Contents

1 Introduction **1**

2 Petri Nets **3**
2.1 Basic Petri Nets . . . 3
2.2 Colored Petri Nets . . . 5

3 Derivation Nets **6**
3.1 Formal Definition . . . 6
3.2 Execution Rules for Marked DNs . . . 8
3.3 Properties . . . 9
3.3.1 Closure of the Model under Queries and Updates . . . 9
3.3.2 Database Closure . . . 9
3.3.3 Uniqueness of Final Markings . . . 13
3.3.4 Semantics of Queries . . . 13
3.3.5 Semantics of Updates . . . 14
3.3.6 Assertions and their Use . . . 14
3.3.7 Observations . . . 15
3.4 Additional Properties of DNs . . . 16
3.4.1 Reachability . . . 16
3.4.2 Reverse Reachability . . . 18
3.4.3 Reversibility . . . 19
3.4.4 Reachability, Reverse Reachability, and Reversibility . . . 20

4 Implementation **20**
4.1 The Gaea Architecture . . . 21
4.2 Data Derivation Management in Gaea . . . 22
4.3 Operational Characteristics . . . 24
4.4 Data Definition and Manipulation . . . 24
4.5 Integrating Data Analysis Tools . . . 26
4.6 Example . . . 27

5 Discussion **29**
5.1 Limitations of the Model . . . 29
5.2 Comparison with Other Formalisms . . . 30
5.2.1 Conceptual and Functional Modeling . . . 31
5.2.2 Lineage and Versioning . . . 31
5.2.3 Scientific Databases . . . 32

6 Conclusions **33**

Derivation Nets: A Petri Net Model for the Management of Data Derivations in Scientific Experiments*

Nabil I. Hachem, Nina Serrao, Michael A. Gennert, and Ke Qiu
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609-2280, USA
e-mail: hachem,nmc,michaelg,qiu@cs.wpi.edu

Abstract

An important aspect of scientific data management is metadata management. One kind of metadata which needs special attention is data derivation information, i.e., how data are generated. In this paper we propose extensions to current database models to include implicit metadata management.

We introduce and analyze Derivation Nets, an extension of Petri Nets, as a tool to represent and manage data derivation relationships between scientific data, and the procedures and algorithms that derive data. We formulate the Derivation Net model, study its properties with respect to database states, and discuss the semantics of queries and updates within this framework.

Derivation Nets have been used to design the metadata manager portion of the Gaea scientific database management system. Their operational characteristics, implementation, and sample use are described.

*This work was supported by the National Science Foundation under Contract IRI-9116988.

1 Introduction

There are several issues in scientific databases which make conventional database techniques insufficient to achieve the goals of data integration and data sharing [12, 15, 51]. One such important issue is the capture of computations and the management of the data derivation mechanisms, involving scientific data objects.

In a scientific database, data may be classified into two categories: *base data* and *derived data* [17]. Base data is viewed as immutable input data to an analysis system, obtained from well-known sources outside the system. Derived data are generated from base data or previously derived data by applying analysis operators. Note that one system's derived data may be another's base data, e.g., radiometrically corrected remotely sensed imagery may be base data for a Geographic Information System (GIS), but derived data for the instrument scientists. Unlike base data, derived data are not always well understood. One important objective for the efficient management of scientific information is to be able to build on pre-existing knowledge, by sharing both base and derived data.

There is a growing need to manage the algorithms applied to scientific data to derive new data. As there are standard mathematics and statistics libraries available to the general scientific community, so too should there be common and consistent algorithms for all components of data analysis. To accomplish this requires the development of methods to manage the development, evolution, verification, and dissemination of algorithms. Another focus of management is in the scientific experiments themselves. The view of some types of investigation as iterative refinement dictates a need to monitor the progression of experiments to best identify future directions of highest potential. Experiment management also helps avoid unnecessary duplication of experiments and may encourage the reuse of aspects of previously performed experiments in the design of new ones. Finally, to facilitate the dissemination, external confirmation, and verification of results, some form of management is needed. Some branches of science have already identified this need, with standard formats for distributing data and reporting experimental results [1].

In GIS and global change research, studies involve gathering many forms of scientific data. This diversity ranges from tabular data such as rainfall or census reports to raster data such as satellite imagery to vector based cartographic data. In these investigations, scientists may evaluate many classification schemes (principle components, maximum likelihood, linear mixture modeling), and perform experiments over diverse regions at different periods of time. Comparison of regions with similar climatic, socio-economic, or geographic characteristics may reveal heretofore undiscovered relationships or trends. However, inconsistencies between different classification methods may prompt the development of entirely different techniques based on different types of data.

Different scientists may employ different methodologies or apply different algorithms to reach the same objective. In order to make use of the results or data obtained by other scientists, we must have a full understanding of the data derivation history—how they are produced. It is only when such metadata are available that shared data can be meaningfully utilized and interpreted.

Consider the following simple scenario: two scientists are working on detecting changes in vegetation index in Africa between 1988 and 1989. One may subtract the NDVI[1] of 1988 from that of 1989, while another divides the NDVI of 1989 by that of 1988. In this case, if only the resultant images are stored (as in common GISs such as IDRISI [14] and GRASS [45]), there is no way to share and compare the produced data unless the derivation procedures are known to both scientists.

It should be observed that the above problem does not exist in business databases. Data stored in a business database are based on descriptions about an existing enterprise, which are commonly accepted by all the users of the database. This is reflected by the global schema in a business database. In scientific environments, individual researchers may share some information but manipulate it using different algorithms or ad hoc experiments to derive new data, which are added to the knowledge pool. Therefore, it is of absolute necessity to manage the data derivation history in scientific databases.

The main contribution of this paper is Derivation Nets (DN), a model for the capture and management of data and metadata derivations in scientific databases. Derivation Nets are based on an extension and interpretation of Petri Nets (PNs) [36]. This model has been implemented in the Gaea spatio-temporal DBMS for global change research [20, 21, 22]. We describe the structure and behavior of the model and analyze it with respect to database states, and the semantics of queries and updates. Our contribution parallels other efforts such as [7, 9, 16, 43], while addressing limitations of current systems such as [14, 45].

We start by overviewing the basis and some important extensions of Petri Nets in Section 2. Section 3 concentrates on the description of the structure of Derivation Nets, and provides an analysis of their behavior. Specifically, we concentrate on the semantics and stability of the model with respect to database states, and queries and updates. We include a discussion of the essential semantic constructs that are proposed and for which the DN model is constructed. In Section 4 we give a brief overall description of the Gaea architecture. This is followed by a description of the use of DNs within the derivation layer of that system. We provide a typical example of derivation specifications and queries that are possible using DNs. In Section 5, we discuss the limitations of the current model and implementation, and relate our work to others. Finally, Section 6 draws some preliminary conclusions from the work described.

[1]NDVI is the normalized difference vegetation index, a qualitative measure of vegetation derived from satellite imagery data.

2 Petri Nets

2.1 Basic Petri Nets

Petri Nets have been proposed as a modeling tool for systems. They are used to develop a mathematical representations of systems and apply these models to study the behavior and analyze the performance of systems [35, 36]. We have proposed to represent data derivation processes with PNs [20]. The advantages of this approach to model the derivation process are [3, 10, 26, 36]:

- The graphical representation of Petri nets is not only easy to understand but has a well-defined semantics which, in an unambiguous way, defines the behavior of the system.
- PNs have proven to be very useful to describe pieces of intended system behavior where process synchronization is of utmost importance and the behavior of the system needs to be analyzed.
- PNs can be used to represent systems in a top-down fashion at various levels of abstraction, i.e., they can be used to model a system hierarchically.
- Most importantly, PNs are uninterpreted models. Hence they can be used in many different environments by using appropriate interpretations.

Informally, a Petri net is an abstract model of the flow of information and control of actions in a system.

Definition 1 *A Petri net structure is the four-tuple* $C = (P, T, I, O)$ *where*

- $P = \{p_1, \ldots, p_m\}$ *is a set of places.*
- $T = \{t_1, \ldots, t_n\}$ *is a set of transitions.*

The relationship between places and transitions is defined using the input function I *and the output function* O. *For each transition* t*:*

- $I(t)$ *defines the set of input places for transition* t*, and*
- $O(t)$ *defines the set of output places for transition* t. [36].

Definition 2 *A marking* $\mu: P \rightarrow N$ *of a Petri net is an assignment of tokens to the places in the net, with* $\mu(t)$ *the number of tokens at place* t. *A* **marked** *Petri net is defined by the 5-tuple* $M = (P, T, I, O, \mu)$.

The two components of a Petri net are places represented using circles, and transitions represented using vertical lines (Figure 1). Arrows interconnect places and transitions. Tokens (black tiny circles) move from place to place according to specific "firing" rules. Firing rules describe the behavior of the structure of a PN model.

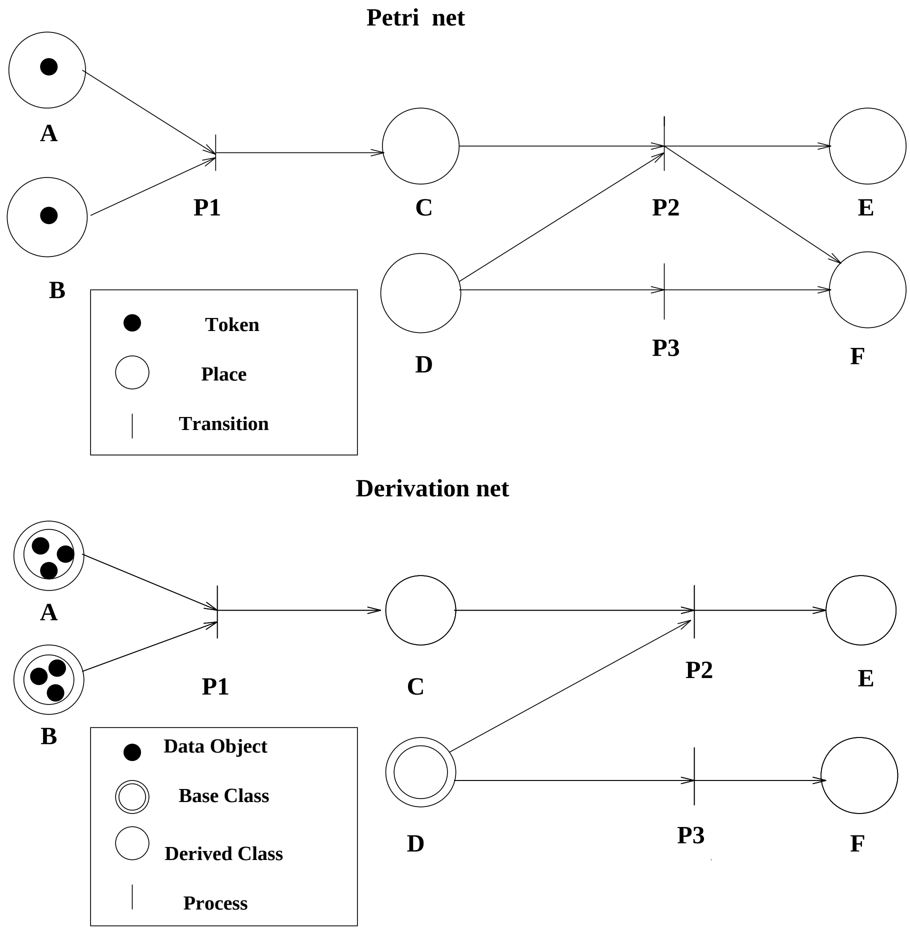


Figure 1: Petri Net and Derivation Net

Definition 3 *The firing rules for a basic PN are as follows:*

- *A transition is said to be* **enabled** *when all of its input places have at least one token.*
- *A transition fires by removing the enabling tokens from their input places and generating new tokens which are deposited in the output places of the transition.*
- *The number of tokens in each place always remains nonnegative when a transition is fired.*
- *Usually, only one of the enabled transitions can fire at a time* [36].

