A Model and Framework for Visualization Exploration

T.J. Jankun-Kelly, Kwan-Liu Ma, and Michael Gertz

ABSTRACT

Visualization exploration is the process of extracting insight from data via interaction with visual depictions of that data. Visualization exploration is more than presentation; the interaction with both the data and its depiction is as important as the data and depiction itself. Previous visualization research has focused on the generation of visualizations—the depiction—and not on the exploratory aspects of the visualization process. However, without formal models of the process, visualization exploration sessions cannot be fully utilized to assist users and system designers. Towards this end, we introduce the P-Set Model of Visualization Exploration for describing this process and a framework to encapsulate, share, and analyze visual explorations. In addition, systems utilizing the model and framework are more efficient as redundant exploration is avoided. Several examples drawn from visualization applications demonstrate these benefits. Taken together, the model and framework provide an effective means to exploit the information within the visual exploration process.

isualization exploration process, visualization, visualization systems, history, derivation, collaboration, XML, software framework, science of visualization

1 INTRODUCTION

The explosive growth of computer graphics processing power and display capability has resulted in the proliferation of graphics in variety of fields. Due to these gains and a decade-long concentrated research effort, scientific and information visualization have become useful tools to information analysts. As the use of visualization becomes more wide-spread, methods to support the use and dissemination of visualization must be developed. A visualization technique with no means of storing its results is wasted. A visualization system which does not communicate to its user where they have been, where they are, and where they could go is inefficient. These problems must be addressed before visualization can become effective for large-scale deployment. This article presents our ongoing efforts to address this problem.

A three-part approach is used to capture and utilize the information within the visualization process. First, a formal model of the process is used to capture the salient aspects of the exploration—what results were generated, how they were generated, and how they were used to generate new results. Second, a representation utilizing the model documents the contents of a visualization session for later analysis or dissemination to collaborators. Finally, a software framework manages instances of the model. These three components have been refined from our previous work [1] based upon our experience applying them to different systems. Specifically, a more complete derivation calculus is introduced along with a specification of the representation and framework, neither of which have been presented before.

It is important to note that a model of the visualization process alone is not sufficient to describe the knowledge of the user before or after the visualization. To completely capture this knowledge and insight, a metadata model for the visualization process also needs to be developed. The ultimate goal of this research is to develop such a metadata model using the process model described here as the basis for the metadata's descriptions. For example, metadata annotations of previous sessions that suggest what results were "good" and which were not could help the session analysis process. Thus, our work is the first step towards that goal; recent visualization ontology efforts [2] point the way towards future development.

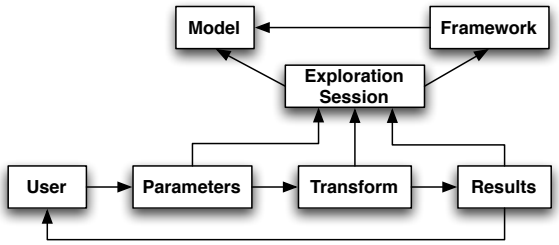


Figure 1: Components of visualization exploration. During exploration, a user manipulates parameters which are then applied to a transform to generate a result; the results are used by the user as feedback to continue the exploration. Our model describes the salient details of this iterative process while the software framework manages the storage and utilization of this information.

There are several benefits of capturing the visualization process. With our process model, users of visualization systems are able to record their visualization sessions at a higher level than simple log systems are able to provide. For example, graphical representations or analysis of the process would be difficult or impossible to perform using a log file due to the lack of information about the process. Our model also allows visualization systems designers to build systems that can share results and process information between different visualization interfaces. Finally, represented with a formal model, the visualization process is opened up to a variety of analyses. Examples of these benefits are presented in Section 6. The process model discussed here addresses visualization exploration in more depth and with greater generality than has been previously presented.

The model and framework introduced here are powerful because they are general—they can be applied to a wide domain of visualization problems. The parameter space abstraction and process model can be used in many visualization tasks. The stored representation can be shared and extended. This research will assist users of visualization to explore, communicate, and understand their results. Finally, we see our model as a stepping stone towards a formal science of visualization. By providing a general and extensible model of the fundamental process of visualization, rigorous discussion, validation, and analysis of visualizations can be performed.

2 A CHARACTERIZATION OF THE VISUALIZATION EXPLORATION PROCESS

Visualization exploration involves many interconnected components (Figure 1). In this section, we review these components and how previous researchers have modeled them. The common fea-

ture of these approaches—iterative interaction during visualization exploration—will lead to a discussion of how visualization systems support this interaction. The properties of these systems will be then distilled to form a fundamental operation of the visualization exploration process. This fundamental operation characterizes the visualization process and is the basis for our model discussed later.

2.1 Visualization Exploration

Upson et al. describe the scientific visualization process as an analysis cycle [3]. According to this model, data is filtered into subsets of interest, mapped onto visual primitives, and then rendered for the user by a function called the *visualization transform*. The visualization generated by this transform is then used by the user to provide feedback into the previous steps, restarting the cycle. A similar cycle of raw data transformation, visual structure generation, and view rendering with user interaction is described by Card et al. for information visualization [4]. The key feature of both models is that the visualization process is an iterative sequence of user controlled transformations. Thus, elements that change during this interaction must be the focus of any description of the visualization process. These elements are arguments to the visualization transform—the visualization parameters. In Upson's model, these parameters control the data filtering process (e.g., by specifying a data threshold value), the visual primitive mapping (e.g., by changing colors assigned to data values), and the actual rendering (e.g., by changing lighting parameters). In Card's model, parameters manipulate the raw data transformation (e.g., by specifying what parts of a data table in a database to utilize), the visual mappings (e.g., by changing the mapping of the nominal variables from color to shape), and the view transformations (e.g., by changing the orientation of the view). As parameters are visualization transformation dependent, a description of this transform is also important to any documentation of the visualization process.

While Upson and Card's models provide an overview for the visualization process, they are not fine grained enough to detail the user's exploration. Models that "unravel" the visualization cycle are needed in order to discuss what the user was doing at any time in the exploration. Two approaches have been taken: visualization space paths and derivation models (for a more detailed comparison, please consult [5]).

Visualization Space Paths Several novel visualization user interfaces [6–9] assume visualization exploration is equivalent to navigating a multidimensional parameter space. In these models, each parameter in the visualization transform corresponds to a dimension in the parameter space. Thus, a visualization result maps to a unique point in the parameter space. Users then trace a path through this space as they explore new results. For example, the Design Galleries [8] interface displays a overview of the entire space by random sampling while an Image Graph [7] follows a more structured path through the space. The models used by these interfaces provide an explicit tie between the visualization results (what the user sees) and the visualization parameters (what the user controls). However, relations between results beyond simple parent–child relationships are not explicit in these models. Derivation models address this limitation.

Derivation Models Derivation models describe how new items are created from previous items. For example, genealogy models the relationships between children and their parents (and previous generations). Previous to our work, two major visualization derivation model efforts had been undertaken: The GRASPARC project [10] and Lee's general data exploration (GDE) model for visual database exploration [11, 12].

