1 Before reading

We assume that the reader is familiar with some UML diagrams: UML Sequence Diagrams, UML State Machine Diagrams, and UML Class diagrams. For further and normative information, she/he is referred to the UML specification [1]. Likewise, we assume that reader is knowledgeable in the PROV data model (PROV-DM) [2] to represent provenance information, and the PROV template approach [3] for designing provenance.

In order to give a systematic explanation of all the patterns, we have set the same structure for all the explanations. This structure is explained in Subsection 1.1. Additionally, in Subsection **??**, we give some notes to take into account before reading the patterns.

1.1 Structure of the patterns

After assigning an unique **identifier** to each of the patterns, we have structured the explanation in four blocks: **Context**, **UML Diagram**, **Mapping to PROV**, and **Discussion**. See the explanation of each block below.

Identifier Unique identifier of the transformation pattern. It is made up of an acronym that refers to the type of UML diagram together with a numeric identifier. The UML `Seq`uence diagram `P`atterns are referred as *SeqPN* where `N` is the numeric identifier. Likewise, *StPN* corresponds to the UML `St`ate Machine `P`atterns, and *ClPN* to the UML `Cl`ass Diagram `P`atterns.

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*...).

Key elements, hereinafter *identified elements*, involved in the aforementioned context will be explained in detail using also the natural language. We remark that sometimes the context may involve the same element in several situations or states; to represent this fact, we will use nested items. The structure used to describe the *identified elements* is as follows.

Identified elements

Name of the element	Description of the element using the natural language.

UML Diagram

This block will depict the UML diagram with the elements modelling the *identified elements*. To see at a glance the correspondence between each identified element and its representation in UML, there is a table in which each row depicts the representation of each *identified element* in UML, together with its rationale. In addition, the UML elements goes with a green label containing a numeric ID for ease its location in the UML diagram, and its mapping to PROV in the following block.

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

Mapping to PROV

This block contains the PROV template generated from previous *UML Diagram*, together with a detailed explanation about the transformation divided into *PROV elements*, *Attributes*, and *Relations*. 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 appear in the PROV template will be labeled with a letter that helps link each relation with its description.

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`	name of attribute / assigned value	Description of the meaning of the attribute and its value.

Relations

PROV relation	Description of the relation.

Discussion

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

2 Index of patterns

Modelled by means of UML Sequence Diagrams

Identifier of pattern	Context	Page
SeqP1	In a system, a component (the sender) interacts with another component (the recipient) by calling an operation in the recipient, and then, it continues immediately. The call causes the recipient to execute an operation.	6
SeqP2	In a system, a component (the sender) interacts with another component (the recipient) by calling an operation in the recipient and waiting for a response. The call causes the recipient to execute the operation and to respond the sender after the execution.	8
SeqP3	This pattern complements *SeqP1* and *SeqP2* contexts. During the execution of a main operation, a nested operation call is sent. After this call, the execution can continue immediately or wait for a response.	11
SeqP4	This pattern complements *SeqP1*, *SeqP2*, and *SeqP3* contexts (the last when its operation's execution waits a response after sending a nested operation call). During the execution of a main operation, it receives the response of a nested operation call. This response is used by the main operation's execution to complete its behaviour.	13

Modelled by means of UML State Machine Diagrams

Identifier of pattern	Context	Page
StP1	An operation has been executed, as a consequence an object is created and immediately reaches its first state.	17
StP2	An operation has been executed, as a consequence of this event the object's behaviour is completed.	20
StP3	An operation has been executed, as a consequence of this event the object that contains the operation switches from one state to another state.	23

Modelled by means of UML Class Diagrams

Identifier of pattern	Context	Page
ClP1	The execution of a class' operation provokes the creation of an instance of such a class, i.e. a new object.	28
ClP2	The execution of an object's operation provokes the destruction of such an object.	32
ClP3	The execution of an object's operation does not provoke the change of its current status. This execution directly returns the value of an object's attribute. It is worth noting that the returned information is not computed but it already existed before the execution.	34
ClP4	The execution of an object's operation does not provoke the change of its current status. This execution computes and returns a new value based on certain object's attributes values that are known beforehand.	37
ClP5	The execution of an object's operation does not provoke the change of its current status. This execution computes and returns a new value based on the current object's status.	40
ClP6	The execution of an object's operation provokes the change of its current status. This execution changes the values of some attributes, which are not known beforehand.	43
ClP7	The execution of an object's operation provokes the change of its current status. This execution directly sets the information passed to the operation as values of certain object's attributes that are known beforehand, without modifying the remainder attributes and without returning information.	47
ClP8	The execution of an object's operation provokes the change of its current status. This execution modifies certain object's attributes that are known beforehand, without modifying the remainder attributes of the object.	50
ClP9	The execution of an object's operation provokes the change of its current status. This execution removes element(s) from an object's attribute collection that is known beforehand, without modifying the remainder attributes of the object.	54
ClP10	The execution of an object's operation provokes the change of its current status. This execution directly adds the information passed to the operation into an object's attribute collection that is known beforehand. This behaviour does not modify the remainder attributes of the object.	58

3 UML Sequence Diagrams

Identifier *Sequence* diagram *Pattern 1* (*SeqP1*)

Context

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

Identified elements

Sender	is the component that makes the operation call.
Operation call	is the actual call that starts the execution of the operation.
Input data	is the information passed into the operation call.
Operation's execution	is the execution of the behaviour specified by the operation.

UML Diagram

Identified Element	UML	Rationale
Sender	`Lifeline` 1▶	It models the sender participant involved in the interaction.
Operation call	`Asynchronous Message` 2▶	It specifies that 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 call.
Operation's execution	`ExecutionSpecification` 4▶	It is started by `Asynchronous Message` 2▶, and shows the period of time in the recipient's `Lifeline` corresponding to the *operation's execution*.

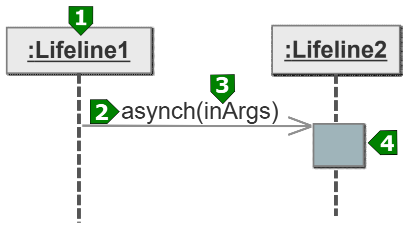
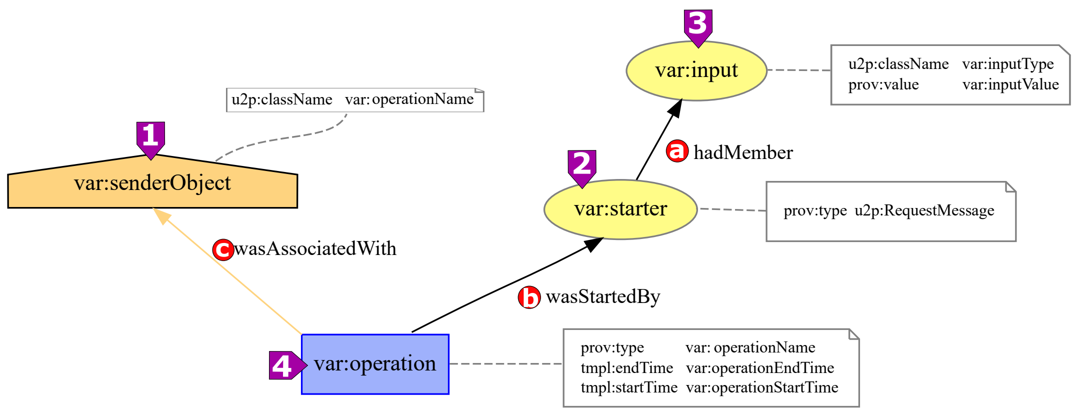


Figure 1. UML representation in *SeqP1*

Mapping to PROV

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` which assumes the responsibility for starting the *operation's execution*.
`Asynchronous Message` 2▶	`prov:Entity` 2▶ / `var:starter`	The `Asynchronous Message` 2▶ that initiates the `ExecutionSpecification` of the recipient is a `prov:Entity` with identifier `var:starter`.
`Input Arguments` 3▶	`prov:Entity` 3▶ / `var:input`	Each Argument of the `Input Arguments` 3▶ is a separated `prov:Entity` identified as `var:input`.
`ExecutionSpecification` 4▶	`prov:Activity` 4▶ / `var:operation`	The `ExecutionSpecification` 4▶ is a `prov:Activity` with identifier `var:operation`.



3
var:input
u2p:className var:inputType
prov:value var:inputValue
u2p:className var:operationName
1
var:senderObject
a hadMember
2
var:starter
prov:type u2p:RequestMessage
c wasAssociatedWith
b wasStartedBy
4
var:operation
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime

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

Attributes

PROV Element	Attribute / Value	Description
`var:senderObject` 1	`u2p:className` / `var:className`	The value `var:className` is the name of the class to which the *sender* belongs to.
`var:starter` 2	`prov:type` / `u2p:RequestMessage`	The value `u2p:RequestMessage` shows that `var:starter` 2 is a request message, i.e. `Asynchronous Message` or `Synchronous Message`.
`var:input` 3	`u2p:className` / `var:inputType`	The value `var:inputType` is the string with the name of the class to which the Argument belongs to.
	`prov:value` / `var:inputValue`	The value `var:inputValue` is the direct representation of `var:input` 3.
`var:operation` 4	`prov:type` / `var:operationName`	The value `var:operationName` is the name of the operation whose execution is modelled by the `ExecutionSpecification` 4.
	`tmpl:endTime` / `var:operationEndTime`	`var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 4.
	`tmpl:startTime` / `var:operationStartTime`	`var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 4.

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`.

Discussion

It is worth noting that Figure 2 contains the responsibility of the sender lifeline (`var:senderObject`) for executing a behaviour (`var:operation`) in a recipient `Lifeline`. However, the recipient `Lifeline` is not modelled in this PROV template, even though it is the participant which executes the behaviour. This decision is based on other patterns' better ability both (1) to better identify the participant responsible for executing that behaviour, and (2) to give a more detailed information about the implications that the execution of that behaviour has in the component.

Identifier *Sequence* diagram *Pattern 2* (*SeqP2*)

Context

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

Identified elements

Sender	is the component that makes the operation call.
Operation call	is the actual call that starts the execution of the operation.
Input data	is the information passed into the operation call.
Operation's execution	is the execution of the behaviour specified by the operation.
Response	is the recipient's response to the operation call.
Output data	is the information contained in the response.

UML Diagram

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

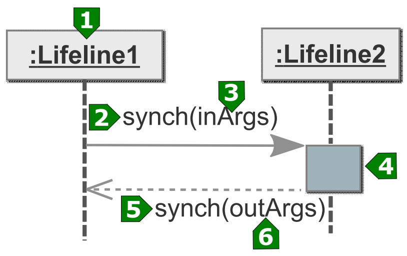
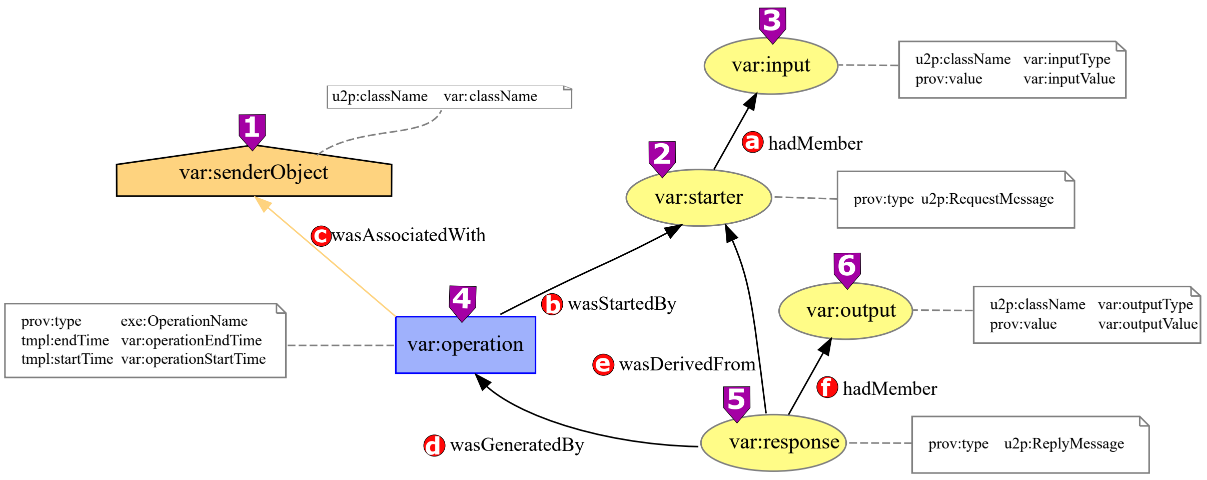


Figure 3. UML representation in *SeqP2*

Mapping to PROV



3
var:input
u2p:className var:inputType
prov:value var:inputValue
u2p:className var:className
1
var:senderObject
a hadMember
2
var:starter
prov:type u2p:RequestMessage
c wasAssociatedWith
6
var:output
u2p:className var:outputType
prov:value var:outputValue
prov:type exe:OperationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
4
var:operation
b wasStartedBy
e wasDerivedFrom
f hadMember
5
var:response
d wasGeneratedBy
prov:type u2p:ReplyMessage

Figure 4. PROV template generated from the UML diagram 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` which assumes the responsibility for starting the *operation's execution*.
`Synchronous Message` 2▶	`prov:Entity` 2▶ / `var:starter`	The `Synchronous Message` 2▶ that initiates the `ExecutionSpecification` of the recipient is a `prov:Entity` with identifier `var:starter`.
`Input Arguments` 3▶	`prov:Entity` 3▶ / `var:input`	Each Argument of the `Input Arguments` 3▶ is a separated `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 the `Output Arguments` 6▶ is a separated `prov:Entity` identified as `var:output`.

Attributes

PROV Element	Attribute / Value	Description
var:senderObject 1	u2p:className / var:className	The value var:className is the name of the class to which the *sender* belongs to.
var:starter 2	prov:type / u2p:RequestMessage	The value u2p:RequestMessage shows that var:starter 2 is a request message, i.e. Asynchronous Message or Synchronous Message.
var:input 3	u2p:className / var:inputType	The value var:inputType is the string with the name of class to which each one of the Input Arguments 3 belongs to.
	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
var:operation 4	prov:type / var:operationName	The value var:operationName is the name of the operation whose execution is modelled by the ExecutionSpecification 4.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 4.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start 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	u2p:className / var:outputType	The value var:outputType is a string with the name of classes to which Output Arguments 6 belong to.
	prov:value / var:outputValue	The value var:outputValue is the direct representation of var:output 6.

