, q_M) \rightarrow F(r_1, r_2, \dots, r_N),$$

consider two other chains C_1 and C_2 which also involve entities D, E, and F:

$$C_1: E(q_1, q_2, \dots, q_M) \rightarrow D(p_1, p_2, \dots, p_L) \rightarrow F(r_1, r_2, \dots, r_N)$$

$$C_2: F(r_1, r_2, \dots, r_N) \rightarrow D(p_1, p_2, \dots, p_L) \rightarrow E(q_1, q_2, \dots, q_M)$$

In chain C_1 , E and F are the in- and out-entities, respectively, for entity D, whereas in chain C_2 , these relationships are reversed. So influence relations can, and most often will, change from chain to chain. The reason influence relations can shift from one chain to another is because each chain represents a different process. To recur to the above example, in the process represented by chain C_1 , it may be that attribute $p_j = R(q_i)$, whereas in the process represented by chain C_2 , $q_h = S(p_k)$, for some functions R and S, and for some $j, k = 1, 2, \dots, L$ and for some $i, h = 1, 2, \dots, M$. That is to say, influence relations change because, for different processes, different entities influence each other's attributes in different ways. In fact, if two chains involving the same set of participating entities influence each other according to the same set of attribute-mapping functions, then these two processes are, as we would expect, identical. Consequently, it is in general necessary to specify in- and out-entities, not only with respect to a given focus entity,

but also with respect to a given chain in which all these entities participate. Moreover, with respect to still another chain, say C_3 , only, say, D and E may participate, i.e., with respect to chain C_3 , entity F is independent of entities D and E. *It is in general not possible to specify influence relations between entities without first specifying a particular chain in which the entities in question participate.*

The process modeling paradigm outlined in this paper will accommodate virtually any process-chain topology, even those that are not monotonically linear, e.g., closed loops representing homeostatic processes as in the example above. Of course, in some cases, particularly those involving closed-loop processes, it might be advisable to model the closed-loop component of a process as a separate process chain. Also, in the case of single entities whose attribute values influence themselves, these processes can usually be modeled in such a way that the entity in question is itself decomposed into its constituent processes. The process modeling architecture outlined in this paper accommodates the description of entities as child processes at a lower level of detail. The consideration of closed-loop processes may make a difference notationally, e.g., the double arrow in the two-entity closed-loop process represented by

$$C: D(p_1, p_2, \dots, p_L) \leftrightarrow E(q_1, q_2, \dots, q_M)$$

But C could also be denoted as a monotonically linear process

$$C: D(p_1, p_2, \dots, p_L) \rightarrow E(q_1, q_2, \dots, q_M) \rightarrow D(p_1, p_2, \dots, p_L)$$

Consequently, in the remainder of the discussion, I shall assume without loss of generality that all process chains are monotonically linear.

