Provenance Segmentation

Rui Abreu
Palo Alto Research Center
rui@parc.com

Dave Archer
Galois, Inc.
dwa@galois.com

Erin Chapman
Galois, Inc.
erin@galois.com

James Cheney
University of Edinburgh
jcheney@inf.ed.ac.uk

Hoda Eldardiry
Palo Alto Research Center
hoda.eldardiry@parc.com

Adrià Gascón
University of Edinburgh
agascon@inf.ed.ac.uk

Abstract

Using pervasive provenance to secure mainstream systems has recently attracted interest from industry and government. Recording, storing and managing all of the provenance associated with a system is a considerable challenge. Analyzing the resulting noisy, heterogeneous, continuously-growing provenance graph adds to this challenge, and apparently necessitates *segmentation*, that is, approximating, compressing or summarizing part or all of the graph in order to identify patterns or features. In this paper, we describe this new problem space for provenance data management, contrast it with related problem spaces addressed by prior work on provenance abstraction and sanitization, and highlight challenges and future directions toward solutions to the provenance segmentation problem.

1. Introduction

The relationship between provenance and security has been considered by a number of authors over the last few years. Some researchers have proposed using provenance as a basis for security (e.g. provenance-based access control [20]) as well as of the security implications of provenance-tracking (e.g. techniques for provenance abstraction, sanitization, or redaction; see [6] for a survey). An emerging application of provenance is its use in analyzing cause and effect in system activity as a means of identifying security threats to that system.

An advanced persistent threat (APT) is a stealthy, long-term application intended to penetrate a system, persist in it, and carry out either pre-determined or dynamically updated instructions from an adversary. Those instructions may result in continuing exfiltration of sensitive data as seen in the Office of Personnel Management (OPM) data breach [19], continuing corruption of data [8], or the capability to shut down [3] or damage [10] critical systems such as the industrial control systems that manage power grids. The stealthiness of APTs makes it difficult to detect them. APTs tend to rely on actions that violate no system security policies, nor any social norms (such as work schedules) of the authorized users of targeted systems.

TaPP 2016, June 8–9, 2016, Washington, DC.

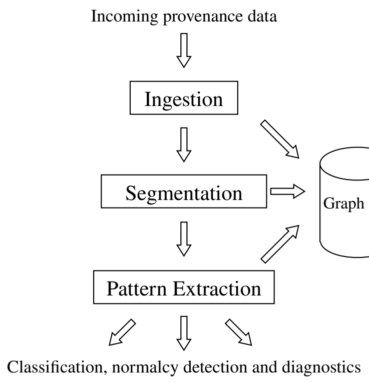

Figure 1. Architectural overview

The increase in such sophisticated threats has motivated the use and mining of *pervasive provenance* to identify relationships among system activities that may indicate malicious actions undetectable by policy-driven approaches. For example, DARPA's recently-funded *Transparent Computing* research program aims to instrument critical systems with pervasive logging of system activities, and apply provenance analysis that continuously monitors such logs to identify actions that could be part of an APT. The idea of monitoring provenance-like records to aid security has appeared in several places [14, 16, 21], but there are numerous challenges to making it a reality.

In a transparent computing system, there is a separation of concerns between *provenance recorders*, which run on the critical systems we wish to protect, and *provenance analysis*, which may run on separate systems, either in forensic (batch processing) or online (stream processing) mode. It is expected that provenance recorders will provide large volumes of data at high velocity (at least in streaming mode). Therefore, provenance analysis systems will need to deal with large and rapidly growing data sets and do so with latencies that allow cyber network defense (CND) operators to mount effective defenses.

