PROV-TC Specification (DRAFT)

Purpose of this Document

This document specifies the PROV-TC conceptual model, a specialization of the W3CPROV provenance model to the domain of computation and its attribution. PROV-TC is intended as the communication standard for data conveyed by TA-1 performers to TA-2 performers in the Transparent Computing program.

Throughout this specification, we use the PROV-TC language to define and illustrate the conceptual model. The PROV-TC language is a dialect of the W3CPROV language (specifically the PROV-N compliant representation of that language). The concrete syntax of PROV-TC used in this specification, and in sample data, is meant to be human-readable and usable by the Transparent Computing program TA1 and TA2 teams for communication. Concise (and thus likely not human-readable) concrete representations are also realizable for the PROV-TC language, and will likely be used for live testing and prototypes. When using the PROV-TC language, if there is disagreement between this specification and the syntax checking tool provided by the ADAPT team, please report the mismatch so we can resolve it. Thanks!

Goals of PROV-TC

PROV-TC aims to provide a conceptual model for causality in general computation and a syntax for expressing instances of that model. The model must be able to represent sufficient causality that an well-formed instance of the model can be communicated to an analysis platform separate from the modeled computation instance, such that tools on that platform can reconstruct and analyze full causality without additional information.

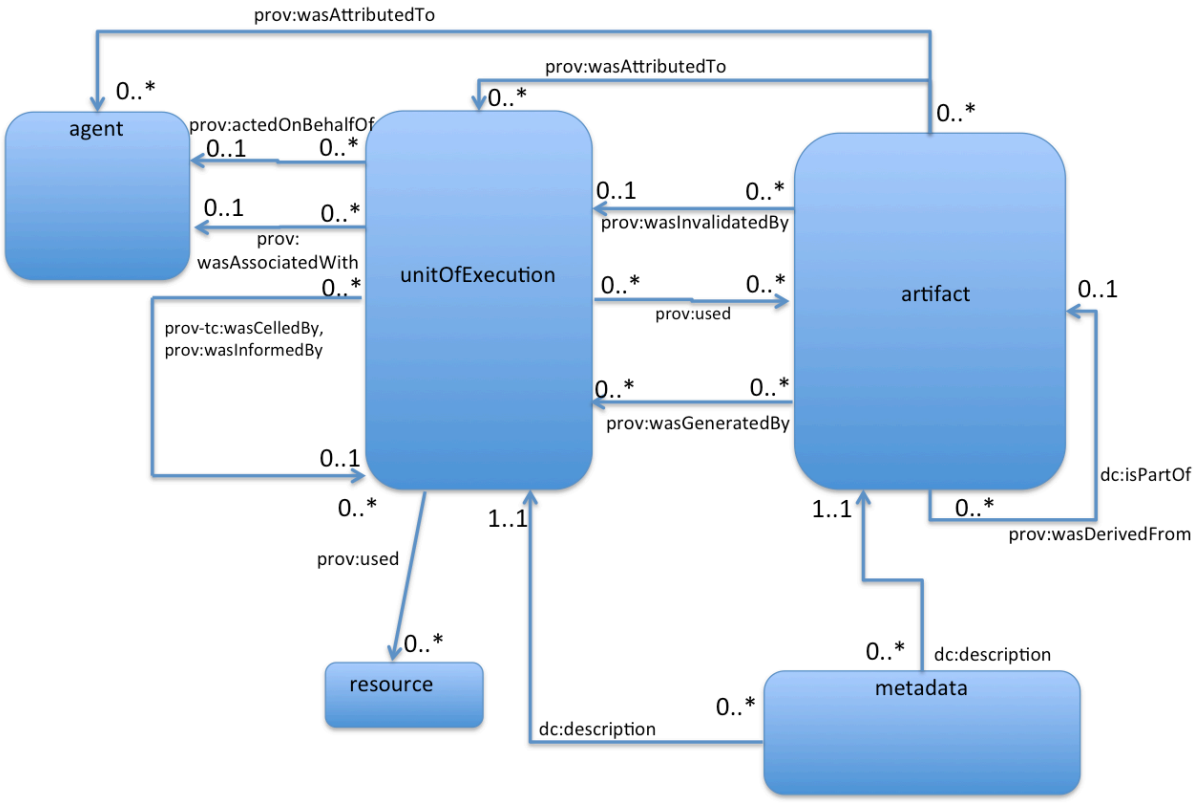
PROV-TC aims to support multiple granularities of computational causality. Entities in a model instance must be able to represent individual instructions or entire processes, individual bytes in a network packet or entire files.