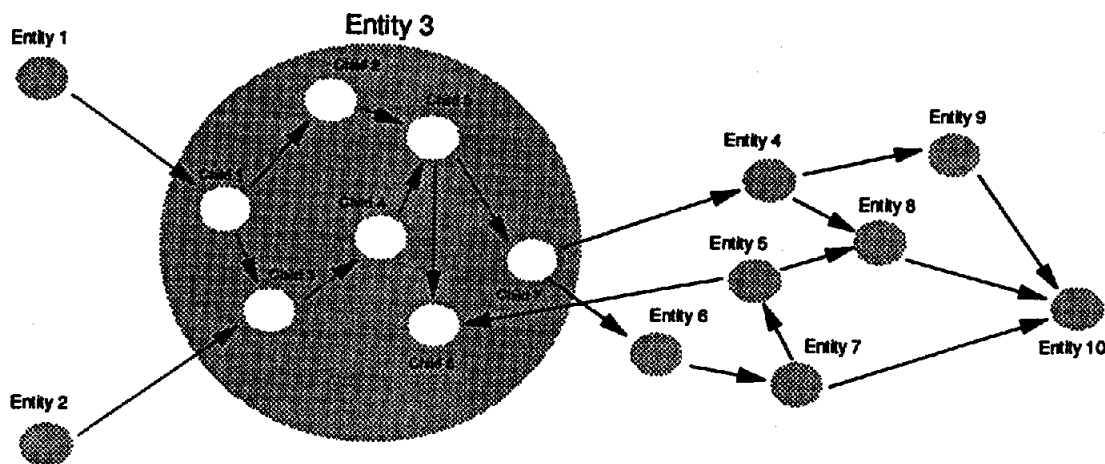
In the SAS system, entities may be represented by a named SCL list. The name of each entry in the SCL list will be the name of the attribute; the value, the value of the attribute. List entries may be of type character or of type numeric. Furthermore, certain entries in the list will themselves be SCL sublists nested within the overall list representing the entity. Now, with the foregoing context established, we are in a position to define all sixteen essential attributes alluded to earlier, and subsequently to define the critical concept of a relevance map. Depending on the nature of its product and business processes, a given enterprise may need more than sixteen essential attributes to define its process entities. Consequently, no inference should be drawn that these sixteen are exhaustive. However, the process modeling paradigm outlined in this paper will need at least these sixteen attributes in order to function successfully. These sixteen essential attributes are defined in the table below. (The entries "Relevance_Map" and "Child_Chain", being pointers to sublists, are not counted as separate essential attributes.)

<i>Attribute</i>	<i>Attribute Type</i>	<i>Description</i>
Entity_Type	Character	This attribute was added to accommodate whatever user/developer-defined entity subclass names may be convenient
Entity_Name	Character	Name of entity
Image_Attachment	Character	File name of bitmap image to be used to represent <Entity_Name>

Image_Attachment_Loc	List	Location of Image_Attachment
Relevance_Map	SCL List	Describes how <Entity_Name> relates to other entities in <Chain_Name>
Chain_Name	Character	Name of chain in which <Entity_Name> participates
In_Entity	SCL List	Name(s) of entity(ies) whose variables influence the variables of <Entity_Name>
In_Entity_Loc	SCL List	Location(s) of entity(ies) that function as in-entity(ies) to <Entity_Name> in chain <Chain_Name>
Out_Entity	SCL List	Name(s) of entity(ies) whose variables are influenced by the variables of <Entity_Name>
Out_Entity_Loc	SCL List	Location(s) of entity(ies) that function as out-entity(ies) to <Entity_Name> in chain <Chain_Name>
Start_Of_Chain	Boolean	"True" if <Entity_Name> is an initial entity in <Chain_Name>, "False" otherwise
End_Of_Chain	Boolean	"True" if <Entity_Name> is a terminal entity in <Chain_Name>, "False" otherwise
Child_Chain	SCL List	If <Entity_Name> is itself composed of interacting entities, this item contains necessary and sufficient information for reconstructing this child process. This function is described more fully below.
Child_Name	Character	Name of child process
Child_Start	SCL List	Initial entity(ies) in child process
Child_Start_Loc	SCL List	Location of initial child entity(ies)
Parent_Entity	Character	Virtual entity (if any) in which <Entity_Name> participates as a child.
Parent_Entity_Loc	SCL List	Location of <Parent_Entity>

