Automating Provenance Capture in Software Engineering with UML2PROV (Apendix)

Carlos Sáenz-Adán[1*], Beatriz Pérez[1], Luc Moreau[2], Simon Miles[2], and Francisco José García Izquierdo[1]

[1] Dept. of Mathematics and Computer Science, Univ. of La Rioja, La Rioja, Spain,
{carlos.saenz,beatriz.perez,francisco.garcia}@unirioja.es
[2] Dept. of Informatics, King's College London, London, UK,
{luc.moreau,simon.miles}@kcl.ac.uk

1 Introduction

In this document, we present supporting material of the paper entitled "Automating Provenance Capture in Software Engineering with UML2PROV" submitted to 7th International Provenance and Annotation Workshop.

- Section 2 presents the case study used.
- Section 3 provides all the transformation patterns, focusing on patterns referring to collections.
- Section 4 discuss the *aspect* automatically generated by UML2PROV.
- Section 5 provides a UML2PROV example of use by means of the case study from Section 2.

2 Case studies

To illustrate our explanations, here we present two different case studies.

2.1 *University courses* case study

Here, we present a case study of a system that manages the enrolment and attendance of students to seminars of a University course. Figure 1 shows an excerpt of the UML Class Diagram, which comprises the *Student* and the *Seminar* classes. We note that each one of the operations is linked to a stereotype, from Subsection 3.1 of the paper, in order to identify the behaviour of the operation at hand.

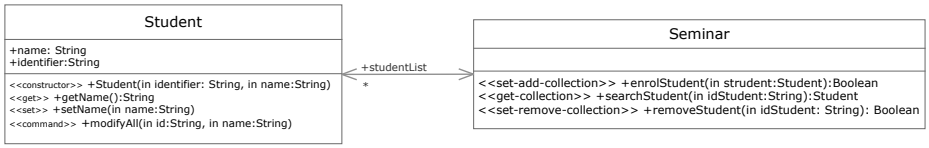

Fig. 1. UML Class diagram showing the structure and the relationship between *Student* and *Seminar* classes

Once this design is implemented in Java, we can provide an example of use. For instance in Figure 4 we can see an example of execution, using an implementation of the design given by the UML diagram from Figure 1 in Java. In particular, this example of execution is made up of 6 method executions, comprising 2 constructors and 4 usual methods.

```
Student student = new Student("id0","Carlos"); // creates a new Student
student.getName();                             // get the name of the Student
student.setName("Bea");                        // set the name of the Student
Seminar seminar = new Seminar();               // creates a new Seminar
seminar.enrolStudent(student);                 // enrol a student in the Seminar
seminar.searchStudent("id0");                  // search the student using the id
```

Fig. 2. Example of execution using the design from Figure 1 implemented in Java

2.2 *Stack* case study

Here, we present a case study of a system that manages the addition and removal of *Person* in a *Stack*. Figure 3 shows an excerpt of the UML Class Diagram, which comprises the *Stack* and the *StackEl*, which is associated to the class *Person*. We note that each one of the operations is linked to a stereotype, from Subsection 3.1 of the paper, in order to identify the behaviour of the operation at hand.

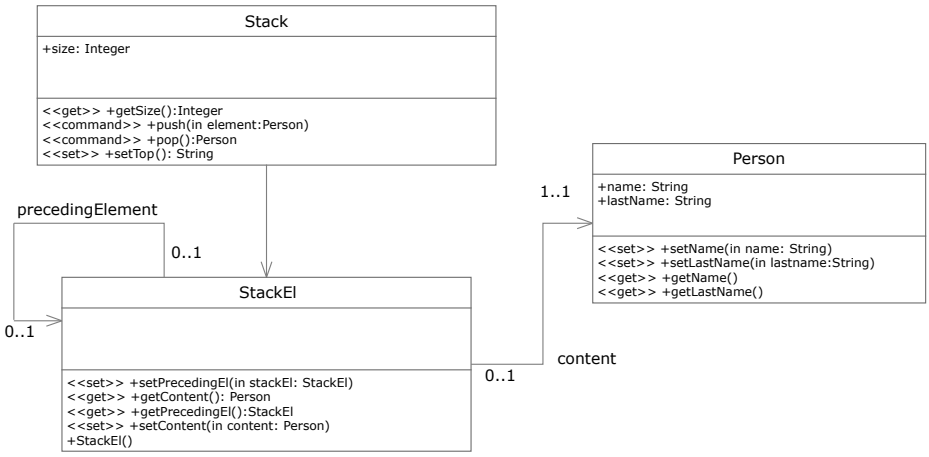

Fig. 3. UML Class diagram showing the structure and the relationship between *Stack*, *Seminar* and *Person* classes