Relations

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

Discussion

It is worth noting that Figure 4 contains the responsibility of the sender lifeline (var:senderObject) for executing a behaviour (var:operation) in a recipient Lifeline. However, the recipient Lifeline is not modelled in this PROV template, even though it is the participant which executes the behaviour. This decision is based on other patterns' better ability both (1) to better identify the participant responsible for executing that behaviour, and (2) to give a more detailed information about the implications that the execution of that behaviour has in the component.

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

Context

This pattern complements *SeqP1* and *SeqP2* contexts. During the execution of a main operation, a nested operation call is sent. After this call, the execution can continue immediately or wait for a response.

Identified elements

(Main) *Operation's execution*	is the execution of the behaviour specified by the main operation. From now on *main operation's execution*.
(Nested) *Operation call*	is the nested operation call sent during the main operation's execution. From now on *nested Operation call*.

UML Diagram

Identified Element	UML	Rationale
Main operation's execution	`ExecutionSpecification` 1▶	It shows the period of time in the recipient's `Lifeline` corresponding to the *main operation's execution*.
Nested Operation call	`Asynchronous Message` 2▶ or `Synchronous Message` 2▶	It is started from the `ExecutionSpecification` 1▶. In case of a `Asynchronous Message` the sender does not wait for a response, whereas with a `Synchronous Message` the sender waits for a response.

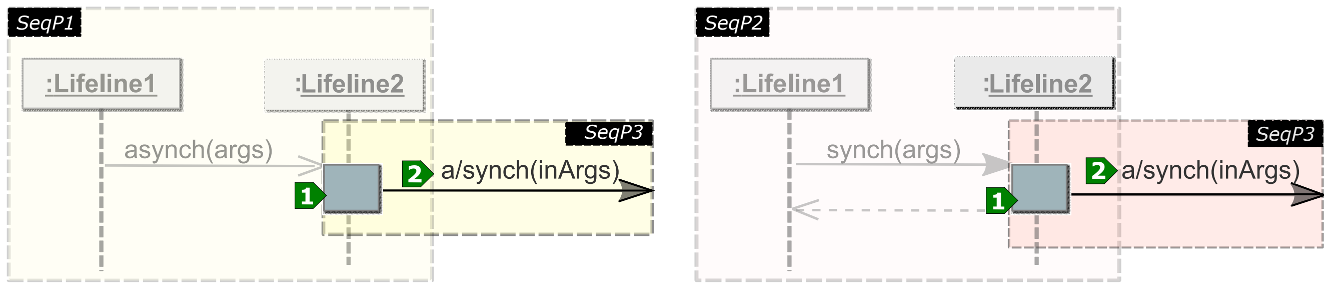


Figure 5. The left hand side is the UML representation of *SeqP1* complemented by *SeqP3*, whereas the right hand side is the UML representation of *SeqP2* complemented by *SeqP3*.

Mapping to PROV

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`.

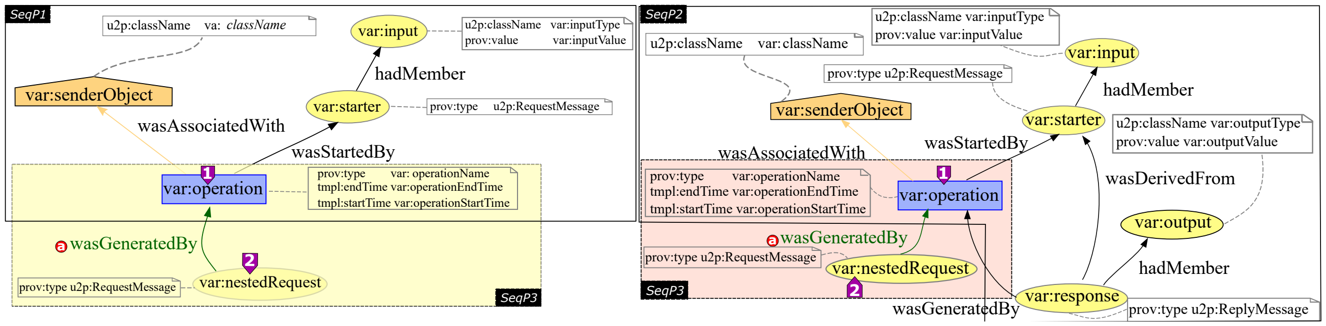

SeqP1
u2p:className va: className
var:input
u2p:className var:inputType
prov:value var:inputValue
hadMember
var:senderObject
var:starter
prov:type u2p:RequestMessage
wasAssociatedWith
wasStartedBy
1
var:operation
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
a wasGeneratedBy
2
prov:type u2p:RequestMessage
var:nestedRequest
SeqP3
SeqP2
u2p:className var:inputType
prov:value var:inputValue
u2p:className var:className
var:input
prov:type u2p:RequestMessage
hadMember
var:senderObject
var:starter
u2p:className var:outputType
prov:value var:outputValue
wasAssociatedWith
wasStartedBy
prov:type var:operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
var:operation
wasDerivedFrom
var:output
a wasGeneratedBy
prov:type u2p:RequestMessage
var:nestedRequest
hadMember
SeqP3
var:response
wasGeneratedBy
prov:type u2p:ReplyMessage

Figure 6. The left hand side is a PROV template generated from the UML diagram in the left side of Figure 5. The right hand side is a PROV template generated from the UML diagram in the right side of Figure 5.

Attributes

PROV Element	Attribute / Value	Description
`var:operation` 1	`prov:type` / `var:operationName`	The value `var:operationName` is the name of the operation whose execution is modelled by the `ExecutionSpecification` 1.
	`tmpl:endTime` / `var:operationEndTime`	The `var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 4.
	`tmpl:startTime` / `var:operationStartTime`	The `var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 4.
`var:nestedRequest` 2	`prov:type` / `u2p:RequestMessage`	The value `u2p:RequestMessage` shows that `var:response` 5 is a request message, i.e. `Asynchronous Message` or `Synchronous Message`.

Relations

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

Discussion

Section **??** explains details about how the synergies between the PROV templates allow for connecting the provenance generated after their expansion. Nevertheless, here it is given an insight about the different roles of the request message in *SeqP3* and *SeqP1* or *SeqP2* (`Asynchronous Message` or `Synchronous Message` 2). From the *SeqP3* point of view, such a request message models a nested operation call started from an `ExecutionSpecification`, which is translated into `var:nestedRequest`. Conversely, in *SeqP1* or *SeqP2* this request message is an operation call that starts the execution of an operation (i.e., the `ExecutionSpecification`), which is translated into `var:starter`. Whilst `var:nestedRequest` and `var:starter` are two different elements of type `prov:Entity` located in two different PROV templates, the values associated to them during the execution of the application will be the same and thereby the `prov:Entity` generated a after merging all the expanded PROV templates will be the same.

Identifier *Sequence diagram Pattern 4* (*SeqP4*)

Context

This pattern complements *SeqP1*, *SeqP2*, and *SeqP3* contexts (the last when its operation's execution waits a response after sending a nested operation call). During the execution of a main operation, it receives the response of a nested operation call. This response is used by the main operation's execution to complete its behaviour.

Identified elements

(Main) *Operation's execution*	is the execution of the behaviour specified by the main operation. From now on *main operation's execution*.
(Nested) *Response*	is the response to a nested operation call. From now on *nested response*.
(Main) *Response*	is the recipient's response to the main operation call. This element is only identified when this pattern complements *SeqP2*. From now on *main response*.

UML Diagram

Identified Element	UML	Rationale
Main operation's execution	`ExecutionSpecification` 1	It shows the period of time in the recipient's `Lifeline` corresponding to the *main operation's execution*.
Nested response	`Reply Message` 2	It specifies the response received in the *main operation's execution*.
Main response	`Reply Message` 3	In case of complementing *SeqP2*, it specifies the response to the main operation call that caused the start of the *main operation's execution*.

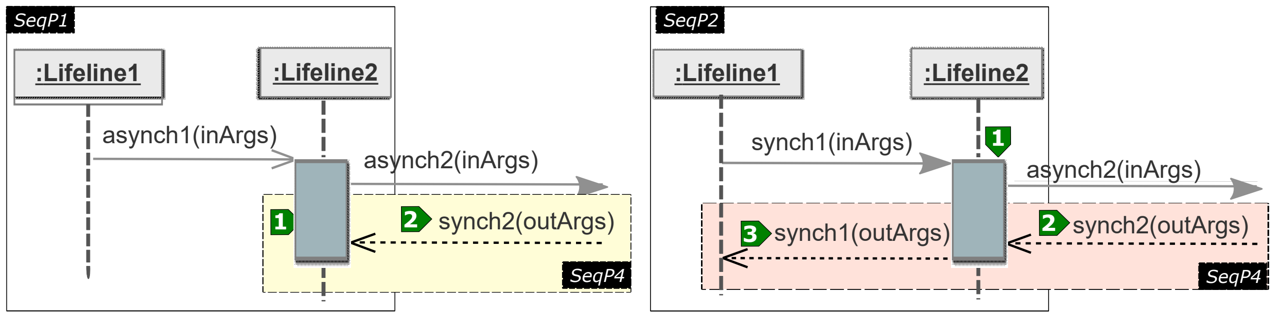


Figure 7. The left hand side is the UML representation of *SeqP1* complemented by *SeqP4*, whereas the right hand side is the UML representation of *SeqP2* complemented by *SeqP4*.

Mapping to PROV

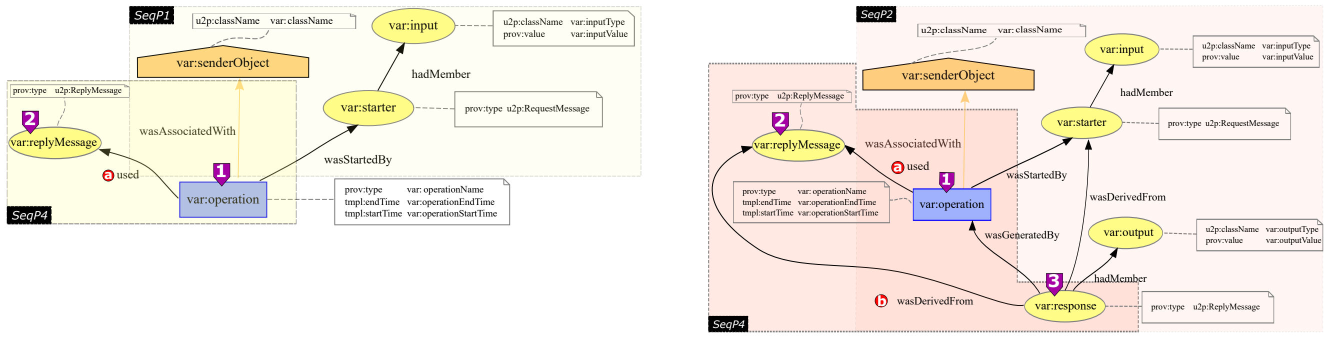

SeqP1
u2p:className var:className
var:input
u2p:className var:inputType
prov:value var:inputValue
var:senderObject
hadMember
prov:type u2p:ReplyMessage
var:starter
prov:type u2p:RequestMessage
2
var:replyMessage
wasAssociatedWith
1
a used
wasStartedBy
var:operation
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
SeqP4
SeqP2
u2p:className var:className
var:input
u2p:className var:inputType
prov:value var:inputValue
var:senderObject
hadMember
prov:type u2p:ReplyMessage
var:starter
prov:type u2p:RequestMessage
2
var:replyMessage
wasAssociatedWith
a used
1
wasStartedBy
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
var:operation
wasDerivedFrom
wasGeneratedBy
var:output
u2p:className var:outputType
prov:value var:outputValue
3
hadMember
b wasDerivedFrom
var:response
prov:type u2p:ReplyMessage
SeqP4

Figure 8. The left hand side is a PROV template generated from the UML diagram in the left side of Figure 7. The right hand side is a PROV template generated from the UML diagram in the right side of Figure 7.

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:replyMessage	The Reply Message 2 that is received in the ExecutionSpecification 1 is a prov:Entity with identifier var:replyMessage.
Reply Message 3	prov:Entity 3 / var:response	In case of complementing *SeqP2*, the Reply Message 3 that is 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 whose execution is modelled by the ExecutionSpecification 1.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 1.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 1.
var:replyMessage 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.

Relations

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

Discussion

As stated in the context, to apply this pattern, it must be assumed that the nested response is used by the main operation's execution. This use causes the relations ⓐ prov:used and ⓑ prov:wasDerivedFrom to appear in the template; the former showing that when the `ExecutionSpecification` ➊ receives the nested `Reply Message` ➋, it *utilises* that `Reply Message` ➋ to complete its behaviour; and the latter showing that the main `Reply Message` ➌ is *influenced* by the nested `Reply Message` ➋ (this last only applies if the main operation's execution is triggered by a synchronous message, i.e. when *SeqP4* complements *SeqP2*). Here we note that in case a specific scenario does not follow this assumption, 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 after expanding all the templates will contain provenance for the nested operation call and its corresponding response, though lacking the relationships ⓐ `prov:used` and ⓑ `prov:wasDerivedFrom`. We refer the reader to Section **??** for further details about how the synergies between the resulting PROV templates allow for connecting the provenance generated after their expansion.

4 UML State Machine Diagrams

Identifier *State* machine diagram *Pattern 1* (*StP1*)

Context

An operation has been executed, as a consequence an object is created and immediately reaches its first state.

Identified elements

Object's states	is the set of states that the object may undergo in the course of its lifetime.
Object's creation	is the execution of the behaviour that creates the object.
Object	is the object that is created. We have identified an object's state that is included in the *Object's states*:

First object's state The object in the state immediately after being created.

UML Diagram

Identified Element	UML	Rationale
Object's states	`StateMachine` 1	It represents the sequence of *Object's states* in which the *object* goes through during its lifetime in response to events.
Object	`Class` 2	Since the *objects* are classified attending to their characteristics and behaviour by means of `classes`, we use `Class` 2 to represent the *object*. *Note*: since `Class` lacks a graphical representation in UML State Machine diagrams, Figure 9 lacks this element.
Object's creation	`Initial Pseudostate` 3	It models the creation of the object.
First object's state	`State` 4	It models the first state of the object within the `StateMachine` 1.