The Child_Chain essential attribute is included to deal with those cases where it may be desirable to treat a process as an entity unto itself. In this case, the process becomes a virtual entity in the sense that it has no type specific attributes in its own right, but is rather only a concatenation of lower-level processes. Consequently, in such cases, the type-specific attribute list is empty. The In_Entity list would consist of all those entities whose attributes influence the value of the attributes of the entities listed in Child_Start. Similarly, the Out_Entity list would consist of all those entities in the chain whose attribute values are influenced by the terminal entities in Child_Name. This point is clarified by the diagram below. The Child_Chain in this case is Entity 3, which has as in-entities Entity 1, Entity 2, and Entity 5, and as out-entities Entity 4 and Entity 6. These entity-connectivity values would be duplicated: they would occur in the SCL list description of the constituent entities of Child_Process and also in the higher-level description of Child_Process as an entity in its own right. This duplication would facilitate the depiction of Entity 3, both as a virtual entity and also, at a lower-level of detail, the individual entities and constitute Entity 3. This two-level schema would, in turn, support the ultimate goal of enabling the user to display a diagram of the process, using the SAS v6.11 process flow diagram widget, at both the lower and higher levels of detail. Analogously, the essential attribute Parent_Entity enables the entity denoted by Entity_Name to point "upward" to whatever virtual entity, if any, is the parent of Entity_Name. So, in the example below, entities Child 1, Child 2, ... ,Child7 would all have entries Parent_Entity="Entity 3", whereas all other entities would have blank entries for Parent_Entity.

Parent-Child Relationships In A Process Chain



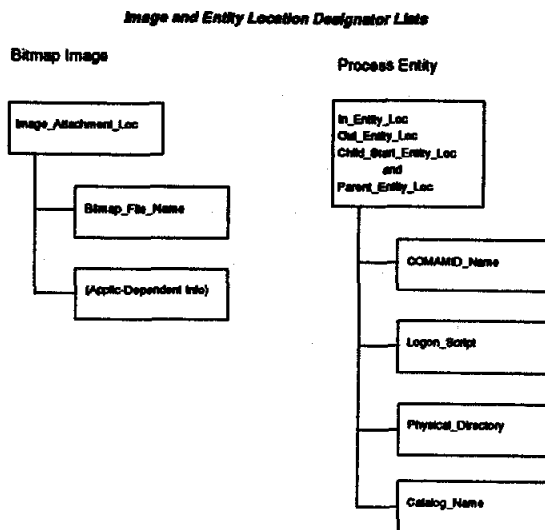
psasware.chilren.pro

6

In order to enable the process modeling paradigm outlined in this paper to function in a distributed computing environment, the essential attributes ending in the text string "_Loc" have been added. In the absence of a particular enterprise or process, it is possible only to describe in rather generic terms what these SCL sublists would contain. In the case of "Image_Attachment_Loc," the contents of this list would simply consist of the fully qualified file name containing the bitmap image to be associated with the entity in the SAS v6.11 organization chart object, together with whatever other information might be needed for an application to establish a connection to a remote site and find that bitmap-image file (e.g., server IP address, etc.). In the case of entities, the intent is to provide SAS/Connect with a "map" to follow for whatever SAS scripts the user may write to maintain, process, and interrogate entities and process chains. However, for both bitmap images and entities, the "_Loc" attribute would be of type list. At a minimum the "_Loc" attributes for entities should contain the following:

- (1) The SAS/Connect communications access method ID that goes in the OPTIONS COMAMID=' statement.
- (2) The fully qualified file name of the script to be used with the SAS/Connect SIGNON statement
- (3) The user would access the entities' SLIST entries by defining a SAS libname that points to a physical directory containing these SLIST entries. The "_Loc" list would have to contain the name of this physical directory.
- (4) Finally, the "_Loc" list would also contain the name of the SAS catalog, under the SAS library defined in (3), that contains the desired SLIST entry.

The structure "Image_Attachment_Loc" and the "_Loc" attributes for entities are depicted schematically below:



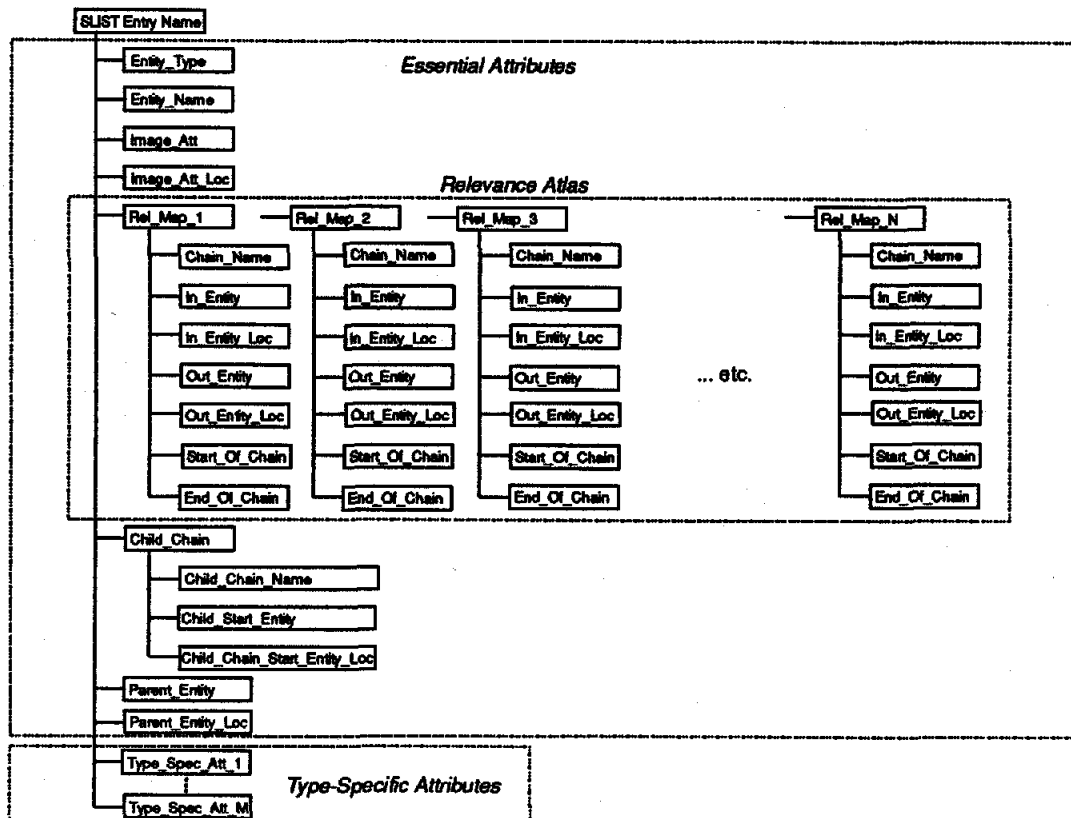
With these parameters defined, the user could, in the case of `Image_Attachment_Loc`, download the desired bitmap image to be associated with `<Entity_Name>`. In the case of entities' `"_Loc"` attributes, the user could either employ SAS/Connect Remote Library Services (RLS) or data transfer services (PROC UPLOAD and PROC DOWNLOAD) to access the SLIST entries containing the desired entity descriptions. In either case, the intent of the `"_Loc"` attribute is to facilitate the implementation of the process modeling architecture described in this paper in a distributed environment. In practice, of course, it would be desirable to keep all such SLIST descriptions in some rough proximity to where those SLIST descriptions are most often used. However, this is an implementation-dependent issue that cannot be addressed at the level of generality of this paper.

The intent of the `"Image_Attachment"` attribute is to function as input to a SAS v6.11 organization-chart representation of the entity. For example, if `<Entity_Name>` was "Drill Press," then `"Image_Attachment"` could be the name of a bitmap image or icon of a drill press contained, say, in file "DRILL.BMP" resident at the location pointed to by `Image_Attachment_Loc`. In any "real-world" realization of this process modeling paradigm, it would be very advantageous to be able to construct a hierarchical representation of an entity, to be able to expand and to collapse the hierarchy in order to view the structure of the entity at successively lower and higher levels, respectively. In this representation schema, each character or numeric attribute would constitute a peer level of the entity-structure hierarchy, while each nested list would constitute a separate child level with its own hierarchical structure among its children, and so on for all the entity's attributes. It would also be advantageous to be able to attach a descriptive image and text to each lower level of the hierarchy. All these goals are very easy to accomplish using the SAS organization chart widget available in release 6.11. The structure of any entity's SCL list representation can be mirrored in a separate SCL list, contained in the WORK catalog, that would drive the organization chart widget. This second list would be identical to the entity's list, except that the sublists of the WORK list would be called "CHILDREN," per the requirements for SCL-list-based organization charts outlined in that section of the SAS *Object Dictionary*. Sublist entries called, as above `Image_Attachment` and `Image_Attachment_Loc` would contain, respectively, the bitmap images and the locations thereof that would be used to label the various sublist levels of the organization chart hierarchy. In fact, a Frames application to be used as an entity-editing front end could incorporate a read-only organization chart widget that would allow the user to see a dynamically changing pictorial representation of the entity structure as that structure was updated "on the fly."