Once this design is implemented in Java, we can provide the set of experiments. For instance in Figure 4 we can see examples of execution, using an implementation of the design given by the UML diagram from Figure 3 in Java.

```
//experiment 1
Stack st = new Stack();
for (int i = 0; i < 25; i++) {
   st.push(new Person("name"+i, "surname"+i));
}
//experiment 2
Stack st = new Stack();
for (int i = 0; i < 50; i++) {
   st.push(new Person("name"+i, "surname"+i));
}
//experiment 3
Stack st = new Stack();
for (int i = 0; i < 100; i++) {
   st.push(new Person("name"+i, "surname"+i));
}
```

```
//experiment 4
for (int i = 0; i < 25; i++) {
   st.pop();
}

//experiment 5
for (int i = 0; i < 50; i++) {
   st.pop();
}

//experiment 6
for (int i = 0; i < 100; i++) {
   st.pop();
}

//experiment 7
Stack st = new Stack();
for (int i = 0; i < 100; i++) {
   st.push(new Person("name"+i, "surname"+i));
}

Stack st2 = new Stack();
for (int i = 0; i < 100; i++) {
   st2.push(st.pop());
}
```

Fig. 4. Examples of execution using the design from Figure 3 implemented in Java

3 Transformation patterns

The set of mappings comprises 8 transformation patterns identified from *CDP1* to *CDP8*, referring *CDP* to *C*lass *D*iagram *P*attern. In Table 1 we shown the patterns explained in Section 3.2 of the paper (*CDP1-CDP6*). Additionally, in this section we complete such a set of patterns with the explanation of patterns referring to collections, *CDP7* (*set-remove-collection*) and *CDP8* (*set-add-collection*), which are presented in Table 5.

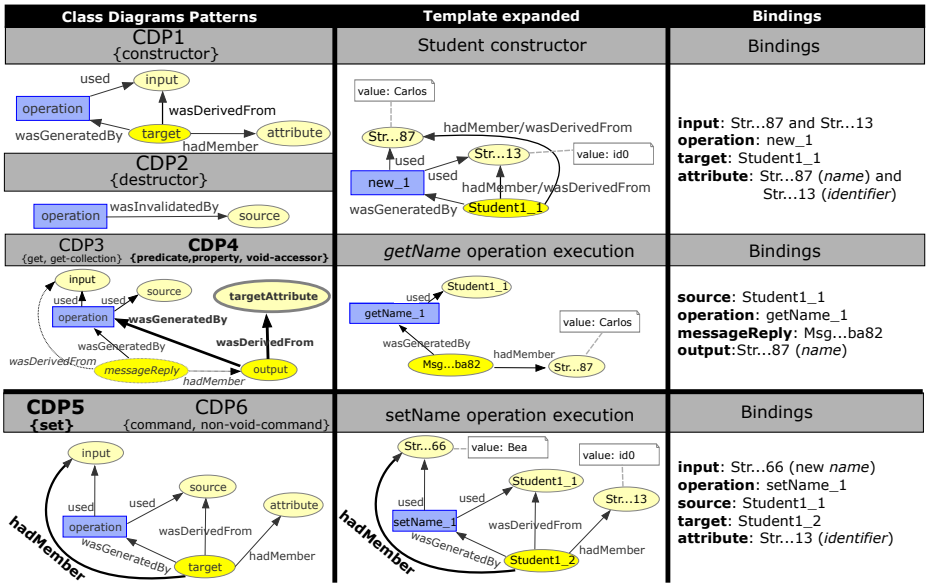

Table 1. Patterns *CDP1-CDP6* including the proposed provenance templates, together with the expanded template and the values of the variables (*bindings*).