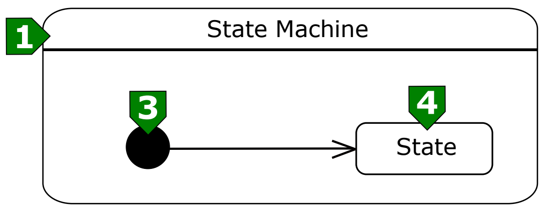


Figure 9. Excerpt of a UML State Machine diagram showing the representation in *StP1*

Mapping to PROV

PROV elements

UML	PROV / id	Rationale
`StateMachine` 1	`prov:Entity` 1 / `var:objectSM`	The `StateMachine` 1 is a `prov:Entity` identified by `var:objectSM`.
`Class` 2	`prov:Agent` 2 / `var:object`	The `Class` 2 is mapped to a `prov:Agent` identified by `var:object` which bears the *object*'s responsibilities.
`Initial Pseudostate` 3	`prov:Activity` 3 / `var:operation`	The `Initial Pseudostate` 3 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`.

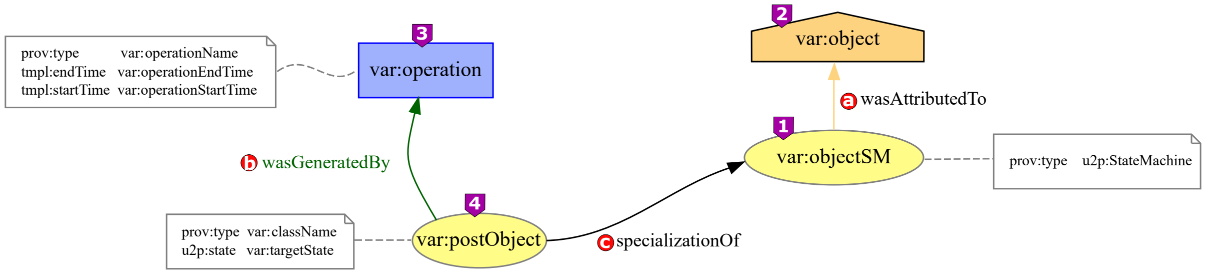

prov:type var:operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
3
var:operation
2
var:object
ⓐ wasAttributedTo
1
var:objectSM
prov:type u2p:StateMachine
ⓑ wasGeneratedBy
4
prov:type var:className
u2p:state var:targetState
var:postObject
ⓒ specializationOf

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

Attributes

PROV Element	Attribute / Value	Description
`var:objectSM` 1▸	`prov:type` / `u2p:StateMachine`	The value `u2p:StateMachine` shows that `var:objectSM` is a state machine.
`var:operation` 3▸	`prov:type` / `var:operationName`	The value `var:operationName` is the name of the operation that creates the object. This operation usually is the constructor.
	`tmpl:endTime` / `var:operationEndTime`	The `var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 3▸.
	`tmpl:startTime` / `var:operationStartTime`	The `var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 3▸.
`var:postObject` 4▸	`prov:type` / `var:className`	The value `var:className` is the name of the `Class` 2▸.
	`u2p:state` / `var:targetState`	The value `var:targetState` is a string with the name of `State` 4▸.

Relations

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

Discussion

Among all the UML elements that may be used in this pattern, the UML design in Figure 9 only contains simple states. This decision has been taken based on the possibility to reach a UML design only with simple states by flattening any UML State Machine diagram -that is, by 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 [4]. Faced with this, the user may be interested in representing the composite states directly into the PROV templates, perhaps as she/he wants a precise information about them or just because she/he does not want to flatten the State Machine diagram. Below, aiming at giving an insight into how the use of composite states is mapped to PROV, Figure 11 depicts a UML representation with the elements from Figure 9 located inside a `Composite State` 5▸, and Figure 12 depicts its transformation into PROV. Both Figure 11 and 12 highlight the elements related to the use of a composite state by blurring the elements coming from Figure 9 and Figure 10, respectively.

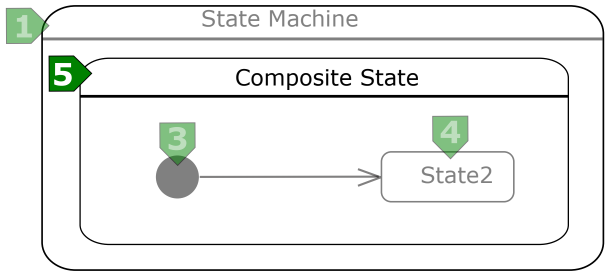


Figure 11. Excerpt of a UML State Machine diagram locating the UML elements from *StP1* in a composite state.

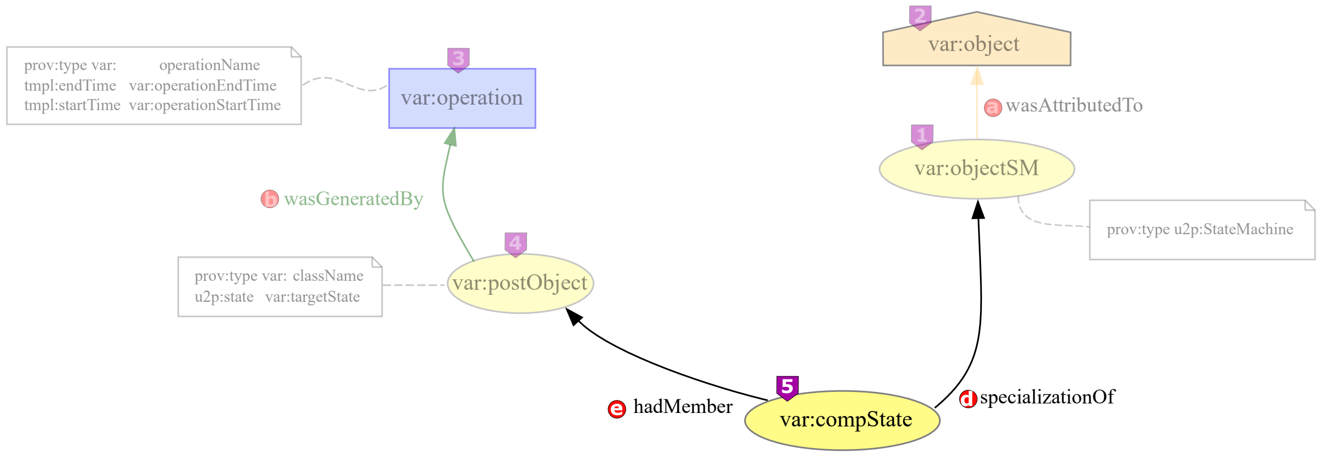


Figure 12. PROV template generated from the UML diagram in Figure 11

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

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 *S*tate machine diagram *P*attern 2 (*StP2*)

Context

An operation has been executed, as a consequence of this event the object's behaviour is completed.

Identified elements

Object's states	is the set of states that the object may undergo in the course of its lifetime.
Object	is the object to which the operation to be executed belongs. We have identified two object's states that are included in the *Object's states*:

Pre-event object	The object in the state before the execution of the behaviour given by the operation.
Post-event object	The object in the state after the execution of the behaviour given by the operation. Concretely, when the object's behaviour is completed.

Event	is the occurrence of an executed operation.

UML Diagram

Identified Element	UML	Rationale
Object's states	`StateMachine` 1	It represents the sequence of *Object's states* in which the *object* goes through during its lifetime in response to events.
Object	`Class` 2	Since the *objects* are classified attending to their characteristics and behaviour by means of `classes`, we use `Class` 2 to represent the *object*. *Note*: since `Class` lacks of a graphical representation in UML State Machine diagrams, Figure 13 lacks this element.
Pre-event object	`State` 3	It models the state of the object within the `StateMachine` 1 before the execution of the behaviour given by the operation, i.e. *pre-event object*.
Post-event object	`FinalState` 4	It models the state of the object within the `StateMachine` 1 after the execution of the behaviour given by the operation, i.e. *post-event object*.
Event	`Event` 5	It specifies an *event* that triggers the switch of states.

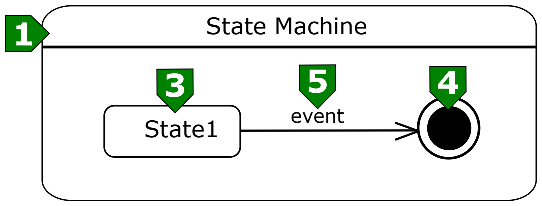


Figure 13. Excerpt of a UML State Machine diagram showing the representation in *StP2*

Mapping to PROV

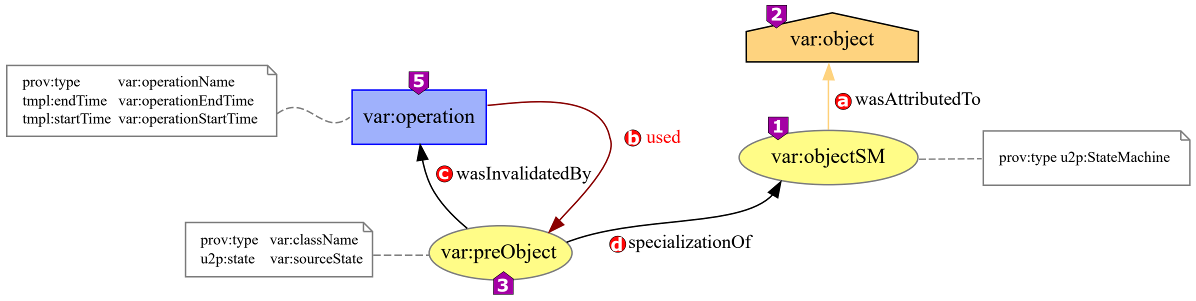

2
var:object
a wasAttributedTo
1
var:objectSM
prov:type u2p:StateMachine
5
var:operation
prov:type var:operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
b used
c wasInvalidatedBy
d specializationOf
var:preObject
3
prov:type var:className
u2p:state var:sourceState

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

PROV elements

UML	PROV / id	Rationale
`StateMachine` 1▶	`prov:Entity` 1▶ / `var:objectSM`	The `StateMachine` 1▶ is a `prov:Entity` identified by `var:objectSM`.
`Class` 2▶	`prov:Agent` 2▶ / `var:object`	The `Class` 2▶ is mapped to a `prov:Agent` identified by `var:object` which bears the *object*'s responsibilities.
`State` 3▶	`prov:Entity` 3▶ / `var:preObject`	The `State` 3▶ is a `prov:Entity` identified by `var:preObject`.
`FinalState` 4▶	None /	There is no mapping. For details, see discussion.
`Event` 5▶	`prov:Activity` 5▶ / `var:operation`	The `Event` 5▶ represents the occurrence of an executed operation. Such an execution is a `prov:Activity` with the identifier `var:operation`.

Attributes

PROV Element	Attribute / Value	Description
`var:objectSM` 1▶	`prov:type` / `u2p:StateMachine`	The value `u2p:StateMachine` shows that `var:objectSM` is a state machine.
`var:preObject` 3▶	`prov:type` / `var:className`	The value `var:className` is the name of the `Class` 2▶.
	`u2p:state` / `var:sourceState`	The value `var:sourceState` is a string with the name of `State` 3▶.
`var:operation` 5▶	`prov:type` / `var:operationName`	The value `var:operationName` is the name of the operation that completes the object's behaviour.
	`tmpl:endTime` / `var:operationEndTime`	The `var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 5▶.
	`tmpl:startTime` / `var:operationStartTime`	The `var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 5▶.

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 semantic (the completion of the object's behaviour) is already included in the PROV template. Concretely, it is given by (1) the relation `prov:wasInvalidatedBy` showing that `var:preObject` is not longer available, and (2) the lack of a `prov:Entity` derived from `var:preObject` that shows the object in a target state. Since the completion of the object's behaviour usually goes with its destruction, this patterns is consistent with *ClP2*

Among all the types of states that may be used in this pattern, the UML design in Figure 9 only contains simple states. This decision has been taken based on the possibility to generate a UML design only with simple states by flattening any UML State Machine diagram -that is, by removing composite and submachine states. In fact, to flatten State Machine diagrams is a very common approach in contexts such as model checking and code generation [4]. Faced with this, the user may be interested in representing the composite states directly into the PROV templates, perhaps as she/he wants a precise information about them or just because she/he is not willing to flatten the State Machine diagram. Below, aiming at giving an insight into how the use of composite states is mapped to PROV, Figure 11 depicts a UML representation with the elements from Figure 9 located inside a `Composite State` 6, and Figure 12 depicts its transformation into PROV. Both Figure 11 and 12 highlight the elements related to the use of a composite state by blurring the elements coming from Figure 9 and Figure 10, respectively.

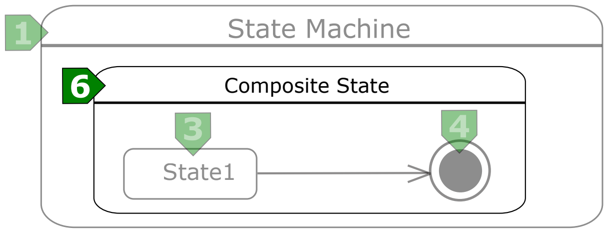


Figure 15. Excerpt of a UML State Machine diagram locating the UML elements from *StP2* in a composite state.

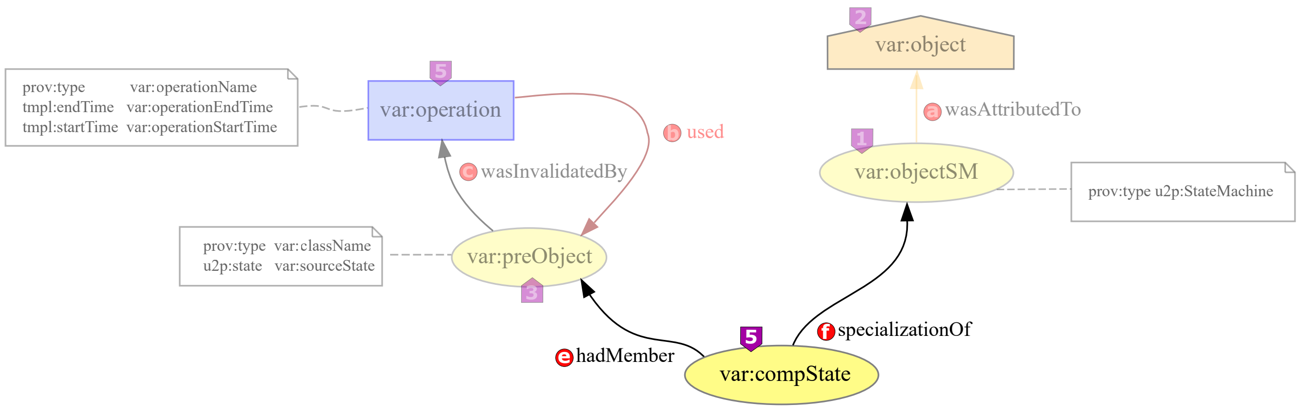


Figure 16. PROV template generated from the UML diagram in Figure 15

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

Relations

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

Identifier *S*tate machine diagram *P*attern *3* (*StP3*)

Context

An operation has been executed, as a consequence of this event the object that contains the operation switches from one state to another state.