The GRASPARC system was designed to assist the search for a solution in a scientific problem solving environment (PSE). Like the visualization exploration process, this search is parameter driven; in this case, the parameters are the control variables for the simulation data in the PSE. A history tree structure is used to communicate and manipulate the PSE control state; nodes in the tree store the solution parameters and related results. A similar tree-like structure could be used for simple visualization explorations; however, as discussed later, the types of interactions a user can have with a visualization system dictate that a more complex structure is needed.

Lee's work in visual database exploration provides a more complete representation of possible derivation relations. Lee uses a graph-based structure to model the visualization process for databases. Vertices in the graph represent the state of the visualization while edges are relationships between states. These relationships are based upon similarities between metadata attributes of the states and the data contained with the states. In Lee's work, the metadata attributes describe structural attributes of the states. A suite of derivations are defined upon these attributes in order to capture the different interactions one can have with visual databases.

Both the visualization space and derivation models contain elements that are used in our model for visualization exploration. An exploration is embedded within a visualization parameter space; the transitions along the exploration path can be described with a derivational model. The derivation model used in this work provides a formal definition of the relationships between different results in this visualization space. Instead of using structural metadata attributes (as in the GDE model), derivations record the origin of the parameters used in a result. In other words, derivations describe how a user's interaction with a visualization system created the parameters that generate a result.

2.2 Modeling User Interaction with Visualization

Visualization user interfaces allow the user to interact with visualizations. Thus, it is important to understand how a user interacts with such systems in order to model the visualization exploration process.

The human-computer interaction (HCI) community has long been concerned with the low-level mechanics of user interface interaction. This previous work has focused on the events generated by user interaction [13] or descriptions of the user interface elements themselves [14]. The interactions pertinent to this work are higher level and tailored for visualization systems—we are not interested in modeling a visualization control widget; instead, we are interested in how the parameter controlled by that widget affects the visualization. Within visualization, taxonomies of visualization tasks and operations have been studied [15–18], but these taxonomies focus on the goal of the user, not how the visualization system was used to achieve these goals. Our process model lies between the low-level syntactic models and high-level semantic models of user interaction. Specifically, it addresses the ways a user manipulates parameter values to produce visualization results.

Most visualization interfaces utilize one of two forms of parameter manipulation: *interactive parameter control* and *dynamic manipulation* [19]. In the former, interactive manipulation of the parameter values does not correspond to interactive updates to the rendered result; the result is only rendered upon user request. For dynamic manipulation interaction, parameter values can vary over a continuous range during manipulation; this range corresponds to a range of rendered results. Interactive parameter control interfaces were the norm in scientific visualization before the wide-spread availability of accelerated graphics hardware; dynamic manipulation is now

common. The only type of interaction not described by the above taxonomy are *function derivations*. In function derivations, parameter values are derived either by some sort of user-applied operator (as in the Image Graph [7] and Tory et al.'s parallel coordinate-based interface [9]) or via a user-defined function (as in the VisSheet [6]). These three interactions describe all the interactions a user can have with parameters controlling a visualization.

2.3 The Fundamental Operation of Visualization Exploration

From the previous discussion, several key properties of the visualization exploration process can be distilled. Visualization exploration is cyclic—parameters are modified repeatedly until the results of interest are generated. The parameter values are generated in one of three ways: A parameter value can be generated from an old parameter value (through interactive parameter control); a range of parameter values can be generated from an old parameter value (through dynamic manipulation); or a set of parameter values can be derived from a different set of parameter values via some operator (through a function derivation). This parameter generation is the essence of visualization exploration:

> The ***fundamental operation of the visualization exploration process*** is the application of a set of parameter values to the visualization transform to generate a visualization result.

The fundamental operation defines the visualization exploration process. It also captures the key action a user performs during visualization—the generation of visualization results. By describing how the user performs this fundamental operation, the entire visualization process is recorded.

3 The P-Set Model of Visualization Exploration

During visualization, a user repeatedly forms sets of parameter values which, when applied to a visualization transform, produces a result. To understand how this new result is related to previous results, a model of this interaction is used. The *P-Set Model of visualization exploration* encapsulates the interactions a user can have with a visualization system and how these interactions are part of the greater exploration session.

Four major elements form a visualization session: The visualization transforms used to create results, the parameter values generated by user interaction, the results created by applying sets of parameter values (*p-sets*) to visualization transforms, and the derivations that describe how the results are related to previous results. The model describes each of these (and their related elements) formally (see Appendix A for the details). The salient details are:

- A *visualization transform* is identified by the parameter types it takes as arguments and the result type it generates (these form its *signature*). Since two transforms may share the same signature but use different methods, each is also identified by a unique label.
- A *visualization parameter* is identified by its type (such as a color map) and its value (the actual color and data values). Data sets are an important class of parameter values. Non-empty groups of parameter values with unique parameter types form *p-sets*; if the set of types match that of a visualization transform's signature, the p-set can be used to generate a result. Parameters from a specific p-set are *bound parameters*; bound parameters are important in describing how results are related.
- *Visualization results* are uniquely identified by the p-set and visualization transform which generated them; since transform signatures may not be unique, both p-sets and transforms are needed to identity a result. Results also identify their actual value (e.g., a raster image) and their corresponding result type (e.g., the set of all raster images). It is important to note results do not store derivations, since it is possible to create the same result multiple times in a visualization session through different interactions.
- *Derivations* are the most detailed portion of the model and are the focus of the rest of this section. The four important pieces of a derivation are the timestamp detailing when the results were created, the parameter derivations (how the user created new parameter values from old parameters), the p-set derivations (how the new parameter values were applied to previous p-sets to create a new p-set), and the results generated (which of the new p-sets where combined with which transforms to create the actual results). Figure 2 illustrates this derivation process.

It is important to note that our model does not specify the components of the visualization transform (i.e., it does not provide a visualization transform model to break it down into sub-operations [3, 15, 20]); neither does it describe the form of the parameter and result values (i.e., it does not provide a data model for the values [4, 21, 22]). These elements are beyond the scope of the work. Transform and data models may be utilized in concert with our work to provide additional detail or as schemas for values in the XML representation in Section 4. For example, a transform model could be used to discuss how the transforms in the session were derived from each other, as in recent work by Kreuseler et al. [23] and Bavoli et al. [24].

The relationships between results, p-sets, and other results and p-sets are described by the model's derivations. As mentioned, there are four components of the derivation: A timestamp, the parameter derivations, the p-set derivations, and the results generated. Informally, derivations can be written using a three part *derivation calculus* expanded from our previous parameter derivation calculus [1] (Figure 2):

$$\begin{aligned}
&\text{Parameters:} && \left(s_1\left[n_1\right],\ldots,s_i\left[n_j\right]\right) \underset{\delta p}{\longrightarrow} \left(p_1,\ldots,p_k\right)\\
&\text{P-Sets:} && \left\{p_1,\ldots,p_m\right\} \mid s_{in} \underset{\delta s}{\longrightarrow} s_{out}\\
&\text{Results:} && t\left(s_{out}\right) \underset{\delta r}{\longrightarrow} r
\end{aligned}$$

A parameter derivation indicates that bound *input parameters* $s_i\left[n_j\right]$ from p-sets s_i were manipulated by the user to create new *output parameters* p_k; n_j represents the parameter type used from the p-set. A p-set derivation describes how a subset of these new parameters (the *applied p-set*) were used to replace the parameter values in an existing, *input p-set* s_{in} to create an *output p-set* s_{out}. Finally, the results derivation lists the actual results r created during the user's interaction. These are identified by the transform t and output p-set used (the output p-set is drawn from one of the p-set derivations). There can be multiple parameter, p-set, or result derivations created by a single user interaction; e.g., this can occur if a function is used to generate a sequence of results in a single timestep.