As far as ***CDP7 and CDP8*** are concerned, CDP7 represents the transformation of *set-remove-collection*, whereas CDP8 shows the mapping of *set-add-collection*. In particular, both *set-remove-collection* and *set-add-collection* modify the state of a collection data member, thus CDP7 and CDP8 represent the new collection as a `prov:Entity` identified by `var:coll_new` and each one of its elements is mapped to a `prov:Entity` identified by `var:collElements`. With the aim of showing that such elements (identified by `var:collElements`) belong to the collection (`var:coll_new`), we use the relationship `prov:hadMember` between them. Similarly, the collection `var:coll_new`, and the target object's state `var:target` are also linked by the relationship `prov:hadMember`. In turn, to show that the collection (`var:coll_new`), which represents the modified data member collection, has been generated by the operation (identified by `var:operation`), we use the relationship `prov:wasGeneratedBy` between them.

As for the differences between CDP7 and CDP8, *set-add-collection* adds new elements to the collection (given by the *input parameters*), and this behaviour has

to be represented explicitly in the template. Hence, the new elements (given by the *input parameters*) are translated into a `prov:Entity` identified by `var:input` conforming the common transformations shared by all the patterns, and they in turn are related to the data member collection to which they have been added (`var:coll_new`) through the relationship `prov:hadMember` (see this relationship in bold in Table 5).

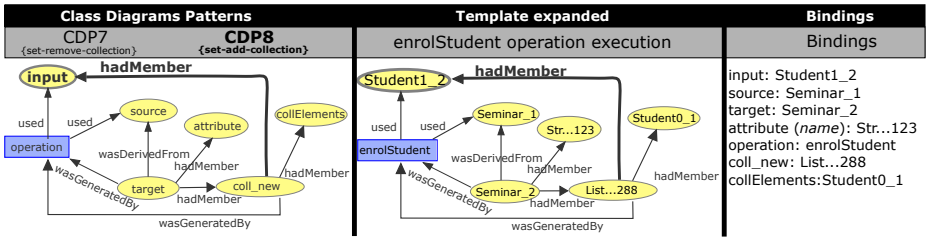

Fig. 5. On the left hand, provenance templates given by CDP3 and CDP4, highlighting in bold the exclusive elements of CDP3. On the right hand, provenance templates given by CDP5 and CDP6, highlighting in bold the exclusive elements of CDP6.

4 Structure of the *aspect* automatically generated

The generated AOP *aspect* in AspectJ implements the behaviour that is to be executed to generate the bindings at specific points in the concrete application code. It is made up of two key elements: the *context-dependent component* comprising the specific points (see Figure 6) and the *context-independent component* with the common functionality located into the *advices* (see Figure 7). In particular, an *advice* brings together a pointcut and a body of code (to run at each of the pointcuts).

The set of *pintcuts* showed in Figure 6 is twofold. First, the list of constructors for which we will generate bindings (e.g. *Student* and *Seminar* constructors), this pointcut is associated with the *advice* identified by *capturedNews*. Second, the list of operations captured for generating bindings (e.g. *getName*, *modifyAll*, *setName*, *enrolStudent*, *searchStudent*, *removeStudent*), this pointcut is related to the *advice* identified by *captureMethods*.

```
import ...;

public aspect provenanceExtractor{

   //list of constructors tackled
   pointcut captureNews():
      initialization(ipaw18.Student.new(..)) ||
      initialization(ipaw18.Seminar.new(..)) ;

   //list of operations tackled
   pointcut captureMethods() :
      call(* ipaw18.Student.getName(..)) ||
      call(* ipaw18.Student.modifyAll(..))||
      call(* ipaw18.Student.setName(..))||
      call(* ipaw18.Seminar.enrolStudent(..))||
      call(* ipaw18.Seminar.searchStudent(..))||
      call(* ipaw18.Seminar.removeStudent(..)) ;
...
```

Fig. 6. Excerpt of an *aspect* in AspectJ showing the *pointcuts*

As we previously said, the *pointcuts* are divided into those tackling the *constructor* operations and those addressing usual operations, and both are related to the *advices capturedNews* and *captureMethods*, respectively.

