Purpose

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 our model. The PROV-TC language is a dialect of the W3CPROV language (specifically the PROV-N compliant syntax 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 ADAPT project team (and hopefully other 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 demonstrations.

Contact 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

Working Prototypes

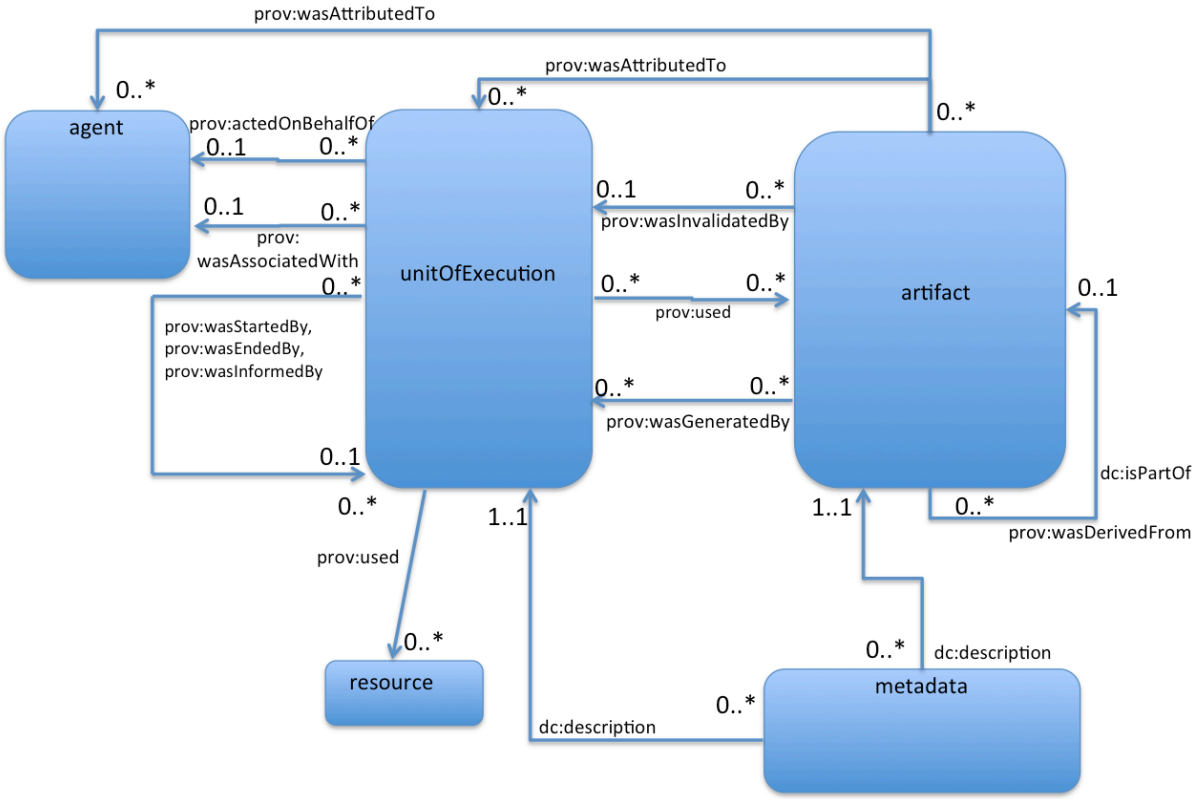
The ADAPT team has demonstrated the following so far in support of developing PROV-TC:

- 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

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.

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 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 extensions to PROV-TC

We are currently collecting ideas for extensions to the basic model. So far, our pending list includes:

- the addition of a *program point* attribute to certain relationships or entities that allows for specifying the location in a program where an entity was used, generated, or invalidated
- 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 addition of attributes that convey lables for integrity and privacy of artifacts

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. We'll clarify as needed.

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.

PROV-TC Entities

PROV-TC defines the following entity classes. In each case, we specify all currently recognized attributes. Those attributes that are required for syntactic correctness are marked as required in the code examples. All others are optional. No alternates to the attributes shown are allowed. That is, the shown attributes are the only option available to represent the semantics they represent. Other semantics may be included, but will be ignored for now.

