Integrating Provenance Capture and UML with UML2PROV: Principles and Experience –Description of patterns–

List of authors

Before reading

Each pattern has been written in a self-contained way. A reader who reads all the patterns globally will find similar explanations, even repeated ones, in several patterns. We have preferred to make the reader suffer this small inconvenience, instead of running the risk that an occasional reader of a particular pattern loses part of the explanations that are discussed elsewhere.

We assume that the reader is familiar with the following UML diagrams: UML Sequence Diagrams (SqDs), UML State Machine Diagrams (SMDs), and UML Class diagrams (CDs). Readers unfamiliar with these diagrams are encouraged to read the UML specification [1]. Additionally, due to the fact that transformations referring to CDs make use of concrete UML stereotypes explained in Appendix A, we refer the reader to this appendix in case of doubts about them.

Likewise, we assume that reader is knowledgeable in both the PROV data model (PROV-DM) [2], to represent provenance information, and the PROV template approach [3], for designing provenance. If this is not the case, she/he is referred to [2] and [3], respectively.

1.1 Notational conventions

More than a terminological nuance, the distinction between the *state* and the *status* of an object is fundamental to understanding this document.

- In SMDs, in accordance to UML terminology [1], the *state* of an object denotes a situation during which some invariant conditions holds.
- In CDs, we use the term object *status* with a broad scope, referring to the values of the object's attributes at some moment, which particularly could correspond to a concrete *state* but not necessarily.

The PROV templates throughout this document are represented following the PROV graph conventions given in [4].

In this document, we use *qualified names* (e.g., `prov:value`) in accordance to PROV-DM [2]. In compliance with PROV-DM, we note that a *qualified name* can be mapped to an Internationalized Resource Identifier (IRI) [5] by concatenating the IRI associated with the prefix (e.g., `prov`) and the local part (e.g., `value`). Every *qualified name* with a prefix refers to the namespace of the prefix. The following namespaces and prefixes are used throughout this document.

prefix	namespace IRI	definition
var	`http://openprovenance.org/var#`	The namespace for template variables
prov	`http://www.w3.org/ns/prov#`	The PROV namespace
xsd	`http://www.w3.org/2000/10/XMLSchema#`	XML Schema namespace
u2p	`http://um2prov.unirioja.es/ns/u2p#`	UML2PROV namespace

Table 1. Prefix and Namespaces used in this specification

1.2 Structure of the patterns

We have structured the explanation of the defined patterns in the same five blocks: **Identifier**, **Context**, **UML Diagram**, **Mapping to PROV**, and **Discussion**. See the explanation of each block below.

1.2.1 Identifier

Unique identifier of the transformation pattern. It is an acronym that refers to the type of UML diagram together with a numeric identifier. The UML *Seq*uence diagram *P*atterns are referred to as *SeqP<N>*, where `N` is the numeric identifier. Likewise, *StP<N>* corresponds to the UML *St*ate Machine *P*atterns, and *ClP<N>* to the UML *Cl*ass Diagram *P*atterns.

1.2.2 Context

The behaviour addressed by the pattern. In order to give an explanation free of context, being as agnostic as possible about any type of modelling language, we will use the natural language including well-known software engineering terminology (e.g., *object*, *operation*...), to identify the part of the domain for which the corresponding pattern proposes a translation.

Each pattern context block will include a detailed description of its *key elements*. When necessary, we will use nested elements to describe the different alternatives through which certain *key elements* participate in the context. We remark that not all the identified *key elements* explicitly appear in the context. Some patterns identify specific *key elements* that are inferred from the context because they play an important role in the pattern.

1.2.3 UML Diagram

This block will depict the excerpt of the UML diagram with the elements that model the previous *key elements*. In addition, we provide a table, whose structure is illustrated below, that explains the representation of each *key element* in UML, by means of UML elements. Additionally, we assign a green label containing a numeric identifier to each UML element, which makes it easier its location in the UML diagram.

Key Element	UML	Rationale
Name of the element	`UML element` id	The fundamental reasons serving to account for the use of the *UML element* for modelling the *key element*.

1.2.4 Mapping to PROV

This block contains the PROV template generated from the previous excerpt of the *UML Diagram*, together with a detailed explanation about the transformation, that is, the generated *PROV elements*, *attributes*, and *PROV relations*. To each PROV element in the template will be assigned the numeric identifier of the UML element from which it comes from. Additionally, the relation that appears in the PROV template will be labeled with a letter that helps link such a relation with its description. The structure used to specify this block is the following:

PROV elements

UML	PROV / id	Rationale
`UML element` id	`PROV element` id / `var:<identifier>`	The explanation of the mapping between *UML element* and *PROV element*.

Attributes

PROV Element	Attribute / Value	Description
`PROV element` id	name of attribute / assigned value	Description of the meaning of the attribute and its value.

PROV relations

PROV relation id	Description of the relation.

NOTE: In PROV, two relationships of the form (B, `prov:used`, A) and (C, `prov:wasGeneratedBy`, B) may be enriched with (C, `prov:wasDerivedFrom`, A) to express the dependency of C on A. This structure is a provenance construction called *use-generate-derive triangle* [3] which explicitly connects a *generated* `prov:Entity` to a *used* `prov:Entity`. In the realm of this work, it may be applied in those templates in which a `prov:Entity` is *used* by a `prov:Activity`, and such a `prov:Activity` *generated* another `prov:Entity`. However, aiming at avoiding the overburden of the PROV template explanations with information that can be inferred, we have decided to include the relation `prov:wasDerivedFrom` only when the context of the pattern refers to such a derivation.

1.2.5 Discussion

Issues related to the transformation of UML to PROV. Concretely, we will focus on the explanation and justification of our transformation decisions together with alternative solutions (if any), and some questions that are likely to come up to the reader.

Index of patterns

UML Sequence Diagrams Patterns

UML State Machine Patterns

UML Class Diagrams Patterns

Pattern identifier	Context	Page
SeqP1	In a system, a participant (the sender) interacts with another participant (the recipient) by calling an operation in the recipient, and then, it continues immediately. The call causes the recipient to execute an operation.	5
SeqP2	In a system, a participant (the sender) interacts with another participant (the recipient) by calling an operation in the recipient and waits for a response. The call causes the recipient to execute the operation and to respond the sender after the execution.	7
SeqP3	During the execution of an operation (main operation), a nested operation call is made. After this call, the execution of the main operation can either continue immediately or wait for a response of that nested operation call. This way, this pattern complements *SeqP1* and *SeqP2*.	10
SeqP4	During the execution of an operation (main operation), a response of a previously issued nested operation call is received. The main operation's execution uses this response to complete its behaviour. This way, this pattern complements *SeqP1* and *SeqP2*, when the execution of the main operation has a call to a nested operation and waits to receive its response (pattern *SeqP3* with `Synchronous Message`).	13

Identifier *Seq*uence diagram *P*attern *1* (*SeqP1*)

Context

In a system, a participant (the sender) interacts with another participant (the recipient) by calling an operation in the recipient, and then, it continues immediately. The call causes the recipient to execute an operation.

Key elements

Sender	It is the participant that makes the operation call.
Operation call	It is the call that starts the execution of the operation.
Input data	It is the information passed to the operation through the *Operation call*.
Operation execution	It is the execution of the operation.

UML Diagram

Key Element	UML	Rationale
Sender	`Lifeline` 1	It models the *Sender* participant involved in the interaction.
Operation call	`Asynchronous Message` 2	It models the *Operation call* when the *Sender* does not wait for a response, but instead continues immediately after sending the message.
Input data	`Input Arguments` 3	They specify the information passed to the operation through the *Operation call*.
Operation execution	`ExecutionSpecification` 4	It shows the period of time that the recipient's participant devotes to the *Operation execution*.

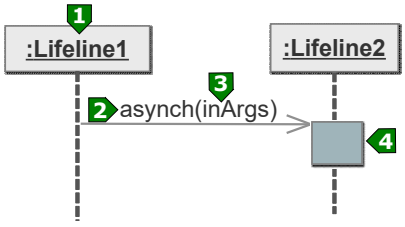

Figure 1. UML representation that models the context given by *SeqP1*

Mapping to PROV

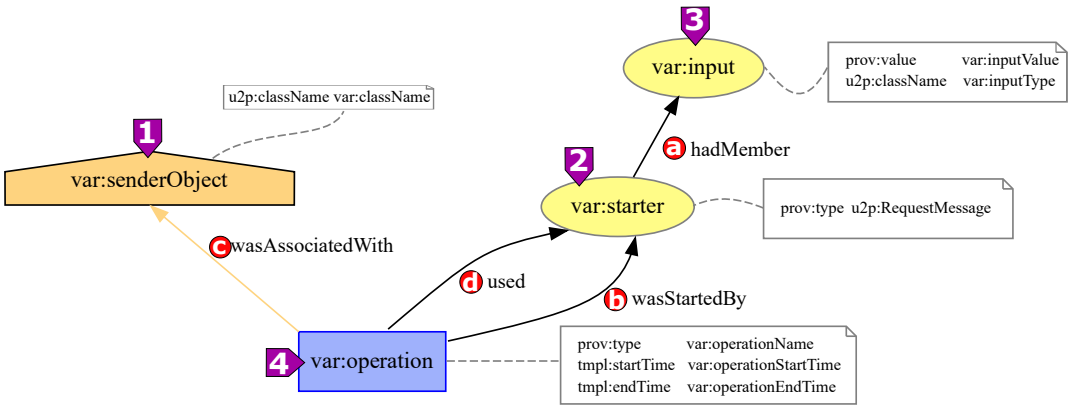

Figure 2. PROV template generated from the UML representation in Figure 1

PROV elements

UML	PROV / id	Rationale
`Lifeline` 1	`prov:Agent` 1 / `var:senderObject`	The sender `Lifeline` 1 is mapped to a `prov:Agent` identified by `var:senderObject`. It assumes the responsibility for starting the `ExecutionSpecification` 4.
`Asynchronous Message` 2	`prov:Entity` 2 / `var:starter`	The `Asynchronous Message` 2 that initiates the `ExecutionSpecification` 4 of the recipient is a `prov:Entity` with identifier `var:starter`.
`Input Arguments` 3	`prov:Entity` 3 / `var:input`	Each argument of `Input Arguments` 3 is a separate `prov:Entity` identified as `var:input`.
`ExecutionSpecification` 4	`prov:Activity` 4 / `var:operation`	The `ExecutionSpecification` 4 is a `prov:Activity` with identifier `var:operation`.

Attributes

PROV Element	Attribute / Value	Description
`var:senderObject` 1	`u2p:typeName` / `var:className`	The value `var:className` is the string with the name of the class to which the `var:senderObject` 1 belongs.
`var:starter` 2	`prov:type` / `u2p:RequestMessage`	The value `u2p:RequestMessage` shows that `var:starter` 2 is a request message.
`var:input` 3	`prov:value` / `var:inputValue`	The value `var:inputValue` is the direct representation of `var:input` 3.
	`u2p:typeName` / `var:inputType`	The value `var:inputType` is the string with the name of the class to which `var:input` 3 belongs.
`var:operation` 4	`prov:type` / `var:operationName`	The value `var:operationName` is the name of the operation `var:operation` 4.
	`tmpl:startTime` / `var:operationStartTime`	`var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 4.
	`tmpl:endTime` / `var:operationEndTime`	`var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 4.

PROV relations

a	`prov:hadMember`	It states that `var:input` is one of the elements in `var:starter`.
b	`prov:wasStartedBy`	`var:operation` is deemed to have been started by `var:starter`.
c	`prov:wasAssociatedWith`	It is the assignment of responsibility to `var:senderObject` for `var:operation`.
d	`prov:used`	It is the beginning of utilizing `var:starter` by `var:operation`.

Discussion

- It is worth noting that Figure 2 depicts the responsibility of the sender lifeline (`var:senderObject`) for executing the operation (`var:operation`) in a recipient lifeline. However, the recipient lifeline is not modelled in this PROV template, even though it is the participant that executes the operation. This decision is based on other patterns' better ability to both (1) identify the participant responsible for executing that operation, and (2) give a more detailed information about the implications that the execution of that operation has in the recipient participant. More specifically, these patterns are: *StP1*-*StP3*, which mainly focus on representing possible changes in an object's state caused by an operation execution; and patterns *ClP1*-*ClP10*, which put more stress on how the execution affects the status of the object responsible for performing such an execution.

Identifier *Seq*uence diagram *P*attern *2* (*SeqP2*)

Context

In a system, a participant (the sender) interacts with another participant (the recipient) by calling an operation in the recipient and waits for a response. The call causes the recipient to execute the operation and to respond the sender after the execution.

Key elements

Sender	It is the participant that makes the operation call.
Operation call	It is the call that starts the execution of the operation.
Input data	It is the information passed to the operation through the *Operation call*.
Operation execution	It is the execution of the operation.
Response	It is the recipient's response to the *Operation call*.
Output data	It is the information contained in the *Response*.

UML Diagram

Key Element	UML	Rationale
Sender	`Lifeline` 1	It models the *Sender* participant involved in the interaction.
Operation call	`Synchronous Message` 2	It models the *Operation call* when the *Sender* waits for a response.
Input data	`Input Arguments` 3	They specify the information passed to the operation through the *Operation call*.
Operation execution	`ExecutionSpecification` 4	It shows the period of time that the recipient's participant devotes to the *Operation execution*.
Response	`Reply Message` 5	It specifies the response to the *Operation call*.
Output data	`Output Arguments` 6	They specify the information contained in the *Response*.

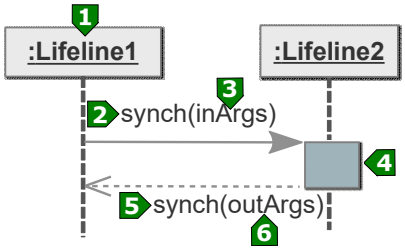

Figure 3. UML representation that models the context given by *SeqP2*

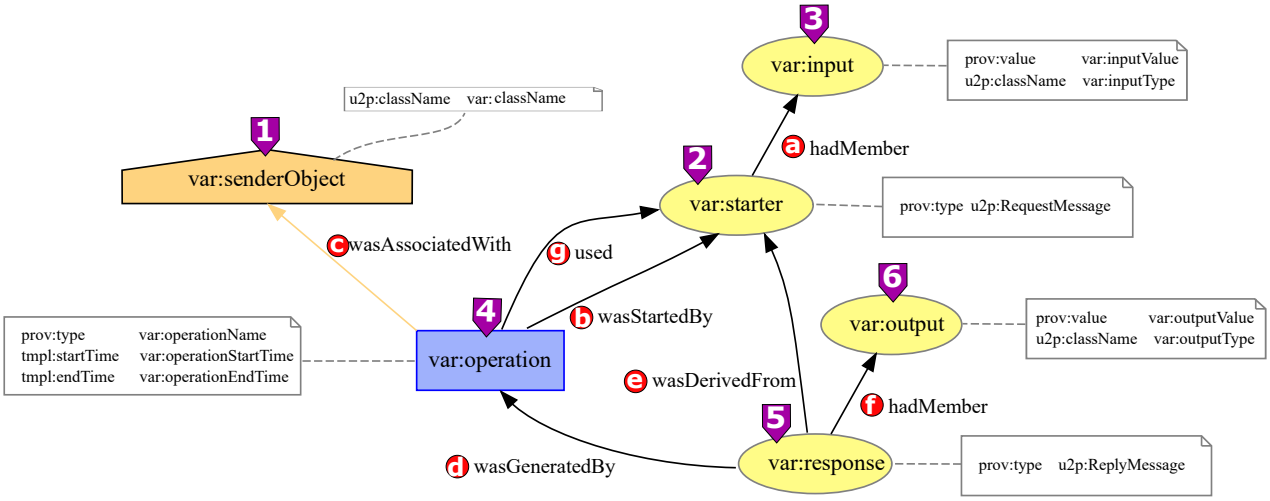

Figure 4. PROV template generated from the UML representation in Figure 3

PROV elements

UML	PROV / id	Rationale
`Lifeline` 1	`prov:Agent` 1 / `var:senderObject`	The sender `Lifeline` 1 is mapped to a `prov:Agent` identified by `var:senderObject`. It assumes the responsibility for starting the `ExecutionSpecification` 4.
`Synchronous Message` 2	`prov:Entity` 2 / `var:starter`	The `Synchronous Message` 2 that initiates the `ExecutionSpecification` 4 of the recipient is a `prov:Entity` with identifier `var:starter`.
`Input Arguments` 3	`prov:Entity` 3 / `var:input`	Each argument of `Input Arguments` 3 is a separate `prov:Entity` identified as `var:input`.
`ExecutionSpecification` 4	`prov:Activity` 4 / `var:operation`	The `ExecutionSpecification` 4 is a `prov:Activity` with identifier `var:operation`.
`Reply Message` 5	`prov:Entity` 5 / `var:response`	The `Reply Message` 5 that responds to the `Synchronous Message` 2 is a `prov:Entity` with identifier `var:response`.
`Output Arguments` 6	`prov:Entity` 6 / `var:output`	Each argument of `Output Arguments` 6 is a separate `prov:Entity` identified as `var:output`.

Attributes

PROV Element	Attribute / Value	Description
var:senderObject 1	u2p:typeName / var:className	The value var:className is the string with the name of the class to which the var:senderObject 1 belongs.
var:starter 2	prov:type / u2p:RequestMessage	The value u2p:RequestMessage shows that var:starter 2 is a request message.
var:input 3	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
	u2p:typeName / var:inputType	The value var:inputType is the string with the name of the class to which the var:input 3 belongs.
var:operation 4	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 4.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 4.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 4.
var:response 5	prov:type / u2p:ReplyMessage	The value u2p:ReplyMessage shows that var:response 5 is a reply message.
var:output 6	prov:value / var:outputValue	The value var:outputValue is the direct representation of var:output 6.
	u2p:typeName / var:outputType	The value var:outputType is a string with the name of the class to which var:output 6 belongs.

PROV relations

a	prov:hadMember	It states that var:input is one of the elements in var:starter.
b	prov:wasStartedBy	var:operation is deemed to have been started by var:starter.
c	prov:wasAssociatedWith	It is the assignment of responsibility to var:senderObject for var:operation.
d	prov:wasGeneratedBy	It is the completion of production of var:response by var:operation.
e	prov:wasDerivedFrom	It is the construction of var:response based on var:starter reception.
f	prov:hadMember	It states that var:output is one of the elements in var:response.
g	prov:used	It is the beginning of utilizing var:starter by var:operation.

Discussion

- It is worth noting that Figure 4 depicts the responsibility of the sender lifeline (var:senderObject) for executing the operation (var:operation) in a recipient lifeline. However, the recipient lifeline is not modelled in this PROV template, even though it is the participant that executes the operation. This decision is based on other patterns' better ability to both (1) identify the participant responsible for executing that operation, and (2) give a more detailed information about the implications that the execution of that operation has in the recipient participant. More specifically, these patterns are: *StP1*-*StP3*, which mainly focus on representing possible changes in an object's state caused by an operation execution; and patterns *ClP1*-*ClP10*, which put more stress on how the execution affects the status of the object responsible for performing such an execution.

Identifier *Seq*uence diagram *P*attern *3* (*SeqP3*)

Context

During the execution of an operation (main operation), a nested operation call is made. After this call, the execution of the main operation can either continue immediately or wait for a response of that nested operation call. This way, this pattern complements *SeqP1* and *SeqP2*.

Key elements

(Main) *Operation execution*	It is the execution of the main operation.
(Nested) *Operation call*	It is the nested operation call sent during the *Main operation execution*.