PROV-TC aims to represent both why-provenance (a representation of the sources that contribute to an computing artifact, and the information flow from sources to artifact) and how-provenance (a representation of how those data came to influence the artifact, and the control flow that accomplished that influence). PROV-TC thus aims to represent the semantics representable in the taint-tracking literature of the computer security field.

PROV-TC also aims to represent security-relevant information about computing artifacts referenced in a model instance, particularly their trustworthiness (as for example might be attested by a digital signature or checksum) and their sensitivity (as for example might be decided by a system administrator). In a similar way, PROV-TC aims to represent trustworthiness of actors referenced in a model instance (as for example might be decided by a user's privilege level or the credentials used to log into the modeled system).

Conceptual Model and Properties

The PROV-TC conceptual model is shown below.



Terminology for describing PROV-TC generally follows the language of E-R models. There are for example five entity classes and twelve relationship classes in the current model. Each instance of an entity or relationship class may have certain required as well as other optional attributes that describe that instance. Details of entities, relationships, and attributes in the model are defined in the Entities and Relationships sections below.

Properties of PROV-TC

It is sometimes convenient to describe a collection of instances in PROV-TC. For example, a valid *provenance linkage* in PROV-TC consists of a relationship instance and the two entity instances that it connects, where each instance in the linkage is characterized by valuations of at least all required attribute terms. Any entity instance in PROV-TC may participate in zero or more provenance linkages. Any relationship instance in our model must participate in exactly one provenance linkage. The *dc:description* relationship class is the only class that does not participate in provenance linkages.

We define a class of provenance linkages, called *direct flow linkages*, that indicate direct flow of information between the entity instances in the linkage. All provenance linkages that contain a relationship in the following classes are direct flow linkages: wasInvalidatedBy, used, wasGeneratedBy, wasDerivedFrom, wasStartedBy, wasEndedBy, wasInformedBy. Compositions of direct flow linkages are particularly useful for expressing "how" provenance, when coupled with attributes that describe how artifacts combine to produce new artifacts.

PROV-TC instances describe provenance relationships found in instances of execution of a software system. PROV-TC is designed such that any valid PROV-TC instance maintains certain properties. Specifically, for any execution of a software system, it is possible to

- create a PROV-TC model such that all provenance linkages that are true are represented in the model (the causal completeness property)
- create a complete PROV-TC model such that only true provenance linkages are represented in the model (the causal soundness property)

PROV-TC model instances may include hierarchies of entity instances, using for example the isPartOf relationship among artifacts, and wasStartedBy or wasInformedBy relationship among units of execution. In a PROV-TC model instance containing only one hierarchical level representing any element, it is possible to

- create a sound PROV-TC model instance such that removing any provenance linkage will make the instance causally incomplete (the causal minimality property)

In addition, there exists a simple syntactic grammar whose well-formed sentences are exactly an expression of PROV-TC model concepts, and for every sound PROV-TC model it is possible to express that model unambiguously in that grammar (the well-formed property).

PROV-TC representations can be composed from other such representation, for example by adding traces from one sensor source to those from another. Because we expect this kind of aggregation to happen often, it is possible to

- create a PROV-TC model by taking the union of other PROV-TC models (closure with respect to union)

PROV-TC representations may be queried using tools such as relational algebra, in order to select certain subsets of models. Thus it is possible to

- create a PROV-TC model by taking the intersection of other PROV-TC models (closure with respect to intersection).

PROV-TC also has the property of non-causal extensibility, where new relationships that do not convey causality can be added. For example, the ADAPT team aims to extend PROV-TC for example by adding structure that conveys hierarchy, allowing representation of subgraphs by individual, higher-level vertices.