Particular attention should be paid to the attributes in the foregoing table that are part of the Relevance Map SCL sublist. These attributes constitute a sublist of the SCL list representing the entity designated by `<Entity_Name>`. Since, as discussed previously, the in-entity and/or out-entity entries will be different for different chains, reflecting the different influence relations among different entities for different processes, each entity's defining SCL list of attributes will contain a collection of such sublists consisting of the attributes `Chain_Name`, `In_Entity`, `In_Entity_Loc`, `Out_Entity`, and `Out_Entity_Loc` - *one such sublist for each chain in which the entity participates*. Each such individual sublist is referred to as an entity's *relevance map* for that entity with respect to `<Chain_Name>`. Each relevance map sublist will have its own list name as an entry in the SLIST description for `<Entity_Name>`, say, `Rel_Map_1`, `Rel_Map_2`, ..., `Rel_Map_N`. This collection of such sublists is referred to as the entity's *relevance atlas*. Finally, consider the attributes `Start_Of_Chain` and `End_Of_Chain`. These are not strictly

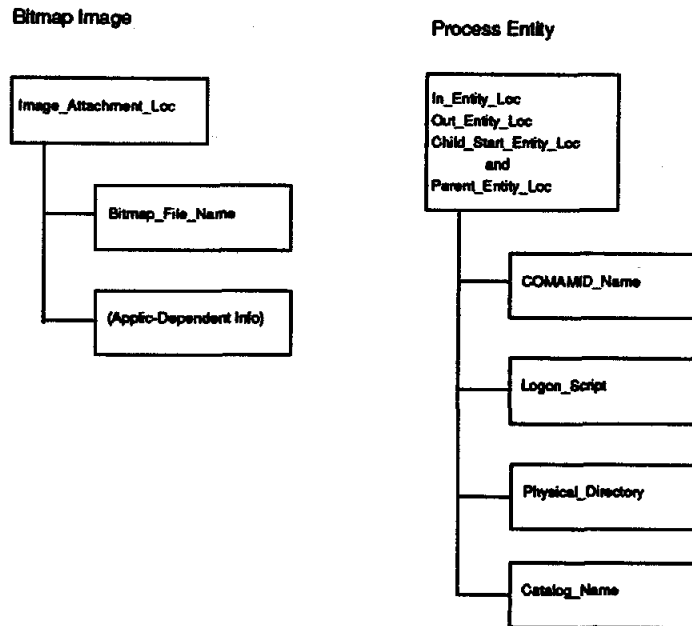
necessary, and are added for convenience. When processing a chain, an application should be intelligent enough to know that it is at the beginning of a process chain when the attribute `In_Entity` is blank, and similarly, that it is at the end of the chain when the attribute `Out_Entity` is blank. However, it should simplify application development somewhat if the application logic could make this determination by simply checking the value of a single Boolean variable resident in the relevance map. One possible disadvantage to incorporating `Start_Of_Chain` and `End_Of_Chain` is that the inclusion of these attributes could slightly complicate the maintenance of the enterprise process library. When a chain was revised to add entities at the beginning or to delete entities at the end, the entities that were initial entities prior to the change would no longer be initial entities, and would have non-blank entries in the `In_Entity` field. Consequently, the chain-modification software would have to incorporate logic to revise `Start_Of_Chain` values accordingly. Similar remarks apply to maintaining `End_Of_Chain` values. This process is known as *chain-structure integrity checking*. For a given implementation of the process modeling architecture describe in this paper, the user would have to determine whether the incorporation of `Start_Of_Chain` and `End_Of_Chain` attributes would save more work on the chain processing end that it adds on the chain-integrity checking end. This is an implementation-specific decision. The following figure gives a schematic representation of the structure of an entity's SLIST description, as it would be depicted in a SAS v6.11 organization chart widget. (The relevance maps `Rel_Map_i`, $i=1,2,\dots,N$, are peer entries in the entity's SLIST description; space limitations on the page necessitate the rather awkward way of picturing them.)