We are among the participants in the ADAPT (A Diagnostic Approach to Advanced Persistent Threat detection) project within the DARPA Transparent Computing program, whose goal is to analyze and recognize advanced persistent threats in provenance graphs. Our approach seeks to combine state-of-the-art provenance and database management technology, statistical anomaly detection, machine learning and diagnostics techniques [1]. In the initial stages of the design of this system, we have identified the interface between the large-scale provenance graph data and the inputs expected by existing anomaly detection and classification and diagnostics techniques as a key design question. In particular, it appears necessary to perform some kind of approximation or downsampling on the raw provenance graph in order to provide inputs of manageable size for later stages. We call this problem *provenance segmentation*.

The ADAPT system (illustrated in Fig. 1) receives provenance graph data emitted by other systems. It *ingests* the data by storing the raw provenance graph in a graph database, then *segments* the graph to define an approximate provenance graph (which we call the *segment layer*) that can be used as a high-level proxy for the raw graph. *Feature extraction* is then performed on segments to summarize their contents. The segment layer and feature annotations are intended for the use of later stages (not shown) that recognize various types of activity detected in the graph, detect normal or abnormal behavior, or diagnose potential APT activity by analyzing causal relationships and matching patterns of behavior against known strategies recorded in a knowledge base. The intent is that most of the work of later stages can be performed at the segment layer, and when potential attacks are identified at that layer, the raw graph of the relevant segments can be inspected in detail.

In this paper, we present our analysis of the segmentation problem, whose requirements and design constraints have turned out to be relatively difficult to pin down, and differs in subtle but important respects from prior work on graph abstraction. This is work in progress, and we do not report on a fully-evaluated research contribution, but rather we hope that carefully describing the problem space will be of interest to other researchers seeking challenges in provenance data management. We believe that the problem(s) of segmentation are likely to be shared by other research projects seeking to analyze large amounts of continuously growing provenance data, and hope that calling attention to this problem will spur progress in this area.

2. Problem statement

We assume standard notation for directed, labeled graphs $G = (V, E \subseteq V \times V, \lambda : V \cup E \to \Sigma)$ where Σ is some set of labels on vertices and edges. We define a *segment* to be a subgraph of G and write $Seg(G)$ for the set of all segments of G.

We consider two high-level forms of the segmentation problem.

- The *batch segmentation problem* takes as input a graph (the *raw graph*), a collection of *segmentation criteria* that define segments and the relationships among them, and produces as output a summary graph (the *segment layer*), along with cross-links from the nodes of the segmentation layer to the associated subgraphs.
- The *incremental segmentation problem* takes as input a raw graph, segmentation criteria and a sequence of insertions to the graph. The problem is to initialize and maintain the segmentation layer efficiently as the updates to the graph are performed.

In both cases, the principal output is the segment layer $G' = (V', E', \lambda')$ along with a mapping $Sg : V' \to Seg(G)$ from vertices of the segment layer to their associated segments, that is, subgraphs of G.

We do not assume that we have control over the behavior of the provenance recorders providing the input graph. Therefore, although we informally refer to the input graph as *the provenance graph* and the edge relationships as *causal*, this terminology should be taken with a grain of salt: it assumes a level of shared under-

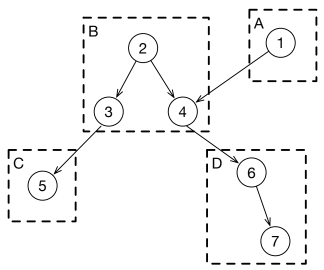

Figure 2. Example of lossy segmentation

standing between recorders and analyzers concerning the semantics of the provenance graphs that has not yet been attained. In this paper, we focus on defining segmentation pragmatically in terms of structural properties of the input graph, and place the question of whether the input graph is "correct" or "complete" outside the scope of this paper.

The provenance recorders will provide data by which to structure G as well as data that will describe the nodes and edges of G. For the moment, we consider two basic kinds of segmentation criteria based on such properties attached to nodes and edges in G. We note that these kinds of segmentation can obviously be combined or generalized:

- *neighborhood* segments consisting of all nodes and edges within a given distance of a given node; for example, all activity involving the process with PID 9002.
- *temporal* segments containing all activity observed (or reported) in some time interval (of any length); e.g. all activity between 11:00 and 11:05 GMT on December 1, 2015, or all activity during December 2015.