In Figure 1, for example, transition P1 is enabled since its input places A and B have at least one token in them. Transitions P2 and P3 are not enabled as their respective input places do not have a token in them. Transition P1 fires by removing one token from each of A and B and then generating one token in C. The effect of firing a transition is illustrated in Figure 2.

PNs evolved to overcome the limitations of finite state machines [36]. Some of the application areas of PNs are in performance analysis [23, 46, 52], communication protocols [49], asynchronous systems modeling [3, 39], hardware modeling [6, 34, 19], and in hypertext systems [38] among others. Different properties of PNs have been investigated as analysis tools, including *boundedness*, *conservation of tokens*, *safety*, *liveness of transitions* [36], and *invariants* [30].

The implications of the concept of *liveness* can be different for different systems modeled using PN. This concept was developed to deal with deadlock problems in operating systems. It is reducible to the *reachability problem* in a PN and can be used for its analysis [36]. The *reachability problem* is stated as follows: Given a marked Petri net (with marking M) and a marking M', is M' reachable from M [36]?. The *reachability set* of a PN is the set of all states into which the net can enter by any possible firing sequence of its transitions. It is the set of markings of the net.

Many extensions to the basic PN formalism have been proposed. Prominent ones are timed PNs [52], stochastic nets [32], G-Nets [11], PNs with inhibitor arcs [3], and free choice nets [36]. Relevant to Derivation Nets, an interesting extension is high-level Petri Nets such as Colored Petri nets (CP-nets) and hierarchical nets [19, 24, 26].

2.2 Colored Petri Nets

One of the PN extensions that comes closest to our work is Colored Petri Nets [26]. CP-nets attach to each token a token-color and relate the input to output token colors by means of functions. In this way, a large Petri Net exhibiting a regular structure can be collapsed into a much smaller and more easily analyzed CPN.

Consider the problem of modeling two processes using the same set of resources in a similar way using basic Petri Nets. Two separate PNs are needed as a single token can be used to fire transitions in either net but not both. As more processes of the same kind are added to the system the PN will become very large and difficult to comprehend. This occurs with most real-time systems as they often contain many processes that are similar yet distinct. In such situations, Colored Petri nets provide a better form of representation by making use of the *token color*.

In a CPN each token has an attached *token color* drawn from a discrete set. The color can be a complex data type like a structure in programming languages. For a given place all tokens must have token colors that belong to a specified set of allowable colors for that place, called the *color set* of the place. Attaching a color to each token and a color set to each place allows the use of fewer places than would be needed in a basic PN. Color sets in CPN are the same as types in programming languages.

A CP-net consists of three different parts: *net structure*, *declarations*, and *net inscriptions*. The net structure is a bipartite directed graph with two kinds of nodes, places and transitions, interconnected by arcs. The declarations describe the different types of data and variables being used in every process. Net inscriptions consist of names for the places, transitions and arcs, the types of data (color sets) and initialization expressions attached to a place, the *guards* attached to a transition and the *arc expressions* attached to an arc. The purpose of guards is to define additional constraints which must be satisfied before the transition is enabled [26].

The major advantage of using CPNs to model a system is that it is possible to prove that a given system has a set of desired properties, including absence of deadlock, the possibility to return to the initial state and an upper bound on the number of tokens.

3 Derivation Nets

In this section we present and discuss the features of Derivation Nets as a model for managing derived data in scientific databases. We show the closure of the model with respect to the database states, followed by semantics of queries and updates. Finally, we present additional interesting properties used for the analysis of and operations on derivation nets.

3.1 Formal Definition

The various extensions to Petri nets which were reviewed are inadequate for the modeling and decision support capabilities needed for scientific database management. By using appropriate interpretations of places, transitions, tokens and markings; and further by modifying the firing rules of PNs, we propose Derivation Nets, a network model for the derivation process.

Definition 4 *A Derivation Net (DN) is the five-tuple*[2] $D = (C, P, I, O, A)$ *where*

- $C = \{c_1, \ldots, c_l\}$ *is a set of classes (net places); each class encapsulates the structural representation of a single type of data.*
- $P = \{p_1, \ldots, p_m\}$ *is a set of processes (net transitions); each process encapsulates the procedures applied to the classes of data.*
- $I: P \to C^*$, $I(p) \subseteq C$ *defines the set of input classes for process* p,
- $O: P \to C^*$, $O(p) \subseteq C$ *defines the set of output class for process* p[3],
- $A: P \to (C^* \to \{\text{false}, \text{true}\})$, $A(p)$ *is an assertion function (predicate). Process* p *can fire, that is, generate output of class* $O(p)$, *only when the set* $I(p)$ *is marked and* $A(p)$ *is true in the context of* $I(p)$. *Allowing arbitrary expressions*

[2]This definition supersedes the one given in [17]; the DNs of that paper are the *marked* DNs in this work.

[3]The implementation discussed in Section 4 assumes that there is only a single output class per process, but this assumption is not critical to the theory.

to appear in $A(p)$, including conjunctions, permits us to treat multiple assertions as a single assertion.

One of the functions of a Derivation Net, in addition to determining data derivation relationships, is to guarantee that derived data are not rederived unnecessarily during the course of responding to a query. To avoid recomputation of known quantities, data objects are uniquely identified by an object identifier and the tokens that mark a DN are also uniquely identified. We now introduce *tokens* and *marked Derivation Nets.*

Definition 5 *A token in a Derivation Net is an element of the set of tokens defined recursively as*

$$x \in X = C \times (B \cup (P \times X^*))$$

where

- $B = \{b_1, \ldots, b_n\}$ *is the set of uniquely identified base data objects, and*
- X^* *indicates zero or more tokens in the input classes from which a token may be derived.*

A token in a DN represents an *instance of a class* (data object), with the class appearing as the first element of the token. The tokens in a class reflect the instances of that class in analogy with tuples in relations. A token can either correspond to base data or it can correspond to derived data. In the former case, the token has the form $x = (c, b)$, indicating that the data is an instance of base class c. Tokens that correspond to derived data are tagged not only with their class, but also the generating process and the *identities of tokens used to instantiate the process.* Thus, derived data carry along with them their derivation history.

For example, let process p_1 be instantiated with base data b_1 in class c_1 and b_2 in class c_2 to generate a new data object in class c_3. The newly generated token will be $(c_3, p_1, (c_1, b_1), (c_2, b_2))$.

Definition 6 *A marked Derivation Net is a pair $M = (D, m)$ where*

- *D is a Derivation Net, and*
- *$m \subseteq X$ is the set of tokens active in the DN.*

Marked Petri Nets ordinarily define the marking to be a function from places to natural numbers and Colored Petri Nets define a marking as a function from places to the multiset of colors. Our definition of a marking differs in that the tokens themselves carry along class information, which plays the same role as colors in CPNs. It is natural to define functions on marked DNs for the class of a token and the tokens of a class.

$$\text{class}: X \to C, \quad \text{class}(x) = \text{first element of } x$$
$$\text{tokens}: C \to X, \quad \text{tokens}(c) = \{x \in X \,|\, \text{class}(x) = c\}$$

For a given net place c, its marking tokens(c) will consist solely of elements of the form (c, b) or else the elements will have the form $(c, p, x_1, x_2, \ldots)$ where $c \in O(p)$ and class$(x_i) \in I(p)$ depending on whether c contains base data or not. A class c is a base class if it is not the output class of any process.

Definition 7 *A (marked) Derivation Net is an* initial marking *if and only if it is* $M = (D, m_0)$ *and* $m_0 \subseteq C \times B$*, the restriction of the tokens to base classes only.*

An initial marking is the set of instances of data objects in the base classes and is thus an image of a state of the database representing base data (in analogy to extensional data in deductive databases). A DN is illustrated in Figure 1. Places represented with concentric circles are base classes, while the others are derived data classes.

In the remainder of this paper, all DNs may be assumed to be marked DNs unless otherwise stated. When data analysis is non-iterative, we may impose on Derivation Nets the additional condition that they be *acyclic.*

Definition 8 *An acyclic Derivation Net is a DN where there are no cycles in the underlying class/process graph.*

The class/process graph of a DN (C, P, I, O, A) is a bipartite directed graph obtained by considering the vertex set $V = C \cup P$ and the edge set E as a subset of $(C \times P) \cup (P \times C)$ with $(c, p) \in E$ iff $c \in I(p)$ and $(p, c) \in E$ iff $c \in O(p)$. A cycle in the class/process directed graph is a chain of classes and processes $(c_1, p_1, c_2, p_2, \ldots, c_N, p_N)$ where $c_i \in I(p_i)$, $c_{i+1} \in O(p_i)$, and $c_1 \in O(p_N)$. It turns out that for many applications, iterative data analysis is extremely important; we conjecture that most of the propositions below can be extended to cyclic DNs by considering the least fixed point of an infinite family of acyclic DNs.

3.2 Execution Rules for Marked DNs

As tokens represent base or derived database objects, they are not consumed by the firing of a process. Firing a transition, or equivalently, instantiating a process p creates new tokens in the output classes of the process. These objects are the result of actually deriving data using a task as defined in Section 4.1. The new objects can be created if the set of assertions $A(p)$ is satisfied. We modify the Petri Net execution rules as follows for Derivation Nets:

- A process p is instantiatable (enabled) if there exists a set of tokens in each input class $c_i \in I(p)$ of p such that $A(p)$ (assertion for process p) is true (satisfied).