Likely future extensions to PROV-TC

We are currently collecting ideas for near-term extensions to the basic model (read: we're trying to get around to adding these as soon as possible!). So far, our pending list includes:

- the addition of a *program point* attribute to certain relationships that allows for specifying the location in a unit of execution where an artifact was used, generated, or invalidated, or another unit of execution was started, stopped, or informed.
- the addition of set constructs to the subject and object of wasDerivedFrom relationships, so that sets of artifacts may be shown to have the same derivation, or so that an artifact may be shown to have been derived from several other artifacts, or both
- the addition of a set construct to the isPartOf relationship, so that sets of artifacts may be shown to be part of the same artifact

The PROV-TC Type System

PROV-TC defines types for all entity instances, relationship instances, and attribute instances. Types of relationship class instances are explicit in the conceptual model diagram, for example:

```
wasGeneratedBy: artifact > unitOfExecution > provenancePredicate
```

Types of entity class instances are the same as the class from which they are drawn.

Types of attributes are intended to be unambiguous in the definitions below. Please let us know if you find missing type declarations or inconsistencies.

PROV-TC Entities

PROV-TC defines the following entity classes. In each case, we specify all currently recognized attributes. No alternates to the attributes shown are allowed. That is, the shown attributes are the only option available to represent the semantics they represent. We make this choice to prevent multiple approaches to expressing the same provenance semantics, which could make analytics more complex. If you see an opportunity to reduce the model in this way, please let us know.

Artifact (also Object)

An artifact is an entity that may be created, referenced, used, or destroyed, but does not take action on its own. Artifact sub-types recognized at present in the TC domain include files, network packets, memory locations, and registry file entries. More may be added later.

An artifact instance is reified as an entity in the PROV model, and includes several attributes. Shown below is an example artifact: a file with path */users/dwa/mystuff/bin/create.exe*, and referred to with the tag *ex:createExe*.

Required attributes include:

- the type of artifact (*prov-tc:entityType*). Enum type with valuations:
    - "file"
    - "network"
    - "memory"
    - "registryEntry"
- a unique identifier for the entity (the type of which depends on the artifact type). Dependent enum type with valuations:
    - *prov-tc:path* for entityType "file"
    - *prov-tc:destinationAddress* for entityType "network"
    - *prov-tc:pageNumber* for entityType "memory"
    - *prov-tc:registryKey* for entityType "registryEntry"
- a location within that item (which also depends on the artifact type). Dependent enum type with valuations:
    - *prov-tc:fileOffset* if the entity type is "file"
    - *prov-tc:packetID* if the entity type is "network"
    - *prov-tc:address* if the entity type is "memory"
- a creation time (*prov-tc:time*). String type with format matching 015-10-16T02:13:07Z
- an owning account (*prov-tc:uid*). String type
- an owning group (*prov-tc:group*). String type
- a "how-derived" attribute (*prov-tc:how-provenance*). Type TBD *TODO: Venkat

Optional attrigutes include

- a version (*prov-tc:hasVersion*). Natural number
- a size (*prov-tc:size*). Natural number
- an attribute that names the source of the information provided (*prov-tc:source*). String
- access permissions (*prov-tc:permissions*). String (for now)
- an indication of trustworthiness of the artifact, that is, a confidence assessment that the artifact's integrity is attested by a trusted authority (*prov-tc:trustworthiness*). Real in range [0..1]
- an indication of the privacy sensitivity of the artifact, that is, a confidence assessment of the extent to which leakage of the artifact to non-owners is a security failure (*prov-tc:privacyLevel*). Real in range [0..1]
- an indication of the integrity sensitivity of the artifact, that is, a confidence assessment of the extent to which modification of the artifact by non-owners is a security failure (*prov-tc:integrityLevel*). Real in range [0..1]

```
entity(ex:createExe, [
            prov-tc:source="/dev/audit",
            prov-tc:entityType="file",
            prov-tc:path="/users/dwa/mystuff/bin/create.exe",
            prov-tc:fileOffset="0x00000000",
            prov-tc:time="015-10-16T02:13:07Z",
            prov-tc:hasVersion="23",
            prov-tc:uid="dwa",
            prov-tc:size="4096",
            prov-tc:how-provenance="<TBD>"])
```

Resource

A resource is a thing that may be used, but not created or destroyed. Typical resources include GPS units, cameras, keyboards, and accelerometers. More may be added later.