We believe that the former has direct application to APT discovery, while the latter may or may not, because APTs are typically extended campaigns with activities that occur over months to years, possibly eluding short-term time-based analysis. Segmentation may also draw on techniques for pattern-based semantic annotation of the graph (which we call "pattern extraction" in our current architecture).

2.1 Requirements

The functional requirements on the solution of the segmentation problems introduced above are related to the specific *purpose* of the segmentation process. The architectural overview depicted in Figure 1, in particular in the case of the *incremental segmentation problem*, can be seen as an instance of *runtime verification*. Runtime verification is a program analysis technique based on monitoring the execution of a running system in order to react to observed behavior satisfying a certain property. In runtime verification, an execution trace, i.e. a fixed representation of the execution, is provided in an online fashion to some inference system that checks the property to be verified. The segmentation process of the ADAPT system can be seen as an abstraction/annotation process of the execution trace to be provided to the inference systems that ultimately detect an ongoing attack.

Although detailing the algorithms that operate on the segmented graph produced by the segmenter is out of the scope of this paper, we identify several classes of requirements.

Updatability The interaction between the segmenter and the classification and diagnosis algorithms could also use feedback from such inference systems to refine the segmentation, as in a Counterexample-Guided Abstraction Refinement (CEGAR) [7]. A scalable approach using this kind of technique would require an efficient way of updating the segmentation. This would require a data model for segments that supports efficient updates, which would in turn provide a way for the algorithms in charge of ultimately finding attacks to "guide" the segmentation.

Validity Intuitively, we would like it to be the case that the segmentation layer does not lose information with respect to the full graph: for example, does not introduce or remove existing paths. However, since the purpose of segmentation is to provide a more compact and tractable representation of the graph, information loss is sometimes inevitable. For example, Figure 2 illustrates a segmentation of a graph and an edge that is required in order to preserve paths, but also introduces a false path. Therefore, "correctness" will generally be a weaker property, such as that all paths are preserved (but some false paths are introduced), or vice versa.

Hierarchical decomposition The segment layer should enable multi-resolution modeling where activity recognition algorithms can operate on high-level representations of the graph, and also zero in on low-level details when necessary. The goal of the activity classifier is to annotate the provenance graph with semantic descriptions of activities within each segment. This can involve recognizing composite activities. For example, an activity "remote shell access" is composed of a sequential set of logged events "browser forking bash", "bash initiating Netcat", and "Netcat listening on new port". The segmentation technique should therefore capture the various levels of semantic abstraction.

Features The segment layer should provide summary information useful for activity classification and normalcy detection models. These models will consume feature vectors that characterize key properties of the provenance graph (e.g., graph normality), perform intelligent analysis of these features, and output normalcy scores or activity descriptions. Different segmentations will provide different "views" of information in the provenance graph that will be useful to construct these features. For example, one view of the data may produce vectors that correspond to login activity or activity related to a system resource, such as a password file. This will enable ADAPT to implement powerful information fusion approaches to combine multiple sources of evidence during provenance analysis [11]. As a starting point, features could consist of any aggregate queries answerable from the segment extent.

Boundary definitions The segment layer should provide boundary definitions that allow useful local activity classifications within segment boundaries. The diagnostic engine can then globally recognize APT activities that span multiple segments, crossing these segment boundaries. Key challenges such as segmentation noise and fuzzy boundaries, where activities span multiple segments, should be considered.

Properties of the segmentation layer It is not yet clear what properties are desirable for the segmentation layer; this is a topic of active discussion. For example, some diagnosis algorithms require a directed acyclic graph, whereas there may easily be cycles in the segmentation layer according to the relationships mentioned at the beginning of this section. Do these cycles need to be removed (as, for example, in PASS [18]), in some principled way? Likewise, the segmentation layer is redundant, in the sense that it may be recomputed. Are segments themselves persistent or volatile? How long should they persist?