Identified elements

Object's states	is the set of states that the object may undergo in the course of its lifetime.
Object	is the object to which the operation to be executed belongs. We have identified two states of the object that are included in the *Object's states*:
	Pre-event object — The object in the state before the execution of the behaviour given by the operation.
	Post-event object — The object in the state after the execution of the behaviour given by the operation.
Event	is the occurrence of an executed operation.

UML Diagram

Identified Element	UML	Rationale
Object's states	`StateMachine` 1	It represents the sequence of *Object's states* in which the *object* goes through during its lifetime in response to events.
Object	`Class` 2	Since the *objects* are classified attending to their characteristics and behaviour by means of `classes`, we use `Class` 2 to represent the *object*. *Note*: since a `Class` lacks a graphical representation in UML State Machine diagrams, Figure 17 lacks this element.
Pre-event object	`State` 3	It models the state of the object within the `StateMachine` 1 before the execution of the behaviour given by the operation, i.e., *pre-event object*.
Post-event object	`State` 4	It models the state of the object within the `StateMachine` 1 after the execution of the behaviour given by the operation, i.e., *post-event object*.
Event	`Event` 5	It specifies an *event* that triggers the switch of states.

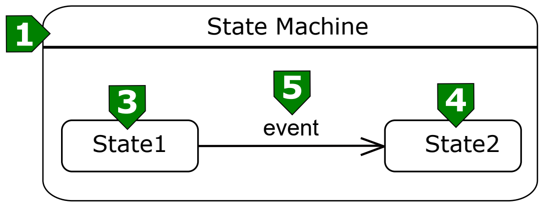


Figure 17. Excerpt of a UML State Machine diagram showing the representation in *StP3*

Mapping to PROV

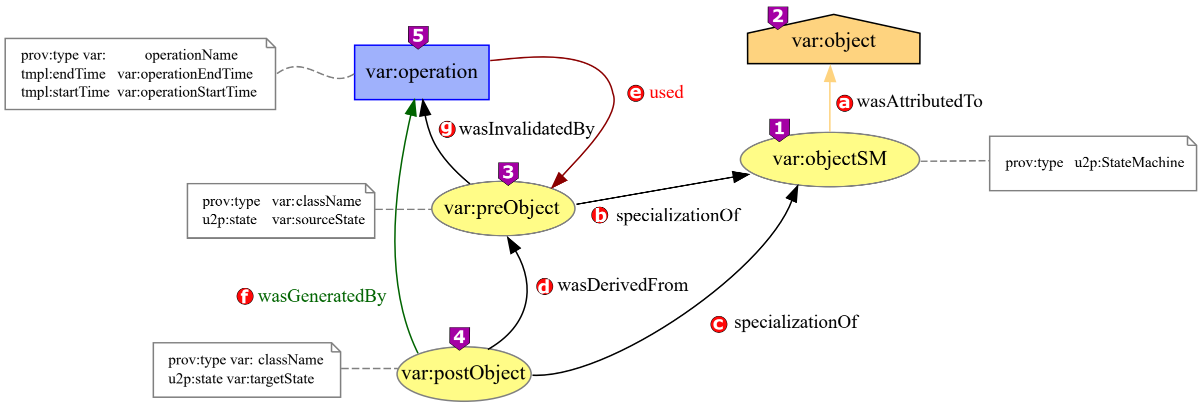


Figure 18. PROV template generated from the UML diagram in Figure 17

PROV elements

UML	PROV / id	Rationale
`StateMachine` 1	`prov:Entity` 1 / `var:objectSM`	The `StateMachine` 1 is a `prov:Entity` identified by `var:objectSM`.
`Class` 2	`prov:Agent` 2 / `var:object`	The `Class` 2 is mapped to a `prov:Agent` identified by `var:object` which bears the *object*'s responsibilities.
`State` 3	`prov:Entity` 3 / `var:preObject`	The `State` 3 is a `prov:Entity` identified by `var:preObject`.
`State` 4	`prov:Entity` 4 / `var:postObject`	The `State` 4 is a `prov:Entity` identified by `var:postObject`.
`Event` 5	`prov:Activity` 5 / `var:operation`	The `Event` 5 represents the occurrence of an executed operation. Such an execution is a `prov:Activity` with the identifier `var:operation`.

Attributes

PROV Element	Attribute / Value	Description
`var:objectSM` 1	`prov:type` / `u2p:StateMachine`	The value `u2p:StateMachine` shows that `var:objectSM` is a state machine.
`var:preObject` 3	`prov:type` / `var:className`	The value `var:className` is the name of the `Class` 2.
	`u2p:state` / `var:sourceState`	The value `var:sourceState` is a string with the name of `State` 3.
`var:postObject` 4	`prov:type` / `var:className`	The value `var:className` is the name of the `Class` 2.
	`u2p:state` / `var:targetState`	The value `var:targetState` is a string with the name of `State` 4.
`var:operation` 5	`prov:type` / `var:operationName`	The value `var:operationName` is the name of the operation whose execution is specified by `Event` 5.
	`tmpl:endTime` / `var:operationEndTime`	The `var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 5.
	`tmpl:startTime` / `var:operationStartTime`	The `var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 5.

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

Among all the UML elements that may be used in this pattern, the UML design in Figure 9 only contains simple states. This decision has been taken based on the possibility to reach a UML design only with simple states by flattening any UML State Machine diagram -that is, by 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 [4]. Faced with this, the user may be interested in representing the composite states directly into the PROV templates, perhaps as she/he wants a precise information about them or just because she/he does not want to flatten the State Machine diagram. Below, aiming at giving an insight into how the use of composite states is mapped to PROV, Figure 11 depicts a UML representation with the elements from Figure 9 located inside a `Composite State` 6, and Figure 12 depicts its transformation into PROV. Both Figure 11 and 12 highlight the elements related to the use of a composite state by blurring the elements coming from Figure 9 and Figure 10, respectively.

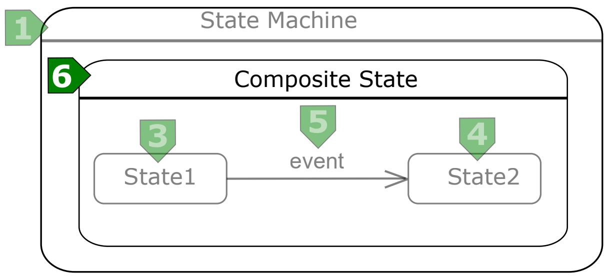


Figure 19. Excerpt of a UML State Machine diagram locating the UML elements from *StP3* in a composite state.

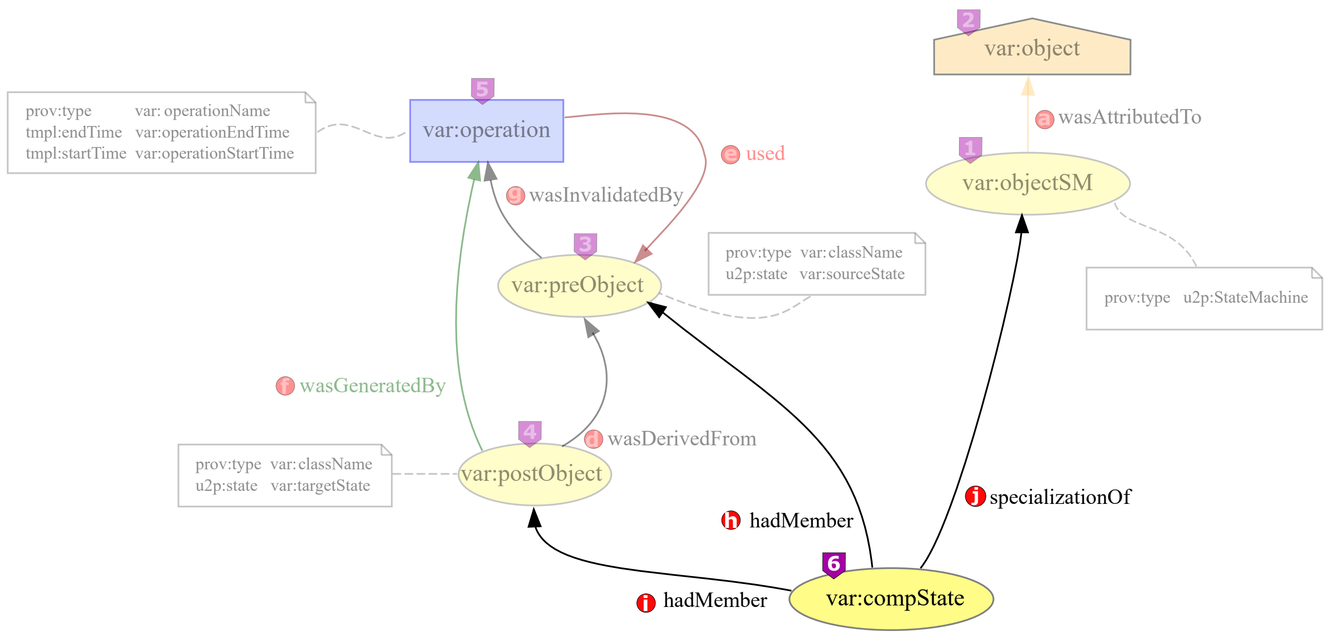


Figure 20. PROV template generated from the UML diagram in Figure 19

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

Relations

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

5 UML Class Diagrams

Identifier *Class* diagram *Pattern 1* (*ClP1*)

Context

The execution of a class' operation provokes the creation of an instance of such a class, i.e. a new object.

Identified elements

Object	is the object to which the operation to be executed belongs. We have identified two object's status (i.e., states).
	Pre-operation object — The object in the status before the execution of the operation.
	Post-operation object — The object in the status after the execution of the operation.
Operation's execution	is the execution of the behaviour specified by the operation.
Input data	is the information (values) passed into the invocation of the behaviour.
Object's attributes	are the characteristics of the *object*. As a consequence of the *operation's execution*, values are assigned to these attributes. Thus, we have identified:
	Attributes values — The values of the *object's attributes* after the execution of the operation.

UML Diagram

Identified Element	UML	Rationale
Object	`Class` 1	The *objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use a `Class` 1 to represent the *object* both before (*pre-operation object*) and after (*post-operation object*) the execution of the behaviour given by the operation.
Operation's execution	`Operation` 2 «`create`»	It represents the behaviour given by the operation. Additionally, it is associated with the stereotype «`create`» to denote that the behaviour creates a new instance of the class, i.e. an object.
Input data	`Input Parameters` 3	They depict the information passed into the operation for its execution.
Object's attributes	`Attributes` 4	They represent the characteristics of the *object*.

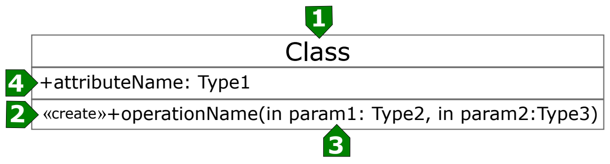


Figure 21. UML representation in *ClP1*

Mapping to PROV

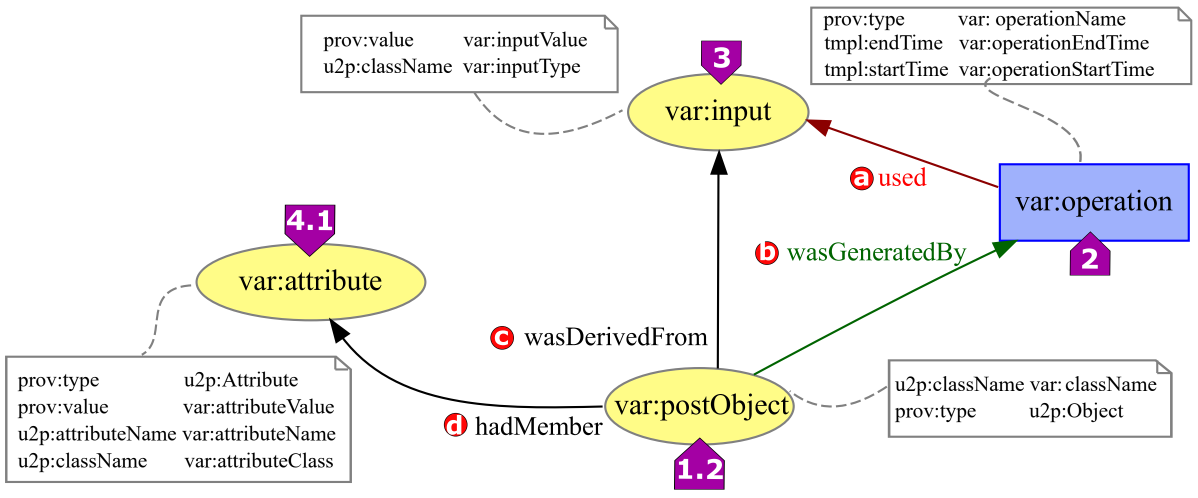

prov:value var:inputValue
u2p:className var:inputType
3
var:input
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
a used
var:operation
2
b wasGeneratedBy
4.1
var:attribute
c wasDerivedFrom
prov:type u2p:Attribute
prov:value var:attributeValue
u2p:attributeName var:attributeName
u2p:className var:attributeClass
d hadMember
var:postObject
1.2
u2p:className var: className
prov:type u2p:Object

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

PROV elements

UML	PROV / id	Rationale
`Class` 1	None /	The *pre-operation object*, i.e. the object in the status before the execution of the operation, is not mapped to PROV. For details, see discussion.
	`prov:Entity` 1.2 / `var:postObject`	The *post-operation object*, i.e. the object in the status after the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:postObject`.
`Operation` 2 «`create`»	`prov:Activity` 2 / `var:operation`	The `Operation` 2, stereotyped with «`create`», models the execution of the behaviour given by the invoked operation. Thus, it is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entiy` 3 / `var:input`	Each parameter of the `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Attributes` 4	`prov:Entity` 4.1 / `var:attribute`	Each attribute of the `Attributes` 4, depicting the characteristics of the *object* after the execution of the operation (i.e., *attributes values*), is a separate `prov:Entity` with identifier `var:attribute`.