Required attributes of resources include:

- an attribute that specifies the resource type (*prov-tc:devType*). Enum with valuations:
    - "GPS"
    - "keyboard"
    - "accelerometer"
    - "camera"
    - "network interface"

Optional attributes of resources include:

- an attribute that names the resource (*prov-tc:devID*). String
- an attribute that names the source of the information provided (*prov-tc:source*). String

```
// a GPS sensor resource called ex:GPSunit
entity(ex:GPSunit, [
            prov-tc:source="/dev/audit",
            prov-tc:devType="GPS",
            prov-tc:devID="Default GPS sensor"])
```

Unit of Execution (UoE) (also Subject)

A UoE is a thread of running computation. It may be created or destroyed, and may take action on its own to create or destroy other UoEs or to affect artifacts. A UoE is reified in PROV as an *activity*. While this is a slight abuse of W3CPROV, this abuse results in an unambiguous and more concise representation than W3CPROV allows. Below we show an example UoE called "parent".

Required attributes include:

- a grain size
    - @TODO: list allowed values. Dave
- a host identifier (*prov-tc:machineID*). String.
- an owner (*foaf:accountName*) . String.
- a group (*prov-tc:group*). String.
- a process ID (*prov-tc:pid*). Natural number.
- a process ID of its parent (*prov-tc:ppid*). Natural number.
- the program name executing (*prov-tc:programName*). String.

Optional attributes include:

- a start time (*prov:startedAtTime*). String type with format matching 015-10-16T02:13:07Z
- an end time (*prov:endedAtTime*). String type with format matching 015-10-16T02:13:07Z
- a privilege level (*prov-tc:privs*). String, format TBD @TODO Dave
- a list of environment variables (*prov-tc:env*). List of key-value pairs, each item a String
- the working filesystem directory (*prov-tc:cwd*). String
- the command line currently executing (*prov-tc:commandLine*). String
- an attribute that names the source of the information provided (*prov-tc:source*). String

```
//
activity(ex:parentb, -, -, [
            prov-tc:source="/dev/audit",
            prov-tc:machineID="0000000100000001",
            prov:startedAtTime="015-10-16T02:13:07Z",
            prov:endedAtTime="015-10-16T02:13:07Z",
            prov-tc:privs="mode=u",
            foaf:accountName="dwa",
            prov-tc:group="Group1",
            prov-tc:pid="12",
            prov-tc:ppid="1",
            prov-tc:cwd="/users/dwa",
            prov-tc:commandLine="xterm",
            prov-tc:programName="xterm"])
//     the connection between them
```

Agent (also Principal)

An agent represents an actor that is not a UoE on a monitored machine. An agent may be human, may be a machine in the target network that has no monitoring, or may be a machine outside the monitored network. Agents have no required attributes.

Required attributes:

- a host identifier - either a domain name or IP address (*prov-tc:machineID*). Qualifying String
- a user name (*foaf:accountName*). String
- a user ID (*prov-tc:uid*). String

- a user group (*prov-tc:group*). String

Optional attributes:

- an authentication indicator such as an ssh key, hashed password, or PKI cert (*prov-tc:authenticator*). Binary object
- an attribute that names the source of the information provided (*prov-tc:source*). String

```
// a remote agent named ex:exernalAgent
agent(ex:externalAgent, [
        prov-tc:source="/dev/audit",
        prov-tc:machineID="0000000100000002"])
```

Metadatum

A metadatum is a thing that describes a UoE or an artifact.