2.2 Design Considerations

Scale and Incremental Processing We are interested in segmentation in order to mediate between a large, and rapidly growing, amount of detailed data and sophisticated classification or diagnosis algorithms. Thus, the segmentation process itself must scale to large amounts of data. This suggests that segments should be relatively small, simple and easy to identify. Likewise, assuming the raw graph is stored in a database, it seems advantageous for segments to be easily definable via queries (and recorded using updates) in the same formalism.

Segment constraints and representations Other forms of provenance abstraction make stronger assumptions on the structure of the abstracted subgraphs, such as convexity (see the next section for additional discussion). We also want to record the relationship between a node in the segment layer and its extent (the actual subgraph it represents). Representing a segment explicitly as a set of nodes and edges could become expensive, however. If segments are assumed convex, then only the set of nodes needs to be recorded and the edges can be inferred. For non-convex segments, efficient representations (and techniques for recovering the extent from the representation) may need to be designed.

Incomplete information Temporal segments can be defined either in terms of the *transaction time* (that is, the time the corresponding raw graph data is recorded or entered into the database) or *valid time* (that is, the time the recorded events actually happened, according to the monitor). Valid time appears preferable, but complete valid time for all events may not be available, which leads to a problem of incomplete information (similar to that studied by Kwasnikowska et al. [15] for OPM). If transaction time is used, this may affect the extent to which time-based information at the segmentation layer is useful to later processing.

3. Related approaches

Provenance segmentation is similar to several previously-studied problems, such as *user views* [4], *hierarchical provenance* [5], *provenance publishing* [9], and *provenance abstraction* [17]. We surveyed several provenance abstraction techniques previously [6]; here we focus only on closely related work.

User views, as implemented in the ZOOM system [4], mediate the full detail of the provenance graph for consumption by users who may only be interested in high-level details. Given a workflow and a specification of the workflow steps that are of interest to the users, ZOOM computes a provenance graph that (roughly speaking) combines uninteresting nodes as much as possible. While the end result is similar to the segmentation layer, ZOOM requires a pre-defined workflow graph that can be annotated with user preferences. In our setting, we have no such workflow and even if we did, it is not clear how to elicit appropriate preferences.

Buneman et al. [5] proposed a hierarchical data model for provenance, and a technique for populating the hierarchical representation starting from simple functional programs, and using the call hierarchy as the basis for extracting abstract "views" of the raw graph. The segmentation layer we consider can be viewed as an instance of this data model (with just one layer of abstraction), but the techniques for populating and viewing the data are not directly applicable in this setting because (as with ZOOM) there is no fixed, known "program" generating the provenance graph.

Archer et al. proposed a hierarchical model for provenance of relational data as part of the Multi-granularity, Multi-Provenance Model (MMP) [2]. MMP encodes provenance at the coarsest granularity for each relational algebra operation. MMP database instances thus store the minimum metadata needed to represent complete provenance of the data. The segmentation layer we consider can be viewed as a data model layer analogous to the higher levels of the relational model such as records or relations. However, the techniques proposed in that work only apply to relational data and the usual operators used to define, manipulate, and query such data.

The ProPub system [9] "publishes" provenance according to a policy, specifying the requirements on what components of the provenance graph to make visible and what components to abstract. The ProvAbs system [17] considers provenance graph abstraction operations such as taking the “convex closure” of a subgraph (to include all nodes and edges between existing nodes of the subgraph), extending subgraphs to make them compatible with an abstracted node type, and replacing a subgraph with an abstracted node. Both ProPub and ProvAbs operate on provenance graphs and do not require a workflow or program to be specified in advance. However, segments need not be convex and the segmentation graph need not be a provenance graph so it is unclear whether ProPub or ProvAbs policies match the requirements for provenance segmentation.