Schematic SAS v6.11 Organization Chart Representation of Entity Structure



We recall here the structure of the respective "_Loc" attribute lists for bitmap images (Image_Attachment) and for entities:

Image and Entity Location Designator Lists



Now, consider once more the process chains referred to earlier:

C: $D(p_1, p_2, \dots, p_L) \rightarrow E(q_1, q_2, \dots, q_M) \rightarrow F(r_1, r_2, \dots, r_N)$.

C1: $E(q_1, q_2, \dots, q_M) \rightarrow D(p_1, p_2, \dots, p_L) \rightarrow F(r_1, r_2, \dots, r_N)$, and

C2: $F(r_1, r_2, \dots, r_N) \rightarrow D(p_1, p_2, \dots, p_L) \rightarrow E(q_1, q_2, \dots, q_M)$

Consider also entity D, which participates in all three chains. If these three chains are the only processes entity D participates in, then the relevance atlas for entity D would consist of three sublists:

Chain_Name: C

In_Entity: _blank_

In_Entity_Loc: _blank_

Out_Entity: E

Chain_Name: C1

In_Entity: E

Out_Entity: F

Chain_Name: C2

In_Entity: F

Out_Entity: E

(Location attributes for C₁ and C₂ have been omitted because they are so highly implementation dependent.) Note that for process chain C, In_Entity and Out_Entity are both blank. This is because, with respect to chain C, entity D is the first entity in the chain. In this case, entity D is said to be the initial process segment. If for still another chain C₃, entity D has in-entities E, F, and G, and out-entities H, J, and K, then the relevance atlas for entity D would include an entry like the following:

Chain_Name: C₃

In_Entity: (E, F, G)

Out_Entity: (H, J, K),

where the parentheses denote two SCL lists with the entries shown. The relevance maps of an entity are important because they determine how that entity influences other entities in all chains of which that entity is a part, as well as the direction in which the influence goes, that is, which entities are relevant to understanding the influence relations among the entities of a given process.

Consequently, in the example above, the SLIST entry representing entity D would comprise four sublists that define how entity D interacts with the constituent entities of process chains C, C₁, C₂, and C₃. Once the previously outlined data structures are in place to represent individual entities, it is possible to construct a directed-graph representation of any process by using the information contained in the SLIST entry, and the Nth entity in the process would contain pointers back to the entities whose attributes influence it (i.e., the in-entities), and forward to the entities whose attribute values it influences (i.e., the out-entities). Moreover, since the in-entity and out-entity attributes are of type list, a given focus entity can accommodate any number of in-entities and out-entities, up to the hardware limitations of the particular platform being used, since the SAS system places no limits on the number of nested sublists a given list may contain. The construction of a process flow diagram from its constituent entities would take place in the following steps (<pdf_name> denotes the name of the process flow diagram):

- (1) Read the Nth entity in the process chain
- (2) Execute a GETITEML to access the Out_Entity list entry in the Nth entity
- (3) For each entity in the Out_Entity list, do the following:
 - (3.1) Execute a CALL NOTIFY(<pdf_name>,'_ADD_NODE_', ...) step to add a node corresponding to each out-entity
 - (3.2) Execute a CALL NOTIFY(<pdf_name>,'_ADD_ARROW_',....) step from the Nth entity to each out-entity
 - (3.3) Use the labeling information (e.g., Entity_Name) to label each new node as it is created
- (4) Repeat steps (1)-(3) for all branches of the process flow diagram, incrementing N,

