Purpose

This document specifies the PROV-TC language, which is a dialect of the W3CPROV language (specifically the PROV-N compliant representation of that language). PROV-TC is meant to be used by the Transparent Computing ADAPT project team (and hopefully other teams) for communication between technical areas one and two.

Contact

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

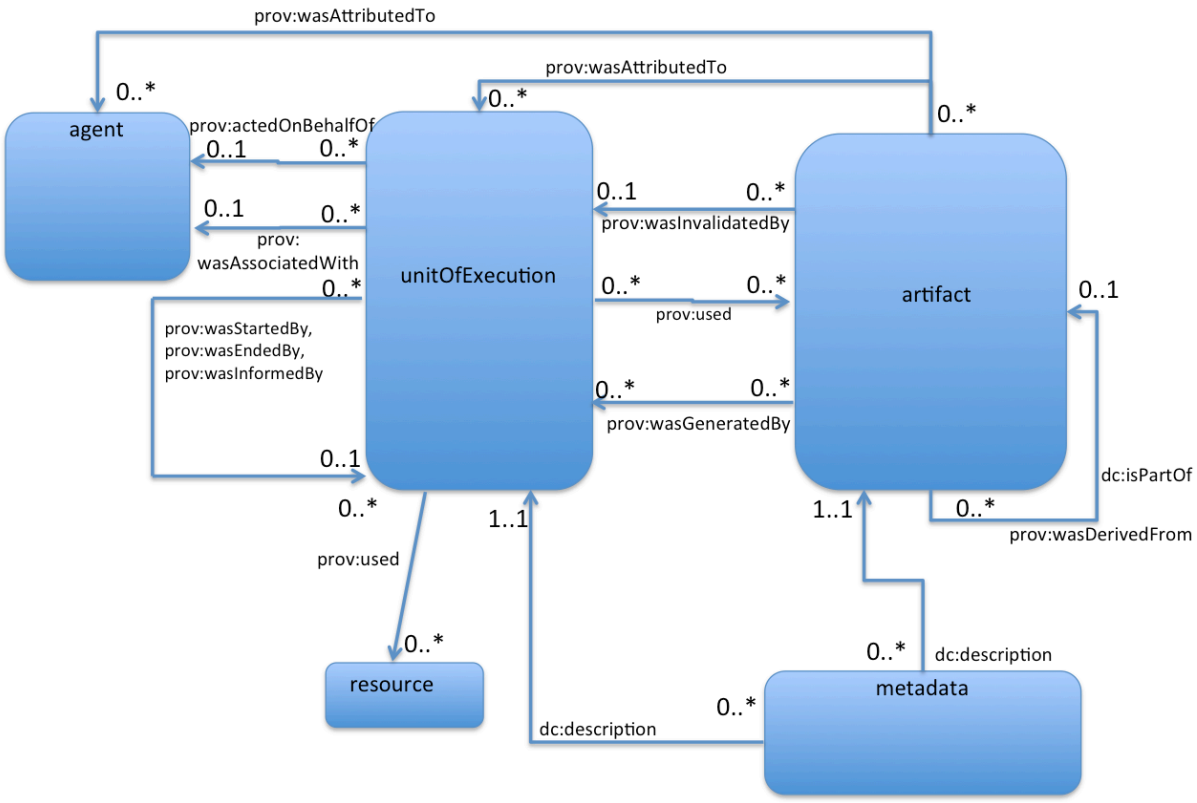
Working Prototypes

The ADAPT team has demonstrated the following so far:

- 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

PROV-TC is a domain-specific language designed to implement a conceptual model for transfer of knowledge between Transparent Computing TA1 and TA2 teams. The conceptual model has been discussed and refined for several months, and is shown below.



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://adapt.org/>
```

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, can be had by contacting the ADAPT team.

PROV-TC Entities

We allow for instances of 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 name */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:artifactType*), 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:type='tcArtifact', // optional, may be required in a later version
            prov-tc:artifactType='file', //required: one of the 4 options here
            // alternately, 'network'
            // alternately, 'memory'
            // alternately, 'registry-entry'
            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, registry-key="stuff"
            prov-tc:fileOffset="0x00000000",
            // alternately, packetID="0x1234"
            // alternately, address="0x00000000"
            // alternately, registry-value="some value"
            prov:generatedAtTime="015-10-16T02:13:07Z",
            prov-tc:hasVersion="23",
            prov:invalidatedAtTime="NA",
            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:type='tcDevice', //optional, may be required later
            prov-tc:devType="GPS",
            prov-tc:devID="Default GPS sensor")]
```

Unit of Execution (UoE)