Artifact

An artifact is an entity that may be created, referenced, used, or destroyed, but does not take action on its own. Artifacts 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, many of which are optional. 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 are marked with comments and include the type of entity (*prov:type*), the type of artifact (*adapt:entityType*), an identifier (the type of which depends on the artifact type), a location within that item (which also depends on the artifact type), a creation time, a version, a destruction time, an owning account, a size, and a 32b taint. Alternative values for the different artifact types are shown in comments.

```
entity(ex:createExe, [
            prov-tc:source="/dev/audit",
            //alternately, "/proc"
            prov-tc:entityType="file", //required: one of the 4 options here
            // alternately, "network"
            // alternately, "memory"
            // alternately, "registryEntry"
            // currently looking at alternatives to add here for MIT
            prov-tc:path="/users/dwa/mystuff/bin/create.exe",
            // alternately, destinationAddress="128.10.125.71"
            // alternately, destinationPort="88"
            // alternately, sourceAddress="127.0.0.1"
            // alternately, sourcePort="1234"
            // alternately, pageID="0x123"
            // alternately, registryKey="stuff"
            prov-tc:fileOffset="0x00000000",
            // alternately, packetID="0x1234"
            // alternately, address="0x00000000"
            // alternately, registryValue="some value"
            prov-tc:time="015-10-16T02:13:07Z",
            prov-tc:hasVersion="23",
            prov-tc:uid="dwa",
            prov-tc:size="4096",
            prov-tc:taint="0x00000001"])
```

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. The *adapt:devType* attribute that specifies the resource type is required. Optional is an attribute *adapt:devID* that names a specific resource.

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

Unit of Execution (UoE)

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". Recognized but optional attributes are an ID string unique to the machine where the UoE runs, the times at which the UoE started and ended, the privileges with which the UoE ran, the account name and group name under which it ran, its process ID and parent process ID, the directory where it started, the command line used, and the command used.

```
//
activity(ex:parentb, -, -, [
            prov-tc:source="/dev/audit",
            //alternately, "/proc"
            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

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. Available attributes include an identifier for a machine, a name, and an account name.

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

Metadatum

A metadatum is a thing that describes a UoE or an artifact. A metadatum is an entity that has an identifier and contains a triple of form (name, type, value).

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

Relationships Among PROV-TC Entities

An artifact such as *ex:createExe* can be created by a UoE such as *ex:parentb* executing an operation *prov-tc:operation* that is one of 'write', 'send', 'connect', 'truncate', 'chmod', or 'touch'. Other attributes include the time at which the generation event occurred, and the permissions with which the entity was created (if applicable).

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

An artifact such as *ex:createExe* can be deleted by a UoE such as *ex:parentb*. The sole attribute, required, is the deletion time.

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

An artifact such as *ex:createExe* can be used by a UoE such as *ex:parentb* executing an operation *prov-tc:operation* that is one of 'read', 'recv', 'accept', or 'execute'. Required attributes include the time of the use action, and the use operation. Optional attributes include an entry address (for example, if useOp is 'execute'), an argument list passed to an executed function, and a return value returned by an executed function.

```
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",
            // alternately: "write", "send", "truncate", "chmod", "recv", "accept"
            prov-tc:time="015-10-16T02:13:07Z"]
                        )
```

Similarly, a resource such as *ex:GPSunit* may be used by a UoE such as *ex:childB*. Required are the start and end time of the use. Options are a command string sent to the device and a value returned from the device.

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

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",
            //alternately, "/proc"])
```

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",
            //alternately, "/proc"])
```

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",
            //alternately, "/proc"])
```

A UoE can start, end, or inform another UoE. The former requires a start time, while the middle requires an end time. The latter requires both a timestamp and the operation that must be one of fork, clone, execve, kill, or setuid.

```
wasStartedBy(ex:childB, -, ex:parentb, 015-10-16T02:13:07Z, [
            prov-tc:source="/dev/audit",
            //alternately, "/proc"])

wasEndedBy(ex:childB, -, ex:parentb, 015-10-16T02:13:07Z, [
            prov-tc:source="/dev/audit",
            //alternately, "/proc"])

wasInformedBy(ex:childB, -, ex:parentb, 015-10-16T02:13:07Z, [
            prov-tc:source="/dev/audit",
            // alternately, "/proc"
            adapt:operation="kill"
            // alternately, "fork", "clone", "execve", "setuid"
            ])
```

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.

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

An artifact can be part of another artifact. We use dc:isPartOf for this construction.

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