- Each of the instantiatable processes can fire at any time.
- Instantiating (firing) a process p does not remove any tokens from the input classes $I(p)$, but adds one token to each output class in $O(p)$. The newly generated tokens are identified with their class, the instantiated process p, and the identifiers of the input tokens.

Thus, instantiating a process with an identifiable set of input tokens always generates the same identifiable output token. This claim will be proven later.

Formally, if process p is instantiated with objects $\{x_1, \ldots\}$ under marking m_1 to yield marking m_2, then m_2 will differ from m_1 by being augmented by the newly generated tokens. To be precise, $m_2 = m_1 \cup \{(c_i \in O(p), p, x_1 \in I(p), \ldots), (c_{i'} \in O(p), p, x_1 \in I(p), \ldots), \ldots\}$.

Therefore, the equivalent to firing a transition in a PN is instantiating a process in a DN. The major differences between the two is that the latter does not remove tokens from its input places when it generates new tokens in the output places.

Note that it is possible to construct an equivalent PN from a DN, ignoring assertions. The construction is straightforward; for each process p, the output classes $O(p)$ are augmented with input classes $I(p)$. Then the usual PN firing rules apply and every time a transition fires, it replenishes its input set.

3.3 Properties

3.3.1 Closure of the Model under Queries and Updates

The modified firing rules lead us to discuss closure of the markings of a DN. With an initial marking m_0, one asks the following questions: What is the relationship between markings? Is there a state which can be used as a basis to determine if a specific object can be derived? Can we assure that the number of tokens remains enumerable? As processes can fire concurrently, does the final database state depend on the sequence of instantiations? These questions are important and should be answered if the model is to be of any use[4].

3.3.2 Database Closure

Based on the firing rules of DNs, we define a *legal marking* and a *final marking* as follows:

Definition 9 *Given an initial marking m_0, a marking m in a DN is* **legal** *if it is the result of zero or more instantiations of processes from the initial marking.*

Definition 10 *The union of all legal markings is called a* **final marking**.

[4]This has correlates with least fixed point semantics and the closed world assumption in deductive databases.

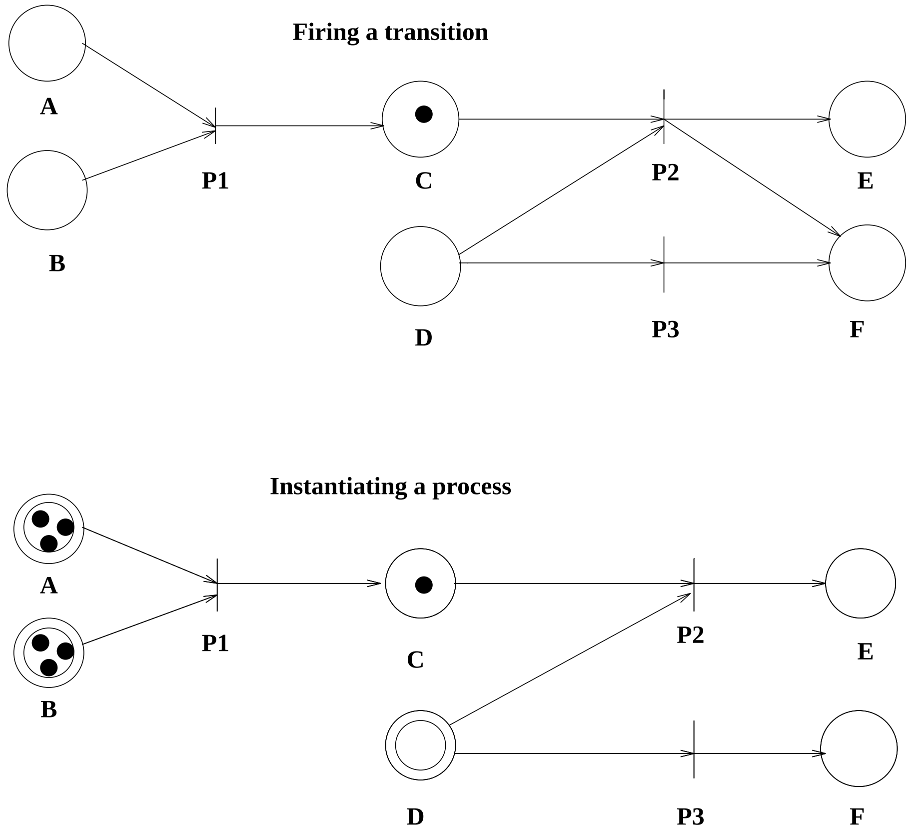


Figure 2: Results of Firings in a Basic Petri Net and a Derivation Net

Consider the DN in Figure 1. The initial marking, m_0 consists of the set of data objects in classes A and B, i.e., a set of three data objects from each of the classes making a total of six data objects. From the legal marking m_0, if P1 is instantiated generating an object in class C, the resultant marking m_1 is a legal marking and is illustrated in Figure 2.

Assume a new data object is added to class C without instantiating process P1. Then the marking $m_1^{'}$ (which includes m_0) is NOT considered to be a legal marking. This is illustrated by the gray token in Figure 3.

Consider the DN in Figure 4. The initial marking m_0 is the set of data objects in classes A, B and D. The marking m_1 is the result of instantiating process P1 from the initial marking m_0, generating an object in class C. The marking m_2 is the result of instantiating process P2 from marking m_1 generating an object in class E. The marking m_3 is the result of instantiating process P3 from marking m_0 generating an object in class F. Assuming for this example that it is not possible for other

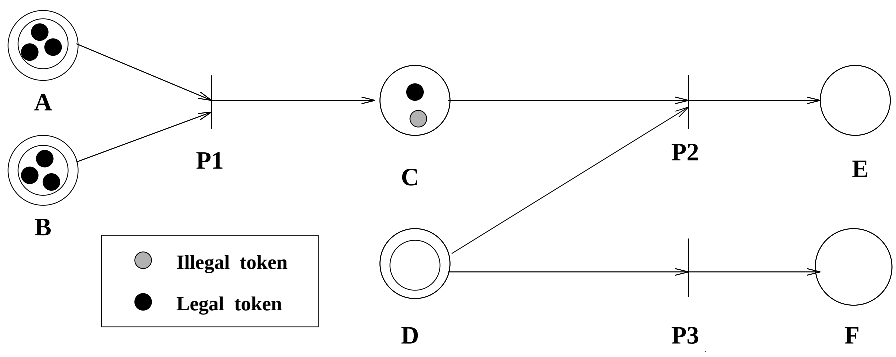


Figure 3: Illegal Marking in a DN

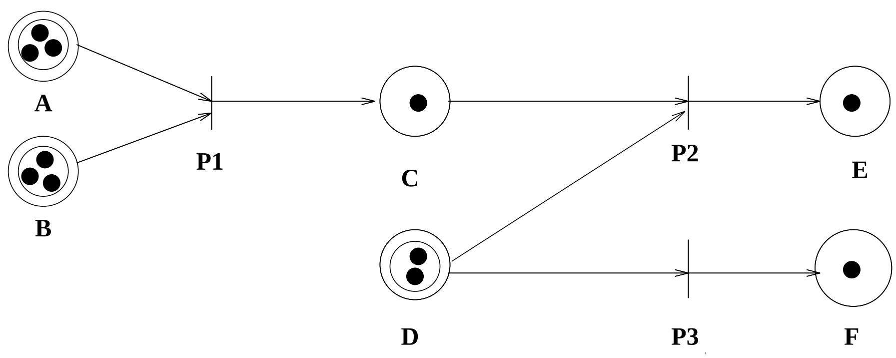


Figure 4: DN with a Final Marking

instantiations to take place[5]. Thus, the final marking, m_f for the DN of Figure 4 is $m_0 \cup m_1 \cup m_2 \cup m_3$.

Tokens represent an abstraction of data objects in the database. If we are to use relational systems and extend them with a DN, then tokens will be individually identified by the keys of the tuples they represent. Using the object-oriented paradigm, tokens may in practice be represented with object identifiers. As tokens are not consumed, then for any firing of a transition, a new legal marking m is generated, with $m_0 \subseteq m$. Since the number of tokens in a class is bounded (finite) over the set of all markings, there is a finite number of markings for an acyclic DN. We restate this as:

[5]It is observed that, as tokens are not consumed, the same input tokens can be reused to instantiate a process as many times as we want. If a new token is derived from a process, any subsequent firing of the same process with the same input tokens will lead to the same output token.

Proposition 1 *The set of possible markings derivable from an initial marking of an acyclic DN is a finite set.*

Proof: Consider the tree constructed by starting with the initial marking m_0 as the root node; children nodes are generated by considering all possible transitions. The size of the tree is limited by the branching factor (the number of active transitions at any node) and the depth of the tree (which cannot exceed the number of processes for an acyclic tree). Thus, the tree is finite.

If the restriction to acyclic DNs is lifted, the tree may be infinite, but enumerable, because the tree depth is enumerable.

Essentially, a marking is a representation of a database state. The final marking is thus the database state that represents all possible objects that one can retrieve, given the initial marking m_0. This final marking m_f completely captures the semantics of the database. The following proposition provides a firm grounding of the relationship between the set of legal markings of a DN.

Proposition 2 *The set of legal markings forms a partially ordered set (poset), ordered by the operation of* **set inclusion** $\subseteq$. *The least upper bound (lub) of the set is the final marking m_f, and the the greatest lower bound (glb) is the initial marking m_0.*

Proof: By the definition of the firing rules for DNs, the marking after process instantiation includes the pre-instantiation marking as a subset, so the poset property hold trivially. The final marking m_f must be the lub because for all other legal markings m it holds that $m \subseteq m_f$. The initial marking m_0 must be the glb because for all other legal markings m derivable from m_0 by a chain of transitions it must hold that $m_0 \subseteq m$ by the transitivity of $\subseteq$.

Illustration: The initial marking m_0 is the glb as it is contained in every marking and the final marking m_f is the lub. The set of legal markings $\{m_0, m_1, m_2, m_3\}$ forms a poset. m_0 is contained in m_1 and m_1 is contained in m_2. Similarly m_0 is contained in m_3. However m_2 and m_3 are not comparable.