On the one hand, the behaviour related to *constructor* operations is located into the *advice* identified by *capturedNews*. The kind of this *advice* is *after*, so the behaviour is called after the actual method execution. This behaviour will collect values of: *operation*, *input*, *target*, and *attribute* variables. This fact is represented in Figure 7 by means of the *after advice* associated to *captureNews*.

On the other hand, the behaviour related to usual operations is located into the *advice* identified by *captureMethods*. This kind of *advice* is *around*, so this advice surrounds the actual method execution and captures variables before (*input*, *source* and *operation*) and after (*output*, *target*, *attribute*, *messageReply*, *coll_new* and *collElements*) the execution of the usual method. This fact is represented in Figure 7 by means of the *around advice* associated to *captureMethods*, so that within this clause we first collect values before the execution, then we execute the actual operation and finally, we collect values after the execution.

```
   ...
   after() : captureNews(){
     //collect values linked with operation, input, target
    }

    Object around() : captureMethods() {
     //collect values before the execution
       //execute the operation
     //collect values after the execution

    }
    ...
}
```

Fig. 7. Excerpt of an *aspect* in AspectJ showing the common behaviour

4.1 UML-Artefacts module

This module, implemented in Xpand, is in charge of generating the *context-dependent components* of the *aspect* in AspectJ, that is, the *pointcuts* associated to the constructors and *methods* (see Figure 6). More specifically, this module goes through all the operations from a class diagram and whether it is a constructor (operation with the same name of the class) it adds such an operation in the pointcut *captureNews*, in case the operation is not a *constructor*, it is added in the pointcut *captureMethods*. The rest of the *aspect*'s code corresponds with the common behaviour executed in order to generate the *bindings*. The structure of this common behaviour is showed in Figure 7. Finally, the result of applying this module to the case study presented in Figure 1 can be seen in Figure 6.

```
[...]
pointcut captureNews():
   <<FOREACH operations AS op->>
      <<IF op.name.compareTo(class.name)==0->>
           initialization(<<cl.name>>.new(..))
           <<IF prop!=operations.last()->>||<<ENDIF->>
        <<ENDIF->>
     <<ENDFOREACH->>;

pointcut captureMethods() :
   <<FOREACH operations AS op->>
      <<IF op.name.compareTo(class.name)!=0->>
           call(* <<op.name>>(..))
            <<IF op!=operations.last()->>||<<ENDIF->>
        <<ENDIF->>
     <<ENDFOREACH->>;
[...]
```

Fig. 8. Excerpt of UML-Artefact module in charge of generating the *aspect*'s pointcuts.

5 UML2PROV example of use

The purpose of this section is to provide a full example of (1) the template generation, (2) the bindings collection, and (3) the final PROV document. To illustrate this example we will use the case study presented in Section 2.

5.1 Templates generated

Here, in Figure 9, we can see an example of all the *templates* generated from the case study depicted in Figure 1. In particular, there is one *template* for each one of the *operations* in the class diagram.