Provenance segmentation also appears related to the problem of *streaming graph partitioning* [12, 22]. The goal of graph partitioning is to distribute graph data across nodes so as to allow efficient distributed query processing. When the graph data is continuously growing, it is of interest (just as in the case of provenance segmentation). However, the end result of (streaming) graph partitioning is a fixed partition of the whole graph, possibly with a query workload in mind [12]. In contrast, segments may overlap and not all nodes are part of a segment.

Provenance segmentation is also related to *graph summarization*. Related work [13] aims to exploit the graph structure to summarize and compress relational knowledge. This is done by extracting patterns and compressing structural knowledge encoded within relational graphs. Potential compression paths are found within repetitive or sequential structures. This makes it possible to augment initial knowledge by adding qualitative relationships over generalized entities.

4. Conclusions and open problems

We have introduced the provenance segmentation problem and discussed it in the context of a specific application, in which classification and diagnostics algorithms are to be performed on an approximate representation of the raw provenance graph. We can generalize this picture to consider any analysis that is too expensive to be performed routinely on the full provenance graph, but may be able to work with an approximation of the graph. Given such an algorithm, the following general problems seem worthwhile to investigate:

- *Validity of segmentation*: What formal guarantees need to hold in order for algorithms working on the segment layer to produce acceptable results with respect to the raw graph?
- *Incremental segmentation*: Given a segmentation strategy, how can we incrementalize it so that whenever the raw graph is (additively) updated, we can quickly produce the new segments and edges for the updated graph?
- *Adaptive segmentation*: Given algorithms that operate on the segment layer, and an annotated training set that we can use to evaluate how well the algorithms are doing using a given segmentation strategy, can we learn a good segmentation strategy?

Acknowledgements This material is based upon work partially supported by the Defense Advanced Research Projects Agency (DARPA) under contract FA8650-15-C-7557.

References

[1] Rui Abreu, Daniel G. Bobrow, Hoda Eldardiry, Alexander Feldman, John Hanley, Tomonori Honda, Johan de Kleer, Alexandre Perez, Dave Archer, and David Burke. Diagnosing advanced persistent threats: A position paper. In *Proceedings of the 26th International Workshop on Principles of Diagnosis (DX-2015)*, pages 193–200, 2015.

[2] David W. Archer, Lois M. L. Delcambre, and David Maier. User trust and judgments in a curated database with explicit provenance. In *In Search of Elegance in the Theory and Practice of Computation*, volume 8000 of *Lecture Notes in Computer Science*, pages 89–111. Springer, 2013.

[3] M. Assante. Confirmation of a coordinated attack on the ukrainian power grid. 2016.

[4] Olivier Biton, Sarah Cohen-Boulakia, Susan B. Davidson, and Carmem S. Hara. Querying and managing provenance through user views in scientific workflows. In *ICDE*, pages 1072–1081. IEEE, 2008.

[5] Peter Buneman, James Cheney, and Egor V. Kostylev. Hierarchical models of provenance. In *TaPP 2012*, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[6] James Cheney and Roly Perera. An analytical survey of provenance sanitization. In *IPAW 2014*, number 8628 in Lecture Notes in Computer Science, pages 113–126. Springer-Verlag, 2015.

[7] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[8] D. Deka, R. Baldick, and S. Vishwanath. Data attacks on power grids: Leveraging detection. *Innovative Smart Grid Technologies Conference (ISGT)*, 2015.

[9] Saumen C. Dey, Daniel Zinn, and Bertram Ludäscher. ProPub: Towards a declarative approach for publishing customized, policy-aware provenance. In *SSDBM*, pages 225–243, 2011.

[10] Marcos Donolo, Armando Guzman, Venkat Mynam, Doug Salmon, and Mark Zeller. Mitigating the aurora vulnerability with existing technology. 2009.

[11] Hoda Eldardiry, Kumar Sricharan, Juan Liu, John Hanley, Robert Price, Oliver Brdiczka, and Eugene Bart. Multi-source fusion for anomaly detection: using across-domain and across-time peer-group consistency checks. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 5(2):39–58, 2014.