Attributes

PROV Element	Attribute / Value	Description
`var:postObject` 1.2	`u2p:className` / `var:className`	The value `var:className` is the name of the `Class` 1.
	`prov:type` / `u2p:Object`	The value `u2p:Object` shows that `var:postObject` is an object.
`Operation` 2	`prov:type` / `var:operationName`	The value `u2p:operationName` is the name of the `Operation` 2.
	`tmpl:endTime` / `var:operationEndTime`	The `var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 4.
	`tmpl:startTime` / `var:operationStartTime`	The `var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 2.
`var:input` 3	`u2p:className` / `var:inputType`	The value `var:inputType` is a string with the name of the class to which the parameter belongs to.
	`prov:value` / `var:inputValue`	The value `var:inputValue` is the direct representation of `var:input` 3.
`var:attribute` 4.1	`prov:type` / `u2p:Attribute`	The value `u2p:Attribute` shows that `var:attribute` is an attribute.
	`prov:value` / `var:attributeValue`	The value `var:attributeValue` is the direct representation of the attribute.
	`u2p:attributeName` / `var:attributeName`	The value `var:attributeName` is the name of the attribute.
	`u2p:className` / `var:attributeClass`	The value `var:attributeClass` is a string with the name of the class to which the attribute belongs to.

Relations

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

Discussion

Since the object is created after the operation's execution, the object in the status before the execution (*pre-operation object*) can be considered as "non-existent. For this reason, the PROV template only includes the object in the status after the execution of the operation (`var:postObject`).

Identifier *Class diagram Pattern 2* (*ClP2*)

Context

The execution of an object's operation provokes the destruction of such an object.

Identified elements

Object	is the object to which the operation to be executed belongs. We have identified one object's status:	
	Pre-operation object	The object in the status before the execution of the operation.
	Post-operation object	The object in the status after the execution of the operation.
Operation's execution	is the execution of the behaviour specified by the operation.	

UML Diagram

Identified Element	UML	Rationale
Object	`Class` 1	The *objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use a `Class` 1 to represent the *object* both before (*pre-operation object*) and after (*post-operation object*) the execution of the operation.
Operation's execution	`Operation` 2 «`destroy`»	It represents the behaviour given by the operation. Additionally, it is associated with the stereotype «`destroy`» to denote that the behaviour destroys the object.

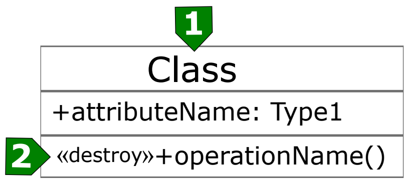
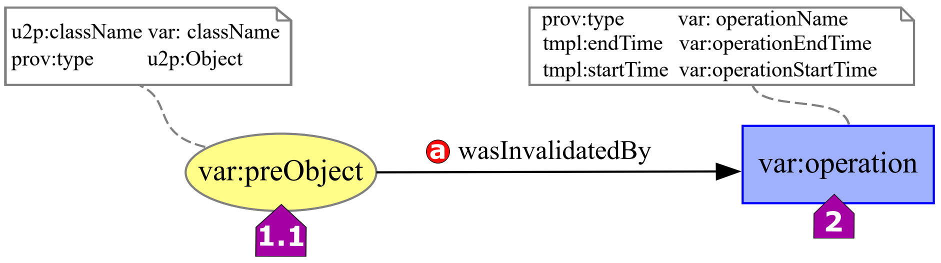


Figure 23. UML representation in *ClP2*

Mapping to PROV



u2p:className var: className
prov:type u2p:Object
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
var:preObject
a wasInvalidatedBy
var:operation
1.1
2

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

PROV elements

UML	PROV / id	Rationale
`Class` 1▶	`prov:Entity` 1.1▶ / `var:preObject`	The *pre-operation object*, i.e. the object in the status before the execution of the operation, which is represented by `Class` 1▶, is a `prov:Entity` identified as `var:preObject`.
	None /	The *post-operation object*, i.e. the object in the status after the execution of the operation, is not mapped to PROV. For details, see discussion.
`Operation` 2▶ `«destroy»`	`prov:Activity` 2▶ / `var:operation`	The `Operation` 2▶, stereotyped with `«destroy»`, models the execution of the behaviour given by the invoked operation. Thus, it is a `prov:Activity` identified by `var:operation`.

Attributes

PROV Element	Attribute / Value	Description
`var:preObject` 1.1▶	`u2p:className` / `var:className`	The value `var:className` is the name of the `Class` 1▶.
	`prov:type` / `u2p:Object`	The value `u2p:Object` shows that `var:preObject` is an object.
`Operation` 2▶	`prov:type` / `var:operationName`	The value `var:operationName` is the name of the `Operation` 2▶.
	`tmpl:endTime` / `var:operationEndTime`	The `var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 4▶.
	`tmpl:startTime` / `var:operationStartTime`	The `var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 2▶.

Relations

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

Discussion

We have decided not to map the object in the status after the execution (*post-operation object*) in the PROV template due to the fact that its semantic is already included in the PROV template. Concretely, it is given by (1) the relation `prov:wasInvalidatedBy` showing that `var:preObject` is not longer available, and (2) the lack of a `prov:Entity` derived from `var:preObject` that shows the object in a status after the operation's execution. Since the destruction of an object usually means the completion of its behaviour, we remark that this patterns is consistent with *StP2*.

Identifier *Class diagram Pattern 3* (*ClP3*)

Context

The execution of an object's operation does not provoke the change of its current status. This execution directly returns the value of an object's attribute. It is worth noting that the returned information is not computed but it already existed before the execution.

Identified elements

Object	is the object to which the operation to be executed belongs. Since the object's status does not change, we have identified one object's status: *Pre/Post-operation object* The object in the status before and after the execution of the operation.
Operation's execution	is the execution of the behaviour specified by the operation.
Input data	is the information (values) passed into the invocation of the behaviour.
Output data	is the information (values) passed out to the execution of the behaviour.

UML Diagram

Identified Element	UML	Rationale
Object	`Class` 1	The *objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use a `Class` 1 to represent the *pre/post-operation object*.
Operation's execution	`Operation` 2 «`get`» / «`search`»	It represents the behaviour given by the operation. Additionally, it is associated with the stereotype «`get`» or «`search`» to model that the information passed out to the operation's behaviour is an attribute («`get`»), or an element in a collection attribute («`search`»).
Input data	`Input Parameters` 3	They depict the information passed into the operation for its execution.
Output data	`Output Parameters` 4	They depict the information passed out to the execution of the operation.

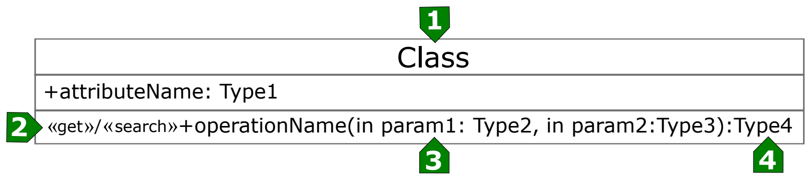

1
Class
+attributeName: Type1
2
«get»/«search»+operationName(in param1: Type2, in param2:Type3):Type4
3
4

Figure 25. UML representation in *ClP3*

Mapping to PROV

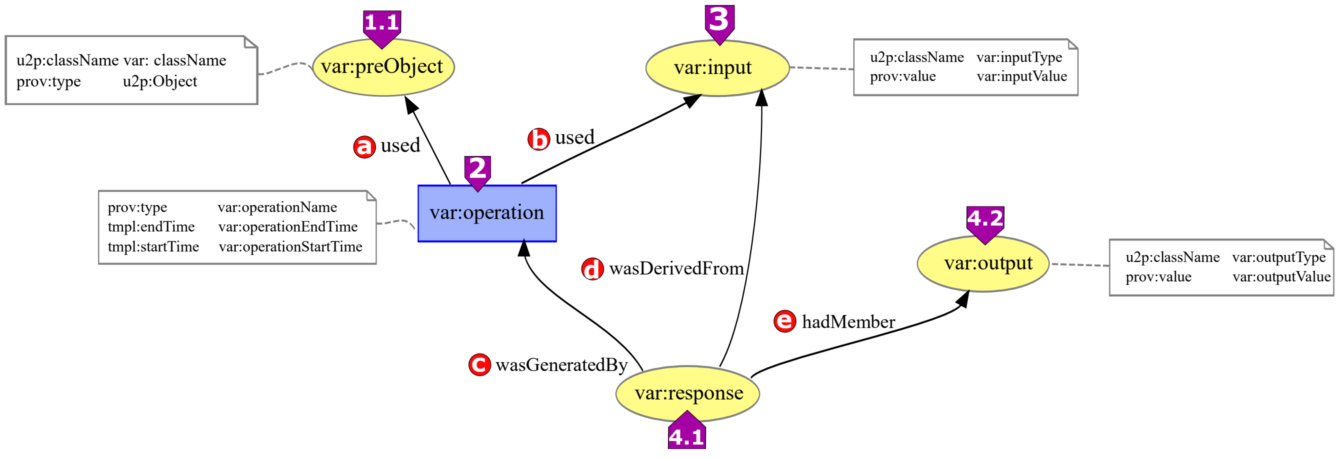

1.1
var:preObject
u2p:className var: className
prov:type u2p:Object
3
var:input
u2p:className var:inputType
prov:value var:inputValue
a used
b used
2
var:operation
prov:type var:operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
d wasDerivedFrom
4.2
var:output
u2p:className var:outputType
prov:value var:outputValue
e hadMember
c wasGeneratedBy
var:response
4.1

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

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1.1 / `var:preObject`	The *pre/post-operation object*, i.e. the object in the status before and after the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:preObject`.
`Operation` 2 «`get`» / «`search`»	`prov:Activity` 2 / `var:operation`	The `Operation` 2, stereotyped with «`get`» or «`search`», models the execution of the behaviour given by the invoked the operation. Thus, it is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entity` 3 / `var:input`	Each parameter of the `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Output Parameters` 4	`prov:Entity` 4.1 / `var:response`	The set of information passed out to the operation's execution is a `prov:Entity` identified by `var:response`.
	`prov:Entity` 4.2 / `var:output`	Each parameter of the `Output Parameters` 4 is a separate `prov:Entity` identified as `var:output`.

Attributes

PROV Element	Attribute / Value	Description
`var:preObject` 1.1	`u2p:className` / `var:className`	The value `var:className` is the name of the `Class` 1.
	`prov:type` / `u2p:Object`	The value `u2p:Object` shows that `var:preObject` is an object.
`var:operation` 2	`prov:type` / `var:operationName`	The value `var:operationName` is the name of the `Operation` 2.
	`tmpl:endTime` / `var:operationEndTime`	The `var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 4.
	`tmpl:startTime` / `var:operationStartTime`	The `var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 4.
`var:input` 3	`u2p:className` / `var:inputType`	The value `var:inputType` is a string with the name of the class to which the parameter belongs to.
	`prov:value` / `var:inputValue`	The value `var:inputValue` is the direct representation of `var:input` 3.
`var:output` 4.2	`u2p:className` / `var:outputType`	The value `var:outputType` is a string with the name of the class to which the parameter belongs to.
	`prov:value` / `var:outputValue`	The value `var:outputValue` is the direct representation of `var:output` 3.

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

Output data (`var:output`) is information not computed, it already existed before the execution. Thus, there is not a `prov:wasGeneratedBy` relationship between `var:operation` and `var:output`. Instead, there is a `prov:Entity` identified by `var:response` that is related to (1) `var:operation` by means of `prov:wasGeneratedBy`, and (2) `var:output` through `prov:hadMember`. This decision has been made so as to be consequent with the strategy followed in *SeqP1*-*SeqP4*. These patterns, which are related to Sequence Diagrams, also passed information out to the execution.

Identifier *Class* diagram *Pattern 4* (*ClP4*)

Context

The execution of an object's operation does not provoke the change of its current status. This execution computes and returns a new value based on certain object's attributes values that are known beforehand.

Identified elements

Object	is the object to which the operation to be executed belongs. Since the object's status does not change, we have identified one object's status:
	Pre/Post-operation object The object in the status before and after the execution of the operation.
Operation's execution	is the execution of the behaviour specified by the operation.
Input data	is the information (values) passed into the invocation of the behaviour.
Output data	is the information (values) passed out to the execution of the behaviour.
Object's attributes	are the characteristics of the *object*. We have identified:
	Source attributes values The values of the attributes used to compute the output information.

UML Diagram

Identified Element	UML	Rationale
Object	`Class` 1	The *objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use a `Class` 1 to represent the *pre/post-operation object*.
Operation's execution	`Operation` 2 «`predicate`»/ «`property`»/ «`void-accessor`»	It represents the behaviour given by the operation. Additionally, it is associated with the stereotype «`predicate`», «`property`», or «`void-accessor`» to model that the information passed out to the operation's behaviour is a computed value based on an object's attribute value.
Input data	`Input Parameters` 3	They depict the information passed into the operation for its execution.
Output data	`Output Parameters` 4	They depict the information passed out to the execution of the operation.
Object's attributes	`Attributes` 5	They represent the characteristics of the *object*.