UML Diagram

Key Element	UML	Rationale
Main operation execution	`ExecutionSpecification` 1	It shows the period of time that takes the *Main operation execution*.
Nested operation call	`Asynchronous Message` 2 or `Synchronous Message` 2	It models the *Nested operation call* either when its sender waits for a response, or when it does not wait for a response, but instead continues immediately after sending the message.

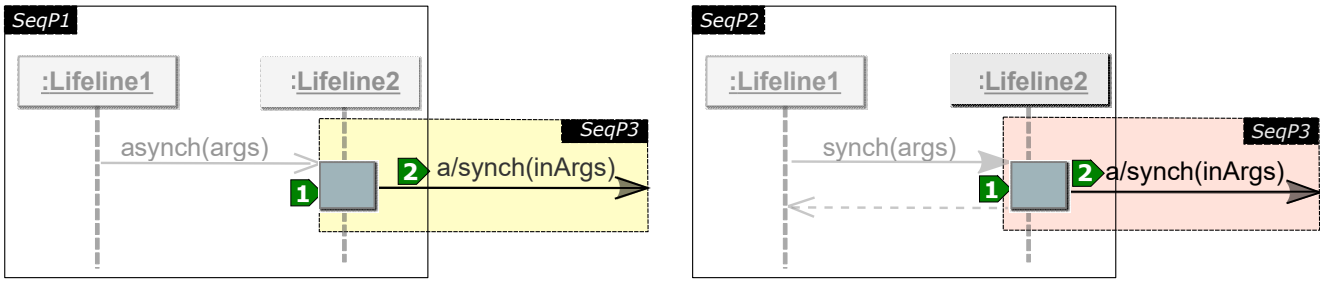

Figure 5. The left hand side is the UML representation of *SeqP1* complemented by *SeqP3*, whereas the right hand side is the UML representation of *SeqP2* complemented by *SeqP3*. Only the shaded areas correspond to the UML elements contributed by this pattern.

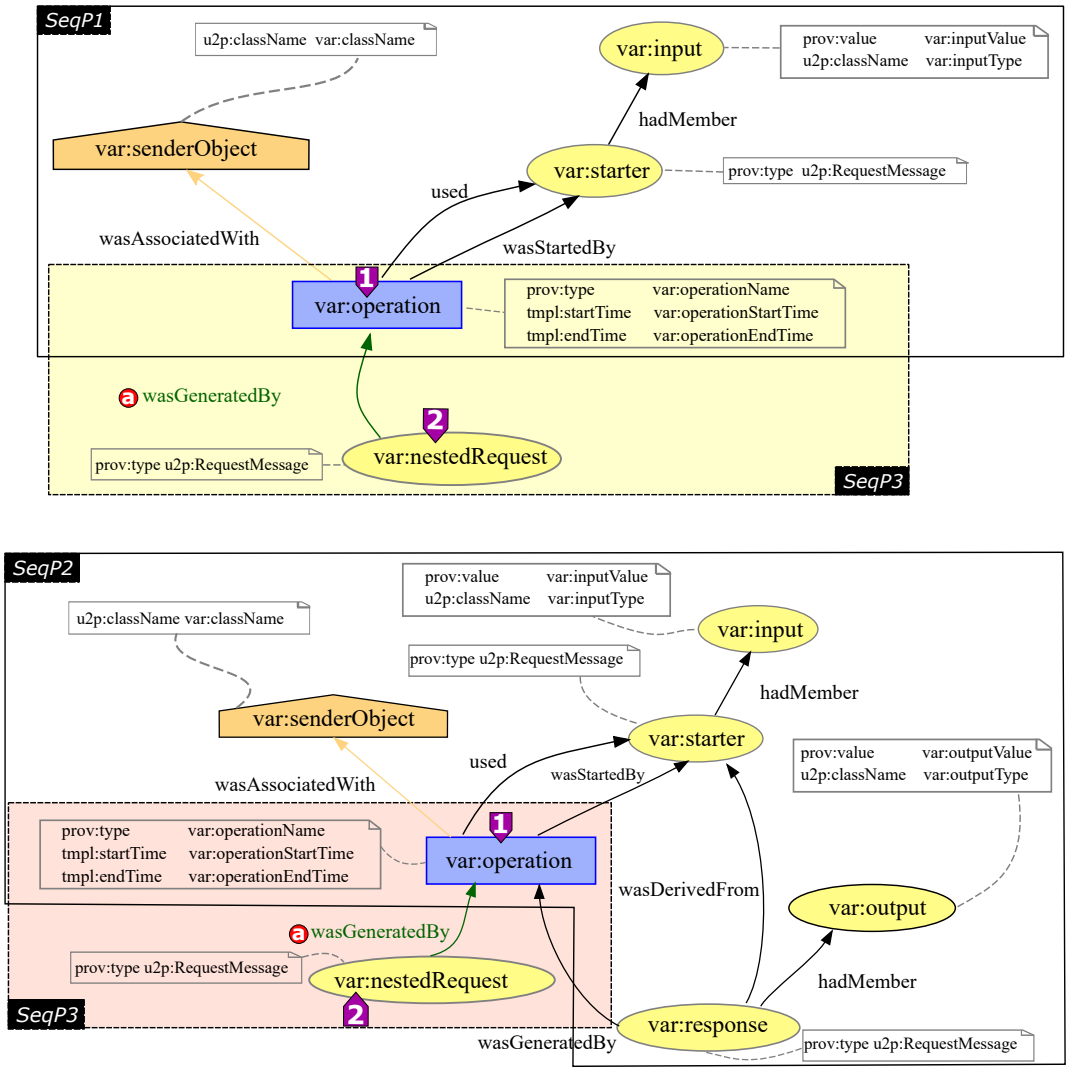

Figure 6. At the top, it is a PROV template generated from the UML representation in the left side of Figure 5. At the bottom, it is a PROV template generated from the UML representation in the right side of Figure 5. Only the shaded areas correspond to the PROV elements contributed by this pattern.

PROV elements

UML	PROV / id	Rationale
`ExecutionSpecification` 1	`prov:Activity` 1 / `var:operation`	The `ExecutionSpecification` 1 is a `prov:Activity` with identifier `var:operation`.
`Asynchronous Message` 2 or `Synchronous Message` 2	`prov:Entity` 2 / `var:nestedRequest`	The `Asynchronous Message` or `Synchronous Message` 2 sent from the `ExecutionSpecification` 1 is a `prov:Entity` with identifier `var:nestedRequest`.

Attributes

PROV Element	Attribute / Value	Description
var:operation 1	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 1
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 4.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 4.
var:nestedRequest 2	prov:type / u2p:RequestMessage	The value u2p:RequestMessage shows that var:nestedRequest 5 is a request message.

PROV relations

prov:wasGeneratedBy	It is the completion of production of var:nestedRequest by var:operation.

Discussion

- Note that the same element (request message in this case) appears in different patterns playing different roles. In *SeqP3* the (nested) request message models the call started from an ExecutionSpecification. However, in *SeqP1* and *SeqP2*, this same request message models the call that starts an ExecutionSpecification. The former way of looking at the request message is translated into var:nestedRequest (in *SeqP3*), and the latter is translated into var:starter (in *SeqP1* and *SeqP2*). Consequently, despite var:nestedRequest and var:starter being two different elements of type prov:Entity appearing in two different PROV templates, both must be assigned to the same value during the execution of the application. Therefore, after merging all the expanded PROV templates, a single prov:Entity will be generated.

Identifier *Seq*uence diagram *P*attern *4* (*SeqP4*)

Context

During the execution of an operation (main operation), a response of a previously issued nested operation call is received. The main operation's execution uses this response to complete its behaviour. This way, this pattern complements *SeqP1* and *SeqP2*, when the execution of the main operation has a call to a nested operation and waits to receive its response (pattern *SeqP3* with `Synchronous Message`).

Key elements

(Main) *Operation execution*	It is the execution of the main operation.
(Nested) *Response*	It is the response to a nested operation call.
(Main) *Response*	It is the response of the *Main operation execution*. This element is only identified when this pattern complements *SeqP2*.

UML Diagram

Key Element	UML	Rationale
Main operation execution	`ExecutionSpecification` 1	It shows the period of time that takes the *Main operation execution*.
Nested response	`Reply Message` 2	It specifies the response received in the *Main operation execution*.
Main response	`Reply Message` 3	In case of complementing *SeqP2*, it specifies the response of the *Main operation execution*.

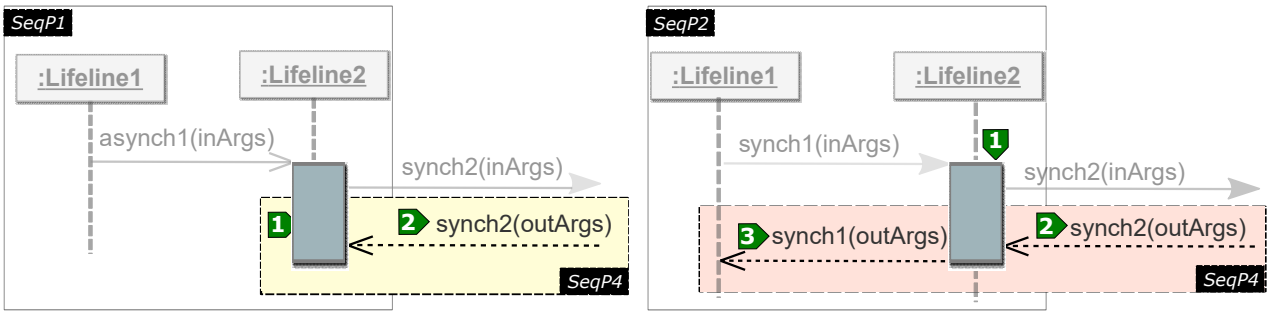

Figure 7. The left hand side is the UML representation that models the context given by *SeqP1* complemented by *SeqP4*, wheres the right hand side is the UML representation that models the context given by *SeqP2* complemented by *SeqP4*. Only the shaded areas correspond to the UML elements contributed by this pattern.

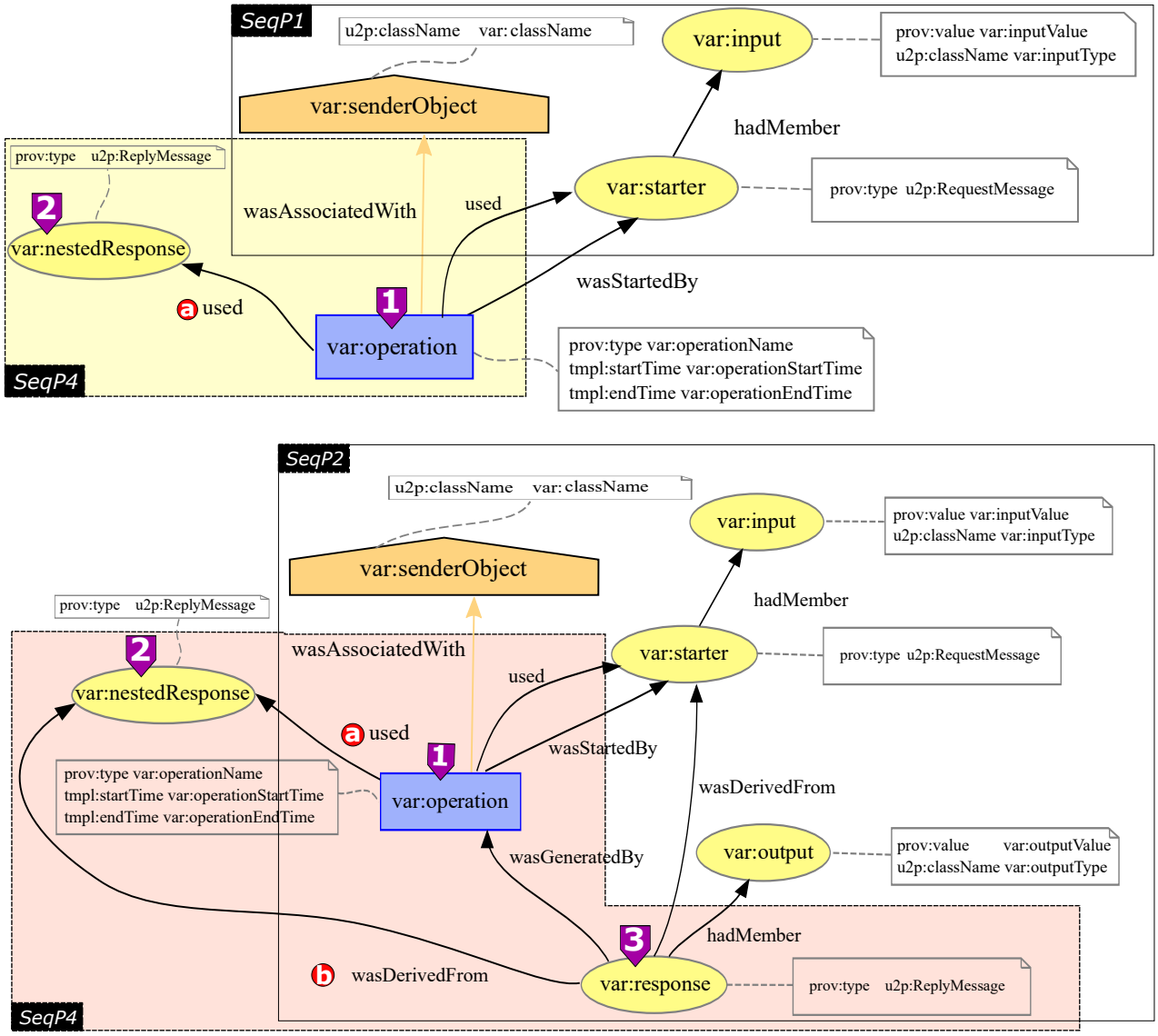

Figure 8. At the top, it is the PROV template generated from the UML representation in the left side of Figure 7. At the bottom, it is a PROV template generated from the UML representation in the right side of Figure 7. Only the shaded areas correspond to the PROV elements contributed by this pattern.

PROV elements

UML	PROV / id	Rationale
ExecutionSpecification 1	prov:Activity 1 / var:operation	The ExecutionSpecification 1 is a prov:Activity with identifier var:operation.
Reply Message 2	prov:Entity 2 / var:nestedResponse	The Reply Message 2 that is received in the ExecutionSpecification 1 is a prov:Entity with identifier var:nestedResponse.
Reply Message 3	prov:Entity 3 / var:response	In case of complementing *SeqP2*, the Reply Message 3 sent from the ExecutionSpecification 1 is a prov:Entity with identifier var:response. For details, see *SeqP2*.

Attributes

PROV Element	Attribute / Value	Description
var:operation 1	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 1.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 1.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 1.
var:nestedResponse 2	prov:type / u2p:ReplyMessage	The value u2p:ReplyMessage shows that var:response 2 is a reply message.
var:response 3	prov:type / u2p:ReplyMessage	The value u2p:ReplyMessage shows that var:response 3 is a reply message.

PROV relations

a prov:used	It is the beginning of utilizing var:nestedResponse by var:operation.
b prov:wasDerivedFrom	It is the construction of var:response based on var:nestedResponse.

Discussion

- As could be inferred from the context, a requirement for this pattern to be applied is that the main operation uses the nested response during its execution. This causes the relations a prov:used and b prov:wasDerivedFrom to appear in the template; the former showing that when the ExecutionSpecification 1 receives the nested Reply Message 2, it *utilises* that Reply Message 2 to complete its behaviour; and the latter showing that the main Reply Message 3 is *influenced* by the nested Reply Message 2 (this last one can only be applied if the main operation execution is triggered by a synchronous message, i.e. when *SeqP4* complements *SeqP2*). If an specific scenario does not meet the abovementioned requirement, i.e., the main operation does not use the nested response or it is not worth recording such dependency, this pattern should not be applied. Even in this case, the merging of the provenance resulting after expanding all the templates will contain provenance for the nested operation call and its corresponding response, though lacking the information regarding the relationships a prov:used and b prov:wasDerivedFrom.

UML State Machine Diagrams

Pattern identifier	Context	Page
StP1	As a consequence of the execution of an operation an object is created and it immediately reaches its first state. This operation is usually the constructor of the object.	17
StP2	As a consequence of the execution of an operation, the behaviour of an object is completed.	20
StP3	As a consequence of the execution of an operation, an object changes its state.	24

Identifier *St*ate machine diagram *P*attern *1* (*StP1*)

Context

As a consequence of the execution of an operation an object is created and it immediately reaches its first state. This operation is usually the constructor of the object.

Key elements

Object — It is the created object.

First object's state — The state immediately reached after the object creation. This is the first state the object may undergo in its lifetime.

Object creation — It refers to the execution of the operation that creates the object.

UML Diagram

Key Element	UML	Rationale
Object	`Object` 1	It represents the created object. *Note*: since `Object` lacks a graphical representation in UML State Machine diagrams, Figure 9 does not depict this element.
	`StateMachine` 2	In UML, a `StateMachine` can be used to express the set of states through which the *Object* might go during its lifetime in response to events.
Object creation	`Initial Pseudostate` 3	It refers to the execution of the operation that creates the *Object*, leading immediately to its first state.
First object's state	`State` 4	It models the first state of the *Object*.

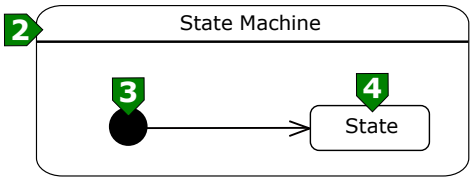

Figure 9. UML representation that models the context given by *StP1*

Mapping to PROV

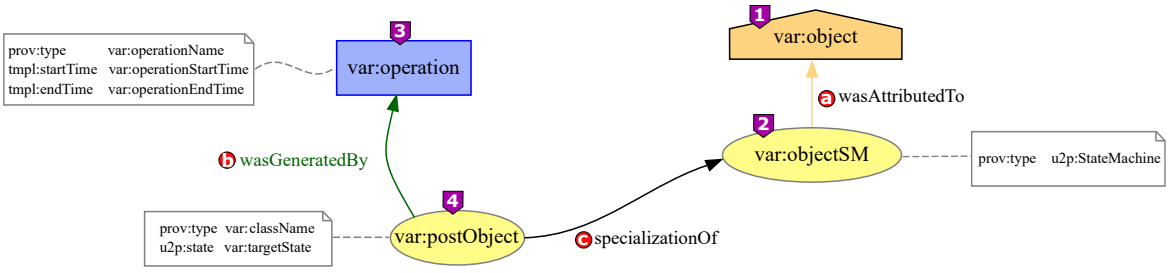

Figure 10. PROV template generated from the UML representation in Figure 9

PROV elements

UML	PROV / id	Rationale
Object 1	prov:Agent 1 / var:object	The Object 1 is mapped to a prov:Agent identified by var:object which bears the responsibilities of the Object 1.
StateMachine 2	prov:Entity 2 / var:objectSM	The StateMachine 2 is a prov:Entity identified by var:objectSM. It reflects the abstraction of the object's states, which will be specialized by each state the object goes through.
Initial Pseudostate 3	prov:Activity 3 / var:operation	The Initial Pseudostate 3, referring to the execution of the operation that creates the Object 1, is a prov:Activity with the identifier var:operation.
State 4	prov:Entity 4 / var:postObject	The State 4 is a prov:Entity identified by var:postObject. We use this name for this identifier because it corresponds to the state of the Object 1 after (post) the object creation.

Attributes

PROV Element	Attribute / Value	Description
var:objectSM 2	prov:type / u2p:StateMachine	The value u2p:StateMachine shows that var:objectSM 2 is a state machine.
var:operation 3	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 3
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 3.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 3.
var:postObject 4	prov:type / var:className	The value var:className is the name of the class to which the object in the state var:postObject 4 belongs.
	u2p:state / var:targetState	The value var:targetState is a string with the name of the state var:postObject 4.