A UoE is a thread of execution running on a processor. 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 both an *agent* and an *activity*. To show that the pair represent the same entity in our model, they are linked by a particular *relationship*. Below we show an example UoE called "parent". A UoE has only one required attribute, *prov:type*, marked in the example. Other 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.

```
//      the agent portion of a UoE, called ex:parenta. optional for now
agent(ex:parenta, [prov:type='adapt:unitOfExecution'])
//
//     the activity portion, called ex:parentb, using all defined optional
attributes
activity(ex:parentb, -, -, [
            prov:type='adapt:unitOfExecution',
            prov-tc:machineID="0000000100000001",
            prov:startedAtTime="015-10-16T02:13:07Z",
            prov:endedAtTime="015-10-16T02:13:07Z",
            prov-tc:privs="mode=u",
            prov-tc: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
wasAssociatedWith(ex:parentb, ex:parenta, 015-10-16T02:13:07Z) // optional for now
// end of parent process record
```

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: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: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 *adapt:genOp* 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:operation='write', \\required
                prov-tc:permissions="0o775"])
```

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)
```

An artifact such as *ex:createExe* can be used by a UoE such as *ex:parentb* executing an operation *adapt:useOp* 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:entryAddress="0x8048170",
            prov-tc:args="",
            prov-tc:returnValue="0",
            prov-tc:operation="execute"] //optional
                        )
```

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. Optionsl 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:devCommand="read",
            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)
```

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)
```

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, -)
```

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)

wasEndedBy(ex:childB, -, ex:parentb, 015-10-16T02:13:07Z)

wasInformedBy(ex:childB, -, ex:parentb, 015-10-16T02:13:07Z, [adapt:execOp='kill'])
```

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:operation="compile",
            prov:atTime="2015-08-31T12:00:00"])
```

An artifact can be part of another artifact. We use dc:isPartOf for this construction. No good examples have been created yet.

A metadatum describes an artifact or UoE. No good examples have been created yet.

Detailed PROV-TC Syntax Guide

This section provides a detailed grammatical structure for composition of the statement examples above. Currently under revision to match the section above, so please refer to the above examples and the related XML file as the gold standard for now.

Token Set

The valid tokens (types) in PROV-TC include:

- Raw strings (IDENT): alphanumeric strings starting with a letter.
- Integers (NUMLIT): Base 10 numbers.
- Quoted Strings (STRINGLIT): Double-quoted strings with any embedded double-quotes escaped via the backslash character.
- Time (TIME): ISO-8601 time format in Zulu (Z).
- Universal Resource Identifiers (URILIT): Commonly known as URIs, such as web addresses.

Recognized keywords

- document
- end document
- prefix
- activity
- agent
- wasAssociatedWith
- entity
- used
- wasStartedBy
- wasGeneratedBy
- wasEndedBy
- wasInformedBy
- wasAttributedTo
- wasDerivedFrom
- actedOnBehalfOf
- wasInvalidatedBy
- description
- isPartOf
- atTime
- args

- returnVal
- operation
- read recv accept execute

Recognized symbols

> *'%%' '=' ':' ',' '(' ')' '[' ']' '-' ';' '\" '/'*

Identifiers

Identifiers are any string starting with a latin character and including characters, numbers, underscores, and optionally a single colon (for qualified names). These are sometimes called scoped names.

```
ident :: { Ident }
  : IDENT ':' IDENT
  | ':' IDENT
  | IDENT
```

Document and Prefixes

Prov-TC documents start with a 'document' string, include a list of prefixes and expressions, and end with either 'endDocument' or 'end document'.

```
prov :: { Prov }
  : 'document' list(prefix) list(expr) 'endDocument'
```

A prefix is a method to define a qualified name for an otherwise unweildy URI. The URI should be valid and provide any visitor with an ontology.

```
prefix :: { Prefix }
  : 'prefix' ident URILIT
```

Expressions

Valid expressions are a sub-set of Prov-N and Dublin Core.

```
expr :: { Expr }
  : provExpr            { $1 }
  | dcExpr              { $1 }

provExpr :: { Expr }
  : entity
  | activity
  | generation
  | usage
  | start
  | end
  | invalidation
  | communication
  | agent
  | association
  | attribution
  | delegation
  | derivation

dcExpr :: { Expr }
  : isPartOf
  | description
```

The entity predicate is valid for use with either the resource or artifact types, where resources are an enumerated list of devices typical of mobile phones and artifacts are common communication idioms including file, packet, and memory. The attribute list included with an entity must include an 'prov:artifactType' field with value of 'adapt:artifact' or 'adapt:resource'. Resources must include a 'devType' in their attributes as well.

```
entity          :: { Expr }
  : 'entity' '(' ident ',' attrs(attrVals) ')'

devType
  : 'gps'
  | 'camera'
  | 'keyboard'
  | 'accelerometer'
  | 'microphone'
```

Activity is used to represent units of execution, may include a start time and end time and an attribute list.

```
activity        :: { Expr }
  : 'activity' '(' ident ',' may(time) ',' may(time) soattrVals ')'
  | 'activity' '(' ident soattrVals ')'
```

Artifact creation is communicated with the 'wasGeneratedBy' predicate.

```
generation      :: { Expr }
  : 'wasGeneratedBy' '(' ident ',' may(ident) ',' may(time)
optAttrs(generationAttr) ')'

generationAttr
 : 'genOp'    '=' '\"' genOp '\"'
 | 'nfo:permissions'   '=' NUM

genOp
 : "write"
 | "send"
 | "connect"
 | "truncate"
 | "chmod"
 | "touch"
```

The 'used' predicate is invalid without the 'useOp' attribute.

```
usage           :: { Expr }
  : 'used' '(' ident ',' ident ',' time ',' attrs(usageAttr) ')'

usageAttr :: { (Key,Value) }
  : 'args'      '=' STRINGLIT
  | 'returnVal' '=' STRINGLIT
  | 'operation' '=' '\"' useOp '\"'

useOp :: { Value }
  : "read"
  | "recv"
  | "accept"
  | "execute"
```

Start and end predicates require the time field, but are otherwise identical to their Prov-N counterparts.