[12] Hugo Firth and Paolo Missier. Workload-aware streaming graph partitioning. In *Workshop on Querying Graph Structured Data (GraphQ 2016)*, 2016. `http://ceur-ws.org/Vol-1558/paper26.pdf`.

[13] Scott E. Friedman. Exploiting graph structure to summarize and compress relational knowledge. In *Workshop on Qualitative Reasoning.*, 2015.

[14] Eleni Gessiou, Vasilis Pappas, Elias Athanasopoulos, Angelos D. Keromytis, and Sotiris Ioannidis. Towards a universal data provenance framework using dynamic instrumentation. In *SEC 2012*, pages 103–114, 2012.

[15] Natalia Kwasnikowska, Luc Moreau, and Jan Van den Bussche. A formal account of the open provenance model. *TWEB*, 9(2):10, 2015.

[16] Michael P. Mesnier, Matthew Wachs, Raja R. Sambasivan, Julio López, James Hendricks, Gregory R. Ganger, and David R. O’Hallaron. //trace: Parallel trace replay with approximate causal events. In *FAST 2007*, pages 153–167, 2007.

[17] Paolo Missier, Jeremy Bryans, Carl Gamble, Vasa Curcin, and Roxana Dánger. ProvAbs: Model, policy, and tooling for abstracting PROV graphs. In *IPAW 2014*, pages 3–15, 2014.

[18] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware storage systems. In *USENIX Annual Technical Conference*, pages 43–56, 2006.

[19] US Office of Personnel Management Office of the Inspector General. Final audit report, report number 4a-ci-00-14-016. 2014.

[20] Jaehong Park, Dang Nguyen, and Ravi S. Sandhu. A provenance-based access control model. In *PST 2012*, pages 137–144, 2012.

[21] Manolis Stamatogiannakis, Paul Groth, and Herbert Bos. Decoupling provenance capture and analysis from execution. In *TaPP ’15*, Edinburgh, Scotland, July 2015. USENIX Association.

[22] Isabelle Stanton and Gabriel Kliot. Streaming graph partitioning for large distributed graphs. In *KDD 2012*, pages 1222–1230, New York, NY, USA, 2012. ACM.

A. Overview of current approach

In this appendix, we provide details of our current prototype approach to segmentation. This is work in progress and likely to change in future versions of the system as we gain experience with real data.

A.1 Segmentation specifications

We use *segmentation specifications* to describe what kinds of segments to construct and how to annotate the resulting segment layer nodes.

The abstract syntax of segmentation specifications is as follows:

```
<rule>  ::= segment <name> (<prop> = <exp>, ...)
            by <specs>
<specs> ::= <spec> | <spec> and <specs>
<spec>  ::= time window <time>
            (from <time>)?
            (starting <var>)?
          | radius <num> from <prop>=<exp>
            (following <edges>)

<prop>   = ... //property names
<var>    = ... // variables
<exp>    = <num> | <string> | <var>
<num>    = ... // numbers
<string> = ... // string literals
<time>   = ... // times/durations
<edges>  = ... // sets of edge names
```

In our prototype implementation, we represent such specifications as JSON data structures to save effort on parsing. An example is shown in Figure 3.

The idea here is that each `spec` is a rule for identifying matching subgraphs, and this matching process may also result in binding some variables to values. The segment name is a special property that every segment has, so that we can tell different kinds of segments apart, and the additional `<prop>=<exp>` pairs are additional properties that get added to the resulting segment node, so that we can see that e.g. a given segment is for PID 42's activity starting at a given time. We can choose to include more information (such as the duration) or less.

For time window segments

```
time window td from ts starting T
```

the window time `td` is the duration of each segment, and the optional time `ts` is the time of the first segment in the window series. `starting T` means that for each resulting segment, `X` is bound to the start time of the segment's time window. (The end time could also be stored, but is recoverable from the segment specification and the start time.)