PROV relations

a	prov:wasAttributedTo	It is the assignment of responsibility to var:object for var:objectSM.
b	prov:wasGeneratedBy	It is the completion of production of var:postObject by var:operation.
c	prov:specializationOf	var:postObject is a specialization of var:objectSM.

Discussion

- Note that Figure 9 only contains simple states. We have decided to deal with neither composite nor submachine states, and focus only on simple states, because the former may be transformed into the latter by resorting to a flattening process consisting of removing composite states as well as submachine states. In fact, to flatten State Machine diagrams is a very common approach in contexts such as model checking and code generation [6]. However, the user might be interested in representing composite states directly into the PROV templates, perhaps because she/he is interested in collecting information about them, or just because she/he does not want to flatten the State Machine diagram. We can give an insight into how composite states can be mapped to PROV by placing the elements from Figure 9 inside a Composite State 5 (see Figure 11). A reader familiar with the UML specification will realize that the semantics of the Initial Pseudostate 3 in Figures 9 and 11 are different, but these semantics nuances would have no effect on the PROV transformation. The transformation of Figure 11 is shown in Figure 12. Both Figure 11 and 12 highlight the added elements by blurring the elements coming from Figure 9 and Figure 10, respectively. Briefly speaking, the new Composite State 5 is translated into a prov:Entity identified by var:compState 5, which is associated with var:objectSM 2 and var:targetState 4 by means of the relations prov:specializationOf and prov:hadMember, respectively. At this point, it is also worth remarking that for this example we have used a *simple composite state* (i.e., Composite State 5),

which means that only one substate is active at a given time within such a state; but we could have used *orthogonal composite states* instead, which means that within such a state several substates are active at the same time. Note that both types of *composite states* would be translated into the same PROV template (see Figure 12); nevertheless, the generated bindings would be different. In case of a *simple composite state*, as there can be only one active substate at the same time, there would be only one value associated with the variable `var:postObject`. Conversely, in case of an *orthogonal composite state*, `var:postObject` will be associated with several values (as many as active states).

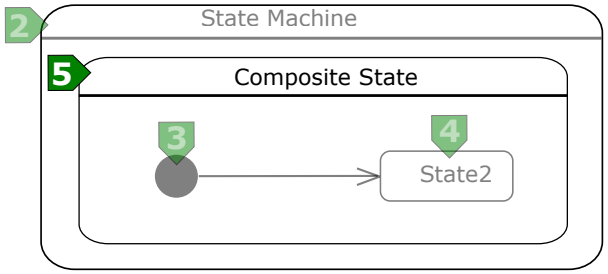

Figure 11. Excerpt of a UML State Machine diagram locating the UML elements from *StP1* in a *simple composite state*.

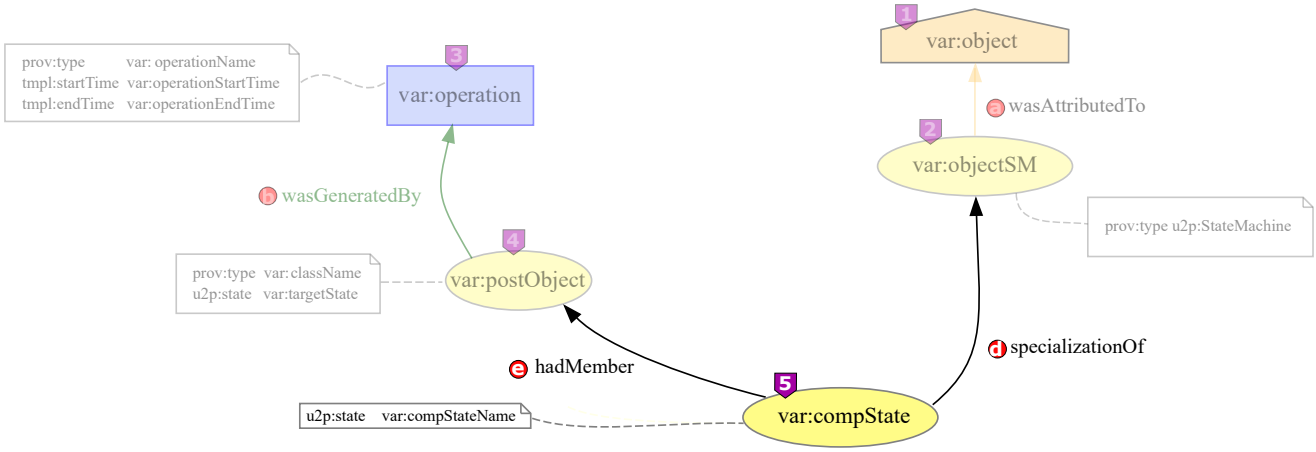

Figure 12. PROV template generated from the UML diagram in Figure 11

PROV elements

UML	PROV / id	Rationale
`Composite State` 5	`prov:Entity` 5 / `var:compState`	The `Composite State` 5 is a `prov:Entity` identified by `var:compState`.

Attributes

PROV Element	Attribute / Value	Description
`var:compState` 5	`u2p:state` / `var:compStateName`	The value `var:compStateName` is a string with the name of the state `var:compState` 5

PROV relations

d	`prov:specializationOf`	`var:compState` is a specialization of `var:objectSM`.
e	`prov:hadMember`	It states that `var:postObject` is one of the elements in `var:compState`.

Identifier *St*ate machine diagram *P*attern *2* (*StP2*)

Context

As a consequence of the execution of an operation, the behaviour of an object is completed.

Key elements

Object	It is the object that completes its behaviour.	
	Pre-operation object's state	The state of the object before the execution of the operation. This is one of the states the object may undergo during its lifetime.
	Final object's state	The state that represents that the object's behaviour is completed.
Operation execution	It is the execution of the operation that leads to the completion of the object's behaviour.	

UML Diagram

Key Element	UML	Rationale
Object	`Object` 1	It represents the object whose behaviour is completed. *Note*: since `Object` lacks a graphical representation in UML State Machine diagrams, Figure 13 does not depict this element.
	`StateMachine` 2	In UML, a `StateMachine` can be used to express the set of states through which the *Object* goes during its lifetime in response to events.
Pre-operation object's state	`State` 3	It models the state of the *Object* before the *Operation execution*.
Final object's state	`FinalState` 4	It models the state of the *Object* after the *Operation execution*.
Operation execution	`Event` 5	It specifies that the *Operation execution* that triggers the switch of states has taken place.

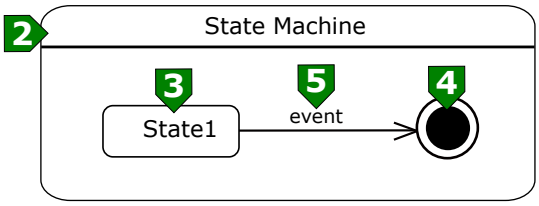

Figure 13. UML representation that models the context given by *StP2*

Mapping to PROV

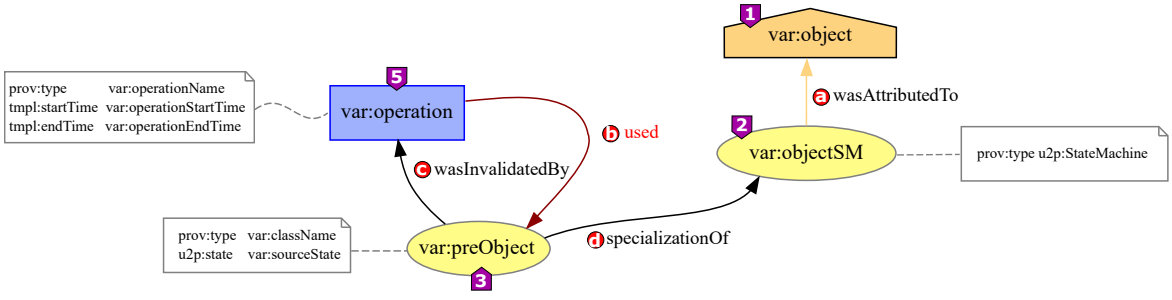

Figure 14. PROV template generated from the UML representation in Figure 13

PROV elements

UML	PROV / id	Rationale
Object 1	prov:Agent 1 / var:object	The Object 1 is mapped to a prov:Agent identified by var:object which bears the responsibilities of the Object 1.
StateMachine 2	prov:Entity 2 / var:objectSM	The StateMachine 2 is a prov:Entity identified by var:objectSM. It reflects the abstraction of the object's states, which will be specialized by each state the object goes through.
State 3	prov:Entity 3 / var:preObject	The State 3 is a prov:Entity identified by var:preObject. We use this name for this identifier because it corresponds to the state of the Object 1 before (pre) the execution of the operation.
FinalState 4	None /	Without mapping (see the discussion block for an explanation about this decision).
Event 5	prov:Activity 5 / var:operation	The Event 5 represents that the execution of an operation has taken place. Such an execution is a prov:Activity with the identifier var:operation.

Attributes

PROV Element	Attribute / Value	Description
var:objectSM 2	prov:type / u2p:StateMachine	The value u2p:StateMachine shows that var:objectSM 2 is a state machine.
var:preObject 3	prov:type / var:className	The value var:className is the name of the class to which the object in the state var:preObject 3 belongs.
	u2p:state / var:sourceState	The value var:sourceState is a string with the name of the state var:preObject 3.
var:operation 5	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 5.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 5.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 5.

PROV relations

a	prov:wasAttributedTo	It is the assignment of responsibility to var:object for var:objectSM.
b	prov:used	It is the beginning of utilizing var:preObject by var:operation.
c	prov:wasInvalidatedBy	It shows that var:preObject is not longer available for use.
d	prov:specializationOf	var:preObject is a specialization of var:objectSM.

Discussion

- We have decided not to map the FinalState in the PROV template due to the fact that its UML semantics (the completion of the object's behaviour) is already included in the PROV template. Concretely, it is given by the relation prov:wasInvalidatedBy showing that var:preObject is not longer available. This pattern is consistent with *ClP2* because the completion of the object's behaviour usually involves its destruction.
- Note that Figure 13 only contains simple states. We have decided to deal with neither composite nor submachine states, and focus only on simple states, because the former may be transformed into the latter by resorting to a flattening process consisting of removing composite states as well as submachine states. In fact, to flatten State Machine diagrams is a very common approach in contexts such as model checking and code generation [6]. However, the user might be interested in representing composite states directly into the PROV templates, perhaps because she/he is interested in collecting information

about them, or just because she/he does not want to flatten the State Machine diagram. We can give an insight into how composite states can be mapped to PROV by placing the elements from Figure 13 inside a `Composite State` 5 (see Figure 15). A reader familiar with the UML specification will realize that the semantics of the `FinalState` 4 in Figures 13 and 15 are different, but these semantics nuances would have no effect on the PROV transformation. The transformation of Figure 15 is shown in Figure 16. Both Figure 15 and 16 highlight the added elements by blurring the elements coming from Figure 13 and Figure 14, respectively. Briefly speaking, the new `Composite State` 6 is translated into a `prov:Entity` identified by `var:compState` 6, which is associated with `var:objectSM` 2 and `var:preObject` 3 by means of the relations `prov:specializationOf` and `prov:hadMember`, respectively. At this point, it is also worth remarking that for this example we have used a *simple composite state* (i.e., `Composite State` 6), which means that only one substate is active at a given time within such a state; but we could have used *orthogonal composite states* instead, which means that within such a state several substates are active at the same time. Note that both types of *composite states* would be translated into the same PROV template (see Figure 16); nevertheless, the generated bindings would be different. In case of a *simple composite state*, as there can be only one active substate at the same time, there would be only one value associated with the variable `var:preObject`. Conversely, in case of an *orthogonal composite state*, `var:preObject` will be associated with several values (as many as active states).

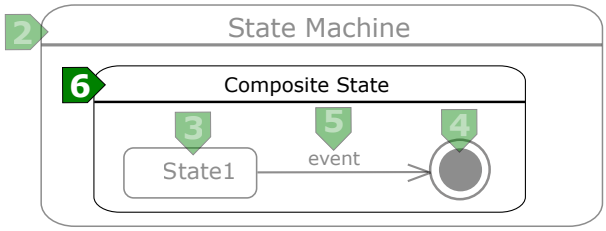

Figure 15. Excerpt of a UML State Machine diagram locating the UML elements from *StP2* in a *simple composite state*.

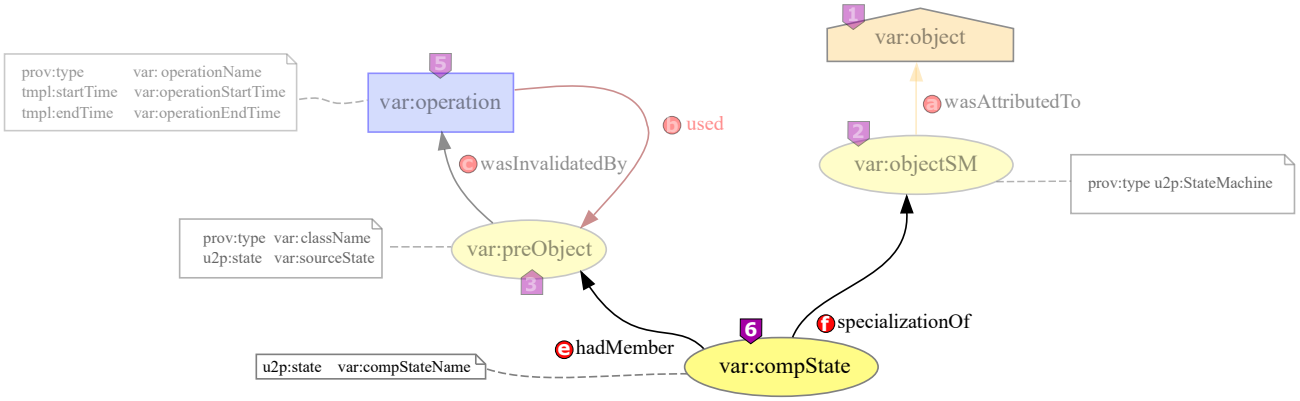

Figure 16. PROV template generated from the UML diagram in Figure 15

PROV elements

UML	PROV / id	Rationale
`Composite State` 6	`prov:Entity` 6 / `var:compState`	The `Composite State` 6 is a `prov:Entity` identified by `var:compState`.

Attributes

PROV Element	Attribute / Value	Description
`var:compState` 6	`u2p:state` / `var:compStateName`	The value `var:compStateName` is a string with the name of the state `var:compState` 6

PROV relations

prov:hadMember	It states that var:preObject is one of the elements in var:compState.
prov:specializationOf	var:compState is a specialization of var:objectSM.

Identifier *St*ate machine diagram *P*attern *3* (*StP3*)

Context

As a consequence of the execution of an operation, an object changes its state.

Key elements

Object — It is the object that changes its state.

- *Pre-operation object's state* — The state of the object before the execution of the operation. This is one of the states the object may undergo during its lifeline.
- *Post-operation object's state* — The state of the object after the execution of the operation. This is one of the states the object may undergo during its lifeline.

Operation execution — It is the execution of the operation that leads to the switch of state.

UML Diagram

Key Element	UML	Rationale
Object	`Object` 1	It represents the object that changes its state. *Note*: since `Object` lacks a graphical representation in UML State Machine diagrams, Figure 13 does not depict this element.
	`StateMachine` 2	In UML, a `StateMachine` can be used to express the set of object's states through which the *Object* goes during its lifetime in response to events.
Pre-operation object's state	`State` 3	It models the state of the *Object* before the *Operation execution*.
Post-operation object's state	`State` 4	It models the state of the *Object* after the *Operation execution*.
Operation execution	`Event` 5	It specifies that the *Operation execution* that triggers the switch of states has taken place.

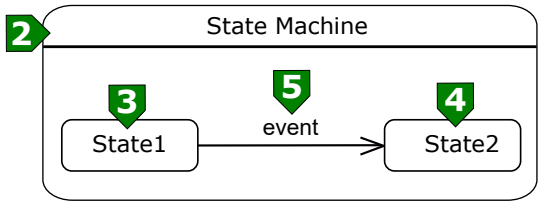

Figure 17. UML representation that models the context given by *StP3*

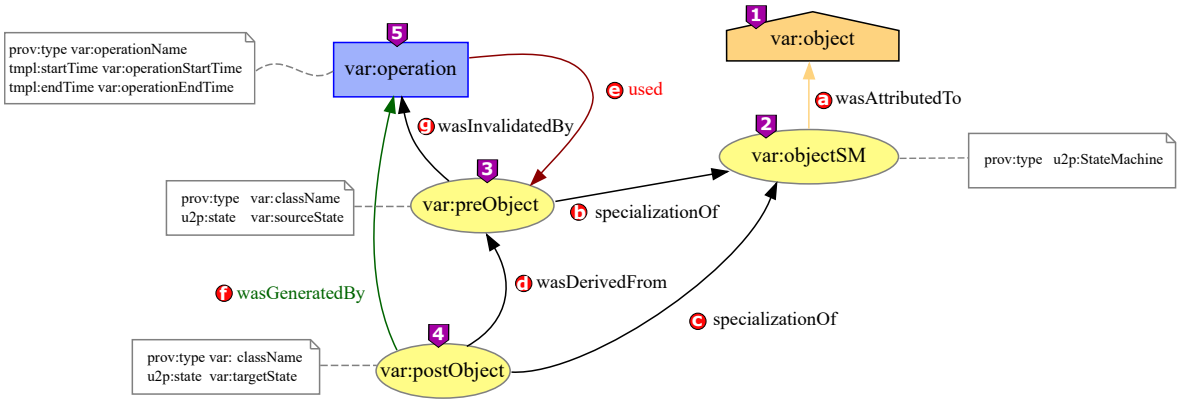

Figure 18. PROV template generated from the UML representation in Figure 17

PROV elements

UML	PROV / id	Rationale
`Object` 1	`prov:Agent` 1 / `var:object`	The `Object` 1 is mapped to a `prov:Agent` identified by `var:object` which bears the *Object*'s responsibilities.
`StateMachine` 2	`prov:Entity` 2 / `var:objectSM`	The `StateMachine` 2 is a `prov:Entity` identified by `var:objectSM`. It reflects the abstraction of the object's states, which will be specialized by each state the object goes through.
`State` 3	`prov:Entity` 3 / `var:preObject`	The `State` 3 is a `prov:Entity` identified by `var:preObject`. We use this name for this identifier because it corresponds to the state of the `Object` 1 before (pre) the execution of the operation.
`State` 4	`prov:Entity` 4 / `var:postObject`	The `State` 4 is a `prov:Entity` identified by `var:postObject`. We use this name for this identifier because it corresponds to the state of the `Object` 1 after (post) the execution of the operation.
`Event` 5	`prov:Activity` 5 / `var:operation`	The `Event` 5 represents that the execution of an operation has taken place. Such an execution is a `prov:Activity` with the identifier `var:operation`.

Attributes

PROV Element	Attribute / Value	Description
var:objectSM 2	prov:type / u2p:StateMachine	The value u2p:StateMachine shows that var:objectSM 2 is a state machine.
var:preObject 3	prov:type / var:className	The value var:className is the name of the class to which the object in the state var:preObject 3 belongs.
	u2p:state / var:sourceState	The value var:sourceState is a string with the name of the state var:preObject 3.
var:postObject 4	prov:type / var:className	The value var:className is the name of the class to which the object in the state var:postObject 4 belongs.
	u2p:state / var:targetState	The value var:targetState is a string with the name of the state var:postObject 4.
var:operation 5	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 5.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 5.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 5.