The following examples illustrate how user interactions are represented using the derivation calculus; Figure 3 depicts these examples using a simple visual language. In the first example, the focus node for a focus+context graph visualization (a MoireGraph [25]) was changed. The corresponding derivation describing this change

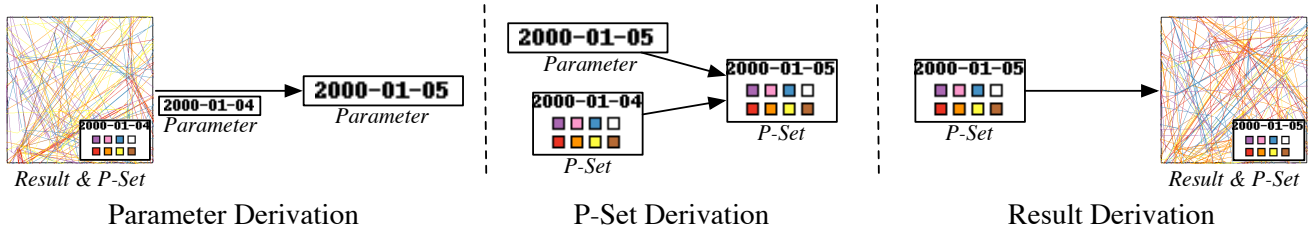


Figure 2: Depiction of a derivation using the P-Set Model. When a user creates a visualization result, three steps occur: A new parameter value is derived from an existing parameter (left), this new parameter is applied to an existing parameter set (*p-set*) to create a new parameter set (middle), and this new parameter set is applied to the visualization transform to create the result (right). In this example, the date parameter for the network visualization discussed in Section 6 was changed.

has one parameter derivation, one p-set derivation, and a single generated result. The parameter derivation uses the focus node of the previous graph (bound to that result's p-set) as its input parameter and the new focus node selected by the interaction as the output parameter. The p-set derivation's applied p-set contains only the new focus node; this parameter value replaces the old focus node in the previous result's p-set to create the output p-set. Finally, this output p-set is applied to the visualization transform to create the result of interest. The derivation clearly encapsulates that the focus node of the last result was the only difference between the last and current graph.

The next example (Figure 3b) describes a dynamic parameter manipulation interaction, in this case the synchronized rendering caused by a view-drag operation in a flow visualization. Depending on the visualization system, multiple results may be generated during this rotation. However, this rotation is considered a single user interaction, and thus all of these rotations occur within one timestep. Each "sub-derivation" within the single timestep is a single parameter derivation where the view parameter is the only parameter that changes. It is up to the visualization system to determine which of the interactively generated results are stored within the session (besides the final view position); good candidates are those the user hovers over during interaction or where the axis of rotation changes.

The final example (Figure 3c) demonstrates that the model is not restricted to derivations involving a single parameter. In the VisSheet, it is possible to combine parameter values from multiple results to create a new result; for example, it is possible to create a derivation where the view position from one result was combined with the zoom factor from another result and applied to the p-set of a third result to create the result of interest. As demonstrated, there are two input parameter values (the view and zoom parameters) which are also the output parameters (no transformation is applied to them). The applied sub-pset consists of these two parameter values which replace the view and zoom position in the input p-set; the resulting image combines the four parameters of interest into a single image. The P-Set Model derivations allow complex visualization derivations to be built from simple atomic blocks.

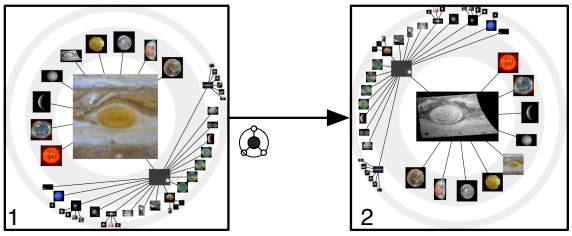
Considerable information can be extracted from the derivations and other elements in a visualization session. The number of parameters generated give a heuristic for how deeply the parameter space was explored. Results are related by the p-sets used, the bound parameters used in the parameter derivations (the p-set containing a bound parameter is a "parent" to the new result), its input p-set (another parent), and by timestamp (which give a history record of interactions). In a metadata model based upon this work, all of these elements could be annotated with the meaning of these relations. It is a fruitful area of future work to determine how these relations can be fully utilized.

4 Representation

In order to share visualizations captured by the P-set model, a common data format is required. To be effective, the format must be extensible to different visualization applications. It is also desirable that the representation can be used by data-mining or analysis tools. These goals are accomplished by using XML (the Extensible Mark-Up Language [26]) to express the visualization process. By expressing the visualization session with XML, the session can be easily shared with collaborators. Specific systems can translate their internal representation into the P-set Model (via XML) which can then be translated again into a representation usable by some other system.

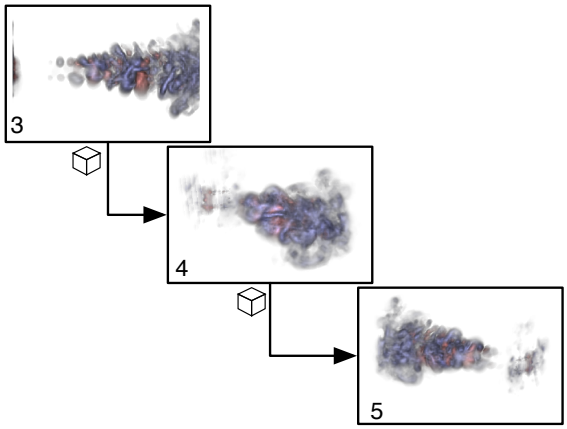
The XML representation of the model is partitioned into seven sections (see Appendix B for a full schema); these sections correspond to the five main components of the process model and additional supporting information:

- *Parameter Types* are stored first. A parameter type is an unique label; each type is given an identifier which is used to refer to the parameter type elsewhere in the XML document.
- *Result Types* are stored next. Like parameter types, each is a unique label and given an id.
- *Visualization Transforms* follow result types, and are stored by listing their signatures (parameter and result type references) and name. Transforms given a unique identifier for referral by results created using the transform.
- *Parameters* are next. Each parameter records its value, its type (by id), and a unique parameter identifier.
- *P-Sets* are stored after parameters. Each p-set contains a list of id references to parameter values composing the p-set and a unique identifier.
- *Results* are next. For each result, a reference to the transform and p-set which generated it, a reference to its type, an identifier, and the result itself are recorded. Note, for interactive systems which can generate results continuously over a range, only the first and last result in that range must be stored, the others only if desired.
- *Derivations* are stored in the last section. Each derivation stores a timestamp, parameter and p-set derivations, and references the results generated.