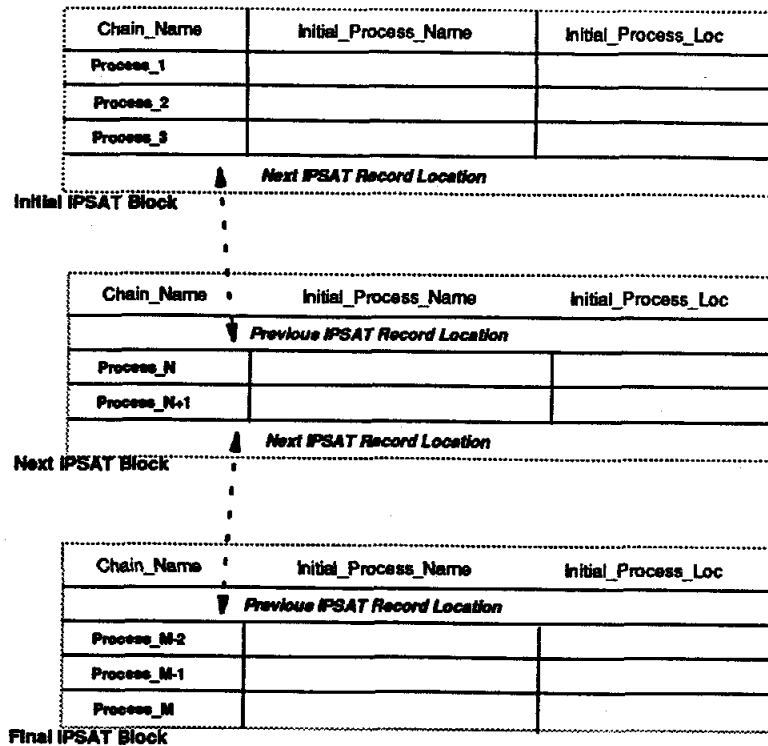
until, for any given branch, one of the following conditions is true:

- (4.1) the focus entity has an empty out-entity list (open chain) (if the variables Start_Of_Chain and End_Of_Chain are being used, this criterion will be satisfied when End_Of_Chain variable has the value "True"), or
- (4.2) there exists a previously accessed entity in the chain whose in-entity is an out-entity (closed chain)

Note that it is not necessary to know how many entities are in a given chain in order for this methodology to work, and that it is never necessary to have more than a single entity's complete SLIST representation in memory in order to drive the process flow diagram widget in the manner outlined above. At most, it would be necessary to have a list of out-entities in memory, together with the entity name of the entity currently being added as a node to the process flow diagram. Note also that, if SAS/Connect Remote Library Services were used to access the constituent entities in process chain, it would not even be necessary to have a copy of any remote entity on the local server or the local client. Finally, note that it is easy to add an entity at any point in a process chain. Merely define the in-entities and out-entities of the entity being added so as to connect that entity with previously existing entities in the chain, and steps (1)-(4) execute as before. Removing an entity from a process chain is equally easy: the revised chain's integrity is preserved by ensuring that the remaining entities still connect to one another via revised influence relations among the remaining entities' attributes.