PROV relations

a	prov:wasAttributedTo	It is the assignment of responsibility to var:object for var:objectSM.
b	prov:specializationOf	var:preObject is a specialization of var:objectSM.
c	prov:specializationOf	var:postObject is a specialization of var:objectSM.
d	prov:wasDerivedFrom	It is the update of var:preObject resulting in var:postObject.
e	prov:used	It is the beginning of utilizing var:preObject by var:operation.
f	prov:wasGeneratedBy	It is the completion of production of var:postObject by var:operation.
g	prov:wasInvalidatedBy	It shows that var:preObject is not longer available for use.

Discussion

- Note that Figure 17 only contains simple states. We have decided to deal with neither composite nor submachine states, and focus only on simple states, because the former may be transformed into the latter by resorting to a flattening process consisting of removing composite states as well as submachine states. In fact, to flatten State Machine diagrams is a very common approach in contexts such as model checking and code generation [6]. However, the user might be interested in representing composite states directly into the PROV templates, perhaps because she/he is interested in collecting information about them, or just because she/he does not want to flatten the State Machine diagram. We can give an insight into how composite states can be mapped to PROV by placing the elements from Figure 17 inside a Composite State 5 (see Figure 19). A reader familiar with the UML specification will realize that the semantics of the UML representation in Figures 17 and 19 are different, but these semantics nuances would have no effect on the PROV transformation. The transformation of Figure 19 is shown in Figure 20. Both Figure 19 and 20 highlight the added elements by blurring the elements coming from Figure 17 and Figure 18, respectively. Briefly speaking, the new Composite State 6 is translated into a prov:Entity identified by var:compState 6, which is associated with var:objectSM 2, var:preObject 3, and var:postObject 3 by means of the relations prov:specializationOf, prov:hadMember, and prov:hadMember, respectively. At this point, it is also worth remarking that for this example we have used a *simple composite state* (i.e., Composite State 6), which means that only one substate is active at a given time within such a state; but we could have used *orthogonal composite states* instead, which means that within such a state several substates are active at the same time. Note that both types of *composite states* would be translated into the same PROV template (see Figure 20); nevertheless, the generated bindings would be different. In case of a *simple composite state*, as there can be only one active substate at the same time, there would be only one value associated with the variable var:preObject and another value with var:postObject. Conversely, in

case of an *orthogonal composite state*, `var:preObject` and `var:postObject` will be associated with several values (as many as active states).

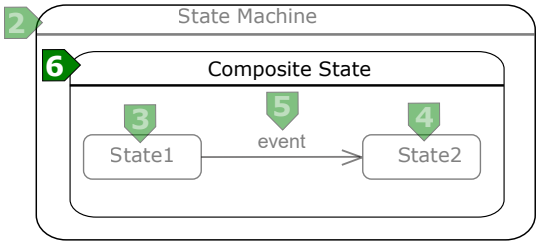

Figure 19. Excerpt of a UML State Machine diagram locating the UML elements from *StP3* in a *simple composite state*.

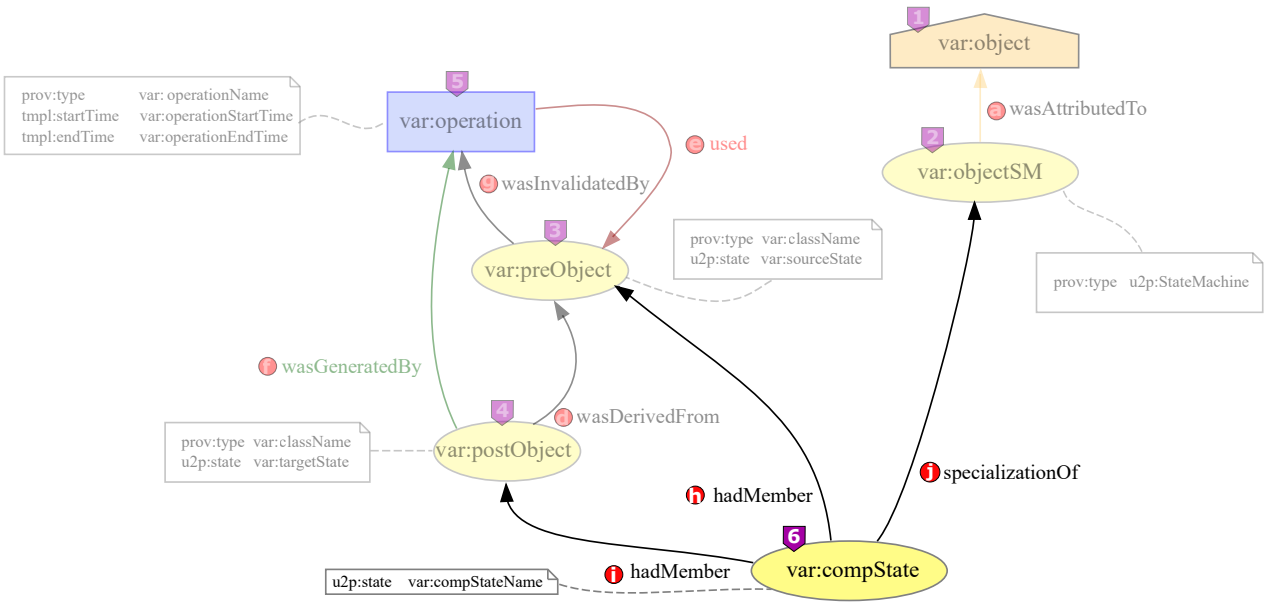

Figure 20. PROV template generated from the UML diagram in Figure 19

PROV elements

UML	PROV / id	Rationale
`Composite State` 6	`prov:Entity` 6 / `var:compState`	The `Composite State` 6 is a `prov:Entity` identified by `var:compState`.

Attributes

PROV Element	Attribute / Value	Description
`var:compState` 6	`u2p:state` / `var:compStateName`	The value `var:compStateName` is a string with the name of the state `var:compState` 6

PROV relations

h	`prov:hadMember`	It states that `var:preObject` is one of the elements in `var:compState`.
i	`prov:hadMember`	It states that `var:postObject` is one of the elements in `var:compState`.
j	`prov:specializationOf`	`var:compState` is a specialization of `var:objectSM`.

UML Class Diagrams

Pattern identifier	Context	Page
ClP1	The execution of an operation provokes the creation of a new object.	29
ClP2	The execution of an operation provokes the destruction of an object.	31
ClP3	The execution of an operation on an object returns the values of concrete object's attributes. This execution does not provoke the change of the object's status. The values are returned as they are, without any further processing.	33
ClP4	The execution of an operation on an object returns values that are computed based on the object's status as a whole (the values of concrete attributes involved in such a computation are unknown or irrelevant). This execution does not provoke the change of the object's status.	36
ClP5	The execution of an operation on an object returns values that are computed based on values of concrete object's attributes. This execution does not provoke the change of the object's status.	39
ClP6	The execution of an operation on an object changes the object's status as a whole (the concrete modified attributes are unknown or irrelevant).	43
ClP7	The execution of an operation on an object directly sets the information passed to the operation as values of concrete object's attributes, thus provoking a change in the object's status.	48
ClP8	The execution of an operation on an object changes the values of concrete object's attributes, thus provoking a change in the object's status.	53
ClP9	The execution of an operation on an object removes element(s) from a concrete object's collection attribute, thus provoking a change in the object's status.	58
ClP10	The execution of an operation on an object directly adds the information passed to the operation as new element(s) of a concrete object's collection attribute, thus provoking a change in the object's status.	63

Identifier *C*lass diagram *P*attern *1* (*ClP1*)

Context

The execution of an operation provokes the creation of a new object.

Key elements

Object	It is the object created as a consequence of the execution of the operation.
Operation execution	It is the execution of the operation.
Input data	It specifies the information (if any) passed into the *Operation execution*.
Object's attributes	They are all the characteristics that conform the *Object*.

UML Diagram

Key Element	UML	Rationale
Object	`Class` 1	*Objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use `Class` 1 to represent the *Object* after the execution of the operation.
Operation execution	`Operation` 2 «`create`»	The `Operation` 2 stereotyped by «`create`» represents the executed operation that creates the *Object*.
Input data	`Input Parameters` 3	They specify the information passed into the *Operation execution*.
Object's attributes	`Attributes` 4	They represent the characteristics of the *Object*.

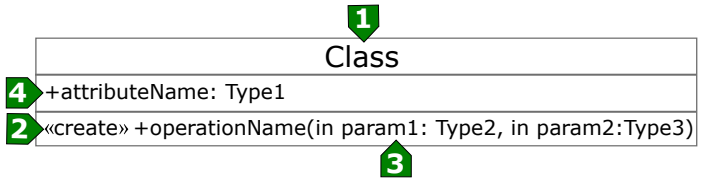

Figure 21. UML representation that models the context given by *ClP1*

Mapping to PROV

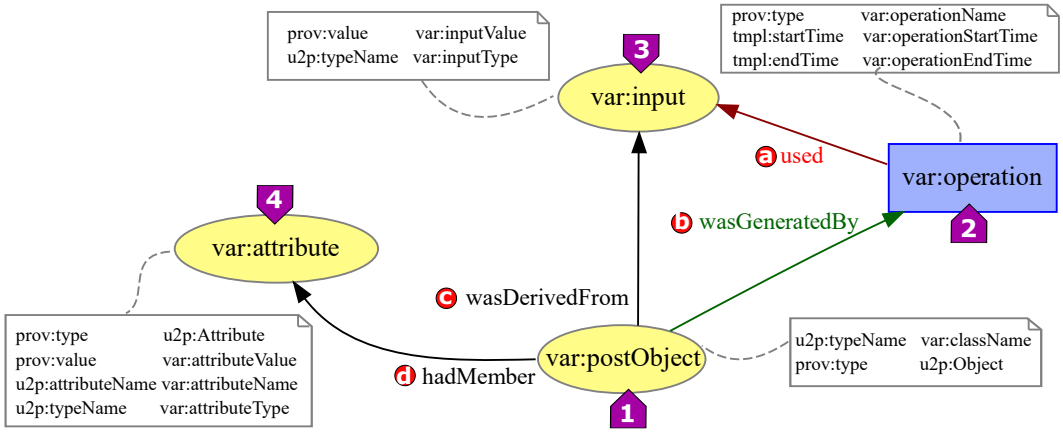

Figure 22. PROV template generated from the UML representation in Figure 21

PROV elements

UML	PROV / id	Rationale
Class 1	prov:Entity 1 / var:postObject	The Class 1 that models the object that is created by the operation is a prov:Entity identified as var:postObject. We use the prefix *post* in this identifier because the object is the result of the executed operation.
Operation 2 «create»	prov:Activity 2 / var:operation	The execution of Operation 2 stereotyped by «create» is a prov:Activity identified by var:operation.
Input Parameters 3	prov:Entity 3 / var:input	Each parameter of Input Parameters 3 is a separate prov:Entity identified as var:input.
Attributes 4	prov:Entity 4 / var:attribute	Each attribute of Attributes 4 is a separate prov:Entity with identifier var:attribute.

Attributes

PROV Element	Attribute / Value	Description
var:postObject 1	u2p:typeName / var:className	The value var:className is the name of the class to which var:postObject 1 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:postObject is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.
var:input 3	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
	u2p:typeName / var:inputType	The value var:inputType is the string with the name of the type of var:input 3.
var:attribute 4	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:attribute 4 is an attribute.
	prov:value / var:attributeValue	The value var:attributeValue is the direct representation of var:attribute 4.
	u2p:attributeName / var:attributeName	The value var:attributeName is the name of var:attribute 4.
	u2p:typeName / var:attributeType	The value var:attributeType is the string with the name of the type of var:attribute 4.

PROV relations

a	prov:used	It is the beginning of utilizing var:input by var:operation.
b	prov:wasGeneratedBy	It is the completion of production of var:postObject by var:operation.
c	prov:wasDerivedFrom	It is the construction of var:postObject based on var:input.
d	prov:hadMember	It states that var:attribute is one of the elements in var:postObject.

Discussion

- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have decided to consider this circumstance with the aim of covering a wider spectrum of cases. When the operation that creates the object lacks *Input data*, the UML representation in Figure 21 will not include Input Parameters 3. As a consequence, the resulting PROV template in Figure 22 will also lack var:input 3 and its associated PROV relations.

Identifier *C*lass diagram *P*attern 2 (*ClP2*)

Context

The execution of an operation provokes the destruction of an object.

Key elements

Object	It is the object destroyed as a consequence of the execution of the operation.
Operation execution	It is the execution of the operation.

UML Diagram

Key Element	UML	Rationale
Object	`Class` 1	*Objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use `Class` 1 to represent the *Object* that is about to be destroyed.
Operation execution	`Operation` 2 «`destroy`»	The `Operation` 2 stereotyped by «`destroy`» represents the executed operation that destroys the *Object*.

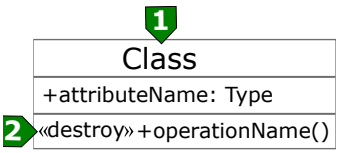

Figure 23. UML representation that models the context given by *ClP2*

Mapping to PROV

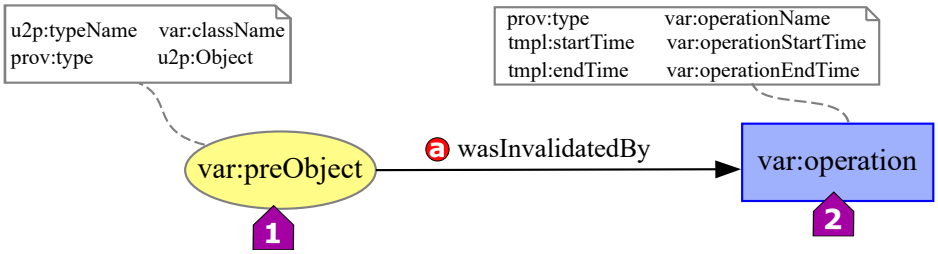

Figure 24. PROV template generated from the UML representation in Figure 23

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1 / `var:preObject`	The `Class` 1 that models the object that is destroyed by the operation is a `prov:Entity` identified as `var:preObject`. We use the prefix *pre* in this identifier because it is the object before the execution of the operation.
`Operation` 2 «`destroy`»	`prov:Activity` 2 / `var:operation`	The execution of `Operation` 2 stereotyped by «`destroy`» is a `prov:Activity` identified by `var:operation`.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1	u2p:typeName / var:className	The value var:className is the name of the class to which var:preObject 1 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject 1 is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.

PROV relations

prov:wasInvalidatedBy	It shows that var:preObject is not longer available for use.

Discussion

- This pattern is consistent with *ClP2* because the completion of the object's behaviour usually involves its destruction.

Identifier *C*lass diagram *P*attern *3* (*ClP3*)

Context

The execution of an operation on an object returns the values of concrete object's attributes. This execution does not provoke the change of the object's status. The values are returned as they are, without any further processing.

Key elements

Object	It is the object on which the operation is executed.
Operation execution	It is the execution of the operation.
Input data	It specifies the information (if any) passed into the *Operation execution*.
Output data	It is the information obtained from the *Operation execution*.

UML Diagram

Key Element	UML	Rationale
Object	`Class` 1	*Objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use `Class` 1 to represent the *Object* on which the operation is executed.
Operation execution	`Operation` 2 «`get`»/«`search`»	The `Operation` 2 stereotyped by «`get`»/«`search`» represents the executed operation. Concretely, operations stereotyped by «`get`» return values of concrete *Object*'s attributes, whereas «`search`» is used when the operation returns elements belonging to a collection attribute of the *Object*.
Input data	`Input Parameters` 3	They specify the information passed into the *Operation execution*.
Output data	`Output Parameters` 4	They depict the information obtained from the *Operation execution*.

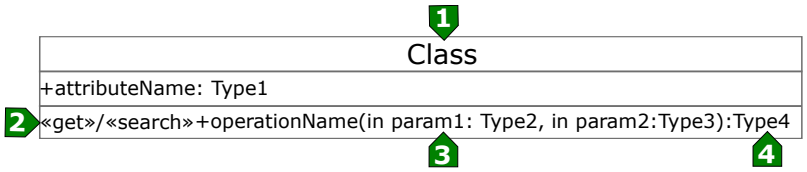

Figure 25. UML representation that models the context given by *ClP3*

Mapping to PROV

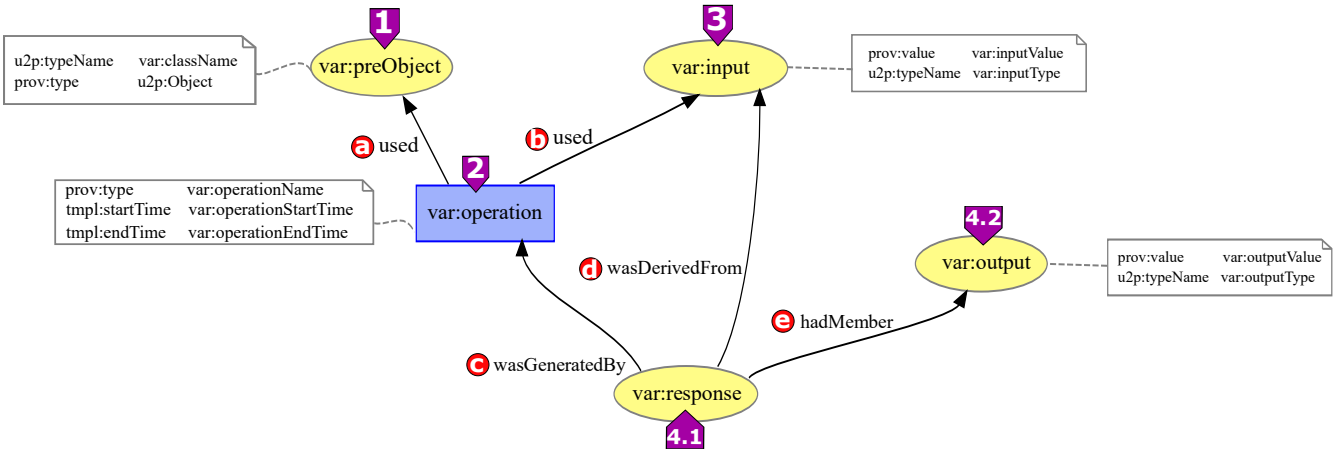

Figure 26. PROV template generated from the UML representation in Figure 25

PROV elements

UML	PROV / id	Rationale
Class 1	prov:Entity 1 / var:preObject	The Class 1 that models the object on which the operation is executed is a prov:Entity identified as var:preObject. We use the prefix *pre* in this identifier because it is the object before the execution of the operation.
Operation 2 «get»/«search»	prov:Activity 2 / var:operation	The execution of Operation 2 stereotyped by «get»/«search» is a prov:Activity identified by var:operation.
Input Parameters 3	prov:Entity 3 / var:input	Each parameter of Input Parameters 3 is a separate prov:Entity identified as var:input.
Output Parameters 4	prov:Entity 4.1 / var:response	The information obtained by the execution of the operation is a prov:Entity identified by var:response. See the discussion block for an explanation about the existence of var:response.
	prov:Entity 4.2 / var:output	Each parameter of Output Parameters 4 is a separate prov:Entity identified as var:output.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1	u2p:typeName / var:className	The value var:className is the name of the class to which var:preObject 1 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject 1 is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the operation Operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.
var:input 3	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
	u2p:typeName / var:inputType	The value var:inputType is a string with the name of the type of var:input 3.
var:output 4.2	prov:value / var:outputValue	The value var:outputValue is the direct representation of var:output 4.2.
	u2p:typeName / var:outputType	The value var:outputType is a string with the name of the type of var:output 4.2.

PROV relations

a	prov:used	It is the beginning of utilizing var:preObject by var:operation.
b	prov:used	It is the beginning of utilizing var:input by var:operation.
c	prov:wasGeneratedBy	It is the completion of production of var:response by var:operation.
d	prov:wasDerivedFrom	It is the construction of var:response based on var:input.
e	prov:hadMember	It states that var:output is one of the elements in var:response.