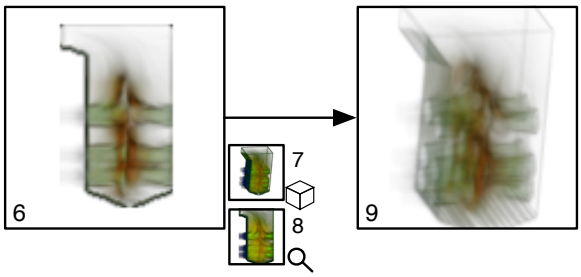
$$(s_1\,[\text{focus}_1]) \underset{\delta p}{\longrightarrow} (\text{focus}_2)$$
$$\{\text{focus}_2\} | s_1 \underset{\delta s}{\longrightarrow} s_2$$
$$t_{\text{graph}}\,(s_2) \underset{\delta r}{\longrightarrow} r_2$$

(a) Change-of-graph-focus derivation



$$(s_3\,[\text{view}_3]) \underset{\delta p}{\longrightarrow} (\text{view}_4)\,, (s_4\,[\text{view}_4]) \underset{\delta p}{\longrightarrow} (\text{view}_5)$$
$$\{\text{view}_4\} | s_3 \underset{\delta s}{\longrightarrow} s_4,\ \{\text{view}_5\} | s_4 \underset{\delta s}{\longrightarrow} s_5$$
$$t_{\text{dvr}}\,(s_4) \underset{\delta r}{\longrightarrow} r_4, t_{dvr}\,(s_5) \underset{\delta r}{\longrightarrow} r_5$$

(b) Dynamic manipulation of a view parameter



$$(s_7\,[\text{view}_7]\,, s_8\,[\text{zoom}_8]) \underset{\delta p}{\longrightarrow} (\text{view}_7, \text{zoom}_8)$$
$$\{\text{view}_7, \text{zoom}_8\} | s_6 \underset{\delta s}{\longrightarrow} s_9$$
$$t_{\text{dvr}}\,(s_9) \underset{\delta r}{\longrightarrow} r_9$$

(c) Application of view and zoom parameters from different results to a third result

Figure 3: Example derivation calculi using the P-Set model; the parameter derivation is listed first, then the p-set derivation, and finally the results. The images give a visual description of the derivations: The left image corresponds to the input p-set, the right to the output p-set, and the icons to the parameter types involved in the derivation. In cases where there are multiple derivations in a single timestep, the resulting images are chained (b). In cases where parameter values from pre-existing p-sets were used, their images are displayed next to the parameter icon (c). In the three images, a focus for a graph visualization was changed (a), a series of view positions were explored (b), and a view and zoom parameter from different results were applied to another result to create a composite result (c).

When generating the XML session document, there are different approaches to how parameters and results are stored. It is possible to embed a representation of these items directly into the XML representation; such in-line storage is especially appropriate for items with native XML formats such as vector graphics [27]. For large or binary elements, such as the data set used or the results themselves, this approach is inadvisable. Instead, each parameter or result element in the XML document could provide a URL describing where the actual parameter or result may be obtained. Linking can be used to reference large data sets over the network while accessing image files locally, avoiding costly transfers. The data set would then be downloaded only when needed. The actual schema for storing parameter and results is left up to the visualization system designer; for maximum interoperability, collaborators should agree on the same open schemas.

Note that the main purpose of the XML description of the model is for transport, not analysis. Analysis is performed on the information encoded by XML (the visualization session from this model), not on the XML document itself. Given an XML document representing a visualization session, tools are expected to parse the document into their own internal structures before operating on the visualization session information.

One possible concern regarding our approach is the growth of the XML representation as visualization sessions become longer. In the worst case, the size of the file can increase quadratically with the number of results (if every new result is derived from all previous results—an unlikely case). In practice the XML document does not approach anywhere near the size of the original data set and can be effectively compressed if needed. This growth is controlled by using externally linked binary representation for results such as images. For example, a visualization session with about 20 results represented as binary PNG images can be stored in about 1 MB (18 kB for the XML, 920 kB for the images). For the same visualization system and an exploration session with over 5500 results, the storage requirements are 285 MB (3 MB for the XML file, and 282 MB for all the images). Both sizes grow linearly. While more data points are necessary to gain a formal understanding of this growth, it is consistent with intuition. Overall, if disk space is an issue for extremely long sessions, only landmark results (chosen by the user) need to be stored at high fidelity (or at all); other results could be regenerated given an implementation of the transforms used since the information to recreate it is provided by the model.

5 Software Framework

The P-Set Model provides a conceptual framework to describe the relationships between results in a visualization exploration session. The representation provides a method to store and transfer these sessions between collaborators in a standard way. Through our experiences with the model and representation over the years (described in Section 6), we have been able to create a common software framework to assist in utilizing the model and representation in different visualization systems.

The framework is a set of software components for managing the results and relations within a visualization session; it is also responsible for storing and loading this information from XML representations. Our framework is designed as a flexible library to be added onto new or existing visualization systems. To be general, the framework is decoupled from any particular visualization method or systems. Thus, any system that uses the framework must communicate its specific behavior (e.g., what transforms its supports) to the framework. The framework has many applications. It can be used as part of a caching mechanism to optimize exploration, as

Tier	Description
Core	P-set Model implementation, session management
Representation	XML representation serialization
Interaction	User interface/visualization session coupling

Figure 4: The three layer in the visualization exploration framework. The Core provides a basis for the other two frameworks; the representation and interaction frameworks are independent of each other.

part of a visualization analysis toolkit, or as part of a logging mechanism for usability studies. This wide range of operational use is accomplished by using a three layer approach (Figure 4).

The *core layer* possesses the data structures vital to the P-Set Model: Session, VisualTransform, VisualParameter, PSet, VisualResult, and Derivation. The transform, parameter, and result classes are abstract interfaces to be implemented by system developers to describe actual transforms, parameters, and results. To assist in their usage, factory classes for these types are provided; once a developer creates a subclass and registers it with the factory, instances of the class (e.g., a parameter value) are created through the factory.

Derivation instances are constructed by calling the Session.addDerivation method with the appropriate parameter and p-set derivations and the generated results; the resulting derivations are added to the session (and new p-sets or other values are added as needed). The Session instance can also be queried for state information. This capability is vital, especially for speeding up responsiveness; the Session instance acts as a cache such that previously generated results can be returned immediately without going through a possibly expensive regeneration.

The *representation layer* builds upon the core tier by allowing Session instances to be serialized and de-serialized via our XML representation. Like the core tier, the representation tier provides customization hooks for serializing parameters and results. These objects, once registered, are automatically generated when result or parameter values are stored or loaded from session files.