The set of legal markings is a pointed complete partial ordering (a poset containing a least element and whose every chain has a lub). Nonetheless, it does not follows that the set of legal markings forms a lattice, because a set of markings may have no unique lub. For example, the marking poset illustrated in Figure 5 generated by a simple DN (not shown) has no unique lub for m_1 and m_2. The complete poset structure of markings is very elegant and very important as it provides us with the **closure of database states**:

Proposition 3 *All possible (legal) database states are derivable from m_0 and converge to m_f.*

Proof: This is a restatement of Definitions 9 and 10. That the set of all legal database states converges to m_f follows from its being the lub of the set of markings.

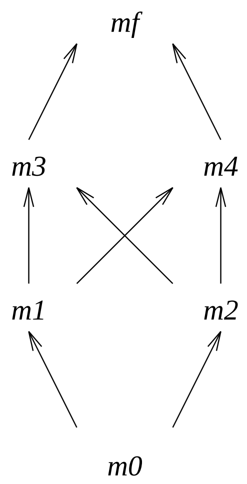


Figure 5: Marking Poset that is not a Lattice.

3.3.3 Uniqueness of Final Markings

Proposition 4 *The final marking derivable from an initial marking is unique and independent of the order of firing of the transitions.*

Proof: The tokens in a marking may be interpreted as functions: tokens in base classes are 0-argument functions (constants) and tokens in derived classes having i inputs are i-ary functions. Firing a transition to add new tokens is akin to performing a reduction on the functional interpretation. The function corresponding to the final marking is a normal form, for which no further reductions are possible. By the Church-Rosser property for function application, the normal form is unique irrespective of the order of reductions. Thus, the final marking is also unique and independent of the order of firing of the transitions.

3.3.4 Semantics of Queries

As m_f is the union of all possible markings, then any object that can be derived from the database using a combination of tokens from m_0 is a token in m_f. A query specified on the database can then be transformed into an equivalent query on m_f.

Proposition 5 *In a DN, a response to a query is a subset of the final marking m_f. Irrespective of the current state of the database, if that state is legal then the answer to a query will be consistent with, and equivalent to, that query with respect to the final marking m_f.*

Proof: The proof follows by induction on the number of process instantiations needed to satisfy a query.

3.3.5 Semantics of Updates

There are two cases of update to consider: derived data updates and base data updates. All updates to derived classes must be performed through the application of derivation procedures; in order to guarantee metadata consistency, it must be illegal for a user to manually perform insertions and deletions to derived classes.

Proposition 6 *Inserting or deleting a derived data object does not alter the final marking.*

Proof: The proof follows from the definition of final marking, which is dependent only on the initial marking developed from base data. Inserting or deleting a derived object is merely a change of state within the poset structure.

On the other hand, base data determines the poset structure, so any modification to it by, for example, adding (deleting) a base token implies a chained addition (deletion) of all possible tokens that could be (have been) generated with that token. This implies modifying the poset structure and the final marking.

Proposition 7 *Inserting or deleting a base data object always alters the final marking, with the new final marking as a superset of the previous final marking in the case of insertion, and a subset in the case of deletion.*

Proof: We prove the claim for the case of insertion; the deletion case is identical. Let m_0 be an initial marking to which base data b is inserted to yield $m'_0 = m_0 \cup \{(c, b)\}$. Clearly, $m_0 \subset m'_0$. By induction on the number of transitions, it must hold that $m_f \subset m'_f$. The two final markings cannot be equal because, at a minimum, token (c, b) will be absent from m_f but present in m'_f.

3.3.6 Assertions and their Use

Assertions in Derivation Nets are similar to *guards* in CP-nets [26]. In CP-nets the guard of a transition is a predicate which must be true before the transition can fire. So far we have ignored the assertions part of a DN. These are part of the process definition (Section 4.1), and are used to provide the following features:

- Guarantee the integrity of data. Conditions on input data can be specified, and through the mapping of a process corresponding conditions are thus defined on output data.

- Guarantee the integrity of data derivation. Some relationship among the input data objects amy be required to obtain meaningful data derivations. For example, a certain process may require its inputs to have the same or overlapping spatio-temporal coverage. This can be expressed in the process template as constraint rules and assertions. Only when such relationships are satisfied will the transition be enabled and fire.

- Assertions can be used to define, capture and express specialized relationships between classes. This differs from the conventional ones used in semantic modeling by being more general, including the ability to define multiple relationships between (possibly the same) classes, pertaining to a specific derivation procedure.

We now consider the implications of the introduction of assertions with respect to the poset structure of the markings of DNs. If no assertions are specified, then instantiating a process is independent of the "nature" of the input tokens. Denote by $P_f = \{m_0, \ldots, m_f\}$ the marking poset structure in this case, which we refer to as the information poset. At the other extreme, we may place an assertion on every transition in a DN such that, for every transition no tokens can be found that will instantiate it. In this case $m_0 = m_f$ and we denote the resultant poset structure by $P_0 = \{m_0\}$. It is straightforward to prove the following:

Proposition 8 *The set of posets that can be generated by starting with no assertions and adding them to a DN, forms a lattice which is partially ordered by the operation of set inclusion $\subseteq$. The least upper bound (lub) of the set is the poset P_f which is generated when no assertions are associated with transitions, and the greatest lower bound (glb) is the poset P_0, which corresponds to the singleton set of markings $\{m_0\}$.*

Proof: Note that the elements of the information structure are themselves posets, i.e., sets of markings. Because the join and meet operations are set union and intersection, respectively, the structure is a lattice. The singleton set of initial markings $\{m_0\}$ is a subset of every poset, thus it is the glb of the lattice. Poset P_f must be the lub because adding assertions to generate another poset P guarantees that $P \subseteq P_f$.

This grounding to a lattice structure of all possible marking posets characterizes the closure of all possible models that can be generated by a DN. The use of assertions does not affect the stability of the model that we propose. It only affects the size of the final marking and thus the composition of the corresponding poset of markings. Assertions are available to enable the scientist to carve a particular poset out of the overall information poset P_f by adding constraints to the DN.

3.3.7 Observations

The DN structure which has been proposed can be extended to a hierarchical network. This is briefly discussed in [20] as a mechanism to represent processes in a top-down fashion at various levels of abstraction.

The problems of *conflict* which arise in PNs do not arise in DNs as tokens are not removed from the input places [3, 36]. Thus, the problem of *deadlock* may not arise in Derivation Nets, because tokens do not represent non-sharable resources. On the other hand, processes can be *dead.* A process is said to be dead in a marking if there is no sequence of process instantiations that can instantiate it. A dead process is one which cannot become instantiatable, and may arise when one or more base classes have no instances. Furthermore the state of *nondeterminism* cannot occur in a DN, since when more than one transition is enabled, they may all fire in some order.

3.4 Additional Properties of DNs

Different properties of PNs have been investigated as analysis tools, namely *boundedness, conservation of tokens, safe nets, liveness of transitions* [36], *S-invariance, T-invariance* [30]. For DNs, the property of *conservation of tokens* is not important as tokens are only created, not consumed. Similarly, the properties of *safety and boundedness* are irrelevant since there is no bound on the number of tokens in any class (place) of the net. Likewise, there are no S-invariants nor T-invariants except for the trivial (zero) invariants, because tokens are only added.

The implications of the concept of *liveness* can be different for different systems modeled using PNs. This concept is reducible to the *reachability problem* in a PN and can be used for its analysis [36]. In a DN, reachability is defined with three different interpretations of the PN, namely the graph based, the class based and the object based interpretations. Furthermore, we define and discuss issues with traversing the net backward and discuss reversibility. These two concepts are important and were used in our implementation of DNs [44].

3.4.1 Reachability

The *reachability set* of a PN is the set of all states into which the net can enter by any possible firing sequence of its transitions. It is the set of legal markings of the net. The *reachability problem* is as follows: Given a marked Petri net (with marking M) and a marking M', is M' reachable from M [36]?

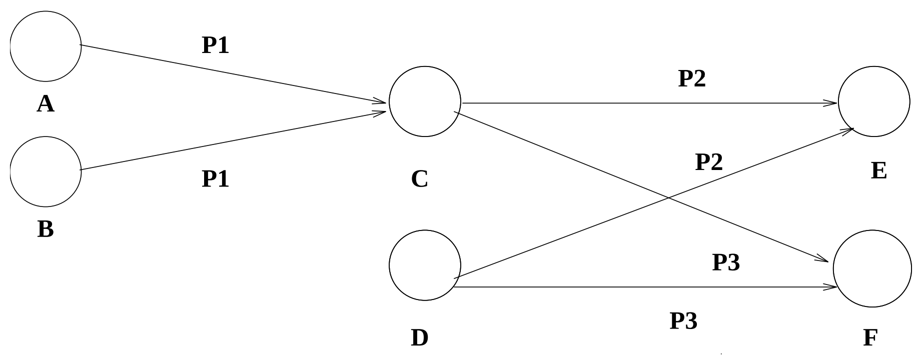


Figure 6: Graph Representation of the Petri Net in Figure 1

A PN can also be viewed as a bipartite directed graph (Figure 6) [3]. In a graph a node n_1 is reachable from a node n_2 if n_1 equals n_2, or there is a path from n_2 to n_1. Therefore a graph based definition of reachability is as follows:

Definition 11 Class reachability (graph based) *A class* **b** *is said to be reachable if it is a base class or a class derived by a process* **p** *such that the set of input classes* $\{c_i\}$ *of process* **p** *is reachable. The set of input classes* $\{c_i\}$ *is said to be reachable if each of the members of the class is reachable. The* **reachability set** *of a set is the transitive closure of the set of reachable classes from a given set of classes.*

The DN can be used to determine if a class is reachable or an instance of a class (data object) is reachable. Hence the definition of reachability of an object differs from the reachability of a class. The definitions of class reachability and object reachability are based on the definition of a *reachability path.*

Definition 12 *The* **reachability path** *of a class* c_i *is the set of processes* $\{p_i\}$ *that must be instantiated from a given set of classes* $\{c_j\}$ *to reach class* c_i.

Definition 13 Class reachability (PN based): *Given a set of input classes* $\{c_i\}$ *with data objects (instances), a class c is said to be reachable if there exists a subset of objects from the set of input classes* $\{c_i\}$ *that instantiate all processes in the reachability path of class c.*

Definition 14 Object reachability: *Given a set of input classes* $\{c_i\}$ *with a set of data objects (instances or tokens), an object b in class c is said to be reachable if the given set of data objects from the set of input classes* $\{c_i\}$ *instantiates all processes in the reachability path of class c to generate object b.*

Illustration: Consider the DN of Figure 1, which has tokens in classes A and B forming the initial marking. Using the *graph-based* definition of class reachability, class E is reachable from class A. The reachability path of class E is the set of processes {P1, P2}. Based on the *PN-based* view, class E is not reachable since process P2 cannot be instantiated to obtain any new data objects in class E. Using the definition of object reachability, there are no objects in class E that are reachable from class A. However, class C is reachable using both the graph- and PN-based definitions of class reachability.