Discussion

- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have decided to consider this circumstance with the aim of covering a wider spectrum of cases. When the executed operation

lacks *Input data*, the UML representation in Figure 25 will not include `Input Parameters`. As a consequence, the resulting PROV template in Figure 26 will also lack `var:input` and its associated PROV relations.

- In order to homogenise the UML Class representations, in *ClP1*-*ClP10*, *Output data* have been specified by `Output Parameters` with *return* direction. Nevertheless, these `Output Parameters` could have been modelled with *inout* and *out* direction, without affecting the transformation into PROV.

- A concrete nuance in this pattern is that the *Output data* (`var:output`) are not computed by the *Operation execution* (`var:operation`); that is, these data already existed before the *Operation execution*. Thus, a `prov:wasGeneratedBy` relation between `var:output` and `var:operation` does not make sense in this pattern. To reflect this pattern's nuance in the PROV template and taking into account the consistency between the different kinds of patterns, we have been inspired by how UML sequence diagram patterns (e.g., *SeqP2*) address the *Output data*. We have made this decision because the semantics of the sequence diagrams patterns (in terms of *Output data*) bears a strong resemblance with this pattern. Thus, we have included a `prov:Entity` identified by `var:response` related to (1) `var:operation` by means of `prov:wasGeneratedBy` (to represent the fact that it is the response who is generated by the *Operation execution*), and (2) `var:output` through `prov:hadMember` (to show that such a response is composed by the concrete output values).

Identifier *C*lass diagram *P*attern *4* (*ClP4*)

Context

The execution of an operation on an object returns values that are computed based on the object's status as a whole (the values of concrete attributes involved in such a computation are unknown or irrelevant). This execution does not provoke the change of the object's status.

Key elements

Object	It is the object on which the operation is executed.
Operation execution	It is the execution of the operation.
Input data	It specifies the information (if any) passed into the *Operation execution*.
Output data	It is the information obtained from the *Operation execution*.

UML Diagram

Key Element	UML	Rationale
Object	`Class` 1	*Objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use `Class` 1 to represent the *Object* on which the operation is executed.
Operation execution	`Operation` 2 «`process`»	The `Operation` 2 stereotyped by «`process`» represents the executed operation. Concretely, operations stereotyped by «`process`» return values that are computed based on the object's status as a whole.
Input data	`Input Parameters` 3	They specify the information passed into the *Operation execution*.
Output data	`Output Parameters` 4	They depict the information obtained from the *Operation execution*.

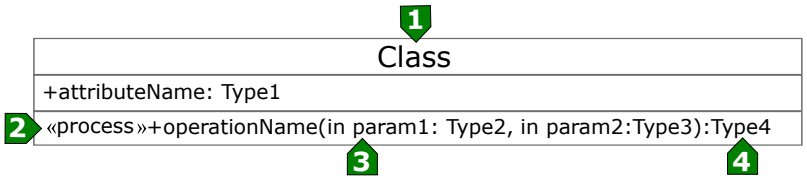

Figure 27. UML representation that models the context given by *ClP4*

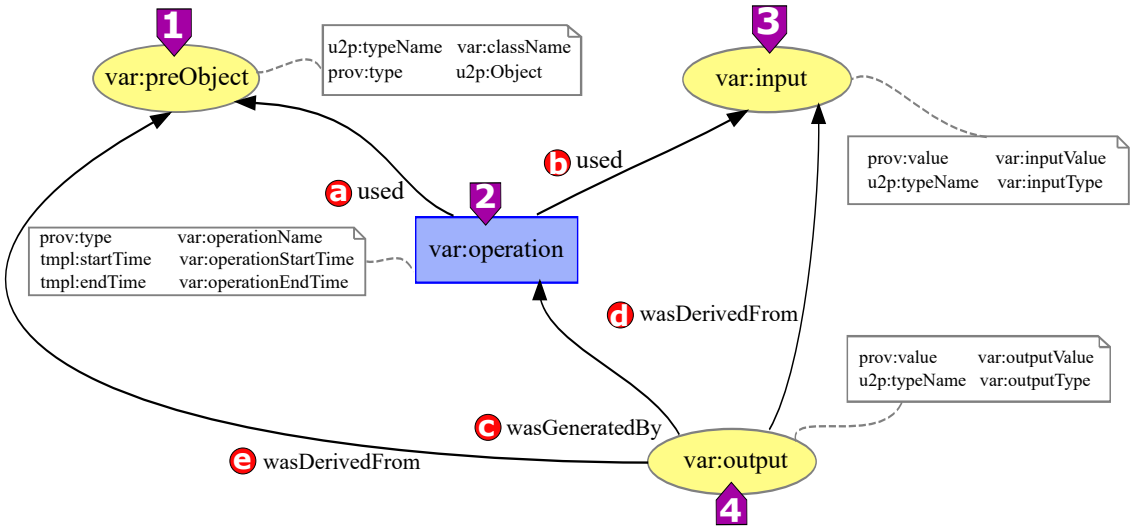

Figure 28. PROV template generated from the UML representation in Figure 27

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1 / `var:preObject`	The `Class` 1 that models the object on which the operation is executed is a `prov:Entity` identified as `var:preObject`. We use the prefix *pre* in this identifier because it is the object before the execution of the operation.
`Operation` 2 «`process`»	`prov:Activity` 2 / `var:operation`	The execution of `Operation` 2 stereotyped by «`process`» is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entity` 3 / `var:input`	Each parameter of `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Output Parameters` 4	`prov:Entity` 4 / `var:output`	Each parameter of `Output Parameters` 4 is a separate `prov:Entity` identified as `var:output`.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1	u2p:typeName / var:className	The value var:className is the name of the class to which var:preObject 1 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject 1 is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the operation Operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.
var:input 3	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
	u2p:typeName / var:inputType	The value var:inputType is a string with the name of the type of var:input 3.
var:output 4	prov:value / var:outputValue	The value var:outputValue is the direct representation of var:output 4.
	u2p:typeName / var:outputType	The value var:outputType is a string with the name of the type of var:output 4.

PROV relations

a	prov:used	It is the beginning of utilizing var:preObject by var:operation.
b	prov:used	It is the beginning of utilizing var:input by var:operation.
c	prov:wasGeneratedBy	It is the completion of production of var:output by var:operation.
d	prov:wasDerivedFrom	It is the construction of var:output based on var:input.
e	prov:wasDerivedFrom	It is the construction of var:output based on var:preObject.

Discussion

- In order to homogenise the UML Class representations, in *ClP1*-*ClP10*, *Output data* have been specified by Output Parameters with *return* direction. Nevertheless, these Output Parameters could have been modelled with *inout* and *out* direction, without affecting the transformation into PROV.
- Among the Class Diagrams patterns, both *ClP4* and *ClP5* address the execution of an operation that returns values computed based on information included on an object. While *ClP4* considers the object's status as a whole (without considering the concrete attributes' values considered for its computation), *ClP5* identifies the concrete attributes used to compute such information. Thus, *ClP4* gives a coarser grained provenance than *ClP5*.
- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have decided to consider this circumstance with the aim of covering a wider spectrum of cases. When the executed operation lacks *Input data*, the UML representation in Figure 27 will not include Input Parameters 3. As a consequence, the resulting PROV template in Figure 28 will also lack var:input 3 and its associated PROV relations.

Identifier *C*lass diagram *P*attern *5* (*ClP5*)

Context

The execution of an operation on an object returns values that are computed based on values of concrete object's attributes. This execution does not provoke the change of the object's status.

Key elements

Object	It is the object on which the operation is executed.
Operation execution	It is the execution of the operation.
Input data	It specifies the information (if any) passed into the *Operation execution*.
Output data	It is the information obtained from the *Operation execution*.
Source Object's attributes	They are the concrete characteristics of the *Object* that are used to compute the *Output data*.

UML Diagram

Key Element	UML	Rationale
Object	`Class` 1	*Objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use `Class` 1 to represent the *Object* on which the operation is executed.
Operation execution	`Operation` 2 «predicate»/ «property»/ «void-accessor»	The `Operation` 2 stereotyped by «predicate»/ «property»/ «void-accessor» represents the executed operation. Each stereotype denotes a behaviour with specific nuances (see the discussion block); nevertheless, all of them process the *Output data* based on values of concrete object's attribute without modifying the object's status.
Input data	`Input Parameters` 3	They specify the information passed into the *Operation execution*.
Output data	`Output Parameters` 4	They depict the information obtained from the *Operation execution*.
Source Object's attributes	`Attributes` 5	They represent the characteristics of the *Object*, whose values are used to compute the *Output data*.

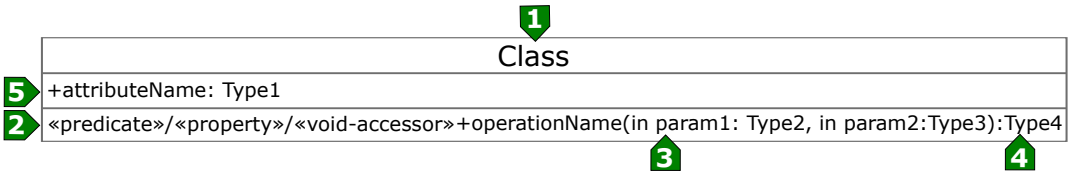

Figure 29. UML representation that models the context given by *ClP5*

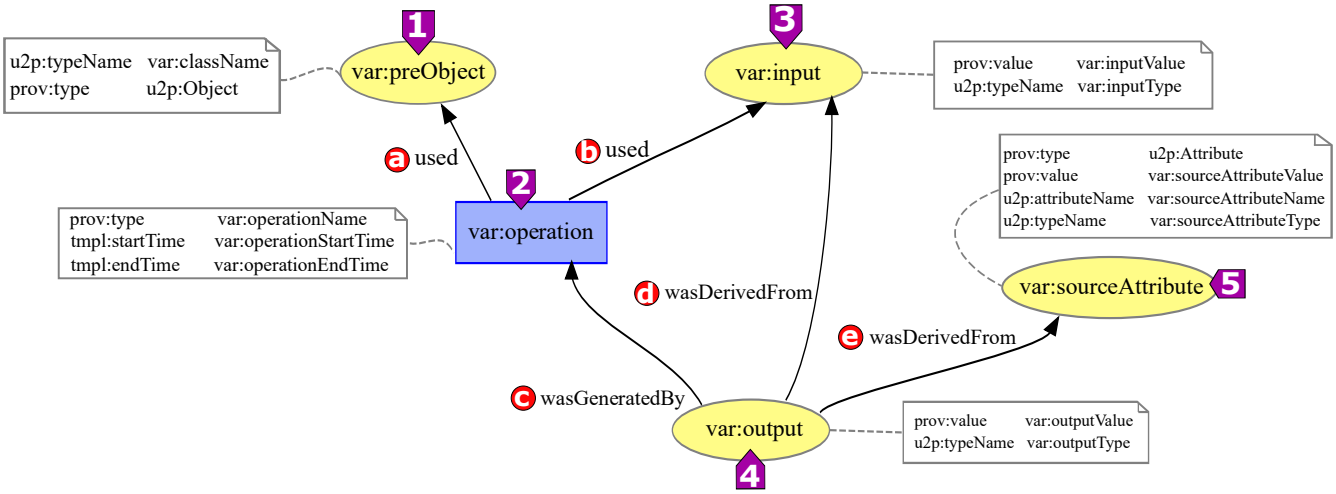

Figure 30. PROV template generated from the UML representation in Figure 29

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1 / `var:preObject`	The `Class` 1 that models the object on which the operation is executed is a `prov:Entity` identified as `var:preObject`. We use the prefix *pre* in this identifier because it is the object before the execution of the operation.
`Operation` 2 «predicate»/ «property»/ «void-accessor»	`prov:Activity` 2 / `var:operation`	The execution of `Operation` 2 stereotyped by «predicate»/ «property»/ «void-accessor» is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entity` 3 / `var:input`	Each parameter of `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Output Parameters` 4	`prov:Entity` 4 / `var:output`	Each parameter of `Output Parameters` 4 is a separate `prov:Entity` identified as `var:output`.
`Attributes` 5	`prov:Entity` 5 / `var:sourceAttribute`	Each attribute of `Attributes` 5 is a separate `prov:Entity` identified by `var:sourceAttribute`.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1	u2p:typeName / var:className	The value var:className is the name of the class to which var:preObject 1 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject 1 is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the operation Operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.
var:input 3	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
	u2p:typeName / var:inputType	The value var:inputType is a string with the name of the type of var:input 3.
var:output 4	prov:value / var:outputValue	The value var:outputValue is the direct representation of var:output 4.
	u2p:typeName / var:outputType	The value var:outputType is the string with the name of the type of var:output 4.
var:sourceAttribute 5	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:sourceAttribute 5 is an attribute.
	prov:value / var:sourceAttributeValue	The value var:sourceAttributeValue is the direct representation of var:sourceAttribute 5.
	u2p:attributeName / var:sourceAttributeName	The value var:sourceAttributeName is the name of var:sourceAttribute 5.
	u2p:typeName / var:sourceAttributeType	The value var:sourceAttributeType is the string with the name of the type of var:sourceAttribute 5.

PROV relations

a	prov:used	It is the beginning of utilizing var:preObject by var:operation.
b	prov:used	It is the beginning of utilizing var:input by var:operation.
c	prov:wasGeneratedBy	It is the completion of production of var:output by var:operation.
d	prov:wasDerivedFrom	It is the construction of var:output based on var:input.
e	prov:wasDerivedFrom	It is the construction of var:output based on var:sourceAttribute.

Discussion

- The stereotypes «predicate», «property», and «void-accessor» denote behaviours with specific nuances. Nevertheless, these nuances do not have impact in the translation into PROV since all of them compute *Output data* based on concrete object's attributes without modifying the object's status. Concretely, «predicate» denotes that the operation returns boolean values, «property» does not restrict the type of the returned values, and «void-accessor» returns the information through a parameter. As we said previously, there is no distinction in the transformation into PROV; however, some of the nuances given by the stereotypes will be included in the generated provenance through the values assigned to the template's variables. For instance, «predicate» defines the *Output data* as boolean, fact that is included in the provenance through the value assigned to var:outputType in var:output.

- Among the Class Diagrams patterns, both *ClP5* and *ClP4* address the execution of an operation that returns values computed based on information included on an object. While *ClP5* identifies the concrete attributes used to compute such information, *ClP4* considers the object's status as a whole (without considering the concrete attributes' values considered for its computation). Thus, *ClP5* gives a finer grained provenance than *ClP4*.
- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have decided to consider this circumstance with the aim of covering a wider spectrum of cases. When the executed operation lacks *Input data*, the UML representation in Figure 29 will not include `Input Parameters`. As a consequence, the resulting PROV template in Figure 30 will also lack `var:input` and its associated PROV relations.

Identifier *C*lass diagram *P*attern *6* (*ClP6*)

Context

The execution of an operation on an object changes the object's status as a whole (the concrete modified attributes are unknown or irrelevant).

Key elements

Object	It is the object on which the operation is executed.	
	Pre-operation object	The object with the status before the execution of the operation.
	Post-operation object	The object with the status after the execution of the operation.
Operation execution	It is the execution of the operation.	
Input data	It specifies the information (if any) passed into the *Operation execution*.	
Object's attributes	They are all the characteristics that conform the *Object*.	

UML Diagram

Key Element	UML	Rationale
Object	`Class` 1	*Objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use `Class` 1 to represent the *Object* both before and after the execution of the operation (*Pre-operation object* and *Post-operation object*, respectively).
Operation execution	`Operation` 2 «`command`»/ «`non-void-command`»	The `Operation` 2 stereotyped by «`command`»/ «`non-void-command`» represents the executed operation. These stereotypes denote that the object changes its status, without considering the concrete modified attributes. They differ in that an operation stereotyped by «`non-void-command`» returns information, while a «`command`» stereotyped operation does not. *Note*: the PROV template depicted in Figure 32 corresponds to an operation stereotyped by «`command`» (see in the discussion block the transformation of the operations stereotyped by «`non-void-command`»).
Input data	`Input Parameters` 3	They specify the information passed into the *Operation execution*.
Object's attributes	`Attributes` 4	They represent the characteristics of the *Object*.

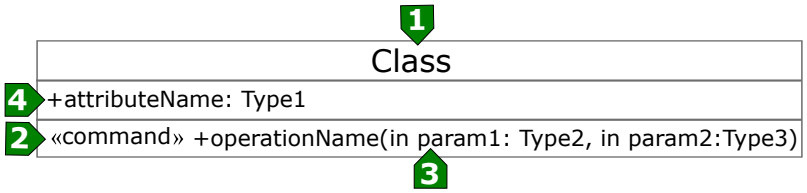

Figure 31. UML representation that models the context given by *ClP6*

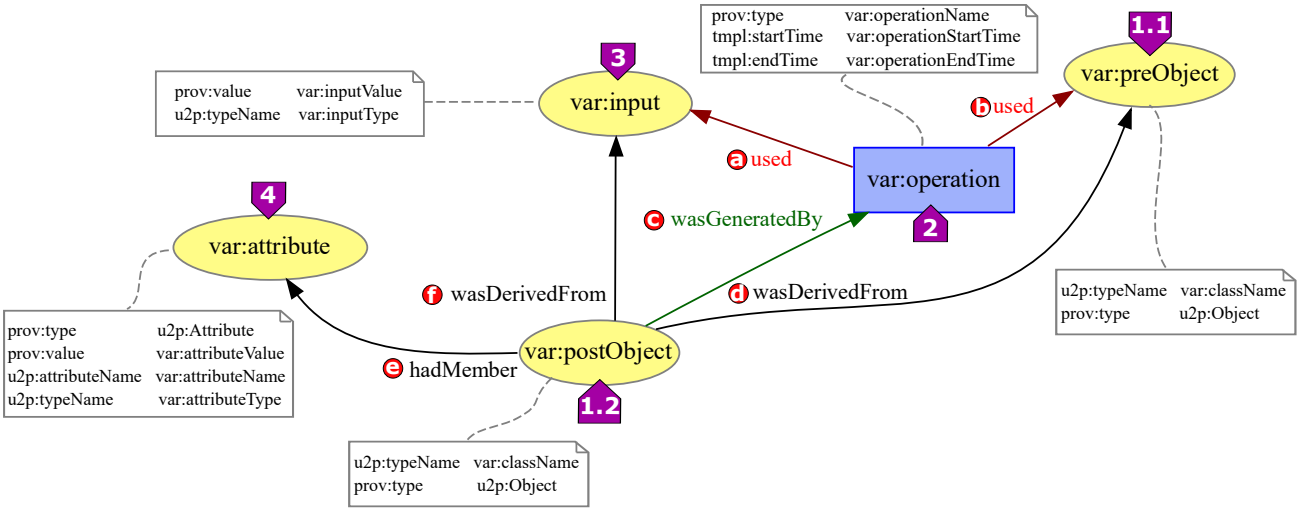

Figure 32. PROV template generated from the UML representation in Figure 31

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1.1 / `var:preObject`	The *pre-operation object*, i.e. the object with the status before the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:preObject`.
	`prov:Entity` 1.2 / `var:postObject`	The *Post-operation object*, i.e. the object with the status after the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:postObject`.
`Operation` 2 «`command`»/ «`non-void-command`»	`prov:Activity` 2 / `var:operation`	The execution of `Operation` 2 stereotyped by «`command`»/«`non-void-command`» is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entity` 3 / `var:input`	Each parameter of `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Attributes` 4	`prov:Entity` 4 / `var:attribute`	Each attribute of `Attributes` 4 is mapped to a separate `prov:Entity` identified by `var:attribute`.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1.1	u2p:typeName / var:className	The value var:className is the name of the class to which var:preObject 1.1 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject 1.1 is an object.
var:postObject 1.2	u2p:typeName / var:className	The value var:className is the name of the class to which var:postObject 1.2 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:postObject 1.2 is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.
var:input 3	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
	u2p:typeName / var:inputType	The value var:inputType is a string with the name of the type of var:input 3.
var:attribute 4	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:attribute 4 is an attribute.
	prov:value / var:attributeValue	The value var:attributeValue is the direct representation of attribute 4.
	u2p:attributeName / var:attributeName	The value var:attributeName is the name of attribute 4.
	u2p:typeName / var:attributeType	The value var:attributeType is the string with the name of the type of var:attribute 4.