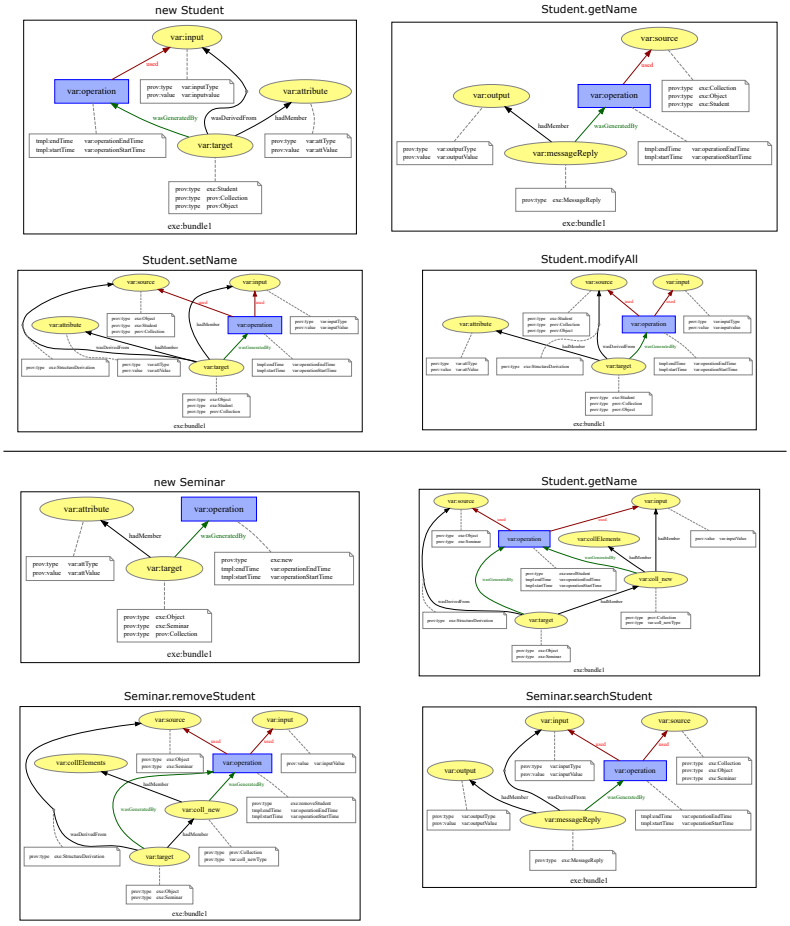

Fig. 9. Set of all the templates automatically generated from the case study presented in Section 2.