The decidability of class and object reachability is settled by the following propositions.

Proposition 9 *Class reachability is a decidable proposition.*

Proof: Class reachability (PN based) is identical to the accessibility problem for PNs, which is shown to be decidable in [41]

Proposition 10 *Class reachability (graph based) is a decidable proposition.*

Proof: By considering an initial marking containing every base class, this follows as a corollary of class reachability (PN based).

Proposition 11 *Object reachability is a decidable proposition.*

Proof: This is also a corollary of class reachability (PN based), assuming that the assertions are decidable, which will be true for any practical DN.

3.4.2 Reverse Reachability

In addition to the concept of reachability, *reverse reachability* can be defined for a class and an object. Intuitively, reverse reachability is the ability to determine the source data provided that the target data exists.

Definition 15 *Given a derived class* **b***,* **reverse class reachability** *is the ability to recursively determine the set of input classes that contain the tokens used to generate at least one new data object in* **b**.

Definition 16 *An object* **o** *is said to be* **reverse reachable** *if the set of tokens that are members of the initial marking* m_0 *used to generate* **o** *can be determined.*

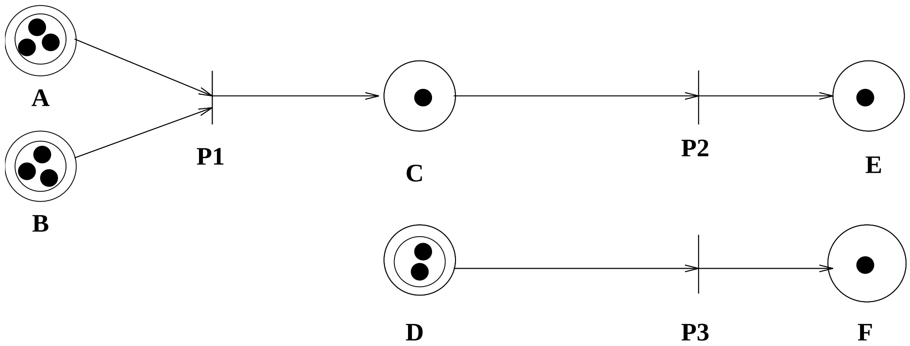


Figure 7: Reverse Class and Object Reachability

Illustration: Consider the DN with the final marking m_f shown in Figure 7. Classes E and F have a token (data object) and hence are reverse reachable. If the structure of the network is known, then it can be determined from which classes in the network the marking needs to be obtained. One can use reverse reachability to determine which portion of the network is relevant to the evaluation of a data object.

This is a selection of a subnet from the global network. This subnet can then be analyzed independently from other network portions, pruning the search space to answer a query. The poset structure of the subnet is contained in the original net. Its reduced size enables us to perform efficient derivations over the DN.

Proposition 12 *Reverse class reachability and reverse object reachability are decidable propositions.*

Proof: Given a marked Derivation Net $M = (C, P, I, O, A, m)$, consider exchanging the input and output classes for all processes to create another marked DN $M' = (C, P, I' = O, O' = I, A, m)$. Reverse class reachability in M is isomorphic to class reachability in M' of the base classes in M. Reverse object reachability in M is isomorphic to object reachability in M' of the base data objects in M. Thus, both reverse properties are decidable.

3.4.3 Reversibility

A process **p** is said to be reversible, if for a given object in its output class, the corresponding object(s) in its input class(es) can be found from the mappings defined as part of the process definition (refer to Section 4.1).

For example, assume the following process definition that is a simple query over a class:

```
DEFINE PROCESS  p1
OUTPUT o1
ARGUMENT (a of in1)
TEMPLATE{
     ASSERTIONS: /* no constraints */
     MAPPINGS:
          o1.timestamp = a.timestamp;
          o1.spatialextent = a.spatialextent;
          o1.someinfo = SQL_TYPE_QUERY_OVER(a);
         }
```

Assume that the `SQLTYPE_QUERY_OVER()` is a linear and bijective function. This process maps class `a` into the output class `o1` over the same spatio-temporal extent. The functions defined by the mappings specified in process `p1` are bijective mappings. Therefore, if the user desires information for “15 Nov 1992” and for the city of “Worcester”, the information can be retrieved from the input class and assigned to the output class. Hence it can be said that the process `p1` is reversible.

An example of a process that is not reversible is:

```
DEFINE PROCESS P2
OUTPUT o2
```

```
ARGUMENTS(a of in2)
TEMPLATE{
     ASSERTIONS: /* no constraints */
     MAPPINGS:
          o2.timestamp = a.timestamp;
          o2.spatialextent = a.spatialextent;
          o2.data = gIMaxlik(gICluster(gMkGroup(a.filename), 12));
   }
```

In the process definition of **p2**, the operators `gIMaxlik`, `gICluster`, and `gMkGroup` are borrowed from GRASS to perform the image functions of Maximum Likelihood, Clustering, and Grouping respectively. In **p2** if an output object is specified with the temporal extent of "15 Nov 1992" and spatial coverage of "Worcester", the input object cannot be found as there is no indication as to which input object should be used. Moreover, even if objects can be found with the specified spatio-temporal extents, they may not all be applicable. Therefore the process is not reversible.

Definition 17 Reversibility *is the process of generating the desired set of data in a class* **b** *by instantiating processes in the reachability path of* **b**, *provided all processes along the path are reversible and all tokens relevant to the derivation can be determined from* m_0.

3.4.4 Reachability, Reverse Reachability, and Reversibility

It is important to differentiate between the different properties just described.

- Reachability generates data if it does not exist.
- Reverse reachability does not generate data. It only helps one find the source of the target data. In order to perform reverse reachability the target data should have been generated earlier.
- For reversibility the target data need not exist. Reversibility generates the target data when the target data does not exist after determining which is the source data that can be used to generate the target data.

4 Implementation

In this section, we provide an overview of the Gaea architecture. We describe our implementation of Derivation Nets within the derivation semantics layer of Gaea. The operational characteristics and some of the Gaea query constructs are described. We provide an example showing how such a system can be used.

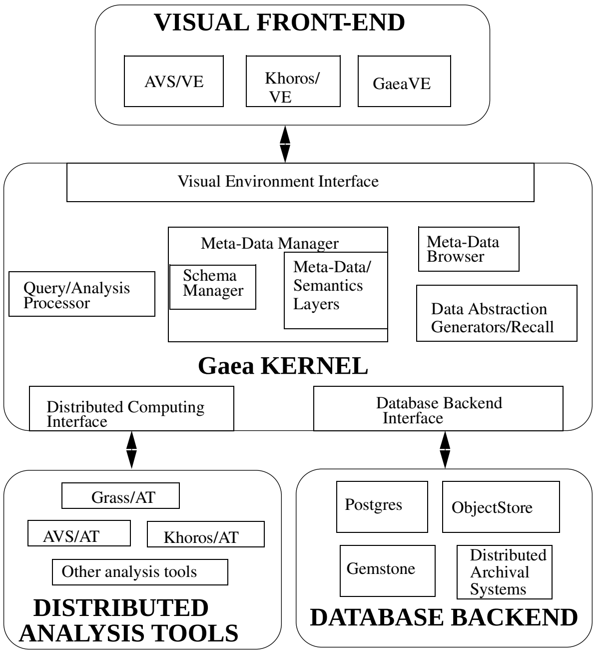


Figure 8: The Architecture of Gaea (from [20]).

4.1 The Gaea Architecture

The Gaea system architecture is designed to meet the needs of scientific research, specifically global change studies. Our view of a scientific data management and analysis environment can be layered along three levels (Figure 8): 1) The visual frontend, which allows the user to pose visual queries, apply analysis operators to data, and visualize data, including analysis results; 2) the Gaea Kernel, which provides support for meta-data, and converts simple queries from the visual frontend into a complex series of database accesses and operations; and 3) the backend, which actually stores the data, providing network and archiving functions. It also provide interface and access to different analysis tools. We describe each subsystem in turn.

The **Visual Frontend** mediates all interaction with the user. Our objective is to provide sufficient flexibility so that a variety of popular visual environments can be interfaced to the Gaea Kernel. There exists many such packages, either commercial (e.g., AVS [4]) or publicly available (e.g., Khoros [40]). These visual environments come with complete analysis subsystems. We would like to make use of the frontends and analysis operators separately, as shown in Figure 8. In addition, we have written our own visual frontend tailored to the Gaea Kernel (GaeaVE) [50].

The most important function of the **Gaea Kernel** is the management of metadata and the semantics of derived data. Users can query metadata to obtain the meaning of derived data. Furthermore, capturing a data object's derivation process information enables the user to repeat that process and derive new data, given different input data. The kernel includes a schema manager which manages the metadata and the associated derivation semantics and analysis operators (Figure 8). The Query/Analysis Processor (QAP) is responsible for processing queries, deriving new data whenever necessary, and using metadata. The kernel includes a semantic and metadata browser to allow a user to find relevant data without knowing specific file and path names. There is also a Data Abstraction Generation and Recall module which allows previously generated data to serve as a template for additional queries, i.e., queries can be abstracted. Generic interfaces to the frontend visual environments and backend distributed computing and distributed databases and archives are provided.

The **Backend System** consists of distributed and archive databases such as Postgres, Object Store and Gemstone [2, pp. 34–93]. The distributed computing environment consists of scientific analysis operators which are available within commercial or public domain software systems. Examples are the analysis tools available within AVS, Khoros, and GRASS. These tools may be imported into Gaea because the metadata manager will have registered information about analysis operators, their domains of application, data types and formats they apply to, among other metadata. The Gaea Kernel may chose from these available tools and use them to provide a seamless integration between analysis and data management for scientific environments.