PROV relations

a	prov:used	It is the beginning of utilizing var:input by var:operation.
b	prov:used	It is the beginning of utilizing var:preObject by var:operation.
c	prov:wasGeneratedBy	It is the completion of production of var:postObject by var:operation.
d	prov:wasDerivedFrom	It is the update of var:preObject resulting in var:postObject.
e	prov:hadMember	It states that var:attribute is one of the elements in var:postObject.
f	prov:wasDerivedFrom	It is the construction of var:postObject based on var:input.

Discussion

- Among the Class Diagrams patterns, patterns from *ClP6* to *ClP10* address the execution of operations that change an object's status. While, *ClP6* changes the object's status as a whole (being the concrete modified attributes unknown or irrelevant), in patterns *ClP7*-*ClP10* the values of the concrete attributes modified by the *Operation execution* are explicitly known. In contrast to *ClP7* which directly sets the information passed into the *Operation execution* as values of concrete object's attributes, the remainder patterns use such an information to change the object's status as a whole or the values of concrete object's attributes. It must also be noted that patterns *ClP9* and *ClP10* address the execution of operations which remove or add elements from/into an object's attribute collection, while patterns *ClP7* and *ClP8* modify the whole attribute by directly setting the input information (*ClP7*), or just only generating a new value for the attribute (*ClP8*).
- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have decided to consider this circumstance with the aim of covering a wider spectrum of cases. When the executed operation

lacks *Input data*, the UML representation in Figure 31 will not include `Input Parameters` 3. As a consequence, the resulting PROV template in Figure 32 will also lack `var:input` 3 and its associated PROV relations.

- A question that might arise is why in Figure 32 `var:attribute` is associated with `var:postObject`, which represents the object with the status after the execution of the operation, but it is not associated with `var:preObject`, (the object with the status before the execution). We have made this decision because another pattern has already registered the generation of the object with such a status (`var:preObject` in this pattern) and has already registered its attributes. This is possible because the collected bindings will associate a concrete value with `var:preObject` in this pattern, and at the same time, they associate the same value with `var:postObject` in another pattern (e.g., *ClP1*).

- Stereotypes «`command`» and «`non-void-command`» denote that the operation performs a complex change to the object's status as a whole. They differ in that an operation stereotyped by «`non-void-command`» returns information, while a «`command`» stereotyped operation does not. Due to the fact that the context of this pattern does not explicitly state that output data is obtained from the *Operation execution*, we represented this context in UML using the stereotype «`command`» (see Figure 31).

  Aiming at giving an insight into how the inclusion of *Output data* affects both UML representation and the resulting PROV template, Figure 33 depicts a UML representation with (1) the stereotype «`non-void-command`» and (2) *Output data* modelled as `Output Parameters` 5 (in this case with *return* direction, though the translation of *inout* and *out* directions would be equivalent). Figure 34 depicts its transformation into PROV. Both Figure 33 and 34 highlight the elements related to the inclusion of the *Output data* by blurring the elements coming from Figure 31 and 32, respectively.

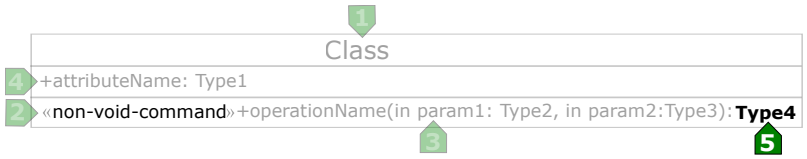

Figure 33. UML representation from Figure 31 including `Output Parameters`.

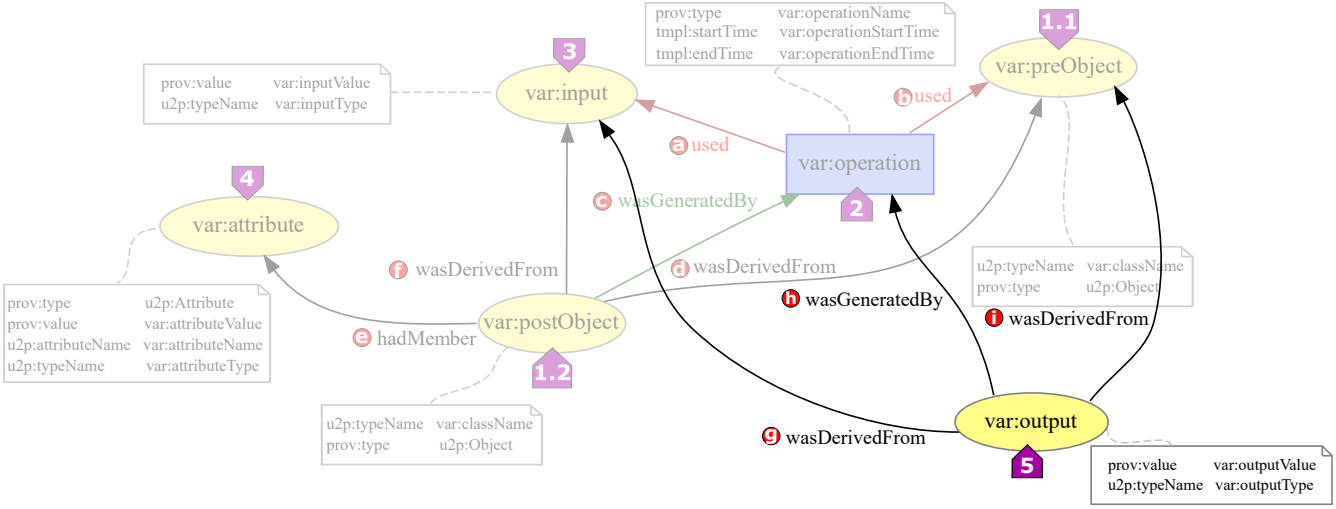

Figure 34. PROV template generated from the UML representation in Figure 33

PROV elements

UML	PROV / id	Rationale
`Output Parameters` 5	`prov:Entity` 5 / `var:output`	Each parameter of `Output Parameters` 5 is a separate `prov:Entity` identified as `var:output`.

PROV relations

g	prov:wasDerivedFrom	It is the construction of var:output based on var:input.
h	prov:wasGeneratedBy	It is the completion of production of var:output by var:operation.
i	prov:wasDerivedFrom	It is the construction of var:output based on var:preObject.

Attributes

PROV Element	Attribute / Value	Description
var:output 5	prov:value / var:outputValue	The value var:outputValue is the direct representation of var:output 5.
	u2p:typeName / var:outputType	The value var:outputType is a string with the name of the type of var:output 5.

Identifier *C*lass diagram *P*attern *7* (*ClP7*)

Context

The execution of an operation on an object directly sets the information passed to the operation as values of concrete object's attributes, thus provoking a change in the object's status.

Key elements

Object	It is the object on which the operation is executed.	
	Pre-operation object	The object with the status before the execution of the operation.
	Post-operation object	The object with the status after the execution of the operation.
Operation execution	It is the execution of the operation.	
Input data	It specifies the information passed into the *Operation execution*.	
Object's attributes	They are all the characteristics that conform the *Object*. Since, as a consequence of the *Operation execution*, the values of some attributes change, we have identified:	
	Modified attributes	The modified *Object's attributes*.
	Unmodified attributes	The not modified *Object's attributes*.

UML Diagram

Key Element	UML	Rationale
Object	`Class` 1	*Objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use `Class` 1 to represent the *Object* both before and after the execution of the operation (*Pre-operation object* and *Post-operation object*, respectively).
Operation execution	`Operation` 2 «`set`»	The `Operation` 2 stereotyped by «`set`» represents the executed operation. Concretely, the stereotype «`set`» denotes that *Input data* is directly set as values of concrete attributes of the object.
Input data	`Input Parameters` 3	They specify the information passed into the *Operation execution*.
Object's attributes	`Attributes` 4	They represent the characteristics of the *Object*.

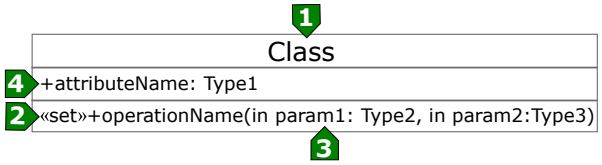

Figure 35. UML representation that models the context given by *ClP7*

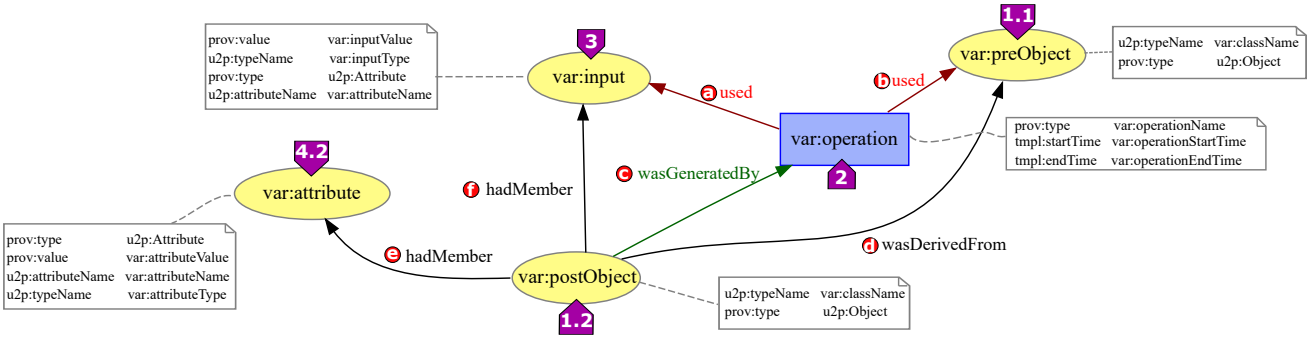

Figure 36. PROV template generated from the UML representation in Figure 35

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1.1 / `var:preObject`	The *Pre-operation object*, i.e. the object with the status before the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:preObject`.
	`prov:Entity` 1.2 / `var:postObject`	The *Post-operation object*, i.e. the object with the status after the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:postObject`.
`Operation` 2 «set»	`prov:Activity` 2 / `var:operation`	The execution of `Operation` 2 stereotyped by «set» is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entiy` 3 / `var:input`	Each parameter of `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Attributes` 4	None /	The *Modified attributes* (belonging to `Attributes` 4) are already mapped to `var:input`. For further information, see the discussion.
	`prov:Entity` 4.2 / `var:attribute`	Each *Unmodified attribute* (belonging to `Attributes` 4) is mapped to a separate `prov:Entity` with identifier `var:attribute`.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1.1	u2p:typeName / var:className	The value var:className is the name of the class to which var:preObject 1.1 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject 1.1 is an object.
var:postObject 1.2	u2p:typeName / var:className	The value var:className is the name of the class to which var:postObject 1.2 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:postObject 1.2 is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.
var:input 3	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
	u2p:typeName / var:inputType	The value var:inputType is a string with the name of the type of var:input 3.
	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:input 3 is an attribute.
	u2p:attributeName / var:attributeName	The value var:attributeName is the name of the attribute var:input 3.
var:attribute 4.2	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:attribute 4.2 is an attribute.
	prov:value / var:attributeValue	The value var:attributeValue is the direct representation of var:attribute 4.2.
	u2p:attributeName / var:attributeName	The value var:attributeName is the name of var:attribute 4.2.
	u2p:typeName / var:attributeType	The value var:attributeType is the string with the name of the type of var:attribute 4.2.

PROV relations

a	prov:used	It is the beginning of utilizing var:input by var:operation.
b	prov:used	It is the beginning of utilizing var:preObject by var:operation.
c	prov:wasGeneratedBy	It is the completion of production of var:postObject by var:operation.
d	prov:wasDerivedFrom	It is the update of var:preObject resulting in var:postObject.
e	prov:hadMember	It states that var:attribute is one of the elements in var:postObject.
f	prov:hadMember	It states that var:input is one of the elements in var:postObject. This is due to the fact that in this context the input information is directly set as values of certain attributes of the *Object*.

Discussion

- Among the Class Diagrams patterns, patterns from *ClP6* to *ClP10* address the execution of operations that change an object's status. While, *ClP6* changes the object's status as a whole (being the concrete modified attributes unknown or irrelevant), in patterns *ClP7*-*ClP10* the values of the concrete attributes modified by the *Operation execution* are explicitly known. In contrast to *ClP7* which directly sets the information passed into the *Operation execution* as values of concrete object's

attributes, the remainder patterns use such an information to change the object's status as a whole or the values of concrete object's attributes. It must also be noted that patterns *ClP9* and *ClP10* address the execution of operations which remove or add elements from/into an object's attribute collection, while patterns *ClP7* and *ClP8* modify the whole attribute by directly setting the input information (*ClP7*), or just only generating a new value for the attribute (*ClP8*).

- A question that might arise is why in Figure 36 `var:attribute` is associated with `var:postObject`, which represents the object with the status after the execution of the operation, but it is not associated with `var:preObject`, (the object with the status before the execution). We have made this decision because another pattern has already registered the generation of the object with such a status (`var:preObject` in this pattern) and has already registered its attributes. This is possible because the collected bindings will associate a concrete value with `var:preObject` in this pattern, and at the same time, they associate the same value with `var:postObject` in another pattern (e.g., *ClP1*).

- This context states that the *Input data* are directly set as values of certain object's attributes, which means that the *Input data* correspond directly to the *Modified attributes*. This fact is represented in the PROV template by means of the pair attribute/value `prov:type`/`u2p:Attribute` of `var:input`, and the relation `prov:hadMember` between `var:postObject` and `var:input`. Additionally, `var:input` has the attribute `u2p:attributeName` whose value `var:attributeName` denotes the name of the modified attribute.

- Among the Class Diagrams patterns, *ClP6*, *ClP7*, *ClP8*, *ClP9*, and *ClP10* address the execution of an operation that provokes a change in the status of the object on which the operation is executed. On the one hand, *ClP6* considers the object's status as a whole, without considering the concrete modified attributes. On the other hand, *ClP7*-*ClP10* identify the concrete modified attributes, giving more detailed information about the behaviour that has led the object's status as it is. Thus, *ClP7* gives a finer grained provenance than *ClP6*.

- Although the *context* of this pattern does not explicitly state that output data should be obtained from the *Operation execution*, this could be the case. However, we have decided not to include this output data in this pattern description to avoid overburden both the UML and PROV explanations with information out of the scope of the *context*.

  Aiming at giving an insight into how the inclusion of *Output data* affects both UML representation and the resulting PROV template, Figure 37 depicts a UML representation with the *Output data* modelled as `Output Parameters` 5 (in this case with *return* direction, though the translation of *inout* and *out* directions would be equivalent). Figure 38 depicts its transformation into PROV. Both Figure 37 and 38 highlight the elements related to the inclusion of the *Output data* by blurring the elements coming from Figure 35 and 36, respectively.

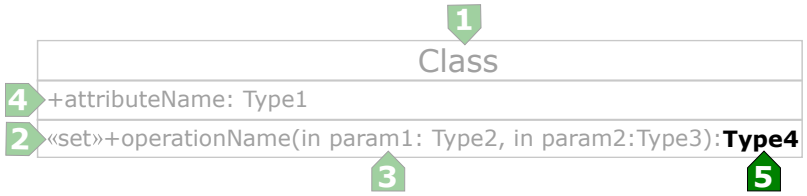

Figure 37. UML representation from Figure 35 including `Output Parameters`.

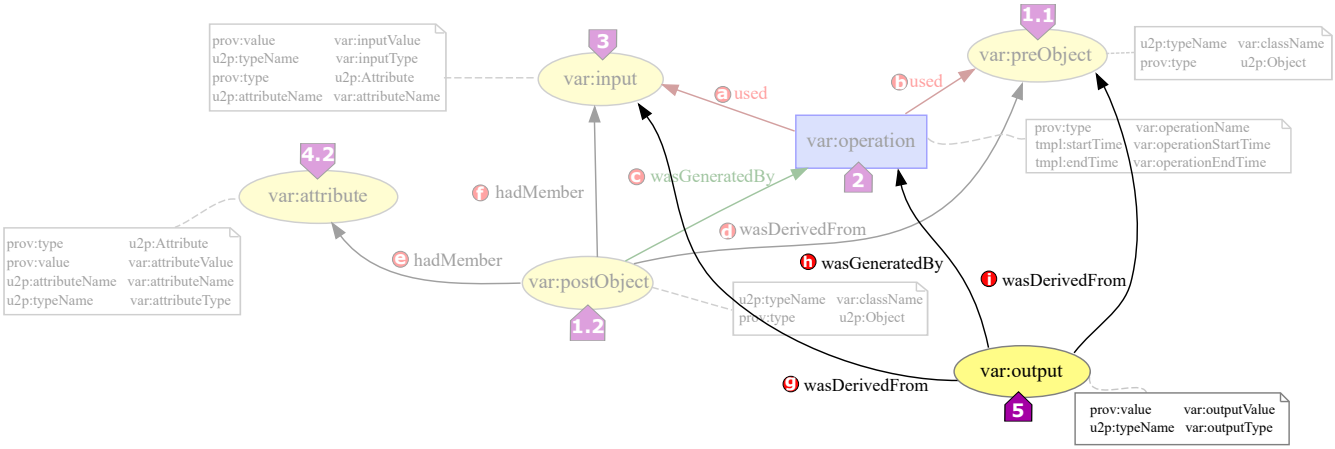

Figure 38. PROV template generated from the UML representation in Figure 37

PROV elements

UML	PROV / id	Rationale
`Output Parameters` 5	`prov:Entity` 5 / `var:output`	Each parameter of `Output Parameters` 5 is a separate `prov:Entity` identified as `var:output`.

PROV relations

g	`prov:wasDerivedFrom`	It is the construction of `var:output` based on `var:input`.
h	`prov:wasGeneratedBy`	It is the completion of production of `var:output` by `var:operation`.
i	`prov:wasDerivedFrom`	It is the construction of `var:output` based on `var:preObject`.

Attributes

PROV Element	Attribute / Value	Description
`var:output` 5	`prov:value` / `var:outputValue`	The value `var:outputValue` is the direct representation of `var:output` 5.
	`u2p:typeName` / `var:outputType`	The value `var:outputType` is a string with the name of the type of `var:output` 5.

Identifier *C*lass diagram *P*attern *8* (*ClP8*)

Context

The execution of an operation on an object changes the values of concrete object's attributes, thus provoking a change in the object's status.

Key elements

Object	It is the object on which the operation is executed.	
	Pre-operation object	The object with the status before the execution of the operation.
	Post-operation object	The object with the status after the execution of the operation.
Operation execution	It is the execution of the operation.	
Input data	It specifies the information (if any) passed into the *Operation execution*.	
Object's attributes	They are all the characteristics that conform the *Object*. Since, as a consequence of the *Operation execution*, the values of some attributes change, we have identified:	
	Modified attributes	The modified *Object's attributes*.
	Unmodified attributes	The not modified *Object's attributes*.