Most predicates in Prov-N can be assigned an optional identifier, syntactically 'predicate([ident1 ;]? .. other args ...)'. In the following these predicates are shown without the identifier because the presented conceptual model does not allow additional references, such as 'description's, to refer to

predicates - thus rendering any predicate identification moot.

```
start           :: { Expr }
  : 'wasStartedBy' '(' ident ',' may(ident) ',' may(ident) ',' time soattrVals ')'

end             :: { Expr }
  : 'wasEndedBy' '(' ident ',' may(ident) ',' may(ident) ',' time ','
attrs(endedAttr) ')'

endedAttr :: { (Key,Value) }
  : 'execOp' '=' '\"' execOp '\"'
```

All inter-activity communication (conceptual, communication between two units of execution) is described by one of a set of enumerated identifiers including fork, clone, execve, kill, and setuid.

```
communication   :: { Expr }
  : 'wasInformedBy' '(' ident ',' ident optAttrs(informedAttr) ')'
  | 'wasInformedBy' '(' ident soattrVals ')'

informedAttr :: { (Key,Value) }
  : 'atTime'    '=' time
  | 'useOp' '=' '\"' execOp '\"'

execOp :: { Value }
  : "fork"
  | "clone"
  | "execve"
  | "kill"
  | "setuid"
```

Agents remain an abstract concept and are legitimate for describing anything from a user to a process or even a threat.

```
agent           :: { Expr }
  : 'agent' '(' ident attrs(agentAttr) ')'

agentAttr     :: { (Key,Value) }
  : 'uniqueID' '=' uuid
  | 'foaf:name' '=' STRINGLIT
  | 'foaf:accountName' '=' STRINGLIT
  | 'machineID'  '=' uuid

uuid
 : STRINGLIT
```

Units of execution can be associated with agents using the prov 'wasAssociatedWith' marking. The three identifiers are, in order, activity (unit of execution), agent, and as-yet unspecified plan identifier.

```
association     :: { Expr }
  : 'wasAssociatedWith' '(' ident ',' ident ',' may(ident) ')'
```

Attribution of artifacts to agents or activities follows naturally from Prov-N.

```
attribution     :: { Expr }
  : 'wasAttributedTo' '(' ident ',' ident ')'
```

Invalidation of an artifact by an activity requires a time.

```
invalidation    :: { Expr }
  : 'wasInvalidatedBy' '(' ident ',' ident ',' time ')'
```

As in Prov, either through analysis or direct observation, activities can be said to act on behalf of agents.

```
delegation      :: { Expr }
  : 'actedOnBehalfOf' '(' ident ',' ident ', ' may(ident) ')'
```

Similarly, artifacts can be derived from other artifacts. Both the subject and the object must be artifact, this should not be used for entities of prov:type resource.

```
derivation      :: { Expr }
I  | 'wasDerivedFrom' '(' ident ',' ident optAttrs(derivationAttr) ')'

deriviationAttr
  : 'prov:atTime' '=' time
  | 'deriveOp'    '=' deriveOp

deriveOp
 : 'rename'
 | 'link'
```

The is-part-of and description fields remain sketches but, as with the entirety of the language including all attributes and enumerations, are expected to grow to meet the needs of the project.

```
isPartOf        :: { Expr}
  : 'isPartOf'  '(' ident ',' ident ')'

description     :: { Expr }
  : 'description' '(' ident soattrVals ')'
```

Some optional fields can use a marker (hyphen) instead of an identifier or other literal. Notice not all fields optional in Prov-N remain optional in this language.

```
may(p)
  : '-'
  | p
```

Optional attribute fields are fields that might be entirely non-existant.

```
optAttrs(avParse)
  : ',' attrs(avParse)
  | {- empty -}

attrs(avParse)
  : '[' sep(',', avParse) ']'        { $2 }
```

Comma-separated option attribute values, 'soattrVals' in the above descriptions, are a sign of a portion of the syntax that remains under-specified and are represented by a catch-all that parses generic key-value pairs.

```
soattrVals :: { [(Key,Value)] }
  : ',' oattrVals
  | {- empty -}

oattrVals :: { [(Key,Value)] }
  : '['  attrVals ']'

attrVals :: { [(Key,Value)] }
  : sep(',', attrVal)

attrVal :: { (Key,Value) }
  : ident '=' literal
```