Finally, the *interaction layer* supplies interactive systems a method to be notified when the session changes. An InteractiveSession object, a subclass of Session, is provided which allows other objects to register for notification of new derivations. This notification is facilitated via a standard Observer pattern [28]. This form of notification is useful when multiple views of the session are used. In addition, interface classes for parameter and result renderers (to display result and parameter values to the user) and parameter editors (to manipulate the parameters) are provided to simplify uniform implementation over several alternate user interfaces. These interface classess should be customized by system developers to suit their specific parameter and result types.

The framework benefits system designers by providing a common base for recording the information within the visualization process. To adapt the framework to a new visualization method, only classes describing the visualization transform, parameters, and results need to be created. Result and parameter types may be common to several transforms and could thus be reused. Earlier versions of this framework were integral in the examples discussed next; it is hoped that future systems will utilize the framework to gain similar benefits.

6 EXAMPLES

Our visualization exploration framework has been used in several successful projects. Its main benefits are that the entire visualization session can be analyzed and reused, promoting dissemination and system efficiency. The following two examples illustrate these points. The first example demonstrates how the framework was used to improve the effectiveness of a network infrastructure visualization; it also demonstrates visualization analysis enabled by the model. The second example discusses the integration of the framework into a grid-computing environment for remote collaboration.

6.1 System Analysis and Development

To manage packet traffic on the Internet, groups of hosts sharing portions of their IP addresses are partitioned into clusters of machines called *autonomous domains* (ASes). The problem of packet routing then simplifies to routing data between these larger entities. To assist network analysts, a tool to visualization AS changes was developed [29] (Figure 5a). The tool uses a quadtree decomposition to represent IP addresses and colored lines drawn from the edge of the quadtree to represent ASes claiming the IP address. The lines represent transfer of IP address ownership and color represents the type of change. The purpose of the tool is to expose anomalies in these transfers via correlated line patterns discovered via visual scanning of the data.

To better understand how users utilized the AS visualization, the system was augmented with the framework presented here. One visualization transform was supported—the quadtree-based event browser—with parameter types including the event type(s) used (chosen from eight types) and the date to display. Once augmented, the system stored the result and how the user generated the result for each visualization created during the visual data mining process. Several explorations were instrumented this way, and their corresponding visualization sessions were stored in XML for further analysis. Previously, we demonstrated how we could quickly augment the VisSheet to allow exploration of these explorations by using the framework [1]; here, we demonstrate graph-based analysis of the explorations.

To analyze the sessions, a graph summarizing the visualization process was used (Figure 5b). The graph utilizes the process model directly in their construction. This graph depicts the similarity of results within the graph; each self-connected cluster indicates a subspace of the exploration containing results that differ by only one parameter value. The enlarged result in the graph corresponds to the result displayed in Figure 5a—the result where the first routing anomaly occured. By analyzing this and other similar graphs (see [30] for details), it was concluded that in regions of interest, the user spent significant time toggling the different event type displays on and off in order to reduce the occlusion of the other lines in the displays. This information was used to redesign the AS event browser; the redesign uses a focus+context method to show all events and each event individually at the same time to remove the need for the "toggling" behavior [31] (Figure 5c). This new depiction allows for a streamlined exploration.

The above example depicts how the model can be used to analyze visualization sessions and visualization interfaces. As validation of visualization techniques becomes more important, the need to capture how a user interacts with an interface increases. Our framework provides the means to collect this information during user studies in more complete manner than transaction logs. Augmented by video capture and user interviews, stored representations can be used to pinpoint exactly what a user did at a given time and why

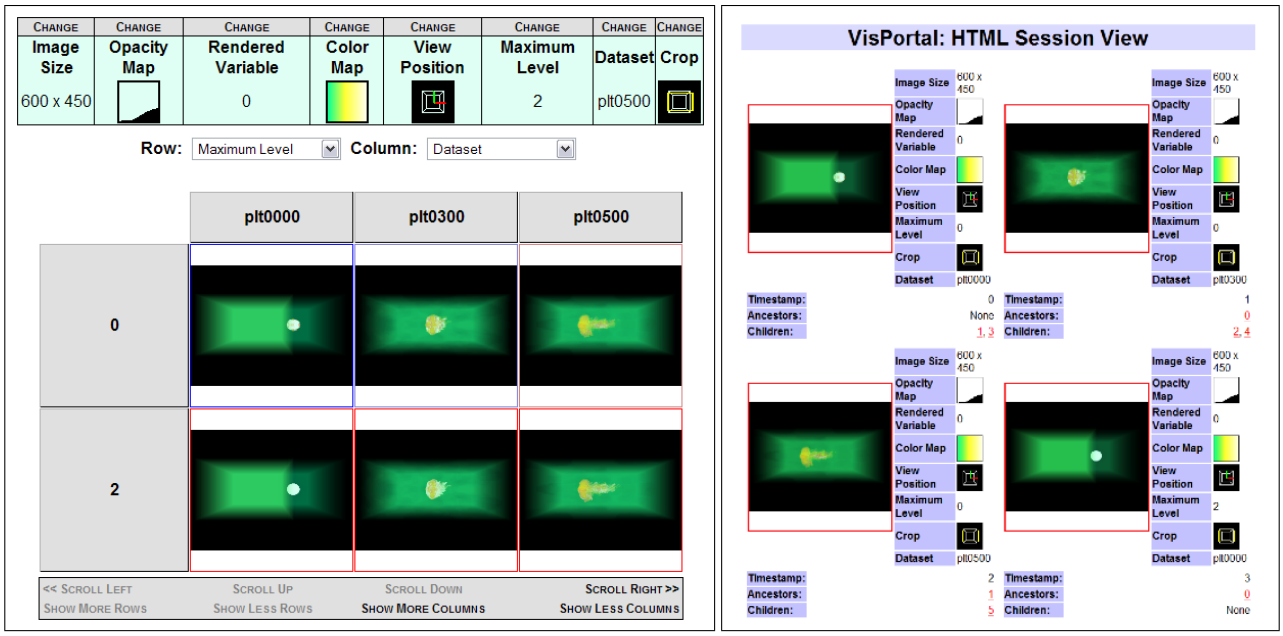


Figure 6: The WebSheet (left) uses our framework to store visualization sessions for collaborators. Sessions can be viewed and extended on-line or downloaded (right).

they performed that action. Since the representation is a complete record of the exploration, the session could then be replayed at a later time, simplifying further analysis.

6.2 Result Dissemination and Collaboration

This last example discusses the deployment of the framework within a visualization portal created by Lawrence Berkeley National Laboratory (LBNL) [32]. The purpose of the portal is to provide a web-accessible single access point for Grid-enabled visualization resources operated by LBNL. Scientists world-wide have access to the portal, and thus it is important that their explorations be stored for their collaborators. In addition, since the visualization server could be on a separate network than the portal, it is important that the visualization be efficient and not require extraneous rendering requests of previous results. Our framework fulfills both of these needs.