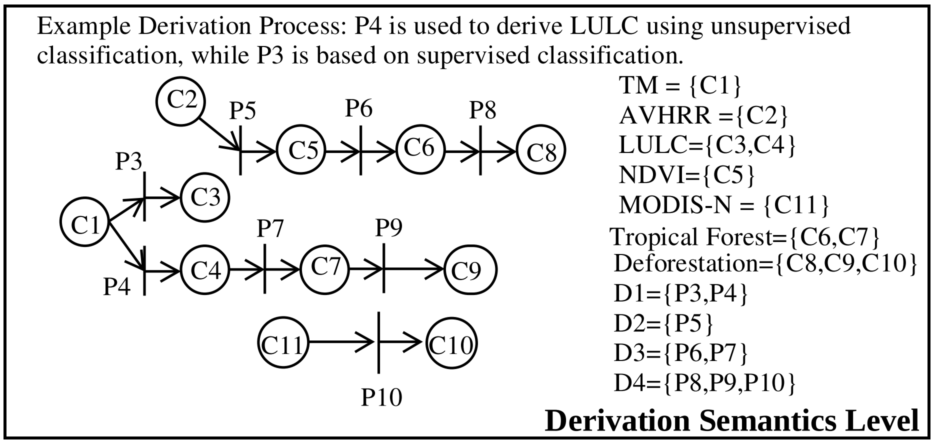


Figure 9: Derivation Management Layer

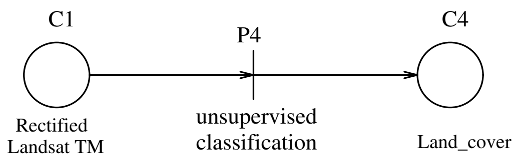
4.2 Data Derivation Management in Gaea

The Meta-data/Semantic layers of Gaea shown in Figure 8 consists of three layers: the high-level semantics, the derivation semantics and system level semantics. DNs are used and implemented for the derivation semantics layer. Additional details about how the other layers are used is found in [20].

The derivation semantics layer provides for the management of (scientific) derivations of data based on the formalism of Derivation Nets. Scientists manipulate objects according to a set of object classes in the derivation semantics layer. These objects correspond to the tokens in the Derivation Nets. Each class represents a set of objects related by the class definition. Objects containing base data can be in base classes, while derived classes hold objects derived from specific derivation procedures.

The derivation semantics layer records the derivation relationships among classes of data in the form of *processes*, which capture the descriptions of scientific procedures used for the generation of new instances of data objects from other data. Typically, when data are not stored in the database, we generate the needed data with the help of such derivation relationships through *process instantiation* with input data objects.

An example of a process for the derivation of land use/land cover (LULC) is illustrated in Figure 9. Base data are Thematic Mapper (TM), Advanced Very High Resolution Radiometer (AVHRR), and MODerate resolution Imaging Spectrometer–Nadir (MODIS-N) data sets in classes C1, C2, and C11, respectively. Process `unsupclass` (P4) derives class `landcover` (C4) which has four attributes: the spatial extent `landcover.spatialextent`, the temporal extent `landcover.timestamp`, the number of land cover classes `landcover.numclass`, and raster image data `landcover.data`. The extents are invariantly transferred from the input classes, while the image data are derived using the functional application of the image operators: `unsuperclassify` and `composite` [13]. The assertions using the rule `common` make sure that the spatio-temporal extents of the input classes are the same or overlap.




```
DEFINE PROCESS    P4
OUTPUT   C4
ARGUMENT (  bands  SETOF   C1  )
TEMPLATE {
  ASSERTIONS:
        card ( bands ) = 3;        // need three bands
        common ( bands.timestamp );
        common ( bands.spatialextent );
  MAPPINGS:
        C4.spatialextent = ANYOF bands.spatialextent;
        C4.timestamp = ANYOF bands.timestamp;
        C4.numclass = 12;
        C4.data = unsuperclassify ( composite ( bands ),  12 );
}
```

Figure 10: Derivation Process for Unsupervised Classification

4.3 Operational Characteristics

Currently, the Gaea system is built on top of the Postgres extensible system [48]. The properties of reachability, reverse reachability, and reversibility discussed in Section 3.4, together with the semantics of DN queries and updates, were used to formulate extensions to Postquel, the query language of Postgres [2, pp. 64–77]. The detailed implementation of these extensions is described in [44].

Reachability and reverse reachability were used to automate the process of data derivation in Gaea, thus making the retrieval of derived data implicit, although not completely transparent to the user. The retrieval mechanism is based on applying reachability and reverse reachability analysis on the network to decide if a non-existing object can be derived from existing data. The basic retrieval algorithm is as follows:

1. Attempt to retrieve the data from the target class. If it exists, return;

2. Else, back propagate the requirements through the derivation net and apply this procedure to the input class(es) of the derivation process. If input data are available, fire the process to generate the needed data; otherwise repeat this step.

3. The procedure is recursively applied until the needed data are generated or back propagation stops at some base class and we fail to generate the needed data.

Using DNs, the above procedure can be formulated on the (virtual) final marking. Given a projected view of the final marking, that is, a subset of the final marking containing the expected answer, try to find a subset of the initial marking which can lead to this view. This identifies a subnet from the global network which is then used to derive the required data.

4.4 Data Definition and Manipulation

Extensions to Postquel include a set of new data definition and manipulation statements designed for Derivation Nets [44]. Some of these are:

- Process definition

  Defining a process registers into the database a process which can be instantiated later. The current implementation allows a process to have only one output class, but many input classes. The arguments to a process are either a single object or a set of objects.

  The body of a process definition is a template for the process, which will be executed by an interpreter when the process is instantiated. A process definition in Gaea is illustrated in Figure 10. and elaborated upon in [20].

  `CARD` and `COMMON` are special functions for set expressions. `CARD` returns the cardinality of a set while `COMMON` is a predicate that checks if some data value holds for all elements in a set.

- Process instantiation

  Process instantiation takes the input objects for the process and generates an output object. The combination of process and its inputs is called a *task*. The input objects could be a list of objects or object set depending on the process definition.

```
INSTANTIATE PROCESS unsupclass
WITH ARGUMENTS ( retrieve (landsattm.all) where
                   landsattm.timestamp = "July 1987"::abstime and
                   landsattm.areaname = "paxton" and
                   (landsattm.bandno = "green" or
                   landsattm.bandno = "red" or
                   landsattm.bandno = "blue"))
```

- Data retrieval

  Gaea has a special retrieve statement which has different semantics from the ordinary retrieve statement of Postquel. When no requested data object is found in the database, the Gaea system will create a new data object by process instantiation if a derivation process is defined and the reverse reachability criteria are satisfied.

```
GRETRIEVE retrieve (human_deforest.all)
WHERE human_deforest.timestamp =
        "January 1987"::abstime AND
      human_deforest.areaname  = "paxton"
DERIVED BY PROCESS
```

  This **GRETRIEVE** query provides automatic retrieval of data from the **human_deforest** class for the location and time specified. If the data already exist, this query is equivalent to a straightforward Postquel **RETRIEVE** statement. If the requested data do not yet exist, then they are derived by executing the appropriate operations using the derivation network. In order for this to work properly, a process must have been defined that is capable of generating data for the **human_deforest** class. If this process exists, but the input data that it needs do not yet exist, then the input data are themselves the targets of internally generated queries and these queries are recursively processed. This recursive query processing is invisible to the user, who is only aware that the original query can or cannot be satisfied.

Other extensions include statements for process retrieval, process deletion, check for reachability, etc. [44].

4.5 Integrating Data Analysis Tools

One important part of the Gaea system is the data analysis operators. In order to make use of the available resources, we have decided to use GRASS as our base for analysis operators because of its popularity in the GIS domain [45]. In the following, we discuss how to integrate GRASS with Postgres to create an environment for change analysis. Then we demonstrate how derivation management is achieved in such an environment.

In GRASS, all data are organized by location and mapset. At one time, the user can only work on a specific mapset of a location. The highest level of the location/mapset tree structure is illustrated in Figure 11.

```
                GISDBASE = /usr/user/grass/data
                                |
                                |
                  --------------------------------
                  |                              |
              location1                      location2
                  |                              |
         -----------------              -----------------
         |                |             |                |
     PERMANENT        mapset1.1     PERMANENT        mapset1.2
```

Figure 11: GRASS data tree structure

For every location, there is a mapset called PERMANENT. This mapset contains non-volatile data for that location that all users will use. It also contains some information about the location itself.

The data are actually stored under the mapsets, which are further divided into subdirectories. The tree structure of a mapset is illustrated in Figure 12. Only the part of the GRASS mapset tree that relates to image data is shown here. Another large category of data is vector data, whose data organization has been omitted for clarity.

Corresponding to every raster data file, the auxiliary information is stored under different directories with the same name as the raster file. Some related directories are:

- cell – the binary raster file itself
- cellhd – header file for raster maps
- cats – category information for raster maps

```
                         mapset
                           |
 -----------------------------------------------------------------
 |      |      |      |     |      |       |       |    ...
WIND   cell  cellhd  cats  colr  group  cell_misc  hist
```

Figure 12: Mapset data tree structure in GRASS

- colr – color tables for raster maps
- cell-misc – miscellaneous raster map support files
- hist – history information for raster maps

When needed, GRASS can look into the relevant directory to find the information about a raster map. GRASS also provides a command **g.remove** which will delete all the information (files) for specific a raster map. This is one way to maintain consistency.

Postgres is an extensible system. It allows the user to define new data types and new functions that can be dynamically loaded for execution. The main problem encountered is the discrepancy in data organization between the two systems. GRASS is file-based, using a private tree structure to organize data, while Postgres is basically a relational database system.

Our approach was to create a GRASS environment when running a GRASS command from Postgres. Data are normally stored in the database, but are transformed into files when needed. The conversion routine takes care of where to put the temporary GRASS files. After the computation is completed, the resultant data is pulled back into the database.

In order to record information necessary to execute GRASS or other procedures from within Gaea, it is necessary to *register* these foreign procedures with the Gaea Kernel. Currently, we have registered a set of GRASS image processing and raster commands into Postgres and have tested them on simple change analysis tasks, such as land cover and land use change in Paxton, Mass using both Landsat TM and SPOT images.

4.6 Example

Based on the Derivation Net of Figure 13, an example of land use/land cover change analysis using Gaea is illustrated in Figures 14, 15 and 16. Referring to Figure 13, we see that the query has requested data from class **human_deforest**, which may be derived by process **chgdetect** (change detection) using **landcover** class data. Process **unsupclass** (unsupervised classification) will be used to estimate landuse

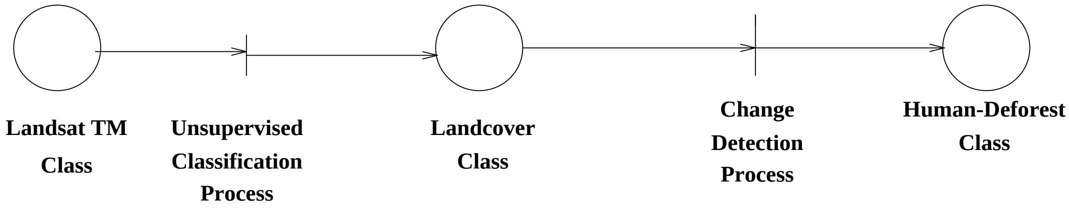


Figure 13: DN for deriving deforestation

using **landsattm** (landsat thematic map) class data. Unsupervised Classification is a process of automatic aggregation of similar regions used in the interpretation of remotely-sensed images.