The construction of a process flow diagram of a given process from its constituent entities and their associated influence relations may be thought of as analogous to the way a PC's operating system reconstructs, say, a text file from its constituent pieces, which may be scattered in various places over the disk. In both cases, with entities represented by SCL list entries and with text files on a PC's disk, there exist pointers that point "backward" to the location of the previous piece (the in-entity/entities) and "forward" to the location of the next piece (the out-entity/entities). While it is theoretically true that any given process chain can be reconstructed from the in- and out-entity information, given the name of chain and any entity in the chain, it would as a matter of practical fact be very highly desirable to have some record, for each process chain, of where the initial entities in that chain are located. This information may be contained in an *initial process segment allocation table (IPSAT)*. The IPSAT in this process modeling schema is therefore functionally analogous to a file allocation table (FAT). Like the FAT, the intent of the IPSAT is function as a map so that chain-processing applications can be aware of which entity(ies) of a given process chain are located where. Of course, "where" in the case of a FAT translates to "which area of the disk," whereas "where" in the case of this process-modeling architecture translates to "which platform communicated with using what protocol to access which physical directory represented by which logical SAS libname." Given a specific process chain, and the Nth entity with-in that chain, it is always possible to find the N-1st entity and the N+1st entity recursively for $N > 1$; the IPSAT starts this recursive chain-reconstruction process by telling a given application where the initial ($N=1$) entity is located.

Structure Of Initial Process Segment Allocation Table



Each IPSAT record should comprise an SCL list minimally consisting of: (1) a character variable denoting the name of the chain, (2) an SCL list containing the name(s) of the initial entity or entities of that chain, and (3) the location(s) of the particular SLIST entry(ies) representing that initial process entity(ies). (In most cases, of course, it would be very useful to store other items of information in the IPSAT, e.g., the number of entities in a given chain, the total disk space occupied by all the SCL lists representing all the entities in the chain, etc.) Items (2) and (3) must be of type list because, in practice, a process will seldom have a single entity as its initial segment. Note that under this process modeling paradigm, to uniquely specify a chain it is necessary and sufficient to specify only: (1) the chain name, and (2) the initial entity(ies) of the chain. Constructing the complete chain from this initial data is only a matter of reading the in- and out-entity information for that chain that is contained in the sublists of each constituent entity's relevance map. Again, in order to take advantage of potential distributed architectures, the IPSAT should be structured so that pieces of it can be located on different servers throughout the enterprise. Consequently, the last record of each IPSAT segment should consist either of the initial segment information previously described in this paragraph, or a "pointer" to the location of the first record of the next IPSAT segment. Analogously, the first record of each IPSAT segment should be either an initial-segment record, or a "pointer" to the location of the last record of the previous IPSAT segment. With this architecture, it would only be necessary to know the location of the first record of the first IPSAT segment in order to reconstitute the entire IPSAT structure.

The traditional nemesis of process modeling is twofold. First of all, in an enterprise complex enough to need

process modeling, the processes involved are so multifarious as to be virtually unmanageable. Secondly, process modeling is an activity that is notorious for being overtaken by events and rendered obsolete. A successful enterprise is a changing enterprise, and in practice this means that the fundamental processes of the enterprise are undergoing periodic revision in response to pressures from the marketplace and from technology itself. Both these sources of change are occurring as the process modeling task proceeds, with the result that the final model usually reflects the enterprise, not as it is, but as it was when the modeling task started. The process modeling architecture outlined in this paper attempts to address both these issues. The emphasis on entities as collections of attributes enables the modeler to "eat the elephant one bite at a time," and the definition of relevance maps and their interrelationships is configured in such a way that, with intelligently designed software as a "front end," the modeler can both create and edit well-defined *pieces* of a given process model without having to recreate the entire process model from the beginning. Moreover, in SAS v6.11 the user can easily depict - and change! - the structure of an entity using the organization chart object and its appropriately encapsulated functions, and similarly can depict - and change! - the interrelationships among entities in a process flow diagram object. Finally, the SAS programming language is of such power and flexibility, particularly those utilities that manage and interrogate the contents of SCL lists, that the user is enabled to concentrate primarily on the process modeling task and only secondarily on the technical details of using the SCL language. The result is that, if the user exercises some professional care in the design and coding of the "front end" software and in the definition of the entities critical to an enterprise, the SAS system enables him or her to create a powerful process modeling tool that faithfully reproduces, not only the critical business processes of an enterprise, but also and with equal fidelity the changes these processes inevitably undergo.