UML Diagram

Key Element	UML	Rationale
Object	`Class` 1	*Objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use `Class` 1 to represent the *Object* both before and after the execution of the operation (*Pre-operation object* and *Post-operation object*, respectively).
Operation execution	`Operation` 2 «`modify`»	The `Operation` 2 stereotyped by «`modify`» represents the executed operation. Concretely, the stereotype «`modify`» denotes that concrete attributes of the object are modified.
Input data	`Input Parameters` 3	They specify the information passed into the *Operation execution*.
Object's attributes	`Attributes` 4	They represent the characteristics of the *Object*.

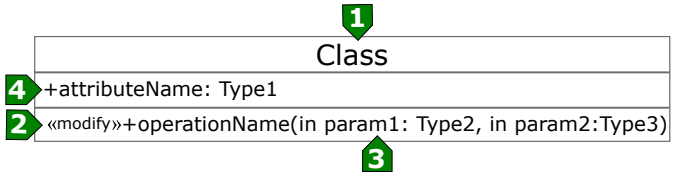

Figure 39. UML representation that models the context given by *ClP8*

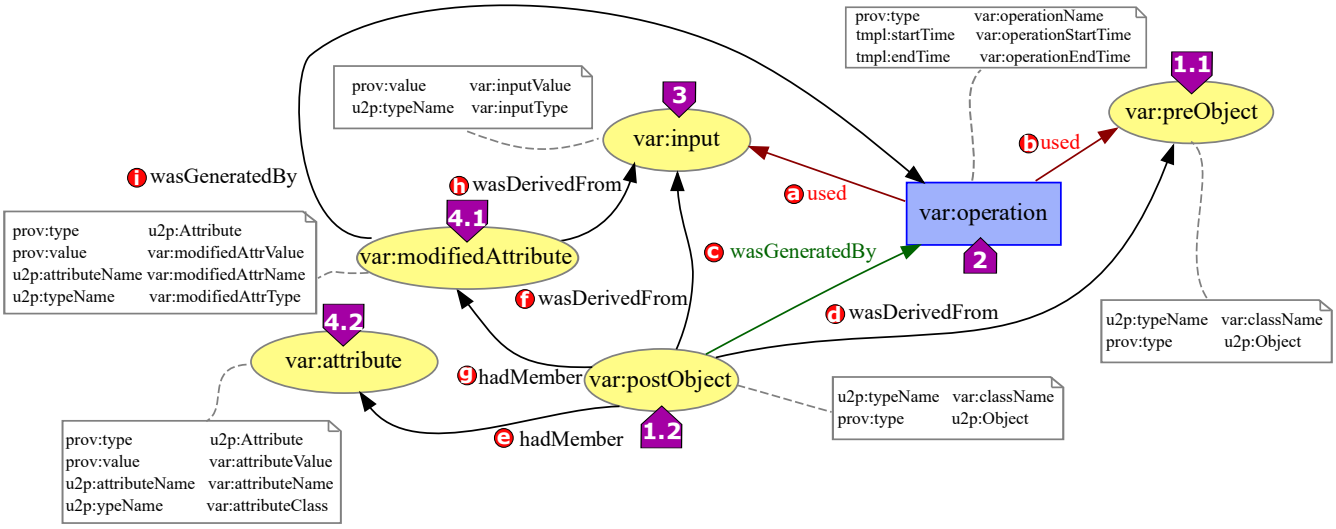

Figure 40. PROV template generated from the UML representation in Figure 39

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1.1 / `var:preObject`	The *Pre-operation object*, i.e. the object with the status before the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:preObject`.
	`prov:Entity` 1.2 / `var:postObject`	The *Post-operation object*, i.e. the object with the status after the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:postObject`.
`Operation` 2 «`modify`»	`prov:Activity` 2 / `var:operation`	The execution of `Operation` 2 stereotyped by «`modify`» is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entiy` 3 / `var:input`	Each parameter of `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Attributes` 4	`prov:Entity` 4.1 / `var:modifiedAttribute`	Each *Modified attribute* (belonging to `Attributes` 4) is mapped to a separate `prov:Entity` with identifier `var:modifiedAttribute`.
	`prov:Entity` 4.2 / `var:attribute`	Each *Unmodified attribute* (belonging to `Attributes` 4) is mapped to a separate `prov:Entity` with identifier `var:attribute`.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1.1	u2p:typeName / var:className	The value var:className is the name of the class to which var:preObject 1.1 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject 1.1 is an object.
var:postObject 1.2	u2p:typeName / var:className	The value var:className is the name of the class to which var:postObject 1.2 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:postObject 1.2 is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.
var:input 3	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
	u2p:typeName / var:inputType	The value var:inputType is a string with the name of the type of var:input 3.
var:modifiedAttribute 4.1	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:modifiedAttribute 4.1 is an attribute.
	prov:value / var:modifiedAttrValue	The value var:attributeValue is the direct representation of var:modifiedAttribute 4.1.
	u2p:attributeName / var:modifiedAttrName	The value var:modifiedAttrName is the name of var:modifiedAttribute 4.1.
	u2p:typeName / var:modifiedAttrType	The value var:attributeType is a string with the name of the type of var:modifiedAttribute 4.1.
var:attribute 4.2	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:attribute 4.2 is an attribute.
	prov:value / var:attributeValue	The value var:attributeValue is the direct representation of var:attribute 4.2.
	u2p:attributeName / var:attributeName	The value var:attributeName is the name of var:attribute 4.2.
	u2p:typeName / var:attributeType	The value var:attributeType is the string with the name of the type of var:attribute 4.2.

PROV relations

a	prov:used	It is the beginning of utilizing var:input by var:operation.
b	prov:used	It is the beginning of utilizing var:preObject by var:operation.
c	prov:wasGeneratedBy	It is the completion of production of var:postObject by var:operation.
d	prov:wasDerivedFrom	It is the update of var:preObject resulting in var:postObject.
e	prov:hadMember	It states that var:attribute is one of the elements in var:postObject.
f	prov:wasDerivedFrom	It is the construction of var:postObject based on var:input.
g	prov:hadMember	It states that var:modifiedAttribute is one of the elements in var:postObject.
h	prov:wasDerivedFrom	It is the construction of var:modifiedAttribute based on var:input.
i	prov:wasGeneratedBy	It is the completion of production of var:modifiedAttribute by var:operation.

Discussion

- A question that might arise is why in Figure 40 `var:attribute` is associated with `var:postObject`, which represents the object with the status after the execution of the operation, but it is not associated with `var:preObject`, (the object with the status before the execution). We have made this decision because another pattern has already registered the generation of the object with such a status (`var:preObject` in this pattern) and has already registered its attributes. This is possible because the collected bindings will associate a concrete value with `var:preObject` in this pattern, and at the same time, they associate the same value with `var:postObject` in another pattern (e.g., *ClP1*).

- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have decided to consider this circumstance with the aim of covering a wider spectrum of cases. When the executed operation lacks *Input data*, the UML representation in Figure 39 will not include `Input Parameters` 3. As a consequence, the resulting PROV template in Figure 40 will also lack `var:input` 3 and its associated PROV relations.

- Among the Class Diagrams patterns, patterns from *ClP6* to *ClP10* address the execution of operations that change an object's status. While, *ClP6* changes the object's status as a whole (being the concrete modified attributes unknown or irrelevant), in patterns *ClP7*-*ClP10* the values of the concrete attributes modified by the *Operation execution* are explicitly known. In contrast to *ClP7* which directly sets the information passed into the *Operation execution* as values of concrete object's attributes, the remainder patterns use such an information to change the object's status as a whole or the values of concrete object's attributes. It must also be noted that patterns *ClP9* and *ClP10* address the execution of operations which remove or add elements from/into an object's attribute collection, while patterns *ClP7* and *ClP8* modify the whole attribute by directly setting the input information (*ClP7*), or just only generating a new value for the attribute (*ClP8*).

- Although the *context* of this pattern does not explicitly state that output data should be obtained from the *Operation execution*, this could be the case. However, we have decided not to include this output data in this pattern description to avoid overburden both the UML and PROV explanations with information out of the scope of the *context*.

  Aiming at giving an insight into how the inclusion of *Output data* affects both UML representation and the resulting PROV template, Figure 41 depicts a UML representation with the *Output data* modelled as `Output Parameters` 5 (in this case with *return* direction, though the translation of *inout* and *out* directions would be equivalent). Figure 42 depicts its transformation into PROV. Both Figure 41 and 42 highlight the elements related to the inclusion of the *Output data* by blurring the elements coming from Figure 39 and 40, respectively.

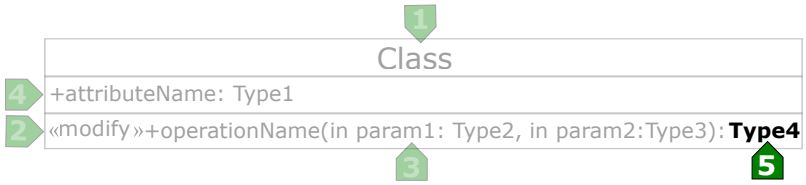

Figure 41. UML representation from Figure 39 including `Output Parameters`.

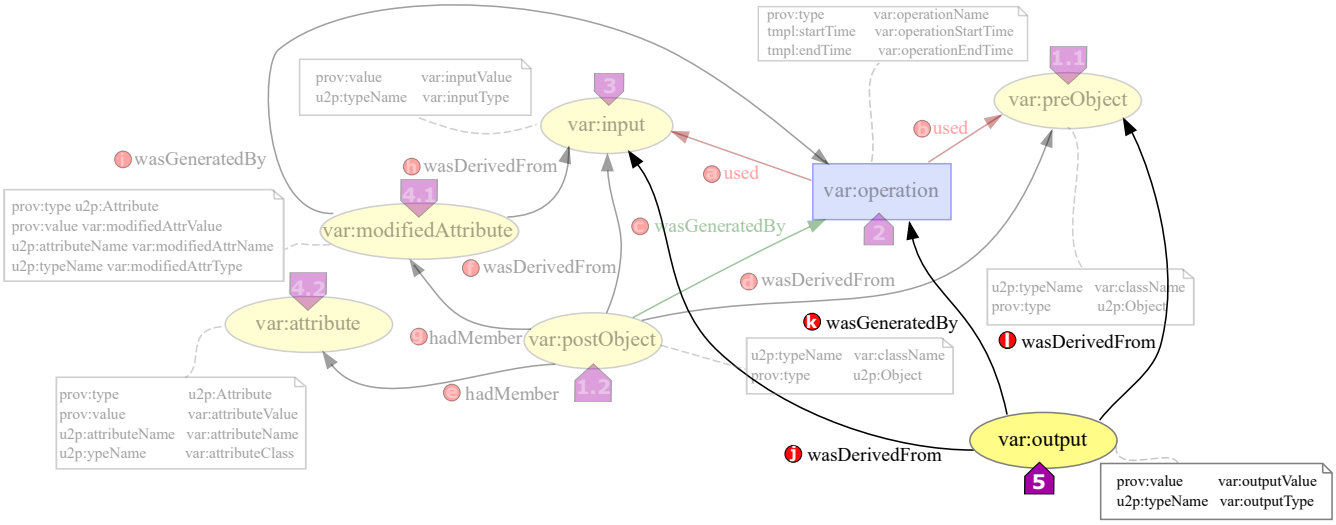

Figure 42. PROV template generated from the UML representation in Figure 41

PROV elements

UML	PROV / id	Rationale
`Output Parameters` 5	`prov:Entity` 5 / `var:output`	Each parameter of `Output Parameters` 5 is a separate `prov:Entity` identified as `var:output`.

PROV relations

j	`prov:wasDerivedFrom`	It is the construction of `var:output` based on `var:input`.
k	`prov:wasGeneratedBy`	It is the completion of production of `var:output` by `var:operation`.
l	`prov:wasDerivedFrom`	It is the construction of `var:output` based on `var:preObject`.

Attributes

PROV Element	Attribute / Value	Description
`var:output` 5	`prov:value` / `var:outputValue`	The value `var:outputValue` is the direct representation of `var:output` 5.
	`u2p:typeName` / `var:outputType`	The value `var:outputType` is a string with the name of the type of `var:output` 5.

Identifier *C*lass diagram *P*attern *9* (*ClP9*)

Context

The execution of an operation on an object removes element(s) from a concrete object's collection attribute, thus provoking a change in the object's status.

Key elements

Object	It is the object on which the operation is executed.	
	Pre-operation object	The object with the status before the execution of the operation.
	Post-operation object	The object with the status after the execution of the operation.
Operation execution	It is the execution of the operation.	
Input data	It specifies the information (if any) passed into the *Operation execution*.	
Object's attributes	They are all the characteristics that conform the *Object*. Since, as a consequence of the *Operation execution*, a concrete collection attribute changes, we have identified:	
	Modified collection attribute	The modified *Object's attribute*.
	Unmodified attributes	The not modified *Object's attributes*.

UML Diagram

Key Element	UML	Rationale
Object	`Class` 1	*Objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use `Class` 1 to represent the *Object* both before and after the execution of the operation (*Pre-operation object* and *Post-operation object*, respectively).
Operation execution	`Operation` 2 «`remove`»	The `Operation` 2 stereotyped by «`remove`» represents the executed operation. Concretely, the stereotype «`remove`» denotes that an element (or elements) of a concrete collection attribute is removed.
Input data	`Input Parameters` 3	They specify the information passed into the *Operation execution*.
Object's attributes	`Attributes` 4	They represent the characteristics of the *Object*.

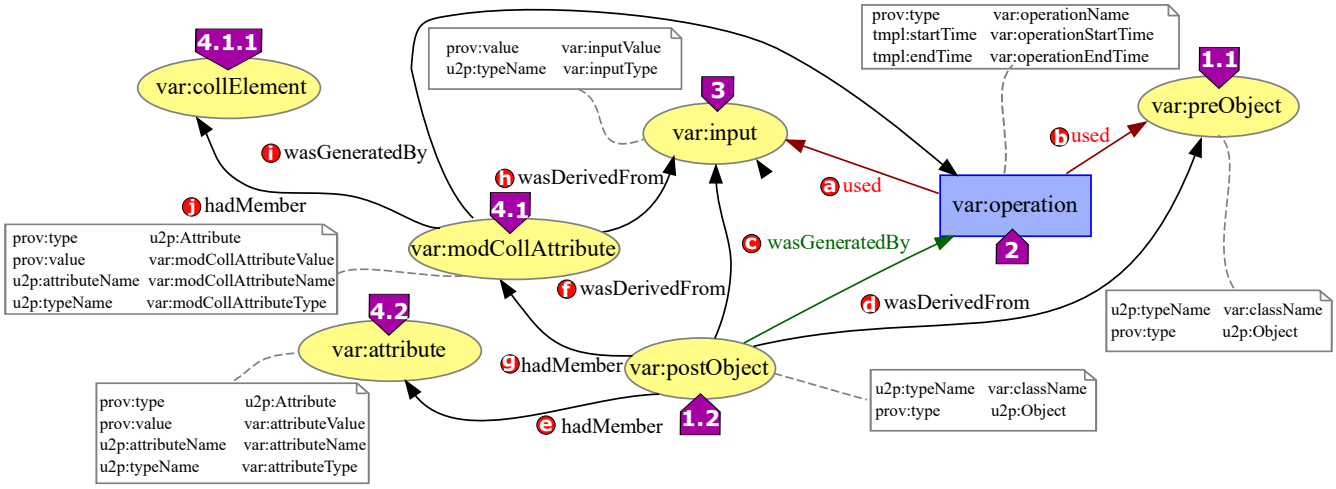

Figure 44. PROV template generated from the UML representation in Figure 43

Mapping to PROV

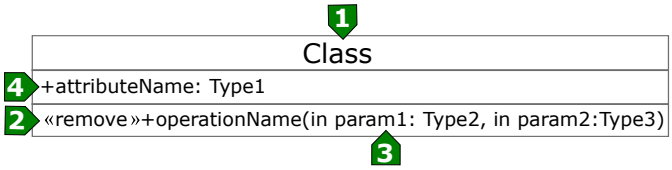

Figure 43. UML representation that models the context given by *ClP9*

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1.1 / `var:preObject`	The *Pre-operation object*, i.e. the object with the status before the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:preObject`.
	`prov:Entity` 1.2 / `var:postObject`	The *Post-operation object*, i.e. the object with the status after the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:postObject`.
`Operation` 2 «remove»	`prov:Activity` 2 / `var:operation`	The execution of `Operation` 2 stereotyped by «remove» is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entity` 3 / `var:input`	Each parameter of `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Attributes` 4	`prov:Entity` 4.1 / `var:modCollAttribute`	The *Modified collection attribute* (belonging to `Attributes` 4) is a `prov:Entity` with identifier `var:modCollAttribute`. Additionally, each element in this collection is a separate `prov:Entity` identified by `var:collElement` 4.1.1
	`prov:Entity` 4.2 / `var:attribute`	Each *Unmodified attribute* (belonging to `Attributes` 4) is mapped to a separate `prov:Entity` with identifier `var:attribute`.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1.1	u2p:typeName / var:className	The value var:className is the name of the class to which var:preObject 1.1 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject 1.1 is an object.
var:postObject 1.2	u2p:typeName / var:className	The value var:className is the name of the class to which var:postObject 1.2 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:postObject 1.2 is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.
var:input 3	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
	u2p:typeName / var:inputType	The value var:inputType is a string with the name of the type of var:input 3.
var:modCollAttribute 4.1	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:modCollAttribute 4.1 is an attribute.
	prov:value / var:modCollAttributeValue	The value var:modCollAttributeValue is the direct representation of var:modCollAttribute 4.1.
	u2p:attributeName / var:modCollAttributeName	The value var:modCollAttributeName is the name of var:modCollAttribute 4.1.
	u2p:typeName / var:modCollAttributeType	The value var:modCollAttributeType is a string with the name of the type of var:modCollAttribute 4.1.
var:attribute 4.2	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:attribute 4.2 is an attribute.
	prov:value / var:attributeValue	The value var:attributeValue is the direct representation of var:attribute 4.2.
	u2p:attributeName / var:attributeName	The value var:attributeName is the name of var:attribute 4.2.
	u2p:typeName / var:attributeType	The value var:attributeType is a string with the name of the type of var:attribute 4.2.

PROV relations

a	prov:used	It is the beginning of utilizing var:input by var:operation.
b	prov:used	It is the beginning of utilizing var:preObject by var:operation.
c	prov:wasGeneratedBy	It is the completion of production of var:postObject by var:operation.
d	prov:wasDerivedFrom	It is the update of var:preObject resulting in var:postObject.
e	prov:hadMember	It states that var:attribute is one of the elements in var:postObject.
f	prov:wasDerivedFrom	It is the construction of var:postObject based on var:input.
g	prov:hadMember	It states that var:modCollAttribute is one of the elements in var:postObject.
h	prov:wasDerivedFrom	It is the construction of var:modCollAttribute based on var:input.

prov:wasGeneratedBy	It is the completion of production of `var:modCollAttribute` by `var:operation`.
prov:hadMember	It states that `var:collElement` is one of the elements in `var:modCollAttribute`.

Discussion

- A question that might arise is why in Figure 44 `var:attribute` is associated with `var:postObject`, which represents the object with the status after the execution of the operation, but it is not associated with `var:preObject`, (the object with the status before the execution). We have made this decision because another pattern has already registered the generation of the object with such a status (`var:preObject` in this pattern) and has already registered its attributes. This is possible because the collected bindings will associate a concrete value with `var:preObject` in this pattern, and at the same time, they associate the same value with `var:postObject` in another pattern (e.g., *ClP1*).