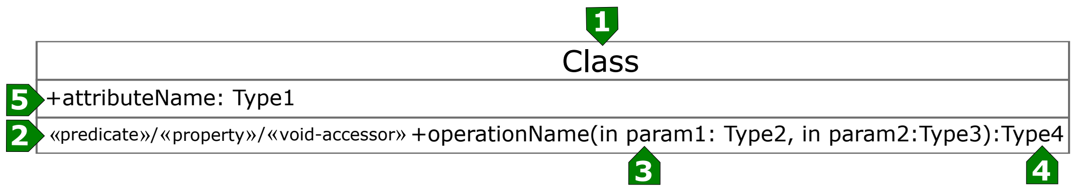
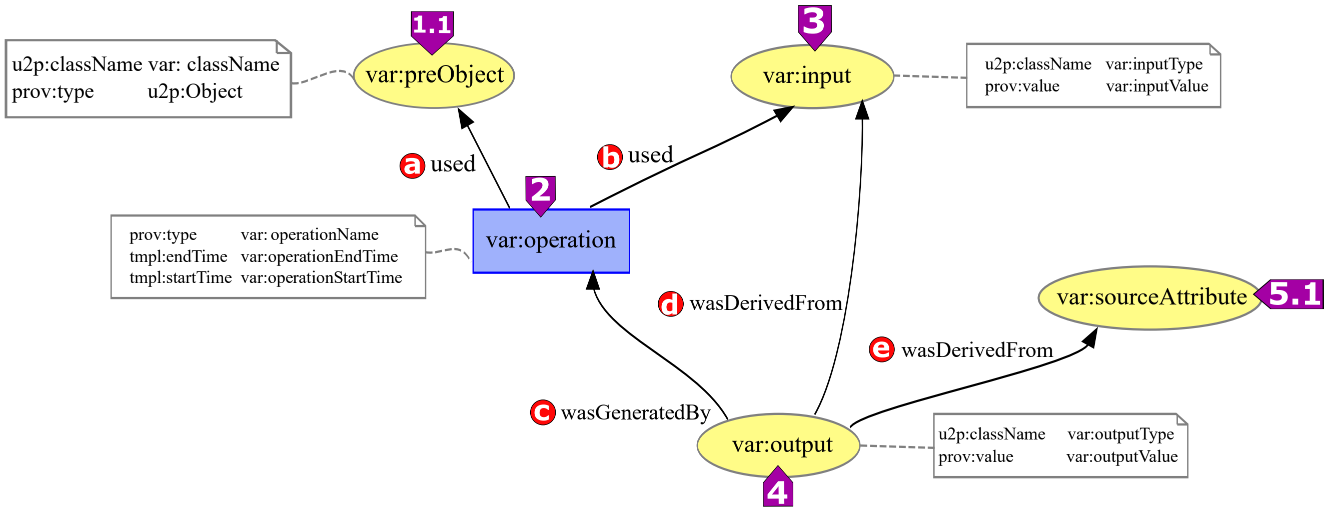

1
Class
5
+attributeName: Type1
2
«predicate»/«property»/«void-accessor» +operationName(in param1: Type2, in param2:Type3):Type4
3
4

Figure 27. UML representation in *ClP4*

Mapping to PROV



1.1
u2p:className var: className
prov:type u2p:Object
var:preObject
3
var:input
u2p:className var:inputType
prov:value var:inputValue
a used
b used
2
var:operation
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
d wasDerivedFrom
var:sourceAttribute
5.1
e wasDerivedFrom
c wasGeneratedBy
var:output
u2p:className var:outputType
prov:value var:outputValue
4

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

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1.1 / `var:preObject`	The *pre/post-operation object*, i.e. the object in the status before and after the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:preObject`.
`Operation` 2 `«predicate»/` `«property»/` `«void-accessor»`	`prov:Activity` 2 / `var:operation`	The `Operation` 2, stereotyped with `«predicate»`, `«property»` or `«void-accessor»`, models the execution of the behaviour given by the invoked operation. Thus, it is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entity` 3 / `var:input`	Each parameter of the `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Output Parameters` 4	`prov:Entity` 4 / `var:output`	Each parameter of the `Output Parameters` 4 is a separate `prov:Entity` identified as `var:output`.
`Attributes` 5	`prov:Entity` 5.1 / `var:sourceAttribute`	Each attribute of the `Attributes` 5 that corresponds to any of the *source attributes values* is a separate `prov:Entity` identified by `var:sourceAttribute`.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1.1	u2p:className / var:className	The value var:className is the name of the Class 1.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the Operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
var:input 3	u2p:className / var:inputType	The value var:inputType is a string with the name of the class to which the parameter belongs to.
	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
var:output 4	u2p:className / var:outputType	The value var:outputType is a string with the name of the class to which the parameter belongs to.
	prov:value / var:outputValue	The value var:outputValue is the direct representation of var:output 4.

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; however, all of them share that they compute *output data* based on object's attribute(s) without modifying the object's status. This pattern is focused on these common facts, representing them with the aforementioned PROV template. Although there are no distinction in the PROV templates generated from these stereotypes, some of their specific nuances will be included in the provenance through the values assigned to the template's variables. For instance, «predicate» defines the *output data* as Boolean, which is included in the provenance through the value assigned to var:outputType in var:output.

Identifier *Class diagram Pattern 5* (*ClP5*)

Context

The execution of an object's operation does not provoke the change of its current status. This execution computes and returns a new value based on the current object's status.

Identified elements

Object	is the object to which the operation to be executed belongs. Since the object's status does not change, we have identified one object's status: *Pre/Post-operation object* The object in the status before and after the execution of the operation.
Operation's execution	is the execution of the behaviour specified by the operation.
Input data	is the information (values) passed into the invocation of the behaviour.
Output data	is the information (values) passed out to the execution of the behaviour.

UML Diagram

Identified Element	UML	Rationale
Object	`Class` 1	The *objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use a `Class` 1 to represent the *pre/post-operation object*.
Operation's execution	`Operation` 2 «`process`»	It represents the behaviour given by the operation. Additionally, it is associated with the stereotype «`process`» to model that the information passed out to the operation's behaviour is a computed value based on an object's current status.
Input data	`Input Parameters` 3	They depict the information passed into the operation for its execution.
Output data	`Output Parameters` 4	They depict the information passed out to the execution of the operation.

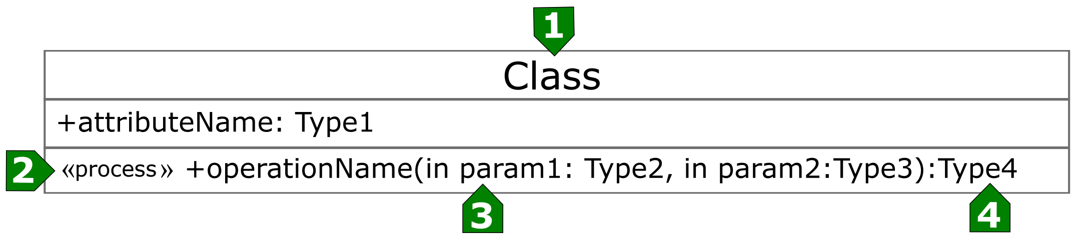
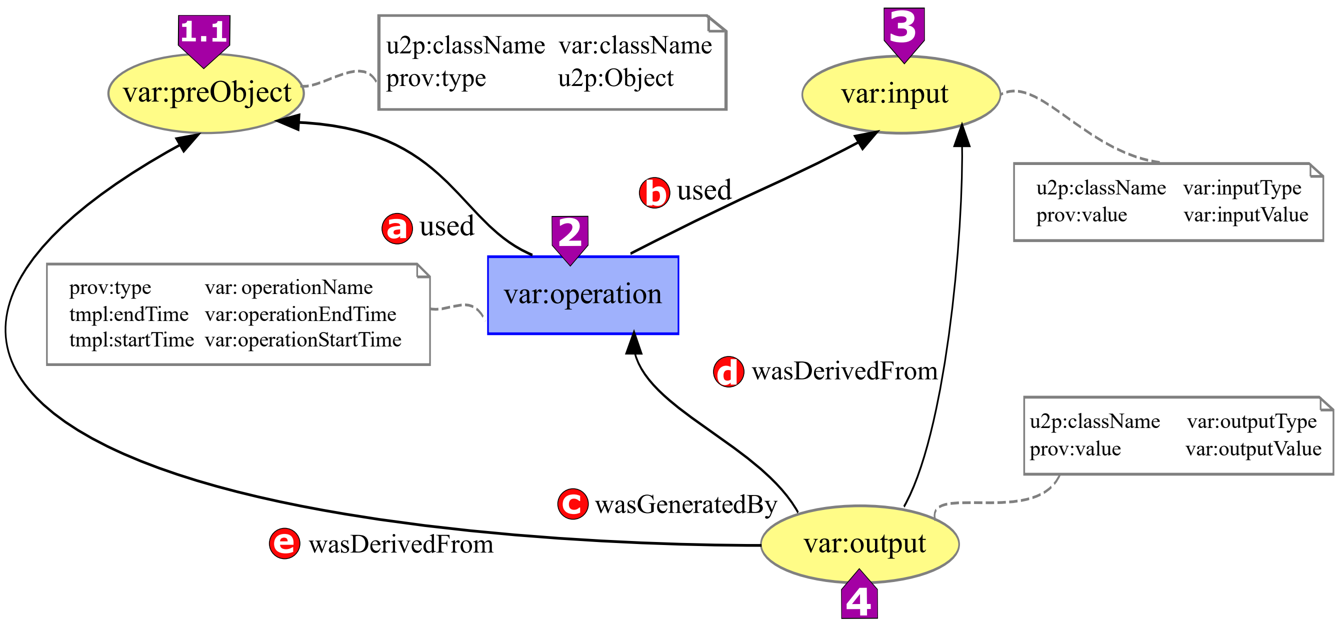

1
Class
+attributeName: Type1
2
«process» +operationName(in param1: Type2, in param2:Type3):Type4
3
4

Figure 29. UML representation in *ClP5*

Mapping to PROV



1.1
var:preObject
u2p:className var:className
prov:type u2p:Object
3
var:input
u2p:className var:inputType
prov:value var:inputValue
a used
b used
2
var:operation
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
d wasDerivedFrom
u2p:className var:outputType
prov:value var:outputValue
c wasGeneratedBy
e wasDerivedFrom
var:output
4

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

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1.1 / `var:preObject`	The *pre/post-operation object*, i.e. the object in the status before and after the execution of the operation, which is represented by `Class` 1, is a `prov:Entity` identified as `var:preObject`.
`Operation` 2 `«process»`	`prov:Activity` 2 / `var:operation`	The `Operation` 2, stereotyped with `«process»`, models the execution of the behaviour given by the invoked operation. Thus, it is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entity` 3 / `var:input`	Each parameter of the `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Output Parameters` 4	`prov:Entity` 4 / `var:output`	Each parameter of the `Output Parameters` 4 is a separate `prov:Entity` identified as `var:output`.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1.1	u2p:className / var:className	The value var:className is the name of the Class 1.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject is an object.
var:operation 2	prov:type / var:operationName	The value var:operationName is the name of the Operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 2.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 2.
var:input 3	u2p:className / var:inputType	The value var:inputType is a string with the name of the class to which the parameter belongs to.
	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
var:output 4	u2p:className / var:outputType	The value var:outputType is a string with the name of the class to which the parameter belongs to.
	prov:value / var:outputValue	The value var:outputValue is the direct representation of var:output 4.

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

Unlike *ClP4*, this pattern computes information based on the whole object's status, it does not specify any object's attribute as source. Thus, var:output (computed information) is directly related to var:preObject (object's status) by means of prov:wasDerivedFrom.

Identifier *Class* diagram *Pattern 6* (*ClP6*)

Context

The execution of an object's operation provokes the change of its current status. This execution changes the values of some attributes, which are not known beforehand.

Identified elements

Object	is the object to which the operation to be executed belongs. Since the object's status changes, we have identified two object's status:
	Pre-operation object — The object in the status before the execution of the operation.
	Post-operation object — The object in the status after the execution of the operation.
Operation's execution	is the execution of the behaviour specified by the operation.
Input data	is the information (values) passed into the invocation of the behaviour.
Output data	is the information (values) passed out to the execution of the behaviour.
Object's attributes	are the characteristics of the *object*. As a consequence of the *operation's execution*, the values of these attributes may change. Thus, we have identified:
	Modified attributes values — The modified values of the *object's attributes* after the execution of the operation.
	Unmodified attributes values — The values of the *object's attributes* not modified by the execution of the operation.

UML Diagram

Identified Element	UML	Rationale
Object	`Class` 1	The *objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use a `Class` 1 to represent the *object* both before (*pre-operation object*) and after (*post-operation object*) the execution of the operation.
Operation's execution	`Operation` 2 «`command`»	It represents the behaviour given by the operation. Additionally, it is associated with the stereotype «`command`» to denote that the behaviour performs a complex change to the object.
Input data	`Input Parameters` 3	They depict the information passed into the operation for its execution.
Output data	`Output Parameters` 4	They depict the information passed out to the execution of the operation.
Object's attributes	`Attributes` 5	They represent the characteristics of the *object* before and after the execution of the operation.

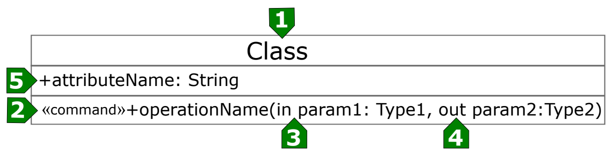
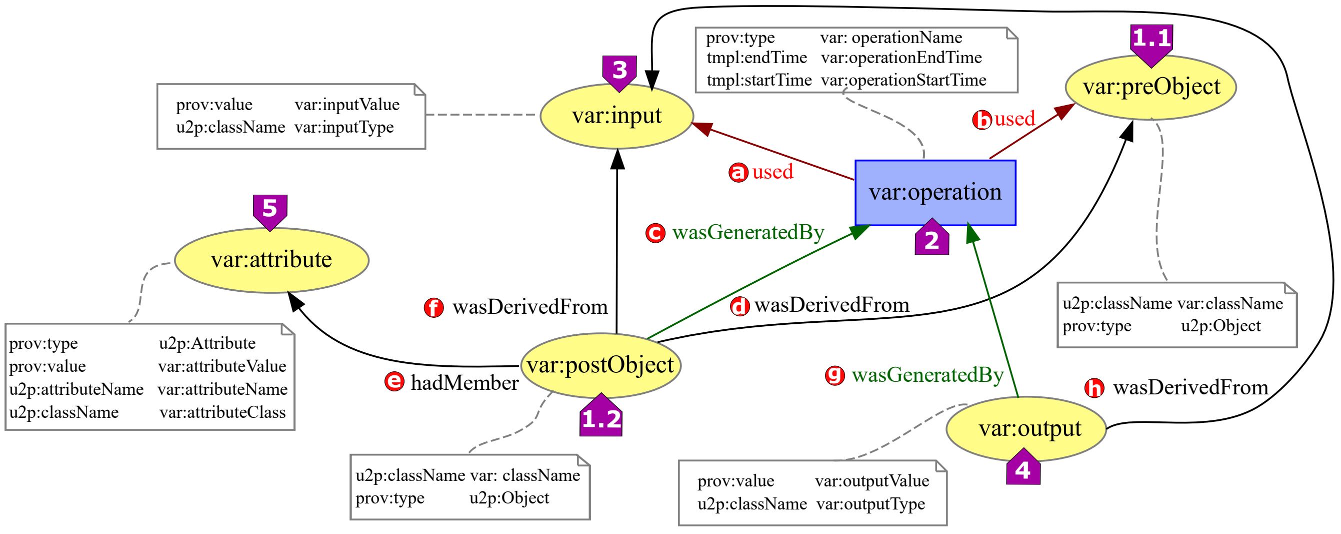


Figure 31. UML representation in *ClP6*

Mapping to PROV



3
var:input
prov:value var:inputValue
u2p:className var:inputType
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
1.1
var:preObject
b used
a used
var:operation
2
5
var:attribute
c wasGeneratedBy
f wasDerivedFrom
d wasDerivedFrom
u2p:className var:className
prov:type u2p:Object
prov:type u2p:Attribute
prov:value var:attributeValue
u2p:attributeName var:attributeName
u2p:className var:attributeClass
var:postObject
e hadMember
g wasGeneratedBy
h wasDerivedFrom
1.2
var:output
4
u2p:className var: className
prov:type u2p:Object
prov:value var:outputValue
u2p:className var:outputType

Figure 32. PROV template generated from the UML diagram 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 in 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 in 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»	`prov:Activity` 2 / `var:operation`	The `Operation` 2, stereotyped with «command», models the execution of the behaviour given by the invoked operation. Thus, it is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entiy` 3 / `var:input`	Each parameter of the `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Output Parameters` 4	`prov:Entiy` 4 / `var:output`	Each parameter of the `Output Parameters` 4 is a separate `prov:Entity` identified as `var:output`.
`Attributes` 5	`prov:Entity` 5 / `var:attribute`	Each attribute of the `Attributes` 5 that belongs both to *modified attributes values* and to *unmodified attributes values* (i.e., all of them) is mapped to a separate `prov:Entity` identified by `var:attribute`.