For radius segments

```
radius n from prop=X following {e1,...,em}
```

the idea is that we start by identifying nodes having the property `prop` (whose value is bound to a variable `X`) and follow up to `n` edges (in either direction) whose labels are among `e1,...,em`.

Each specification produces (conceptually) a set of subgraphs and a binding of the variables in the specification. This results in a new segment node linked to (the nodes of the) subgraph, with a segment name property set to the declared name, and any additional properties set using the values of the variables. The `and` operation (conceptually) combines the results of two specifications as follows: for each pair (SG_1, env_1) in the result of $spec_1$ and (SG_2, env_2) in the result of $spec_2$, if SG_1 overlaps with SG_2 and env_1 and env_2 are compatible environments, produce $(SG_1 \cap SG_2, env_1 \uplus env_2)$ where $env_1 \uplus env_2$ is the merge of the two environments. (It probably makes sense to restrict attention to rules that do not reuse variables, so that combining environments is always possible).

A.2 Formal semantics

Fix sets Γ of property names, Σ of edge labels, and D of data values. We assume that there is a special attribute $time$ associating certain nodes with times.

We extend the graph model to allow for properties and their values. Let $G = (V, E, \alpha, \lambda)$ be a graph where $\alpha : V \times \Gamma \to D_\perp$ associates each property name with an optional data value, and $\lambda : V \cup E \to \Sigma$ associates a label to each vertex and edge.

A graph $G' = (V', E', \alpha', \lambda')$ is a subgraph of G if (as usual) $V' \subseteq V$, $E' \subseteq E$ and the labels in G' are compatible with those in G, that is, for all $v \in V'$ and all $p \in \Gamma$ we have $\alpha'(v, p) = \alpha(v, p)$, and similarly for all $e \in E'$ we have $\lambda'(e) = \lambda(e)$. Since α' and λ' are obtainable by restriction, we can represent a subgraph just as a set of nodes and edges. A *valuation* is a partial mapping $\rho : Var \to D_\perp$ from variables to data values. The result of segmentation of a graph G is a set of pairs (SG, ρ) where SG is a subgraph of G and ρ is a valuation.

In what follows, we fix an ambient graph G by which all subsequent definitions are parameterized and we discuss how to segment a graph $G' = (V', E')$ which may be G itself (initially) or a subgraph of G.

To segment according to rule

```
radius N from p = X following S
```

suppose N is the radius, p is the property name and S is the set of possible edge labels to follow. Suppose $v \in V'$. Then we define $Radius(G', v, N, S)$ to be the subgraph of G' obtained by adding v and all nodes reachable from v by paths of length $\leq N$ following edges whose labels are in S. Then the result of segmentation is

$$\{(Radius(G', v, N, S), [X = d] \mid \alpha(v, p) = d\}$$

that is we generate one segment for each node in G' having $p = d$ for some d, and we record this choice in the valuation.

To segment according to the rule

```
time window T_s from T_s starting X
```

suppose t_s is the start time, and t_w is the window length. Consider the periodic intervals starting at t_w with period t_d as follows:

$$[t_s, t_s + t_d), [t_s + t_d, t_s + 2t_d), \ldots, [t_s + nt_d, t_s + (n+1)t_d), \ldots$$

In a finite graph G' there is a finite subset of these intervals that contains all $time$ property values present in G'. Let $Starts(G')$ be the set of starting times of such intervals, that is,

$$Starts = \{t_s + nt_d \mid \exists v \in V'.t_s + nt_d \leq \alpha(v, time) < t_s + (n+1)t_d\}$$

We define $Time(G', I)$ as the subgraph of G' obtained by retaining all nodes with a time property in interval I, and all edges between such nodes.

$$\{(Time(G', [t, t + t_d)), [X = t]) \mid t \in Starts(G')\}$$

To segment the conjunction of two rules $spec_1$ `and` $spec_2$, let Y_1 be the result of segmenting G' according to $spec_1$ and Y_2 the result of segmenting according to $spec_2$. The result of segmenting according to `spec1 and spec2` is:

$$\{(SG_1 \cap SG_2, \rho_1 \uplus \rho_2) \mid (SG_1, \rho_1) \in Y_1, (SG_2, \rho_2) \in Y_2\}$$

where we consider only the compatible pairs ρ_1, ρ_2 that agree on any common variable values.