The portal uses a web-based implementation of the VisSheet (the WebSheet, Figure 6, left) as its interface. When a user requests a result, the web browser translates the action into a web server request. This request is treated as query upon the contents of the model—the URL contains references to the parameters in the p-set being used to create the result (the parameters are stored on the server). If the result already exists (if it is stored in the session), it is returned to the browser immediately, reducing computation. Only if a result is not found in the cache is a render request sent to the visualization server. These new result is stored by the model to reduce computation time on later requests.

During explorations with the WebSheet, the framework captures the entire session. This session is stored using the representation. This allows scientists to return to their explorations at a later date or view and extend explorations performed by others (Figure 6, right). In addition, since the exploration is encapsulated within an XML file and the result image files, a user can opt to download these files for off-line examination. The complete representation of visualization sessions provided by our framework is vital to the scientist's workflow—it provides a means for collaborating remotely and provides documentation for validation purposes.

7 Conclusions

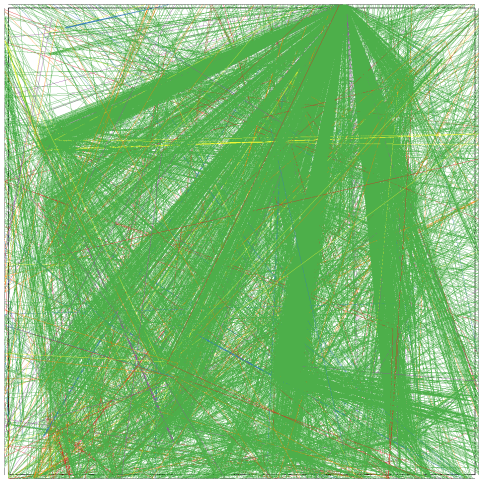
The visualization exploration process contains a wealth of information; our work has demonstrated a model to describe this information and a representation to share the information. Both the visualization technique performed and the process used to generate visualization results are captured by the model and representation. The framework discussed brings these benefits to visualization systems developers quickly and effectively.

This work impacts the user of visualization in several ways. Systems utilizing the process model assist in reuse since they clearly track where a user has been, where they are, and possibly suggest where to go. Visualizations represented using this formalism can be used in heterogeneous visualization interface environments, enabling large-scale collaboration. The salient details of the visualization process are documented, allowing others to reproduce the process. Finally, others can use the formal model to operate upon or analyze their results in a rigorous manner.

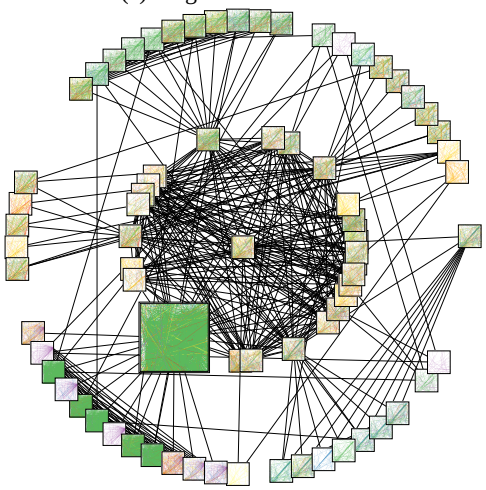
This work also contributes to the understanding of the visualization process. A characterization of user interactions with parameters during the visualization process has been performed. This characterization has led to the development of a derivation calculus to describe the relationships between results created during a visualization session. Information stored using this calculus can be analyzed and further visualized to gain insight into the visualization process itself.

7.1 Future Work

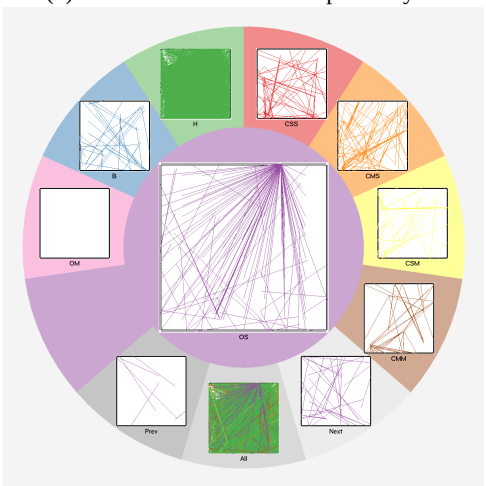
This research can be extended in several ways. The derivation calculus represents a wealth of information that has not been fully ex-



(a) Original Visualization



(b) Visualization Session Graph Analysis



(c) Refined Visualization

Figure 5: Analysis and evolution of the AS Event browser. The original interface (a) uses colored lines connected to the edges of a square to detect AS ownership changes. Our framework was used to capture user interaction with the tool, and these sessions were analyzed in order to improve the interface (b). The redesigned interface (c) makes the exploration more efficient by displaying all event types individually and combined.

ploited. Different graphical visualizations and metrics based upon the calculus need to be investigated. The visual language used in Figure 3, the relationship graphs used in our previous work [1] and in Section 6 are initial examples of this effort. These sorts of visualizations may help users or designers gain insight into previous visualization sessions.

Another important aspect in many scientific visualization sessions is the change of visualization transforms. This model does not currently store any information about modifications to the transform beyond what transforms were used. A "visualization transform derivation" model is needed. However, before this can be realized, more research unifying scientific and information visualization transform representations should be performed to assure that any subsequent work were to be general. Currently, efforts specialized for data-flow scientific visualizations are underway [24].

As visualization matures, it becomes important to understand how it fits into the workflow of actual users. User studies and validation thus becomes important. As demonstrated, our model can capture the interactions the user had with the visualization system. A more complete coupling with other validation tools (audio and video logs) to create a robust user validation framework is worth investigating.

Finally, the model does not currently store any metadata. Metadata can be used to annotate any portion of the model, including results, parameter settings, or the steps in the process. A scientist's notes about a particular result or the operation performed during a state transition are all examples of metadata to store. The video and audio logs for validation is another example of useful metadata. Metadata would provide important semantic information about the visualization process and help capture the visualization user's insight gained from the process. Like the current model, the metadata model needs to be flexible, allowing users to customize it to their specific application. More work in visualization specific and application specific ontologies—such as the effort by Duke et al. [2]—needs to be conducted before this goal can be reached.

Acknowledgments

This work was supported by the National Science Foundation and the Lawrence Livermore and Lawrence Berkeley National Laboratories. The authors would like to thank Melanie Tory and Torsten Möller for their insightful discussions, and Soon Tee Teoh for the use of the Border Gate Protocol visualization tool. Finally, we would like to thank the members of the Mississippi State Visualization, Imaging, and Analysis Laboratory and the UC Davis Visualization and Graphics Research Group for their input and assistance.

References

[1] T. J. Jankun-Kelly, K.-L. Ma, and M. Gertz, "A model for the visualization exploration process," in *Proceedings of the 13th IEEE Conference on Visualization (Vis '02)*, R. J. Moorhead, M. Gross, and K. I. Joy, Eds., 2002, pp. 323–330.

[2] D. J. Duke, K. W. Bordlie, D. A. Duce, and I. Herman, "Do you see what I mean?" *IEEE Computer Graphics and Applications*, vol. 25, no. 3, pp. 6–9, May/June 2005 2005.