- Among the Class Diagrams patterns, patterns from *ClP6* to *ClP10* address the execution of operations that change an object's status. While, *ClP6* changes the object's status as a whole (being the concrete modified attributes unknown or irrelevant), in patterns *ClP7*-*ClP10* the values of the concrete attributes modified by the *Operation execution* are explicitly known. In contrast to *ClP7* which directly sets the information passed into the *Operation execution* as values of concrete object's attributes, the remainder patterns use such an information to change the object's status as a whole or the values of concrete object's attributes. It must also be noted that patterns *ClP9* and *ClP10* address the execution of operations which remove or add elements from/into an object's attribute collection, while patterns *ClP7* and *ClP8* modify the whole attribute by directly setting the input information (*ClP7*), or just only generating a new value for the attribute (*ClP8*).

- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have decided to consider this circumstance with the aim of covering a wider spectrum of cases. When the executed operation lacks *Input data*, the UML representation in Figure 43 will not include `Input Parameters` 3. As a consequence, the resulting PROV template in Figure 44 will also lack `var:input` 3 and its associated PROV relations.

- Although the *context* of this pattern does not explicitly state that output data should be obtained from the *Operation execution*, this could be the case. However, we have decided not to include this output data in this pattern description to avoid overburden both the UML and PROV explanations with information out of the scope of the *context*.

  Aiming at giving an insight into how the inclusion of *Output data* affects both UML representation and the resulting PROV template, Figure 45 depicts a UML representation with the *Output data* modelled as `Output Parameters` 5 (in this case with *return* direction, though the translation of *inout* and *out* directions would be equivalent). Figure 46 depicts its transformation into PROV. Both Figure 45 and 46 highlight the elements related to the inclusion of the *Output data* by blurring the elements coming from Figure 43 and 44, respectively.

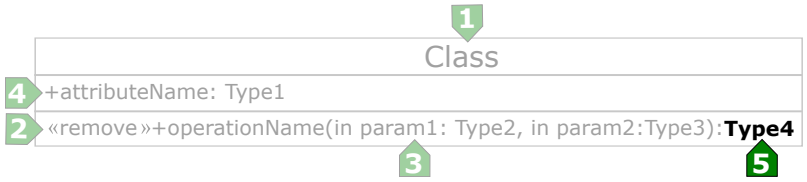

Figure 45. UML representation from Figure 43 including `Output Parameters`.

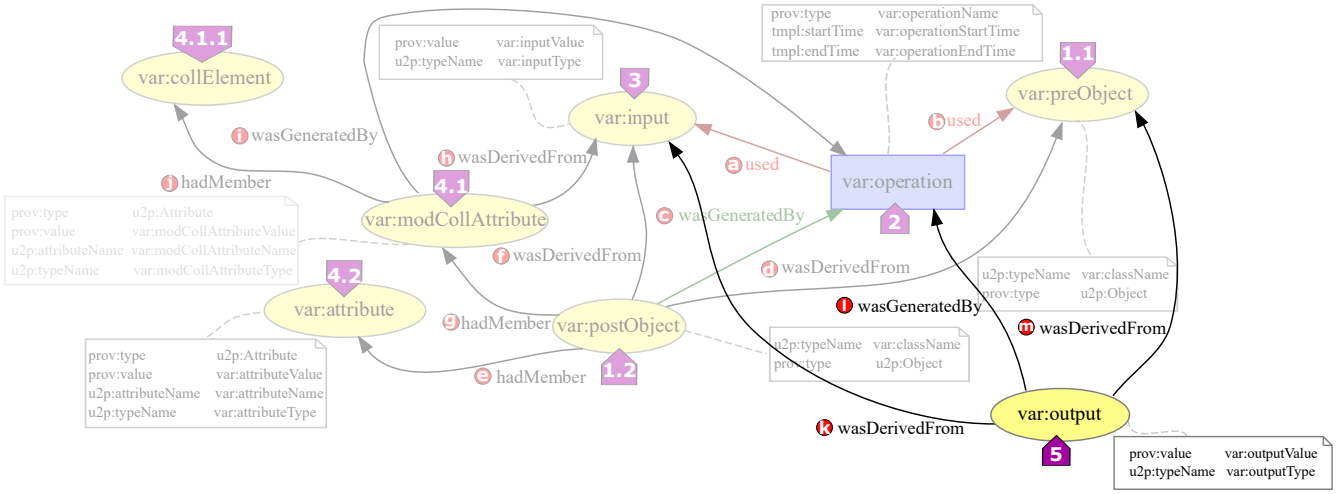

Figure 46. PROV template generated from the UML representation in Figure 45

PROV elements

UML	PROV / id	Rationale
`Output Parameters` 5	`prov:Entity` 5 / `var:output`	Each parameter of `Output Parameters` 5 is a separate `prov:Entity` identified as `var:output`.

PROV relations

k	`prov:wasDerivedFrom`	It is the construction of `var:output` based on `var:input`.
l	`prov:wasGeneratedBy`	It is the completion of production of `var:output` by `var:operation`.
m	`prov:wasDerivedFrom`	It is the construction of `var:output` based on `var:preObject`.

Attributes

PROV Element	Attribute / Value	Description
`var:output` 5	`prov:value` / `var:outputValue`	The value `var:outputValue` is the direct representation of `var:output` 5.
	`u2p:typeName` / `var:outputType`	The value `var:outputType` is a string with the name of the type of `var:output` 5.

Identifier *C*lass diagram *P*attern *10* (*ClP10*)

Context

The execution of an operation on an object directly adds the information passed to the operation as new element(s) of a concrete object's collection attribute, thus provoking a change in the object's status.

Key elements

Object	It is the object to which the operation to be executed belongs.	
	Pre-operation object	The object with the status before the execution of the operation.
	Post-operation object	The object with the status after the execution of the operation.
Operation execution	is the execution of the behaviour specified by the operation.	
Input data	It specifies the information passed into the *Operation execution*.	
Object's attributes	They are all the characteristics that conform the *Object*. Since, as a consequence of the *Operation execution*, a concrete collection attribute changes, we have identified:	
	Modified collection attribute	The modified *Object's attribute*.
	Unmodified attributes	The not modified *Object's attributes*.

UML Diagram

Key Element	UML	Rationale
Object	`Class` 1	*Objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use `Class` 1 to represent the *Object* both before and after the execution of the operation (*Pre-operation object* and *Post-operation object*, respectively).
Operation execution	`Operation` 2 «`add`»	The `Operation` 2 stereotyped by «`add`» represents the executed operation. Concretely, the stereotype «`add`» denotes that a new element (or elements) is directly added to a concrete collection attribute.
Input data	`Input Parameters` 3	They specify the information passed into the *Operation execution*.
Object's attributes	`Attributes` 4	They represent the characteristics of the *Object*.

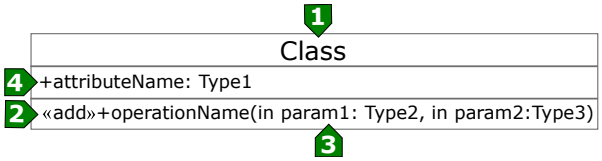

Figure 47. UML representation that models the context given by *ClP10*

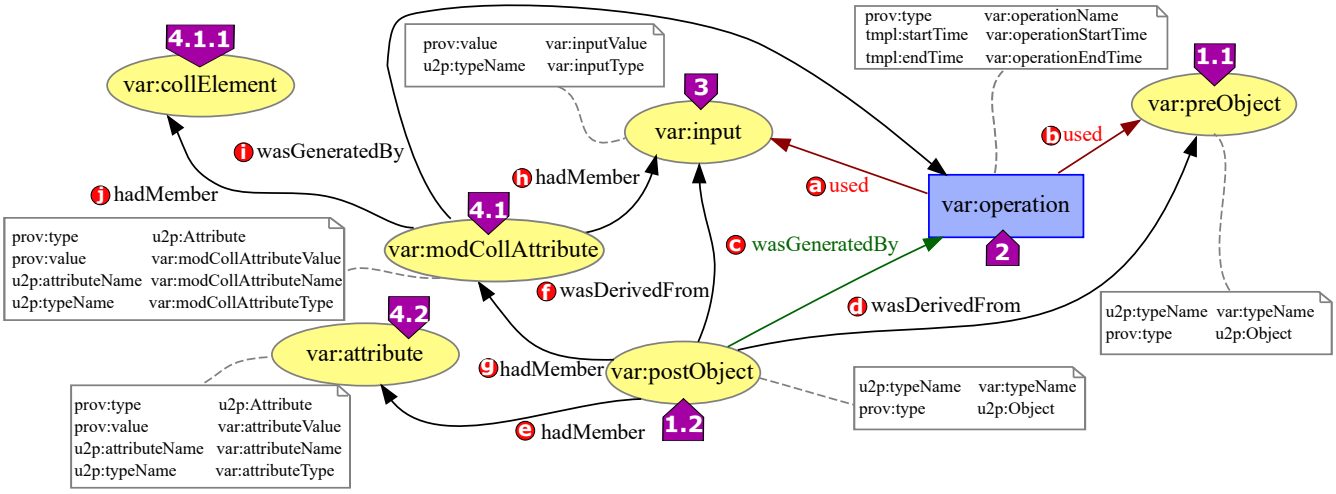

Figure 48. PROV template generated from the UML representation in Figure 47

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1.1 / `var:preObject`	The *Pre-operation object*, i.e. the object with the status before the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:preObject`.
	`prov:Entity` 1.2 / `var:postObject`	The *Post-operation object*, i.e. the object with the status after the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:postObject`.
`Operation` 2 «add»	`prov:Activity` 2 / `var:operation`	The execution of `Operation` 2 stereotyped by «add» is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entity` 3 / `var:input`	Each parameter of `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Attributes` 4	`prov:Entity` 4.1 / `var:modCollAttribute`	The *Modified collection attribute* (belonging to `Attributes` 4) is a `prov:Entity` with identifier `var:modCollAttribute`. Additionally, each element in this collection is a separate `prov:Entity` identified by `var:collElement` 4.1.1
	`prov:Entity` 4.2 / `var:attribute`	Each *Unmodified attribute* (belonging to `Attributes` 4) is mapped to a separate `prov:Entity` with identifier `var:attribute`.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1.1	u2p:typeName / var:className	The value var:className is the name of the class to which var:preObject 1.1 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject 1.1 is an object.
var:postObject 1.2	u2p:typeName / var:className	The value var:className is the name of the class to which var:postObject 1.2 belongs.
	prov:type / u2p:Object	The value u2p:Object shows that var:postObject 1.2 is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.
var:input 3	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
	u2p:typeName / var:inputType	The value var:inputType is a string with the name of the type of var:input 3.
var:modCollAttribute 4.1	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:modCollAttribute 4.1 is an attribute.
	prov:value / var:modCollAttributeValue	The value var:modCollAttributeValue is the direct representation of var:modCollAttribute 4.1.
	u2p:attributeName / var:modCollAttributeName	The value var:modCollAttributeName is the name of var:modCollAttribute 4.1.
	u2p:typeName / var:modCollAttributeType	The value var:modCollAttributeType is a string with the name of the type of var:modCollAttribute 4.1.
var:attribute 4.2	prov:type / u2p:Attribute	The value u2p:Attribute shows that attribute 4.2 is an attribute.
	prov:value / var:attributeValue	The value var:attributeValue is the direct representation of attribute 4.2.
	u2p:attributeName / var:attributeName	The value var:attributeName is the name of attribute 4.2.
	u2p:typeName / var:attributeType	The value var:attributeType is a string with the name of the type of attribute 4.2.

PROV relations

a	prov:used	It is the beginning of utilizing var:input by var:operation.
b	prov:used	It is the beginning of utilizing var:preObject by var:operation.
c	prov:wasGeneratedBy	It is the completion of production of var:postObject by var:operation.
d	prov:wasDerivedFrom	It is the update of var:preObject resulting in var:postObject.
e	prov:hadMember	It states that var:attribute is one of the elements in var:postObject.
f	prov:wasDerivedFrom	It is the construction of var:postObject based on var:input.
g	prov:hadMember	It states that var:modCollAttribute is one of the elements in var:postObject.

h	prov:hadMember	It states that var:input is one of the elements in var:modCollAttribute. This is due to the fact that in this context the input information is directly added to the object's collection attribute.
i	prov:wasGeneratedBy	It is the completion of production of var:modCollAttribute by var:operation.
j	prov:hadMember	It states that var:collElement is one of the elements in var:modCollAttribute.

Discussion

- A question that might arise is why in Figure 48 var:attribute is associated with var:postObject, which represents the object with the status after the execution of the operation, but it is not associated with var:preObject, (the object with the status before the execution). We have made this decision because another pattern has already registered the generation of the object with such a status (var:preObject in this pattern) and has already registered its attributes. This is possible because the collected bindings will associate a concrete value with var:preObject in this pattern, and at the same time, they associate the same value with var:postObject in another pattern (e.g., *ClP1*).

- Among the Class Diagrams patterns, patterns from *ClP6* to *ClP10* address the execution of operations that change an object's status. While, *ClP6* changes the object's status as a whole (being the concrete modified attributes unknown or irrelevant), in patterns *ClP7*-*ClP10* the values of the concrete attributes modified by the *Operation execution* are explicitly known. In contrast to *ClP7* which directly sets the information passed into the *Operation execution* as values of concrete object's attributes, the remainder patterns use such an information to change the object's status as a whole or the values of concrete object's attributes. It must also be noted that patterns *ClP9* and *ClP10* address the execution of operations which remove or add elements from/into an object's attribute collection, while patterns *ClP7* and *ClP8* modify the whole attribute by directly setting the input information (*ClP7*), or just only generating a new value for the attribute (*ClP8*).

- Although the *context* of this pattern does not explicitly state that output data should be obtained from the *Operation execution*, this could be the case. However, we have decided not to include this output data in this pattern description to avoid overburden both the UML and PROV explanations with information out of the scope of the *context*.

  Aiming at giving an insight into how the inclusion of *Output data* affects both UML representation and the resulting PROV template, Figure 49 depicts a UML representation with the *Output data* modelled as Output Parameters 5 (in this case with *return* direction, though the translation of *inout* and *out* directions would be equivalent). Figure 50 depicts its transformation into PROV. Both Figure 49 and 50 highlight the elements related to the inclusion of the *Output data* by blurring the elements coming from Figure 47 and 48, respectively.

Figure 49. UML representation from Figure 47 including Output Parameters.

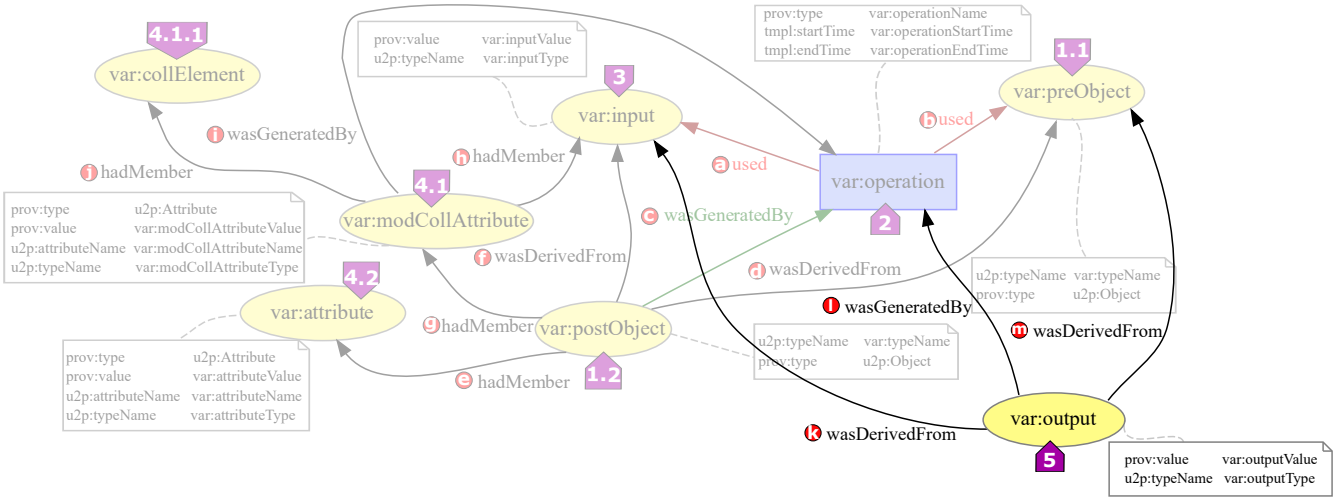

Figure 50. PROV template generated from the UML representation in Figure 49

PROV elements

UML	PROV / id	Rationale
`Output Parameters`	`prov:Entity` / `var:output`	Each parameter of `Output Parameters` is a separate `prov:Entity` identified as `var:output`.

PROV relations

k	`prov:wasDerivedFrom`	It is the construction of `var:output` based on `var:input`.
l	`prov:wasGeneratedBy`	It is the completion of production of `var:output` by `var:operation`.
m	`prov:wasDerivedFrom`	It is the construction of `var:output` based on `var:preObject`.

Attributes

PROV Element	Attribute / Value	Description
`var:output`	`prov:value` / `var:outputValue`	The value `var:outputValue` is the direct representation of `var:output`.
	`u2p:typeName` / `var:outputType`	The value `var:outputType` is a string with the name of the type of `var:output`.

References

[1] OMG, "Unified Modeling Language (UML). Version 2.5," 2015. Document formal/15-03-01, March, 2015.

[2] L. Moreau and P. Missier (eds.), "PROV-DM: The PROV Data Model," W3C Recommendation REC-prov-dm-20130430, World Wide Web Consortium, 2013.

[3] N. Kwasnikowska, L. Moreau, and J. V. D. Bussche, "A formal account of the open provenance model," *ACM Trans. Web*, vol. 9, pp. 10:1–10:44, May 2015.

[4] "PROV Graph Conventions." Last accessed March, 2019.

[5] M. Dürst and M. Suignard., "Internationalized Resource Identifiers (IRIs) (RFC 3987)." January, 2005.

[6] A. Knapp and S. Merz, "Model checking and code generation for uml state machines and collaborations," *Proc. 5th Wsh. Tools for System Design and Verification*, pp. 59–64, 2002.

[7] N. Dragan, M. L. Collard, and J. I. Maletic, "Automatic identification of class stereotypes," in *Proceedings of the 26th IEEE International Conference on Software Maintenance*, pp. 1–10, 2010.

Taxonomy of operations

Table 2 shows our taxonomy of operations, which is used to stereotype the operations that appear in the CDs patterns in order to link such an operation with the concrete behaviour given by the stereotype. This taxonomy extends the presented by Dragan et al. in [7], with additional stereotypes we have defined (they are marked with an asterisk).

Table 2. Extension of the taxonomy given in [7] showing the categories used in our proposal. Stereotypes with an asterisk denote those which extends the proposal.

Category	Stereotype name	Description
Creational	create	The operation creates an object.
	destroy	The operation destroys an object.
Structural ***Accessor***	get	The operation returns values of concrete attributes of an object.
	search*	The operation returns elements belonging to a concrete collection attribute of an object.
	process*	The operation returns values that are computed based the object's status as a whole.
	predicate	The operation returns boolean values that are computed based on concrete attributes of an object.
	property	The operation returns values (of any type) that are computed based on concrete attributes of an object.
	void-accessor	The operation, by means of a parameter, returns values (of any type) that are computed based on concrete attributes of an object.
Structural ***Mutator***	command	The operation changes the status of an object as a whole (the modified attributes are unknown or irrelevant). It does not return information.
	non-void-command	The operation changes the status of an object as a whole (the modified attributes are unknown or irrelevant). It does return information.
	set	The operation directly sets the information passed to the operation as values of concrete attributes of an object.
	modify*	The operation modifies concrete attributes of an object.
	remove*	The operation removes an element from a concrete collection attribute of an object.
	add*	The operation adds an element on a concrete collection attribute of an object.