SC-PROV-N

Milan Markovic University of Aberdeen, UK

1 Introduction

SC-PROV-N estends PROV-N, which is a syntax for realising concepts defined by PROV-DM in a human-readable form. This makes it easier to communicate the provenance describtions for the puroses of teaching, illustrating, formalizing, and discussing provenance related issues.[1]

1.1 Grammar Overview

The notation adopts the functional style syntax (predicate name and ordered list of terms) with the grammar defined as a subset of Extended Backus-Naur Form (EBNF)[2]. The following applies to the production rules defined as part of PROV-N:

```
A nonterminal symbol are defined as  expr ::= term.
A terminal symbols are defined as <TERMINAL> ::= term.
```

The right-hand side of a rule can be defined by the following terms to match strings of one or more characters:

```
- nonterminal symbol expression
- TERMINAL symbol expression
- "abc" to match strings inside the quotes
- (term)? to match term or nothing}
- (term)+ to match one or more occurrences of term
- (term)* to match zero or more occurrences of term
- (term | term) to match one of the two terms
```

The notation also defines following identifiers that are used to denote *entity, activity, agent, generation, usage, and collection* respectively:

```
eIdentifier     ::=     identifier
aIdentifier     ::=     identifier
agIdentifier    ::=     identifier
gIdentifier     ::=     identifier
uIdentifier     ::=     identifier
cIdentifier     ::=     identifier
```

The identifier is then defined as:

```
identifier    ::=    QUALIFIED_NAME
```

[1] http://www.w3.org/TR/prov-n/
[2] http://www.w3.org/TR/2006/REC-xml11-20060816/

QUALIFIED NAME defines a name consisting of optional prefix and local name, and is a subject to namespace interpretation[3].

2 SC-PROV-N Productions per Component

For completeness, this section also includes the mappings of P-PLAN concepts, as to date these have not been published by the original authors.

2.1 Component 1: SC-PROV Types

2.1.1 Step

```
conditionExpression    ::=    "step" "(" identifier optionalAttributeValuePairs ")"
```

2.1.2 Variable

```
conditionExpression    ::=    "variable " "(" identifier optionalAttributeValuePairs ")"
```

2.1.3 Condition

```
conditionExpression    ::=    "condition" "(" identifier optionalAttributeValuePairs ")"
```

2.1.4 SocialActorSpec

```
conditionExpression    ::=    "socialActorSpec " "(" identifier optionalAttributeValuePairs ")"
```

2.1.5 Incentive

```
conditionExpression    ::=    "incentive " "(" identifier optionalAttributeValuePairs ")"
```

2.1.6 EvaluationContext

```
conditionExpression    ::=    "evaluationContext " "(" identifier optionalAttributeValuePairs ")"
```

2.1.7 ParameterCollection

```
conditionExpression    ::=    "parameterCollection " "(" identifier optionalAttributeValuePairs ")"
```

Furthermore, the following applies to all productions of Component 1:

```
optionalAttributeValuePairs    ::=    ( "," "[" attributeValuePairs "]" )?
attributeValuePairs    ::=    ( | attributeValuePair ( "," attributeValuePair )* )
attributeValuePair    ::=    attribute "=" literal
```

2.2 Component 2: SC-PROV Relations

2.2.1 isImposedOn

```
conditionAssociationExpression    ::=    "isImposedOn" "(" optionalIdentifier conditionIdentifier "," stepIdentifier
optionalAttributeValuePairs ")"
```

[3] http://www.w3.org/TR/prov-n/#prod-QUALIFIED_NAME

2.2.2 hasIncentive

```
incentiveassociationExpression    ::=    "hasIncentive" "(" optionalIdentifier stepIdentifier ","
incentiveIdentifier optionalAttributeValuePairs ")"
```

2.2.3 isConditionOfPlan

```
isConditionOfPlanExpression    ::=    "isConditionOfPlan" "(" optionalIdentifier conditionIdentifier ","
eIdentifier optionalAttributeValuePairs ")"
```

The *eIdentifier* **must** correspond to the identifier of an entity expresses as an *entityExpression* with attributes *prov:type='prov:Plan', prov:type='p-plan:Plan'*.

2.2.4 isVariableOfPlan

```
isVariableOfPlanExpression    ::=    "isVariableOfPlan" "(" optionalIdentifier stepIdentifier "," eIdentifier
optionalAttributeValuePairs ")"
```

The *eIdentifier* **must** correspond to the identifier of an entity expresses as an *entityExpression* with attributes *prov:type='prov:Plan', prov:type='p-plan:Plan'*.

2.2.5 isStepOfPlan

```
isStepOfPlanExpression    ::=    "isStepOfPlan" "(" optionalIdentifier stepIdentifier "," eIdentifier
optionalAttributeValuePairs ")"
```

The *eIdentifier* **must** correspond to the identifier of an entity expresses as an *entityExpression* with attributes *prov:type='prov:Plan', prov:type='p-plan:Plan'*.

2.2.6 hasParameter

```
hasParameterExpression    ::=    "hasParameter" "(" optionalIdentifier conditionIdentifier ","
variableIdentifier | socialActorIdentifier | incentiveIdentifier optionalAttributeValuePairs ")"
```

2.2.7 performs

```
performsExpression    ::=    "performs" "(" optionalIdentifier socialActorIdentifier "," stepIdentifier
optionalAttributeValuePairs ")"
```

2.2.8 requests

```
requestsExpression    ::=    "requests" "(" optionalIdentifier socialActorIdentifier "," stepIdentifier
optionalAttributeValuePairs ")"
```

2.2.9 hadResult

```
resultExpression    ::=    "hadResult" "(" optionalIdentifier eContextIdentifier "," eIdentifier
optionalAttributeValuePairs ")"
```

2.2.10 hadParameterCollection

```
parameterCollectionExpression    ::=    "hadParameterCollection" "(" optionalIdentifier eContextIdentifier
"," parCollIdentifier  optionalAttributeValuePairs ")"
```

2.2.11 hadCondition

```
hadConditionExpression    ::=    "hadConditionCollection" "(" optionalIdentifier eContextIdentifier ","
conditionIdentifier  optionalAttributeValuePairs ")"
```

2.2.12 hadEvaluationSubject

```
hadConditionExpression    ::=    "hadConditionCollection" "(" optionalIdentifier eContextIdentifier ","
aIdentifier  optionalAttributeValuePairs ")"
```

2.2.13 correspondsToSocialActor

```
corespondsToSocActExpression    ::=    "correspondsToSocialActor" "(" optionalIdentifier
agtIdentifier "," socialActorIdentifier  optionalAttributeValuePairs ")"
```

2.2.14 correspondsToVariable

```
corespondsToVarExpression    ::=    "correspondsToVariable" "(" optionalIdentifier
eIdentifier "," variableIdentifier | incentiveIdentifier  optionalAttributeValuePairs ")"
```

2.2.15 correspondsToStep

```
corespondsToStepExpression    ::=    "correspondsToSocialActor" "(" optionalIdentifier
aIdentifier "," stepIdentifier  optionalAttributeValuePairs ")"
```

2.3 Identifiers

The following identifiers are used in the SC-PROV-N

```
variableIdentifier     ::=   identifier
stepIdentifier         ::=   identifier
socialActorIdentifier  ::=    variableIdentifier
incentiveIdentifier    ::=   variableIdentifier
conditionIdentifier    ::=   identifier
eContextIdentifier     ::=   identifier
parCollIdentifier      ::=   cdentifier
```