[3] C. Upson, T. A. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam, "The application visualization system: A computational environment for scientific visualization," *IEEE Computer Graphics and Applications*, vol. 9, no. 4, pp. 30–42, 1989.

[4] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, 1999.

[5] T. J. Jankun-Kelly, "Visualizing visualization: A model and framework for visualization exploration," Ph.D. dissertation, University of California, Davis, Jun 2003.

[6] T. J. Jankun-Kelly and K.-L. Ma, "Visualization exploration and encapsulation via a spreadsheet-like interface," *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, no. 3, pp. 275–287, 2001.

[7] K.-L. Ma, "Image Graphs—A novel approach to visual data exploration," in *Proceedings of the 10th IEEE Conference on Visualization (Vis '99)*, D. Ebert, M. Gross, and B. Hamann, Eds., 1999, pp. 81–89, 513.

[8] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber, "Design Galleries: A general approach to setting parameters for computer graphics and animation," in *Proceedings of ACM SIGGRAPH 1997*, ser. Computer Graphics Procceedings, Annual Conference Series, 1997, pp. 389–400.

[9] M. Tory, S. Potts, and T. Möller, "A parallel coordinates style interface for exploratory volume visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 1, pp. 71–80, 2005.

[10] K. Brodlie, A. Poon, H. Wright, L. Brankin, G. Banecki, and A. Gay, "GRASPARC–A problem solving environment integrating computation and visualization," in *Proceedings of the 4th IEEE Conference on Visualization (Vis '93)*, G. M. Nielson and R. D. Bergeron, Eds., 1993, pp. 102–109.

[11] J. P. Lee and G. G. Grinstein, "An architecture for retaining and analyzing visual explorations of databases," in *Proceedings of the 6th IEEE Conference on Visualization (Vis '95)*, G. M. Nielson and D. Silver, Eds., 1995, pp. 101–108.

[12] J. P. Lee, "A systems and process model for data exploration," Ph.D. dissertation, U. of Massachuesetts Lowell, 1998.

[13] D. M. Hilbert and D. F. Redmiles, "Extracting usability information from user interface events," *ACM Computing Surveys*, vol. 32, no. 4, pp. 384–421, 2000.

[14] D. Duke, G. Faconti, M. Harrison, and F. Paternò, "Unifying views of interactors," in *Proceedings of the Workshop on Advanced Visual Interfaces (AVI '94)*, T. Catarci, M. F. Costabile, S. Levialdi, and G. Sanctucci, Eds. ACM Press, 1994, pp. 143–152.

[15] E. H.-H. Chi and J. T. Riedl, "An operator interaction framework for visualization systems," in *Proceedings of the 1998 IEEE Symposium on Information Visualization (InfoVis '98)*, J. Dill and G. Wills, Eds., 1998, pp. 63–70.

[16] M. C. Chuah and S. F. Roth, "On the semantics of interactive visualizations," in *Proceedings 1996 IEEE Symposium on Information Visualization (InfoVis '96)*, S. Card, S. Eick, and N. Gershon, Eds. IEEE Computer Society, 1996, pp. 29–36.

[17] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proceedings of the 12th IEEE Symposium on Visual Languages (VL '96)*. IEEE Computer Society Press, 1996, pp. 336–343.

[18] S. Wehrend and C. Lewis, "A problem-oriented classification of visualization techniques," in *Proceedings of the First IEEE Conference on Visualization (Vis '90)*, A. Kaufman, Ed. IEEE Computer Society Press, 1990, pp. 139–143, 469.

[19] P. Rheingans, "Are we there yet? Exploring with dynamic visualization," *IEEE Computer Graphics and Applications*, vol. 22, no. 1, pp. 6–10, 2002.

[20] W. L. Hibbard, C. R. Dyer, and B. E. Paul, "A lattice model for data display," in *Proceedings of the 5th IEEE Conference on Visualization (Vis '94)*, R. D. Bergeron and A. E. Kaufman, Eds., 1994, pp. 310–317.

[21] D. M. Butler and M. H. Pendley, "A visualization model based on the mathematics of fiber bundles," *Computers in Physics*, vol. 3, no. 5, 1989.

[22] L. Treinish, "A function-based data model for visualization," in *Proceedings of IEEE Visualization '99 Late Breaking Hot Topics*, 1999, pp. 73–76.

[23] M. Kreuseler, T. Nocke, and H. Schumann, "A history mechanism for visual data mining," in *Proceedings 2004 IEEE Symposium on Information Visualization (InfoVis '04)*, M. Ward and T. Munzner, Eds. IEEE Computer Society, 2004, pp. 49–56.

[24] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "Vistrails: Enabling interactive multiple-view visualizations," in *Proceedings of the 16th IEEE Conference on Visualization (Vis '05)*, C. T. Silva, E. Gröller, and H. Rushmeier, Eds., 2005, pp. 135–142.

[25] T. J. Jankun-Kelly and K.-L. Ma, "MoireGraphs: Radial focus+context visualization and interaction for graphs with visual nodes," in *Proceedings of the 2003 IEEE Symposium on Information Visualization*, T. Munzner and S. North, Eds., IEEE Computer Society TCVG. IEEE Computer Society Press, 2003, pp. 59–66.

[26] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, "Extensible Markup Language (XML) 1.0 (Second Edition)," World Wide Web Consortium, Tech. Rep., 2000, http://www.w3.org/TR/REC-xml.

[27] J. Ferraiolo, J. Jun, and D. Jackson, "Scalable vector graphics (svg) 1.1 specification," World Wide Web Consortium, Tech. Rep., 2003.

[28] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns; Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[29] S. T. Teoh, K.-L. Ma, F. Wu, and X. Zhao, "Case study: Interactive visualization for internet security," in *Proceedings of 13th IEEE Conference on Visualization (Vis '02)*, R. Moorhead, M. Gross, and K. I. Joy, Eds., 2002, pp. 505–508.

[30] T. J. Jankun-Kelly, "Using visualization process graphs to improve visualization exploration," Department of Computer Science and Engineering, Mississippi State University, Tech. Rep. MSU-060315-2, 2006.

[31] S. T. Teoh, , T. J. Jankun-Kelly, K.-L. Ma, and F. Wu, "Visual data analysis for detecting flaws and intruders in computer network systems," *IEEE Computer Graphics and Applications*, vol. 24, no. 5, September/October 2004.

[32] T. J. Jankun-Kelly, O. Kreylos, J. M. Shalf, K.-L. Ma, B. Hamann, K. I. Joy, and E. W. Bethel, "Deploying web-based visual exploration tools on the grid," *IEEE Computer Graphics and Applications*, vol. 23, no. 2, pp. 40–50, 3 2003.