Assuming that the necessary base data in class **landsattm** are available, the derivation network may be examined to determine that the original query is satisfiable. That is, starting with an initial marking of the base data classes, a final marking can be found that includes the output class **human_deforest**.

```
DEFINE PROCESS unsupclass
OUTPUT landcover
ARGUMENTS (blueband of landsattm, greenband of landsattm,
           redband of landsattm)
TEMPLATE {
ASSERTIONS:
      blueband.timestamp = greenband.timestamp;
      greenband.timestamp = redband.timestamp;
      blueband.spatialexten = greenband.spatialexten;
      greenband.spatialexten = redband.spatialexten;
MAPPINGS:
  landcover.timestamp = blueband.timestamp;
  landcover.spatialextent = blueband.spatialextent;
  landcover.area = blueband.area;
  landcover.numcls = 12;
  landcover.data = gRReclass (gMakeImg (gGetImgNumRow(blueband.data),
            gGetImgNumCol(blueband.data), gGetImgPixType(blueband.data),
            gIMaxlik (gICluster( gMkGroup3(blueband.data,
                      greenband.data, redband.data), 12))));
```

Figure 14: Process definition for land use and change analysis

```
INSTANTIATE PROCESS unsupclass
WITH ARGUMENTS (retrieve (landsattm.all)
               where landsattm.timestamp = "Jan 1 1982"::abstime and
                         landsattm.area = "paxton" and
                         landsattm.bandno = ''blue'',

                  retrieve (landsattm.all)
                   where landsattm.timestamp = "Jan 1 1982"::abstime and
                         landsattm.area = "paxton" and
                         landsattm.bandno = ''green'',

                  retrieve (landsattm.all)
                   where landsattm.timestamp = "Jan 1 1982"::abstime and
                         landsattm.area = "paxton" and
                         landsattm.bandno = ''red'')


INSTANTIATE PROCESS chgdetect
WITH ARGUMENTS (retrieve (landcover.all)
                   where landcover.timestamp = "Jan 1 1982"::abstime and
                         landcover.area = "paxton",

                  retrieve (landcover.all)
                   where landcover.timestamp = "Jan 1 1987"::abstime and
                         landcover.area = "paxton" )
```

Figure 15: Instantiating tasks for land use and change analysis

5 Discussion

5.1 Limitations of the Model

It is important to identify some of the limitations of our model and propose possible extensions to overcome them.

1. The current DN is a single-level structure and should be extended to hierarchical nets. A *hierarchical* DN is an extension of DNs where places or transitions can themselves be considered as abstractions of Derivation Nets. Just as subroutines and functions became a necessity as programming projects grew in size so also managing descriptions of large and complex systems using PNs necessitated the development of Hierarchical Petri Nets [19] and Hierarchical Colored Petri Nets [24]. Using Hierarchical CP-nets, a number of individual CPNs may be

```
GRETRIEVE retrieve (human_deforest.all)
WHERE human_deforest.timestamp =
        "January 1987"::abstime AND
      human_deforest.areaname  = "paxton"
DERIVED BY PROCESS
```

Figure 16: Using Gaea retrieve

related to each other in a formal way. Hierarchy constructs include *substitution of transitions*, *substitution of places*, *invocation of transitions*, *fusion of places*, and *fusion of transitions* [26].

At this time, the Gaea class structure is flat; classes do not inherit from other classes. Although current scenarios for global change research do not require class inheritance, future applications may require it. Developing the theory for Hierarchical Derivation Nets would facilitate this.

2. We made the assumption throughout most of this paper that all DNs are acyclic. Thus, no loops are allowed and no output class may be used as input to its generating process or any process on its generation path. This is needed to guarantee that the set of markings converges to a finite final marking m_f. Otherwise, we would be dealing with infinite posets[6]. We are currently investigating extensions of the model that would support cyclic DNs.

3. Finally, in the current implementation of derivation nets, no interaction can be specified in a process definition. There are many situations in data analysis that require the user to decide how to proceed with the based on an intermediate result. One can envision a procedure followed by a scientist which demands the specification or modification of input parameters based on some temporary result visualized on the screen. A typical example is supervised classification [14], which by its nature requires interaction to complete successfully.

   This limitation is due to our interpretation of DNs and is not induced by the DN model. It seems fairly easy to modify this interpretation of derivation nets to capture interaction. We foresee the use of special transitions and places for that purpose. We are currently investigating that extension.

5.2 Comparison with Other Formalisms

In this section, we review other proposed mechanisms that relate to our work and make some comparisons.

[6]This view has a correspondence with 1st-order logic (without functions).

5.2.1 Conceptual and Functional Modeling

The extended E-R approach is used in [31] to model both the functional and structural components of an information system. The basic idea is to represent a process as a relationship and apply existential constraints to express the partial order implied in a process. We do not believe that the E-R approach is sufficient to represent derivation relationships among data classes for reasons discussed in [20].

One may find similarities between our work and functional modeling in the system analysis stage of business database applications. However they are different in their purpose and the methods used. One popular method for functional modeling is Data Flow Analysis [33]. In data flow analysis, an information system is considered as a process that maps input data to output data, and can be represented as a data flow diagram. Then the transformation process is further decomposed into subprocesses until each is basic enough to be implemented with a piece of simple program.

Although functional analysis is also concerned with a process, the purpose is different from that of derivation management in scientific databases. A process in functional analysis is used to develop application programs, while in Gaea it is used to define derivation relationships among data classes. Furthermore, a task, the instantiation of a process, is of no interest in functional analysis, while in Gaea, individual tasks define the derivation relationship among a set of data objects.

5.2.2 Lineage and Versioning

Different means of capturing lineage information in GIS (Geographic Information Systems) have been studied and critiqued [28]. Lineage information consists of input/output relationships, transformations, and source data. Methods of automatically capturing lineage information include history files, version control systems, map librarians, and polygon attributes. Each of these methods captures lineage details but does not efficiently manage the lineage information captured. There is no easy way to retrieve and view this information as and when desired. The solutions to the problem of lineage in GIS suggested in [28] are manual. This approach cannot provide the ease of retrieval that an automated system provides.

A solution for the representation of lineage information in GIS, based on semantic networks and frames, is proposed in [29]. The proposed procedure to maintain lineage information is suitable when the data size is small. Since GIS datasets are very large, semantic nets may not be effective without a database to store the data.

Version modeling concepts are similar to lineage in GIS and audit trails in management information systems (MIS) [18]. They focus on version histories, time-varying configurations, and equivalences among objects of different types. Version modeling has been applied to computer aided design databases in [27]. Version concepts appear relevant to process histories in Gaea. The ability to maintain the derivation history of a process is crucial in Gaea. On the other hand, the idea of currency within the version history suggested in [27] is irrelevant in Gaea. Since version histories can branch widely, a control mechanism to identify a preferred version from which new derivatives can be created is identified with a currency indicator. In Gaea, version histories may be of concern when the process definition changes and data generated from an old definition still exists. That is effectively version control. The metadata used by the derivation semantics may also be used to keep track of versions, but note that the derivation semantics per se does not solve versioning problems.

5.2.3 Scientific Databases

Experiment management is also the goal in [9, 25]. The former application domain is the modeling of experiments in computational chemistry, while the latter is simulation; both are based on the object-oriented paradigm [5]. They provide mechanisms for managing the definition, preparation, monitoring and interpretation of experiments. We address the same problems, but identify differences between experiment management and data derivation management. By using different formalisms to model them, we have introduced different semantics into our system.

Semantic networks are an appropriate tool to capture the relationships among a set of data objects. This formalism has been used in the USD system [43]. Although the intention was to make use of the flexibility of semantic networks to represent unstructured data, it can also be adequately used to model an experiment. The problem with semantic networks is that they might become too complex with a large database system. In addition, data derivation relationships are not explicitly represented in the network.

Data Analysis Management is necessary to help an inexperienced analyst learn how expert analysts conduct experiments [8]. It requires interaction and recording of intermediate events, to prove that the analysis was rigorous and complete. The proposed technique is based on the concept of *save-state*, a collection of metadata and data that captures significant information about the state of the analysis at a certain point in the analysis process. Although save-states encapsulate metadata, no indication on how to implicitly manage save-states is provided.

The process-oriented scientific data model, proposed in [37], is used to capture, organize, manipulate, and retrieve experimental process data such as seed growing into a plant [37]. The proposed model is such that in a given process, when a state is changed, knowledge of the previous state is lost. For example, once the seed is transformed into a plant, the seed state does not exist anymore with respect to the same entity (plant). As the entity gets transformed, any further information about the original entity cannot be obtained at a later point in time.

The objectives of the MDBS project [47] is to overcome the problems of large, distributed and heterogeneous sources of data. At present, the data are assumed to be in files and lacks database support. The authors propose modeling scientific data around the concept of domains of entities and transformations between such domains. Two kinds of domains are proposed: **conceptual domains** (C-domains), which relate to abstract views of entities and transformations, and corresponding

representational domains (R-domains) which relate to symbolic representation of the entities and transformations. An example of a C-domain is a *polygon* with a variety of R-domains based on sequences of *points*, sequences of *line-segments* and sets of *half-planes*. New R-domains are created through the application of constructors.

Although the MDBS system and Gaea have similar goals, the approach taken is very different. One similarity lies in the notion of a *project* which is similar to the concept of an *experiment* in Gaea. Also, in the application of *constructors* to previously defined domain elements to obtain new R-domains is similar to the *operators* in Gaea. Finally, The notion of representing C-domains and R-domains is interesting. The structure of an R-domain is similar to the structure of a process definition in some respects. Both include the name of the domain, sets of transformations on the domain elements and the constraints of the domain elements. However a process definition does not include the structure of the domain elements, the latter is provided by the class definition.

The Sequoia 2000 research project aims at providing the basic computational resources necessary to support global change research and to build a data store and server for virtually unlimited geographic information [16]. The Sequoia prototype provided GIS functionality within an enhanced DBMS environment. The project was called POST-GRASS and it was implemented with GRASS as the engine for GIS primitives and Postgres as the DBMS. There are a lot of similarities between the Gaea project and Sequoia 2000 as both encompass a lot of aspects of global change research. However the philosophies are different and the Sequoia 2000 research does not address the problem of implicitly managing derivation metadata and data.