5.2 Bindings collected

The bindings depicted in this section have been generated from the execution of the Java source showed in Figure 4.

```
//Student student = new Student("id0","Carlos");
entity(var:target,[tmpl:value_0 =
    'exe:Student-44c85cad-92e2-4f38-8b9e-9cfdc5856802_1'])
entity(var:operation,[tmpl:value_0 =
    'exe:new-477aea06-21e0-4855-8bea-31f756946131'])
entity(var:input,[tmpl:value_0 =
    'exe:String-32901bbf-705b-4a27-bdd9-56bfb2210f93_1',
  tmpl:value_1 = 'exe:String-b3f08f02-0c82-4e1e-bff6-09ed98f02f0c_1'])
entity(var:inputValue,[tmpl:2dvalue_0_0 = 'exe:id0',
  tmpl:2dvalue_1_0 = 'exe:Carlos'])
entity(var:inputType,[tmpl:2dvalue_0_0 = 'exe:java.lang.String',
  tmpl:2dvalue_1_0 = 'exe:java.lang.String'])
entity(var:attribute,[tmpl:value_0 =
    'exe:String-32901bbf-705b-4a27-bdd9-56bfb2210f93_1',
  tmpl:value_1 = 'exe:String-b3f08f02-0c82-4e1e-bff6-09ed98f02f0c_1'])
entity(var:attValue,[tmpl:2dvalue_0_0 = 'exe:id0',
  tmpl:2dvalue_1_0 = 'exe:Carlos'])
entity(var:attType,[tmpl:2dvalue_0_0 = 'exe:java.lang.String',
  tmpl:2dvalue_1_0 = 'exe:java.lang.String'])
```

```
//student.getName();
entity(var:operation,[tmpl:value_0 =
    'exe:getName-2b7c2d1e-9060-4290-93e2-3e1c401298ba'])
entity(var:source,[tmpl:value_0 =
    'exe:Student-44c85cad-92e2-4f38-8b9e-9cfdc5856802_1'])
entity(var:operationStartTime,[tmpl:2dvalue_0_0 = "2018-03-16T19:13:16"
    %% xsd:dateTime])
entity(var:operationEndTime,[tmpl:2dvalue_0_0 = "2018-03-16T19:13:16" %%
    xsd:dateTime])
entity(var:target,[tmpl:value_0 =
    'exe:Student-44c85cad-92e2-4f38-8b9e-9cfdc5856802_1'])
entity(var:messageReply,[tmpl:value_0 =
    'exe:d742781c-db73-4816-a6e4-8e6016fd5683'])
entity(var:output,[tmpl:value_0 =
    'exe:String-b3f08f02-0c82-4e1e-bff6-09ed98f02f0c_1'])
entity(var:outputValue,[tmpl:2dvalue_0_0 = 'exe:Carlos'])
entity(var:outputType,[tmpl:2dvalue_0_0 = 'exe:java.lang.String'])
```

```
//student.setName("Bea");
entity(var:operation,[tmpl:value_0 =
    'exe:setName-828cd123-f57e-46bb-91bf-112b342bf2db'])
entity(var:source,[tmpl:value_0 =
    'exe:Student-44c85cad-92e2-4f38-8b9e-9cfdc5856802_1'])
```

```
entity(var:operationStartTime,[tmpl:2dvalue_0_0 = "2018-03-16T19:13:16"
    %% xsd:dateTime])
entity(var:operationEndTime,[tmpl:2dvalue_0_0 = "2018-03-16T19:13:16" %%
    xsd:dateTime])
entity(var:target,[tmpl:value_0 =
    'exe:Student-44c85cad-92e2-4f38-8b9e-9cfdc5856802_2'])
entity(var:messageReply,[tmpl:value_0 =
    'exe:c82b9fb8-d77d-4184-8d10-ce483072a827'])
entity(var:input,[tmpl:value_0 =
    'exe:String-3b891511-36c2-4e14-81c0-783b86781512_1'])
entity(var:inputValue,[tmpl:2dvalue_0_0 = 'exe:Bea'])
entity(var:inputType,[tmpl:2dvalue_0_0 = 'exe:java.lang.String'])
entity(var:attribute,[tmpl:value_0 =
    'exe:String-32901bbf-705b-4a27-bdd9-56bfb2210f93_1',
  tmpl:value_1 = 'exe:String-3b891511-36c2-4e14-81c0-783b86781512_1'])
entity(var:attValue,[tmpl:2dvalue_0_0 = 'exe:id0',
  tmpl:2dvalue_1_0 = 'exe:Bea'])
entity(var:attType,[tmpl:2dvalue_0_0 = 'exe:java.lang.String',
  tmpl:2dvalue_1_0 = 'exe:java.lang.String'])
```

```
//Seminar seminar = new Seminar();
entity(var:target,[tmpl:value_0 =
    'exe:Seminar-3a36f92c-3408-45ff-ab9a-0205bc297386_1'])
entity(var:operation,[tmpl:value_0 =
    'exe:new-ff919776-0689-46eb-bb09-d3aa4dd51fbc'])
entity(var:attribute,[tmpl:value_0 =
    'exe:ArrayList-5927eee1-f3f7-41f5-8ede-1c36b618bf41_1'])
entity(var:attValue,[tmpl:2dvalue_0_0 = 'exe:empty'])
entity(var:attType,[tmpl:2dvalue_0_0 = 'exe:List'])
```

```
//seminar.enrolStudent(student);
entity(var:operation,[tmpl:value_0 =
    'exe:enrolStudent-4aa84f66-292a-4dc1-b959-53bb760b6652'])
entity(var:source,[tmpl:value_0 =
    'exe:Seminar-3a36f92c-3408-45ff-ab9a-0205bc297386_1'])
entity(var:operationStartTime,[tmpl:2dvalue_0_0 = "2018-03-16T19:13:16"
    %% xsd:dateTime])
entity(var:operationEndTime,[tmpl:2dvalue_0_0 = "2018-03-16T19:13:16" %%
    xsd:dateTime])
entity(var:target,[tmpl:value_0 =
    'exe:Seminar-3a36f92c-3408-45ff-ab9a-0205bc297386_2'])
entity(var:messageReply,[tmpl:value_0 =
    'exe:180db494-d326-40e3-8b03-0f1623f36d82'])
entity(var:coll_new,[tmpl:value_0 =
    'exe:ArrayList-9ce80395-5f7e-428a-a3d3-f8f8bd743164_1'])
entity(var:input,[tmpl:value_0 =
    'exe:Student-44c85cad-92e2-4f38-8b9e-9cfdc5856802_2'])
entity(var:inputValue,[tmpl:2dvalue_0_0 = 'exe:ipaw18.Student@704a52ec'])
```

```
entity(var:inputType,[tmpl:2dvalue_0_0 = 'exe:ipaw18.Student'])
entity(var:output,[tmpl:value_0 =
    'exe:Boolean-58a4fb76-12f2-4b8b-9339-3cc98d5fbd85_1'])
entity(var:outputValue,[tmpl:2dvalue_0_0 = 'exe:true'])
entity(var:outputType,[tmpl:2dvalue_0_0 = 'exe:java.lang.Boolean'])
```

```
//seminar.searchStudent("id0");
entity(var:operation,[tmpl:value_0 =
    'exe:searchStudent-2d7cc5a5-4466-43e7-914a-d2333dc09bbd'])
entity(var:source,[tmpl:value_0 =
    'exe:Seminar-3a36f92c-3408-45ff-ab9a-0205bc297386_2'])
entity(var:operationStartTime,[tmpl:2dvalue_0_0 = "2018-03-16T19:13:16"
    %% xsd:dateTime])
entity(var:operationEndTime,[tmpl:2dvalue_0_0 = "2018-03-16T19:13:16" %%
    xsd:dateTime])
entity(var:target,[tmpl:value_0 =
    'exe:Seminar-3a36f92c-3408-45ff-ab9a-0205bc297386_2'])
entity(var:messageReply,[tmpl:value_0 =
    'exe:eaaa663d-7652-4df1-9721-8bfed1b88e21'])
entity(var:input,[tmpl:value_0 =
    'exe:String-f3e6cff8-1e2d-435f-879d-c6fc0c7e8253_1'])
entity(var:inputValue,[tmpl:2dvalue_0_0 = 'exe:id0'])
entity(var:inputType,[tmpl:2dvalue_0_0 = 'exe:java.lang.String'])
entity(var:output,[tmpl:value_0 =
    'exe:Student-44c85cad-92e2-4f38-8b9e-9cfdc5856802_2'])
entity(var:outputValue,[tmpl:2dvalue_0_0 =
    'exe:ipaw18.Student@704a52ec'])
entity(var:outputType,[tmpl:2dvalue_0_0 = 'exe:ipaw18.Student'])
```