Attributes

PROV Element	**Attribute / Value**	**Description**
var:preObject 1.1	u2p:className / var:className	The value var:className is the name of the Class 1.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject is an object.
var:postObject 1.2	u2p:className / var:className	The value var:className is the name of the Class 1.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject is an object.
Operation 2	prov:type / var:operationName	The value var:operationName is the name of the Operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 4.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 4.
var:input 3	u2p:className / var:inputType	The value var:inputType is a string with the name of the class to which the parameter belongs to.
	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
var:output 4	u2p:className / var:outputType	The value var:outputType is a string with the name of the class to which the parameter belongs to.
	prov:value / var:outputValue	The value var:outputValue is the direct representation of var:output 4.
var:attribute 5	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:attribute is an attribute.
	prov:value / var:attributeValue	The value var:attributeValue is the direct representation of the attribute.
	u2p:attributeName / var:attributeName	The value var:attributeName is the name of the attribute.
	u2p:className / var:attributeClass	The value var:attributeClass is a string with the name of the class to which the attribute belongs to.

Relations

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

Discussion

The taxonomy of stereotypes described both «command» and «non-void-command» for operations that perform a complex change to the object's status. The difference is that if the operation returns information, it is stereotyped with «command», otherwise it is stereotyped with «command». The PROV template generated is the same due to the fact that at the moment of expanding the template, if there is no values associated to `var:output`, this `prov:Entity` together with its relations will not appear in the expanded document.

In this pattern, both the *modified attributes values* and the *unmodified attributes values* are translated into `var:attribute`. This is due to we know, because of «`command`», that some attributes change, but we don't know beforehand what they are. Later, after expanding and merging the template with the remainder expanded documents, we will be able to know what attributes where modified and those unmodified. Concretely, the modified attributes are those that are related to the expanded `var:postObject` by means of `prov:hadMember`, but they are not related to the expanded `var:preObject` through `prov:hadMember`.

Identifier *Class* diagram *Pattern 7* (*ClP7*)

Context

The execution of an object's operation provokes the change of its current status. This execution directly sets the information passed to the operation as values of certain object's attributes that are known beforehand, without modifying the remainder attributes and without returning information.

Identified elements

Object	is the object to which the operation to be executed belongs. Since the object's status changes, we have identified two object's status:
	Pre-operation object — The object in the status before the execution of the operation.
	Post-operation object — The object in the status after the execution of the operation.
Operation's execution	is the execution of the behaviour specified by the operation.
Input data	is the information (values) passed into the invocation of the behaviour.
Object's attributes	are the characteristics of the *object*. As a consequence of the *operation's execution*, the values of these attributes may change. Thus, we have identified:
	Modified attributes values — The modified values of the *object's attributes* after the execution of the operation.
	Unmodified attributes values — The values of the *object's attributes* not modified by the execution of the operation.

UML Diagram

Identified Element	UML	Rationale
Object	`Class` 1	The *objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use a `Class` 1 to represent the *object* both before (*pre-operation object*) and after (*post-operation object*) the execution of the operation.
Operation's execution	`Operation` 2 «`set`»	It represents the behaviour given by the operation. Additionally, it is associated with the stereotype «`set`» to model that the information passed into the operation is directly set as values of certain attributes.
Input data	`Input Parameters` 3	They depict the information passed into the operation for its execution.
Object's attributes	`Attributes` 4	They represent the characteristics of the *object* before and after the execution of the operation.

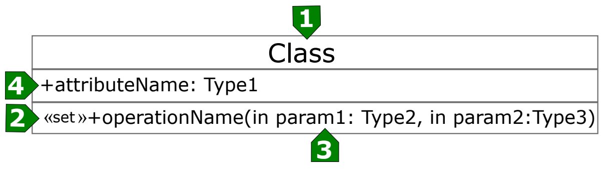


Figure 33. UML representation in *ClP7*

Mapping to PROV

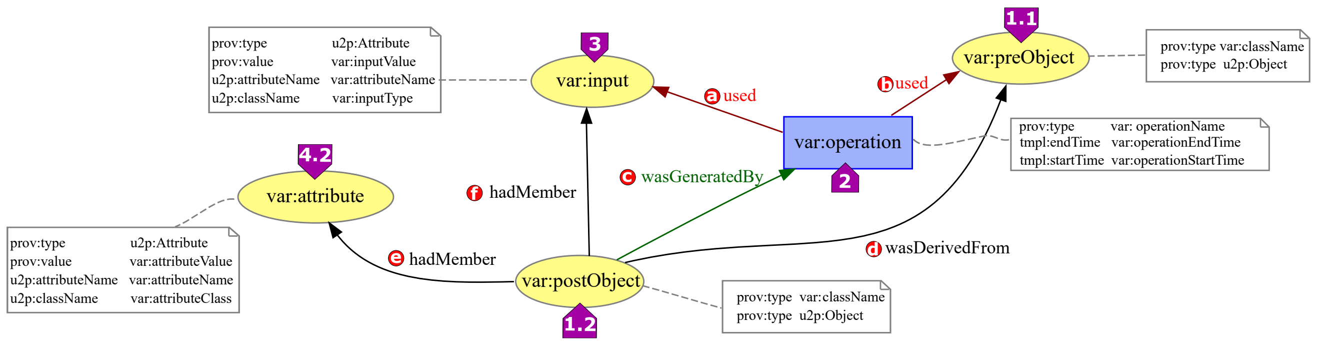

prov:type u2p:Attribute
prov:value var:inputValue
u2p:attributeName var:attributeName
u2p:className var:inputType
3
var:input
1.1
var:preObject
prov:type var:className
prov:type u2p:Object
a used
b used
var:operation
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
2
4.2
var:attribute
f hadMember
c wasGeneratedBy
prov:type u2p:Attribute
prov:value var:attributeValue
u2p:attributeName var:attributeName
u2p:className var:attributeClass
e hadMember
var:postObject
d wasDerivedFrom
prov:type var:className
prov:type u2p:Object
1.2

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

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1.1 / `var:preObject`	The *pre-operation object*, i.e. the object in 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 in 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 `Operation` 2, stereotyped with «set», models the execution of the behaviour given by the invoked operation. Thus, it is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entiy` 3 / `var:input`	Each parameter of the `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Attributes` 4	None /	Those attributes included in `Attributes` 4 that belong to *modified attributes values* are already mapped to `var:input`. For further information, see the discussion.
	`prov:Entity` 4.2 / `var:attribute`	Each attribute of the `Attributes` 4 that belongs to *unmodified attributes values* is a `prov:Entity` with identifier `var:attribute`.

Attributes

PROV Element	Attribute / Value	Description
`var:preObject` 1.1	`u2p:className` / `var:className`	The value `var:className` is the name of the `Class` 1.
	`prov:type` / `u2p:Object`	The value `u2p:Object` shows that `var:preObject` is an object.
`var:postObject` 1.2	`u2p:className` / `var:className`	The value `var:className` is the name of the `Class` 1.
	`prov:type` / `u2p:Object`	The value `u2p:Object` shows that `var:preObject` is an object.
`Operation` 2	`prov:type` / `var:operationName`	The value `var:operationName` is the name of the `Operation` 2.
	`tmpl:endTime` / `var:operationEndTime`	The `var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 4.
	`tmpl:startTime` / `var:operationStartTime`	The `var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 4.
`var:input` 3	`prov:type` / `u2p:Attribute`	The value `u2p:Attribute` shows that `var:input` is an attribute.
	`u2p:attributeName` / `var:attributeName`	The value `var:attributeName` is the name of the attribute.
	`u2p:className` / `var:inputType`	The value `var:inputType` is a string with the name of the class to which the parameter belongs to.
	`prov:value` / `var:inputValue`	The value `var:inputValue` is the direct representation of `var:input` 3.
`var:attribute` 4.2	`prov:type` / `u2p:Attribute`	The value `u2p:Attribute` shows that `var:attribute` is an attribute.
	`prov:value` / `var:attributeValue`	The value `var:attributeValue` is the direct representation of the attribute.
	`u2p:attributeName` / `var:attributeName`	The value `var:attributeName` is the name of the attribute.
	`u2p:className` / `var:attributeClass`	The value `var:attributeClass` is a string with the name of the class to which the attribute belongs to.

Relations

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

This context claims that the *input data* is directly set as values of certain object's attributes, which means that the *input data* correspond directly to the *modified attributes values*. This fact is represented in the PROV template by means of the 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 attribute modified.

Identifier *Class* diagram *Pattern 8* (*ClP8*)

Context

The execution of an object's operation provokes the change of its current status. This execution modifies certain object's attributes that are known beforehand, without modifying the remainder attributes of the object.

Identified elements

Object	is the object to which the operation to be executed belongs. Since the object's status changes, we have identified two object's statuss:
	Pre-operation object — The object in the status before the execution of the operation.
	Post-operation object — The object in the status after the execution of the operation.
Operation's execution	is the execution of the behaviour specified by the operation.
Input data	is the information (values) passed into the invocation of the behaviour.
Object's attributes	are the characteristics of the *object*. As a consequence of the *operation's execution*, the values of these attributes may change. Thus, we have identified:
	Modified attributes values — The modified values of the *object's attributes* after the execution of the operation.
	Unmodified attributes values — The values of the *object's attributes* not modified by the execution of the operation.

UML Diagram

Identified Element	UML	Rationale
Object	`Class` 1	The *objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use a `Class` 1 to represent the *object* both before (*pre-operation object*) and after (*post-operation object*) the execution of the operation.
Operation's execution	`Operation` 2 «`modify`»	It represents the behaviour given by the operation. Additionally, it is associated with the stereotype «`modify`» to model that only certain object's attributes are modified.
Input data	`Input Parameters` 3	They depict the information passed into the operation for its execution.
Object's attributes	`Attributes` 4	They represent the characteristics of the *object* before and after the execution of the operation.

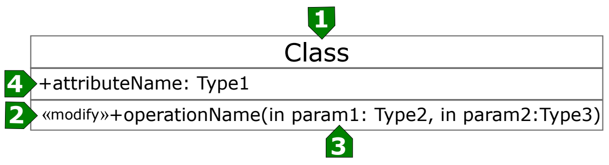


Figure 35. UML representation in *ClP8*

Mapping to PROV

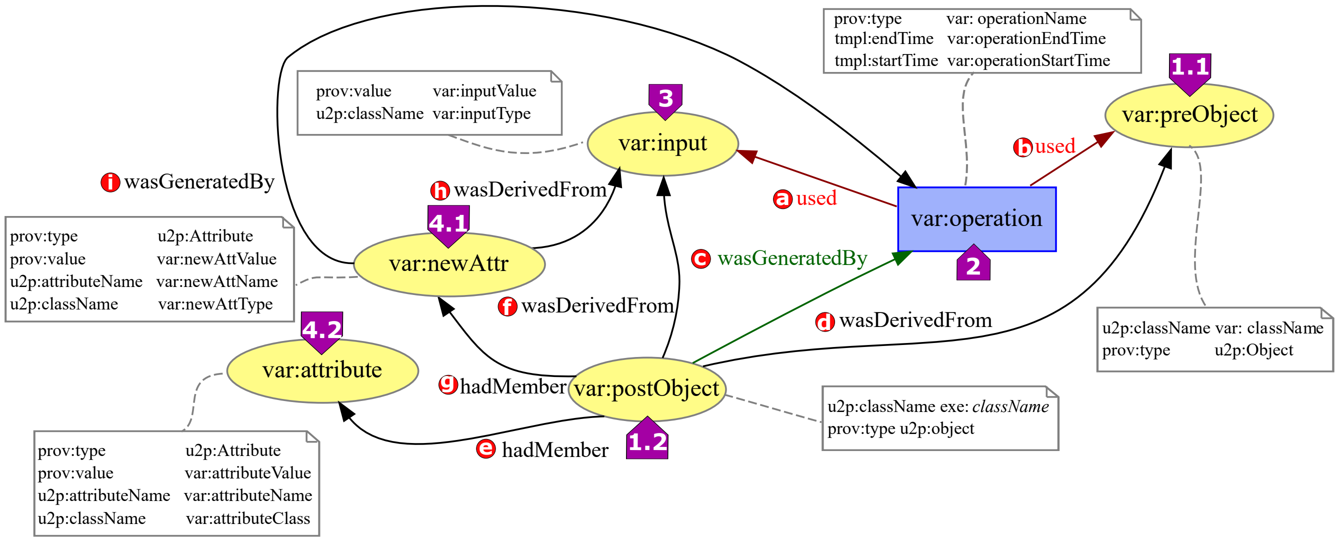

prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
1.1
var:preObject
prov:value var:inputValue
u2p:className var:inputType
3
var:input
b used
i wasGeneratedBy
h wasDerivedFrom
a used
var:operation
prov:type u2p:Attribute
prov:value var:newAttValue
u2p:attributeName var:newAttName
u2p:className var:newAttType
4.1
var:newAttr
c wasGeneratedBy
2
f wasDerivedFrom
d wasDerivedFrom
u2p:className var: className
prov:type u2p:Object
4.2
var:attribute
g hadMember
var:postObject
u2p:className exe: className
prov:type u2p:object
e hadMember
1.2
prov:type u2p:Attribute
prov:value var:attributeValue
u2p:attributeName var:attributeName
u2p:className var:attributeClass

Figure 36. PROV template generated from the UML diagram 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 in 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 in 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 `Operation` 2, stereotyped with «`modify`», models the execution of the behaviour given by the invoked operation. Thus, it is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entiy` 3 / `var:input`	Each parameter of the `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Attributes` 4	`prov:Entity` 4.1 / `var:newAttr`	Each attribute of the `Attributes` 4 that belongs to *modified attributes values* is a separate `prov:Entity` with identifier `var:newAttr`.
	`prov:Entity` 4.2 / `var:attribute`	Each attribute of the `Attributes` 4 that belongs to *unmodified attributes values* is a separate `prov:Entity` with identifier `var:attribute`.