6 Conclusions

We have presented a model for the management of data derivation relationships so that data can be shared in scientific databases. The main contributions include:

- Derivation Nets, an extension of Petri Nets, were introduced to represent and manage the semantics of data derivation in scientific databases. Derivation Nets can be used to 1) browse data following their derivation relationships, 2) compare derivation procedures and their resulting data classes, and 3) derive data not stored in the database.

- Some properties of Derivation Nets were stated and proven, including closure with respect to database states, order independence, query and update semantics, reachability, reverse reachability, and reversibility.

- A three-layered view of the Gaea scientific database management systems, which includes DNs as a core component, was described.

The proposed Derivation Net model has many potential long term extensions: 1) They provide a knowledge acquisition environment that can be used for learning and automated derivation of scientific data. 2) There are some known limitations of the model, i.e., the current model does not provide for nested abstractions of derivations nor does it provide for the capture and management of scientific user interactions during an experiment. To overcome these limitations, we are currently looking into extensions of the model to include hierarchical networks which may also represent user interaction.

References

[1] "Contents Standards for Digital Geospatial Metadata," Federal Geographic Data Committee, Washington, D.C., 1994.

[2] "Next-Generation Database Systems," Comm. of the ACM, Special Issue, Vol. 34, No. 10, Oct. 1991.

[3] T. Agerwala, "Putting Petri Nets to Work," IEEE Computer Magazine, pp. 85–94, Dec. 1979.

[4] Advanced Visual System Inc., 300 5th Avenue, Waltham, MA 02154. *AVS Technical Overview*, Oct. 1992.

[5] M. Atkinson, F. Bancilhon, D. DeWitt, D. Maier, and S. Zdonik, "The Object-Oriented Database System Manifesto," Proc. Int. Conf. Deductive and Object-Oriented Databases, pp. 40–57, 1989.

[6] J.-L. Baer, "Modeling Architectural Features With Petri-Nets," Lecture Notes in Computer Science, Springer-Verlag, No. 255, pp. 258–277, 1986.

[7] A. Beller, "Spatial/Temporal Events in GIS," GIS/LIS 91, Vol. 57, No. 4, pp. 407–411, 1991.

[8] P.J. Cowley, M.A. Whiting, "Managing Data Analysis Through Save-States," 17th Symp. Interface Of Computers In Kentucky, Pacific Northwest Laboratory PNL-SA-13023, Mar. 1985.

[9] J.B. Cushing, et. al. "Object-Oriented Database Support for Computational Chemistry," Proc. SSDM '92, pp. 58–76, 1992.

[10] A.M. Davis, "A comparison of techniques for the specification of external system behavior," Comm. of the ACM, Vol. 31, No. 9, pp. 1098–1115, Sept. 1988.

[11] Y. Deng, and S.-K. Chang, "A G-Net Model for Knowledge Representation and Reasoning," IEEE Trans. Knowledge and Data Engineering, Vol. 2, No. 3, pp. 295–310, Sept. 1990.

[12] J. Dozier, "Access to data in NASA's Earth Observing System," Keynote Address, Proc. ACM SIGMOD Int. Conf. Management of Data, San Diego, 1992.

[13] J.R. Eastman and J.McKendry, *Change and Time Series Analysis in GIS*, UNITAR, 1991.

[14] J.R. Eastman, *IDRISI—-A Grid-Based Geographic Analysis System (User's Manual)*, Clark University, Worcester, MA, Nov. 1990.

[15] J.C. French, A.K. Jones, and J.L. Pfaltz, "Summary of the Final Report of the NSF Workshop on Scientific Database Management," SIGMOD Rec. **19**-4, pp. 32–40, 1990.

[16] K. Gardels, "Sequoia 2000 and the Post-Grass project," GrassClippings, pp. 40–41, Winter 1992.

[17] M.A. Gennert, N.I. Hachem, N. Serrao, and A. Bansal, "Distributing Computations Among GIS Servers," to appear in Proc. 7th Int. Conf. Parallel and Distributed Computing Systems, Las Vegas, NV, Oct. 1994.

[18] R. K. Grady, "Data Lineage in Land and Geographic Information Systems," Proc. GIS/LIS '88, Falls Church, VA: American Congress Surveying and Mapping, pp. 722–730, 1988.

[19] N.I. Hachem, "Petri-Net Driven Knowledge Base System for Automated Microcode Generation in VLSI," Proc. IASTED Int. Symp. MODELING and SIMULATION, Calgary, Canada, July 1991.

[20] N.I. Hachem, K. Qiu, M.A. Gennert, and M.O. Ward, "Managing Derived Data in the Gaea Scientific DBMS," Proc. VLDB Conf., pp. 1–13, 1993.

[21] N. I. Hachem, M.A. Gennert, and M.O. Ward, "Distributed Database Management for Scientific Data Analysis," Proc. Int. Wkshp. Global GIS, Int. Soc. Photogrammetry and Remote Sensing WG IV/6, Tokyo, Japan, pp. 85–93, Aug. 1993.

[22] N.I. Hachem, M.A. Gennert, and M.O. Ward, "A DBMS Architecture for Global Change Research," Proc. ISY Conf. Earth and Space Science Info. Systems, Pasadena, CA, pp. 186–194, Feb. 1992.

[23] M.A. Holliday and M.K. Vernon, "A Generalized Timed Petri Net Model for Performance Analysis," IEEE Trans. Soft. Eng., Vol. 13, Dec. 1987.

[24] R. Huber, K. Jensen. R.M. Shapiro, "Hierarchies in Colored Petri Nets," Lecture Notes in Computer Science **483**, pp. 313–341, Springer-Verlag, 1990.

[25] Y. Ioannidis et. al., "Desktop Database Management," IEEE Data Engineering Bulletin, Vol. 16, No. 1, pp. 19–23, Mar. 1993.

[26] K. Jensen, "Coloured Petri Nets: A High Level Language for System Design and Analysis," EATCS Monographs on Theoretical Computer Science, pp. 342–416, 1992.

[27] R.H. Katz, E. Chang, and R. Bhateja, "Version Modeling Concepts for Computer-Aided Design Databases", Proc. ACM SIGMOD Conf., pp. 379–386, May 1986.

[28] D.P. Lanter, "Lineage in GIS: The Problem and a Solution," NCGIA Technical Paper 90-6 Sept. 1990.

[29] D.P. Lanter, "Design of a Lineage-based Meta-data base for GIS," Cartography and Geographic Information Systems, Vol.1 8, No. 4, pp. 255–261, 1991.

[30] K. Lautenbach, "Linear Algebraic Techniques for Place/Transition Nets," Lecture Notes in Computer Science **254**, pp. 142–167, Springer-Verlag, 1986.

[31] V.M. Markowitz, "Representing Processes in the Extended Entity-Relationship Model," Proc. Int. Conf. Data Engineering, pp. 103–110, 1990.

[32] M.A. Marsan, G. Balbo, A. Bobbio, and G. Conti, "A Class of Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," ACM Trans. Computer Systems, Vol. 2, No. 2, pp. 93–122, May 1984.

[33] J. Martin and C. McClure, *Diagraming Techniques for Analysis and Programming*, Prentice-Hall, New Jersey, 1985.

[34] J.D. Noe "A Petri Net Model of the CDC 6400," Proc. ACM SIGOPS Wkshp. System Performance Evaluation, pp. 362–378, 1971.

[35] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall, 1981.

[36] J. L. Peterson, "Petri Nets," Computing Surveys, Vol. 9, No. 3, pp. 223–252, Sept. 1977.

[37] J.M. Pratt and M. Cohen, "A Process-Oriented Scientific Database Model," SIGMOD Record, Vol. 21, No. 3, pp. 17–25, Sept. 1992.

[38] R. Rada, P.E.S. Dunne, and J. Barlow, "EXPERTEXT: From Semantic Nets to Logic Petri Nets," Expert Systems with Applications, Vol. 1, pp. 51–62, 1990.

[39] C.V Ramamoorthy and G.S. Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," IEEE Trans. Soft. Eng., Vol. 6, Sept. 1980.

[40] J. Rasure, D. Argiro, T. Sauer, and C. Williams, "A Visual Language and Software Development Environment for Image Processing," Int. J. Imaging Systems and Technology, Vol. 2, pp. 183–199, 1990.

[41] C. Reutenauer, *The Mathematics of Petri Nets*, Prentice-Hall, 1990.

[42] J.A. Richards, "Multispectral Transformations of Image Data," Chapter 6 in Remote Sensing Digital Image Analysis, Springer-Verlag, pp. 127–145, 1986.

[43] R. R. Johnson, M. Goldner, M. Lee, K. McKay, R. Shectman, and J. Woodruff, "USD—A Database Management System for Scientific Research," Video presentation at the ACM SIGMOD Int. Conf. on Management of Data, San Diego, 1992.

[44] N. Serrao, "Design and Implementation of the Derivation Semantics Layer in the Gaea Prototype," M.S. Thesis, Worcester Polytechnic Institute, Worcester, MA, Dec. 1993.

[45] M. Shapiro et. al., "GRASS 4.0 Programmer's Manual (Draft)," U.S. Army Construction Engineering Research Laboratory, April 1992.

[46] C.U. Smith, "Robust Models for the Performance Evaluation of Software/Hardware Design," Int. Wkshp. Timed Petri Nets, 1985, pp. 172–180.

[47] T. R. Smith, J. Su, D. Agrawal, and A.E. Abbadi, "Database and Modeling System for the Earth Sciences," Data Engineering, Vol. 16, No. 1, pp. 33–37, Mar. 1993.

[48] M. Stonebraker, L.A. Rowe, and M. Hirohima, "The Implementation of POSTGRES," IEEE Trans. Knowledge and Data Eng., Vol. 2, No. 1, pp. 125–142, 1990.

[49] T. Suzuki, S.M Shatz, and T. Murata, "A Protocol Modeling and Verification Approach Based on a Specification Language and Petri Nets," IEEE Trans. Soft. Eng., Vol. 16, No. 5, pp. 523–536, May 1990.

[50] Y. Zhang, M.O. Ward, N.I. Hachem, and M.A. Gennert, "A Visual Programming Environment for Supporting Scientific Data Analysis," Proc. Int. Wkshp. Visual Prog. Languages, Aug. 1993. (extended version as WPI-CS-TR 93-01, Mar. 1993)

[51] Y. Zhou, M.A. Gennert, N.I. Hachem, and M.O. Ward, "Requirements of a Database Management System for Global Change Studies," ASPRS/ACSM/RT 92, pp. 186–194, 1992.

[52] W.M. Zuberek, "Timed Petri Nets and Performance Evaluation," Proc. Symp. Computer Architecture, May 1980.