A. Formal P-Set Model Definition

A visualization exploration session $E = (T, P, S, R, \Delta)$ consists of sets of visualization transforms T, parameter values P, parameter value sets (p-sets) S, result values R, and result derivations Δ. For such a session, the following holds:

- **Transforms:** $T = \{ t | t = (n_t, N_p, n_r) \}$ such that $n_t, n_r \in N, n_t \neq n_r$, are the transform and result names respectively (N is a set of unique identifiers/labels) and $N_p \subset N$ are the names of the parameter types used by the transform. The non-empty set of parameter types and the result type form the transform's *signature* (i.e., $t(N_p) \mapsto n_r$). As it is conceivable that multiple transforms may share the same signature but utilize different rendering methods, the transform's name provides a unique identifier for the transform.

- **Parameters:** $P = \{ p | p = (n_p, v_p) \}$ where $n_p \in N$ specifies the parameter's type and $v_p \in V$ is the parameter's value. V is the universe of possible values for parameters and results.

- **P-Sets:** $S = \{ s | s \subset \mathscr{P}(P)$ where there exists a transform $t = (n_t, N_p, n_r) \in T$ such that there exists a bijective map $f : s \mapsto N_p$ where $f(s) = \{ n_i | n_i \in p_i$ for each $p_i \in s \} \}$. A p-set s is a set of parameters. There is a one-to-one correspondence between parameter types within the p-set and the parameter types of a transform in the session. A *sub-pset* relaxes this constraint such that some parameter types are missing (though a single parameter type is not repeated within a

different parameter value); sub-psets are used in p-set derivations.

- **Results:** $R = \{ r | r = (t, s, n_r, v_r) \}$ where $t \in T$, $s \in S$, $n_r \in N$ is the result type, and $v_r \in V$ is the result's value. Both a p-set s and the transform t used with that p-set are needed to uniquely identity a result. Like parameter types, result types are unique labels.
- **Derivations:** $\Delta = \{ \delta | \delta = (\tau, \Delta P, \Delta S, R_\tau) \}$ where for a given derivation δ, τ is the derivation's unique timestamp, ΔP are the parameter derivations, ΔS are the p-set derivations, and R_τ are the results generated by the derivation such that
  - $\tau \in \mathbb{R}$. Timestamps are monotonically increasing.
  - $\Delta P = \{ \delta p | \delta p = (P_{in}, P_{out}) \}$ such that $P_{in} \subset \mathscr{P}(P \times S)$ where $\forall (p_i, s_i) \in P_{in},\ p_i \in s_i$; and $P_{out} \subseteq \mathscr{P}(P) - \emptyset \}$. A parameter/p-set pair $(p_i, s_i) \in \mathscr{P}(P \times S)$ where $p_i \in s_i$ is called a *bound parameter* as it directly references the p-set from which the parameter was used—bound parameter are vital to understanding which results were the genesis of the derived results. Note, it is possible for $P_{in} = \emptyset$; this occurs when the parameter was not derived from any other parameter.
  - $\Delta S = \{ \delta s | \delta s = (s_a, s_{in}, s_{out}) \}$ where $s_a = \left\{ p = (n_p, v) \middle| \exists (P_{in}, P_{out}) \in \Delta P \text{ such that } p \in P_{out} \text{ and } \nexists p' = (n'_p, v') \in s_a \text{ such that } n_p = n'_p \right\}$ is a sub-pset formed about of output parameters, $s_{in} \in S \cup \emptyset$, and $s_{out} \in S$ such that $s_{out} = \left\{ p = (n_p, v) \middle| p \in s_{applied} \vee \left(p \in s_{in} \wedge \nexists p' = (n'_p, v') \in s_a \text{ such that } n_p = n'_p \right) \right\}$. The parameters in s_a replace the parameters in s_{in} to generate s_{out}. Like in the parameter derivations, $s_{in} = \emptyset$ only if the no previous p-set was used to generate the result; in this case, $s_a = s_{out}$.
  - $R_\tau \subseteq \mathscr{P}(R) - \emptyset$ where $\forall r = (t, s_{out}, n_r, v) \in R_\tau, \exists \delta s = (s_a, s_{in}, s_{out})$ such that $\delta s \in \Delta S$. R_τ represents all the results r derived from the various output p-sets s_{out} created by the derivation.

B. XML Representation Schema

The following provides a Document Type Definition (DTD) [26] schema for XML representations of visualization sessions using the P-Set Model. The XML namespace name for the representation is *http://vis.cse.msstate.edu/vex/core*.

```
<!ELEMENT visualization (parameterTypes, resultTypes, transforms,
    parameters, parameterSets, results, derivations)>

<!ELEMENT parameterTypes (parameterType)+>

<!ELEMENT parameterType name>
<!ATTLIST parameterType
  id ID #REQUIRED >

<!ELEMENT name #CDATA>
<!ATTLIST name
  shortName CDATA #IMPLIED >

<!ELEMENT resultTypes (resultType)+>

<!ELEMENT resultType name>
<!ATTLIST resultType
  id ID #REQUIRED >

<!ELEMENT transforms (transform)+>

<!ELEMENT transform (name, ptypeRef+, rtypeRef)>

<!ELEMENT ptypeRef EMPTY>
<!ATTLIST ptypeRef
  ref IDREF #REQUIRED >

<!ELEMENT rtypeRef EMPTY>
<!ATTLIST rtypeRef
  ref IDREF #REQUIRED >

<!ELEMENT parameters parameter*>

<!ELEMENT parameter (ptypeRef, value)>
<!ATTLIST parameter
  id ID #REQUIRED>

<!ELEMENT value #PCDATA>
<!ATTLIST value
  href CDATA #IMPLIED >

<!ELEMENT parameterSets parameterSet*>

<!ELEMENT parameterSet parameterRef+>
<!ATTLIST parameterSet
  id ID #REQUIRED>

<!ELEMENT parameterRef EMPTY>
<!ATTLIST parameterRef
  ref IDREF #REQUIRED >

<!ELEMENT results result*>

<!ELEMENT result (rtypeRef, psetRef, transformRef, value)>
<!ATTLIST result
  id ID #REQUIRED>

<!ELEMENT psetRef EMPTY>
<!ATTLIST psetRef
  ref IDREF #REQUIRED >

<!ELEMENT transformRef EMPTY>
<!ATTLIST transformRef
  ref IDREF #REQUIRED >

<!ELEMENT derivations derivation*>

<!ELEMENT derivation (timestamp, parameterDeltas, psetDeltas,
    resultsGenerated)>

<!ELEMENT timestamp #PCDATA>

<!ELEMENT parameterDeltas parameterDelta+>

<!ELEMENT parameterDelta (inputParameters, outputParameters)>

<!ELEMENT inputParameters (boundParameter* | undefined)>

<!ELEMENT boundParameter (parameterRef, psetRef)>

<!ELEMENT undefined EMPTY>

<!ELEMENT outputParameters parameterRef+>

<!ELEMENT psetDeltas psetDelta+>

<!ELEMENT psetDelta (appliedPset, inputPset, outputPset)>
```

```
<!ELEMENT appliedPset parameterRef+>

<!ELEMENT inputPset (undefined|psetRef)>

<!ELEMENT outputPset psetRef>

<!ELEMENT resultsGenerated resultRef+>

<!ELEMENT resultRef EMPTY>
<!ATTLIST resultRef
   ref IDREF #REQUIRED >
```