5.3 Final PROV document

Here in Figure 10 we can see the final PROV document given by the expansion algorithm. This algorithm has used as source the PROV templates from Subsection 5.1 and the set of bindings from Subsection 5.2. Due to the size of this diagram, we refers the reader to `https://provenance.ecs.soton.ac.uk/store/documents/117950/`.

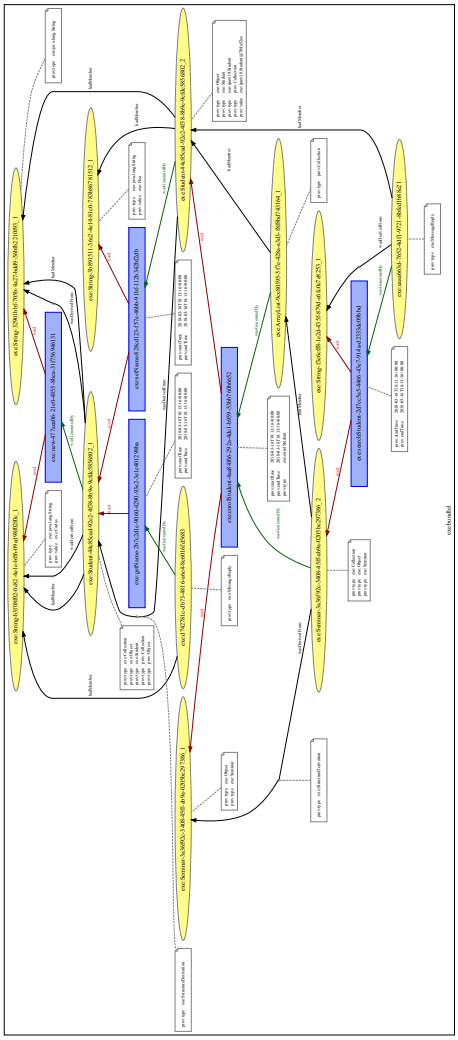

Fig. 10. Set of all the templates automatically generated from the case study presented in Section 2.