Required attributes: * a triple of form (name, type, value) (*prov-tc:metadata*). Triple of Strings

Optional attributes: * an attribute that names the source of the information provided (*prov-tc:source*). String

```
// a metadatum named ex:mdata1
entity(ex:mdata1, [
    prov-tc:source="/dev/audit",
    prov-tc:metadata="name, type, value"]) //required
```

Event Relationships Among Entities

Event (E) relationships represent actions performed by subjects that affect objects, other subjects, or resources. The E relationships we support from subjects (UoEs) to objects (artifacts) are wasGeneratedBy, wasInvalidatedBy, and used. Supported E relationships between subjects are wasStartedBy, wasEndedBy, and wasInformedBy. The sole E relationships from UoEs to resources is usedResource. These general relationships conform to W3CPROV, and provide an intuitive notion of the kind of event that occurred. Some of these relationships are further typed to give more detail about the event.

wasGeneratedBy

A wasGeneratedBy relationship indicates that an artifact (the object of the relationship) such as *ex:createExe* in the example below was created by a UoE (the subject of the relationship) such as *ex:parentb* executing the operation *prov-tc:operation*="*write*".

Required attributes include:

- the operation performed on the artifact (*prov-tc:operation*). Enum with valuations:
    - 'write'
    - 'send'
    - 'connect'
    - 'truncate'
    - 'chmod'
    - 'touch'
    - 'create'

Optional attributes include:

- a tag that specifies, in more detail than the granularity of the UoE, the *program point* that initiated this event. @TODO type for this
- a list of arguments used in the operation initiated by that program point operation (*prov-tc:args*). List of Strings
- a return value from the operation (*prov-tc:returnVal*). String
- the time at which the generation event occurred (*prov-tc:time*). See time definition above
- the permissions with which the entity was created (*prov-tc:permissions*). See perms definition above
- an attribute that names the source of the information provided (*prov-tc:source*). String

```
// newprog.exe was created by the parent process (activity ex:parentb)
wasGeneratedBy(ex:createExe, ex:parentb, - , [
                prov-tc:source="/dev/audit",
                prov-tc:operation='write',
                prov-tc:permissions="0o775",
                prov-tc:time="015-10-16T02:13:07Z"])
```

wasInvalidatedBy

A wasInvalidatedBy relationship indicates that an artifact (the object of the relationship) such as *ex:createExe* in the example below was deleted by a UoE (the subject of the relationship) such as *ex:parentb*.

Required attributes:

- deletion time (*prov-tc:time*)
- the operation performed (*prov-tc:operation*). Enum with valuations:
    - 'delete'
    - 'unlink'

```
wasInvalidatedBy(ex:createExe, ex:parentb, 015-10-16T02:13:07Z, [
            prov-tc:source="/dev/audit",
            prov-tc:operation='delete',
            prov-tc:time="015-10-16T02:13:07Z"])
```

used

A used relationship indicates that an UoE such as *ex:parentb* in the example below (the subject of the relationship) gained information from, or modified, but did not due to this relationship create nor delete, a pre-existing artifact such as *ex:createExe* (the object of the relationship).

Required attributes:

- the operation performed (*prov-tc:operation*). Enum with valuations:
    - 'open'
    - 'bind'
    - 'connect'
    - 'accept'
    - 'read'
    - 'mmap'
    - 'mprotect'
    - 'close'
    - 'link'
    - 'modAttributes'
    - 'execute'

Optional attributes:

- a tag that specifies, in more detail than the granularity of the UoE, the *program point* that initiated this event
- a list of arguments used in the operation initiated by that program point operation (*prov-tc:args*). List of Strings
- a return value from the operation (*prov-tc:returnVal*). String
- the time at which the generation event occurred (*prov-tc:time*)
- an attribute that names the source of the information provided (*prov-tc:source*). String
- an offset in the artifact at which the use event began (*prov-tc:entryAddress*). Unsigned64

```
used(ex:parentb, ex:createExe, 015-10-16T02:13:07Z,[
            prov-tc:source="/dev/audit",
            //alternately, "/proc"
            prov-tc:entryAddress="0x8048170",
            prov-tc:args="[]",
            prov-tc:returnValue="0",
            prov-tc:operation="execute",
            prov-tc:time="015-10-16T02:13:07Z"]
                        )
```

usedResource

A resource such as *ex:GPSunit* in the example below may be used by a UoE (subject) such as *ex:childB*. Required attributes:

- time of the use (*prov-tc:time*)

Optional attributes:

- command string sent to the device (*prov-tc:operation*)
- value returned from the device (*prov-tc:returnValue*)
- an attribute that names the source of the information provided (*prov-tc:source*)

```
used(ex:childB, ex:GPSunit, 015-10-16T02:13:07Z, [
            prov-tc:source="/dev/audit",
            prov-tc:operation="read",
            prov-tc:returnValue="45.52119128833272,-122.67789063043892"
            ])
```

Event Relationships Among UoEs

A UoE can start, end, or inform another UoE. These relationships represent control flow in the monitored system.

wasInformedBy

A UoE can be influenced by another UoE. That is, one may start, end, communicate with, or modify behavior of another. Required attributes:

- the time of influence (*prov-tc:time*)
- the operation that caused the influence (*prov-tc:operation*). Enum with valuations:
    - "fork"
    - "clone"

    - "execve"
    - "signal" - may need to be further refined
    - "setuid"
    - "kill"
    - "follows"

Optional attributes:

- an attribute that names the source of the information provided (*prov-tc:source*)

```
wasInformedBy(ex:childB, -, ex:parentb, 015-10-16T02:13:07Z, [
            prov-tc:source="/dev/audit",
            adapt:operation="signal"])
```

Note: the operations listed above supercede the need for use of the W3CPROV relationships *wasStartedBy* and *wasEndedBy*, which have been removed from this model.

wasCalledBy

A UoE can be a composition of other UoEs. For example, a process may execute functions. We use isCalledBy to represent the hierarchical structure of a UoE as needed. Note that order of execution is important, because state modifications made by one UoE can influence others that follow. However, timestamps may not be precise enough to indicate sequential flow. We use *wasInformedBy* with *tc:operation="follows"* to indicate sequential flow in threads of execution. Because each execution of a code element in an iterative structure such as a loop is represented by a distinct UoE instance, this relationship structure is capable of representing iterations. *isCalledBy* has no required attributes.

Optional attributes:

- an attribute that names the source of the information provided (*prov-tc:source*)

```
wasCalledBy(ex:callee, -, ex:caller, -, [prov-tc:source="/dev/audit"])
```

Attribution Relationships Among Entities

Attribution (A) relationships among entity instances represent accountability. Optional attributes are the same for all of these relationships:

- an attribute that names the source of the information provided (*prov-tc:source*)

An artifact such as *ex:createExe* can be attributed to a UoE or an agent such as *ex:parenta*.

```
wasAttributedTo(ex:createExe, ex:parenta, [prov-tc:source="/dev/audit"])
```

A UoE such as *ex:parenta* can act on behalf of an agent such as *ex:externalAgent*. Note that the activity portion of the UoE is shown in the third argument position:

```
// the parent process acted on behalf of this remote agent
actedOnBehalfOf(ex:parenta, ex:externalAgent, ex:parentb, [prov-
tc:source="/dev/audit"])
```

A UoE can be associated with an agent. This relationship is somewhat looser and less well defined than the above case.

```
// the parent process was also associated with this remote agent
wasAssociatedWith(ex:parentb, ex:externalAgent, -, [prov-tc:source="/dev/audit"])
```

An artifact such as *ex:newprogExe* can be derived from another artifact such as *ex:newprogSrc* using an *adapt:deriveOp* operation that is one of rename, link, or compile.