To segment the sequential composition of two rules $spec_1$ `then` $spec_2$, let Y_1 be the result of segmenting G' according to $spec_1$. For any subgraph SG of G', let $Y_2(SG)$ be the result of segmenting SG

(a)

```
segment byPidTime(pid=X, startTime=T)
     by radius 3 from PID=X
         following {"wasDerivedFrom", "used", "wasGeneratedBy",
                    "wasAssociatedWith", "wasInvalidatedBy"}
    and time window 24:00:00 from 2013-03-16T00:00:00 starting T
```

(b)

```
{"segmentation_specification" :
 {"segment" :
  {"name": "byPidTime",
   "args": [{"property" : "pid", "value" : {"var" : "X"}},
            {"property" : "startTime", "value" : {"var" : "T"}}],
   "specifications" : [
       {"radius" : { "r" : 3,
                     "from" : {"property" : "PID", "var": "X"},
                     "edges" : ["wasDerivedFrom",
                                "used",
                                "wasGeneratedBy",
                                "wasAssociatedWith",
                                "wasInvalidatedBy"]}},
       {"time" : {"window" : {"days" : 0,
                              "hours" : 24,
                              "minutes" : 0,
                              "seconds" : 0},
                  "from" : "2013-03-16T00:00:00",
                  "starting" : {"var" : "T"}}}
   ]
  }
 }
}
```

Figure 3. An example segmentation specification and its JSON representation

according to $spec_2$. Then the result is:

$$\{(SG_2, \rho_1 \uplus \rho_2) \mid (SG_1, \rho_1) \in Y_1, (SG_2, \rho_2) \in Y_2(SG_1)\}$$

where we again consider only the compatible pairs ρ_1, ρ_2 that agree on any common variable values.

The difference between and and $then$ is that and considers each specification independently on the current graph, and takes the intersection of the resulting subgraphs, whereas $then$ segments using the first specification, then "focuses" on each resulting subgraph and segments each one using the second specification. The variable bindings are combined using $\uplus$ in either case.

In our current prototype, each segment in the final result is recorded in the graph database as a new segment node, whose name property is set to the declared segment name, and other properties in the segment header are set to the values obtained from the corresponding variable bindings. The segment node is also linked to all of the nodes in the segment.

Notice that we include empty segments in the result, as it may be interesting to record the fact that a segment with given metadata was considered and is indeed empty (e.g. process 42 had no activity between midnight and 1am on Friday.) It may, of course, be worthwhile to consider compression techniques to avoid storing large numbers of empty segments explicitly.

A.3 Edges among segments

We can consider several kinds of relationships among segments, defined in terms of the raw graph, that can be recorded as edges in the segment layer. We say that a segment *overlaps* another segment if they have vertices in common, otherwise the segments are *disjoint*. Two segments are *contiguous* if the shortest path between them is of length 1. A segment S_2 *depends on* S_1 if there is a directed path from S_1 to S_2 in G. Two segments are *concurrent* if each depends on the other. Two segments are *independent* if neither depends on the other. If S_2 depends on S_1 but S_1 does not depend on S_2 then we say that S_1 *happens before* S_2.

Currently we do not define explicit edges among segments during segmentation. However, since the segments are linked to the nodes of their subgraphs, such edges can be defined implicitly as queries. For example, we could define an edge relationship between segment nodes to hold if there is a (directed) path from a node in one segment to a node in another. It is not yet clear whether we want to (or need to) persist such edges in the database or simply generate them on the fly when the segment layer is analyzed by later stages, such as the diagnostic engine [1].