Derivation and Composition Relationships Among Artifacts

Derivation-Composition (DC) relationships among artifacts represent data flow or data composition.

wasDerivedFrom

This relationship indicates that the derived artifact has as part of its provenance the original artifact. Note that a derived artifact may have several artifacts in its provenance, including artifacts that indirectly influence it, such as executable images used by UoEs in creating or modifying an artifact. The *how-provenance* attribute of the derived artifact describes how these source artifacts combined to yield the derived artifact. Required attributes:

- the operation used on the parent to result in the child (*prov-tc:operation*). Enum with valuations:
    - "compile"
    - "project" in the sense of the relational algebra operator of the same name
    - "computation input"
    - "rename"
    - "link"
    - "execute"
- the time of the derivation (*prov-tc:time*)

Optional attributes:

- an attribute that names the source of the information provided (*prov-tc:source*)

```
// showing the derivation of newprog.exe from newprog.c
wasDerivedFrom(ex:newprogExe, ex:newprogSrc, [
            prov-tc:source="/dev/audit",
            prov-tc:operation="compile",
            prov-tc:time="015-10-16T02:13:07Z"])
```

isPartOf

An artifact can be part of another artifact. We use dc:isPartOf for this construction. This relationship has no attributes.

```
dc:isPartOf(ex:childThing, ex:parentThing)
```

Representing the "How" of Provenance in PROV-TC

PROV-TC allows for representing provenance in three distinct ways:

- exclusively in the "data plane" via DC relationships and prov-tc:how-provenance attributes
- exclusively in the "control plane" via the E-relationship types that connect UoEs to other UoEs
- by connections between the "control plane" and "data plane" via the E-relationship types that connect UoEs to artifacts and resources

Each way supports one or more distinct analytic approaches that may be used to TA-2 teams. We encourage TA-1 teams to provide as much information as possible in each of these ways, and to make the reported information consistent between them where there is possibility of redundancy.

@TODO: One aspect of PROV-TC that is not yet specified is the representation of the prov-tc:how-provenance attribute. We're working on this. One possible representation could be an adaptation of Green's provenance polynomials.

The Mechanics of PROV-TC Syntax

PROV-TC Prelude and Postlude

Every valid PROV-TC document has a prelude consisting of at least the following

```
document
// declare the ontologies used in the code

// namespace for instance names of entities and relations
prefix ex <http://example.org/>

// namespace for our specific attributes
prefix prov-tc <http://spade.csl.sri.com/rdf/audit-tc.rdfs#>.
```

Every valid PROV-TC document concludes with

```
end document
```

Quick checking for basic syntax correctness

Because we specify a dialect of PROV-N, an easy on-line tool can be used as a basic check of syntactic correctness prior to using our ingester tools to check more deeply. You can find this tool here.

A more complete check, including checking of attributes specific to PROV-TC and full type checking of PROV-TC instances, can be had by contacting the ADAPT team.

Working Examples

The ADAPT team has demonstrated the following so far in support of developing PROV-TC:

- a translator from the 5-Directions sample data syntax to PROV-TC
- successful translation of all 46 5-Directions data samples into PROV-TC (these are available, just ask)
- an XML file specifying PROV-TC specific attributes to PROV-N constructs that is used to generate SRI SPADE data compliant to PROV-TC
- successful generation of several SPADE samples in PROV-TC
- an ingestor that parses, syntax checks, and type checks PROV-TC, and creates a prototype graph database from the ingested data
- successful ingest of all 5-Directions and SPADE samples generated so far in PROV-TC
- the examples used in this literate specification of PROV-TC

Contact List and Contributors

Please contact Dave Archer, dwa@galois.com, with questions regarding this specification.

Many thanks to those who have contributed to the PROV-TC model to date. It's likely that there are

"unsung heroes" who are not credited here due to oversight by the journaling author. Please feel free to abuse him for such oversights. In alphabetical order:

- Armando Caro - acaro@bbn.com
- James Cheney - jcheney@inf.ed.ac.uk
- Mukesh Dalal - mukesh.dalal@baesystems.com
- Tom DuBuisson - tommd@galois.com
- Ashish Gehani - gehani@csl.sri.com
- Joud Khoury - jkhoury@bbn.com
- Rui Maranhao - Rui.Maranhao@parc.com
- Alex Perez - Alexandre.Perez@parc.com
- Jeff Perkins - jhp@csail.mit.edu
- Martin Rinard - rinard@csail.mit.edu
- Venkat Venkatakrishnan - venkat@uic.edu