Attributes

PROV Element	Attribute / Value	Description
var:preObject 1.1	u2p:className / var:className	The value var:className is the name of the Class 1.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject is an object.
var:postObject 1.2	u2p:className / var:className	The value var:className is the name of the Class 1.
	prov:type / u2p:Object	The value u2p:Object shows that var:preObject is an object.
Operation 2	prov:type / var:operationName	The value var:operationName is the name of the Operation 2.
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 4.
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 4.
var:input 3	u2p:className / var:inputType	The value var:inputType is a string with the name of the class to which the parameter belongs to.
	prov:value / var:inputValue	The value var:inputValue is the direct representation of var:input 3.
var:newAttr 4.1	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:attribute is an attribute.
	prov:value / var:newAttValue	The value var:attributeValue is the direct representation of the attribute.
	u2p:attributeName / var:newAttName	The value var:newAttName is the name of the attribute.
	u2p:className / var:newAttType	The value var:attributeClass is a string with the name of the class to which the attribute belongs to.
var:attribute 4.2	prov:type / u2p:Attribute	The value u2p:Attribute shows that var:attribute is an attribute.
	prov:value / var:attributeValue	The value var:attributeValue is the direct representation of the attribute.
	u2p:attributeName / var:attributeName	The value var:attributeName is the name of the attribute.
	u2p:className / var:attributeClass	The value var:attributeClass is a string with the name of the class to which the attribute belongs to.

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:newAttr is one of the elements in var:postObject.
h	prov:wasDerivedFrom	It is the construction of var:newAttr based on var:input.
i	prov:wasGeneratedBy	It is the completion of production of var:newAttr by var:operation.

Discussion

This pattern bears a strong resemblance with *ClP6*. In *ClP6* the *modified attributes values* are not known beforehand, and both the *modified attributes values* and the *unmodified attributes values* are translated into a separate `var:attribute`. Conversely, in this pattern we know the attributes to be modified beforehand and therefore, we distinguish between `var:newAttr` (*modified attributes values*) and `var:attribute` (*unmodified attributes values*).

Identifier *Class* diagram *Pattern 9* (*ClP9*)

Context

The execution of an object's operation provokes the change of its current status. This execution directly adds the information passed to the operation into an object's attribute collection that is known beforehand. This behaviour does not modify the remainder attributes of the object.

Identified elements

Object	is the object to which the operation to be executed belongs. Since the object's situation changes, we have identified two object's situations:	
	Pre-operation object	The object in the situation before the execution of the operation.
	Post-operation object	The object in the situation after the execution of the operation.
Operation's execution	is the execution of the behaviour specified by the operation.	
Input data	is the information (values) passed into the invocation of the behaviour.	
Object's attributes	are the characteristics of the *object*. As a consequence of the *operation's execution*, the values of these attributes may change. Thus, we have identified:	
	Modified attributes values	The modified values of the *object's attributes* after the execution of the operation.
	Unmodified attributes values	The values of the *object's attributes* not modified by the execution of the operation.

UML Diagram

Identified Element	UML	Rationale
Object	`Class` 1	The *objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use a `Class` 1 to represent the *object* both before (*pre-operation object*) and after (*post-operation object*) the execution of the operation.
Operation's execution	`Operation` 2 `«remove»`	It represents the behaviour given by the operation. Additionally, it is associated with the stereotype `«remove»` to model the specific behaviour of the operation.
Input data	`Input Parameters` 3	They depict the information passed into the operation for its execution.
Object's attributes	`Attributes` 4	They represent the characteristics of the *object* before and after the execution of the operation.

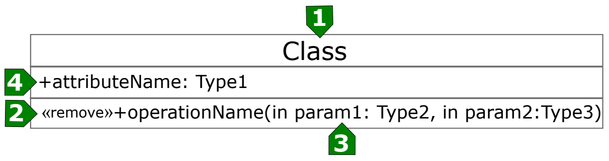


Figure 37. UML representation in *ClP9*

Mapping to PROV

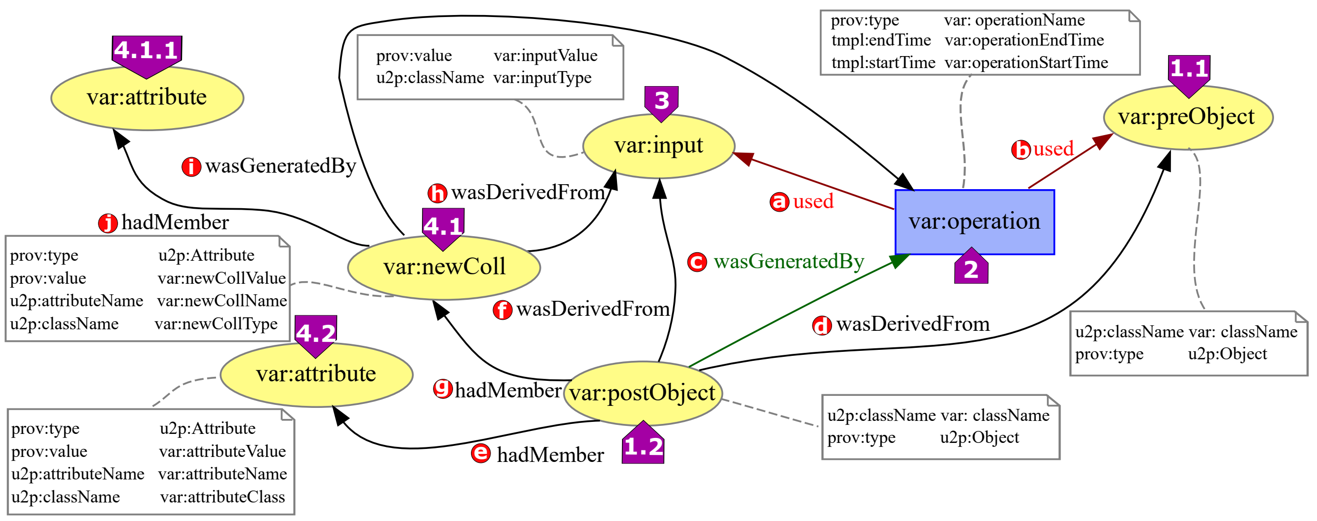

4.1.1
var:attribute
i wasGeneratedBy
j hadMember
prov:type u2p:Attribute
prov:value var:newCollValue
u2p:attributeName var:newCollName
u2p:className var:newCollType
prov:value var:inputValue
u2p:className var:inputType
3
var:input
h wasDerivedFrom
4.1
var:newColl
f wasDerivedFrom
4.2
var:attribute
g hadMember
var:postObject
e hadMember
1.2
prov:type u2p:Attribute
prov:value var:attributeValue
u2p:attributeName var:attributeName
u2p:className var:attributeClass
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
a used
b used
c wasGeneratedBy
d wasDerivedFrom
var:operation
2
1.1
var:preObject
u2p:className var: className
prov:type u2p:Object
u2p:className var: className
prov:type u2p:Object

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

PROV elements

UML	PROV / id	Rationale
`Class` 1	`prov:Entity` 1.1 / `var:preObject`	The *pre-operation object*, i.e. the object in the situation 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 in the situation 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 `Operation` 2, stereotyped with «`remove`», models the execution of the behaviour given by the invoked operation. Thus, it is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entiy` 3 / `var:input`	Each parameter of the `Input Parameters` 3 is a separate `prov:Entity` identified as `var:input`.
`Attributes` 4	`prov:Entity` 4.1 / `var:newColl`	The collection attribute of the `Attributes` 4 that belongs to *modified attributes values*, is a `prov:Entity` with identifier `var:newColl`. 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 attribute of the `Attributes` 4 that belongs to *unmodified attributes values* is a separate `prov:Entity` with identifier `var:attribute`.

Attributes

PROV Element	Attribute / Value	Description
`var:preObject` 1.1	`u2p:className` / `var:className`	The value `var:className` is the name of the `Class` 1.
	`prov:type` / `u2p:Object`	The value `u2p:Object` shows that `var:preObject` is an object.
`var:postObject` 1.2	`u2p:className` / `var:className`	The value `var:className` is the name of the `Class` 1.
	`prov:type` / `u2p:Object`	The value `u2p:Object` shows that `var:preObject` is an object.
`Operation` 2	`prov:type` / `var:operationName`	The value `var:operationName` is the name of the `Operation` 2.
	`tmpl:endTime` / `var:operationEndTime`	The `var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 4.
	`tmpl:startTime` / `var:operationStartTime`	The `var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 4.
`var:input` 3	`u2p:className` / `var:inputType`	The value `var:inputType` is a string with the name of the class to which the parameter belongs to.
	`prov:value` / `var:inputValue`	The value `var:inputValue` is the direct representation of `var:input` 3.
`var:newColl` 4.1	`prov:type` / `u2p:Attribute`	The value `u2p:Attribute` shows that `var:newColl` is an attribute.
	`prov:value` / `var:newCollValue`	The value `var:newCollValue` is the direct representation of the attribute.
	`u2p:attributeName` / `var:newCollName`	The value `var:newCollName` is the name of the attribute.
	`u2p:className` / `var:newCollType`	The value `var:newCollType` is a string with the name of the class to which the attribute belongs to.
`var:attribute` 4.2	`prov:type` / `u2p:Attribute`	The value `u2p:Attribute` shows that `var:attribute` is an attribute.
	`prov:value` / `var:attributeValue`	The value `var:attributeValue` is the direct representation of the attribute.
	`u2p:attributeName` / `var:attributeName`	The value `var:attributeName` is the name of the attribute.
	`u2p:className` / `var:attributeClass`	The value `var:attributeClass` is a string with the name of the class to which the attribute belongs to.

Relations

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

Discussion

This pattern shows the modification of a collection attribute that belongs to an object. In this case the *modified attributes values* correspond to such a collection attribute, which is translated into `var:newColl`. The remainder attributes of the object that belongs *unmodified attributes values* are translated into `var:attribute`.

Identifier *Class* diagram *Pattern 10* (*ClP10*)

Context

The execution of an object's operation provokes the change of its current status. This execution directly adds the information passed to the operation into an object's attribute collection that is known beforehand. This behaviour does not modify the remainder attributes of the object.

Identified elements

Object	is the object to which the operation to be executed belongs. We have identified two object's statuss:	
	Pre-operation object	The object in the status before the execution of the operation.
	Post-operation object	The object in the status after the execution of the operation.
Operation's execution	is the execution of the behaviour specified by the operation.	
Input data	is the information (values) passed into the invocation of the behaviour.	
Object's attributes	are the characteristics of the *object*. As a consequence of the *operation's execution*, the values of these attributes may change. Thus, we have identified:	
	Modified attributes values	The modified values of the *object's attributes* after the execution of the operation.
	Unmodified attributes values	The values of the *object's attributes* not modified by the execution of the operation.

UML Diagram

Identified Element	UML	Rationale
Object	`Class` 1	The *objects* are classified attending to their characteristics and behaviour by means of `classes`. Thus, we use a `Class` 1 to represent the *object* both before (*pre-operation object*) and after (*post-operation object*) the execution of the behaviour given by the operation.
Operation's execution	`Operation` 2 «`add`»	It represents the behaviour given by the operation. Additionally, it is associated with the stereotype «`add`» to model the specific behaviour of the operation.
Input data	`Input Parameters` 3	They depict the information passed into the operation for its execution.
Object's attributes	`Attributes` 4	They represent the characteristics of the *object* before and after the execution of the operation.

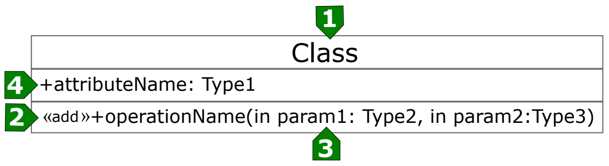


Figure 39. UML representation in *ClP10*

Mapping to PROV

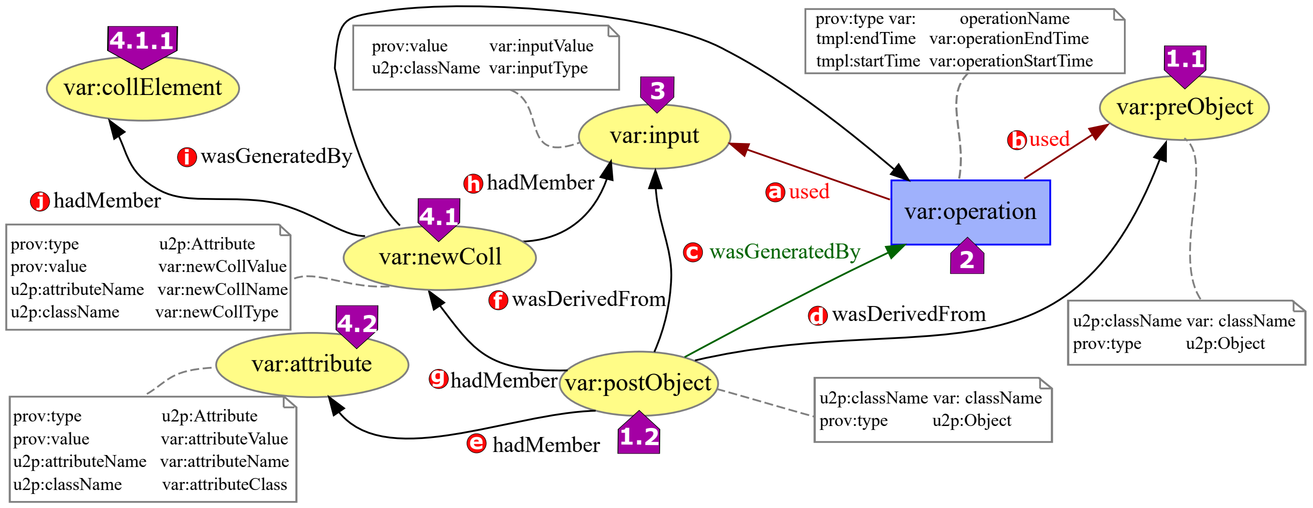

4.1.1
var:collElement
prov:value var:inputValue
u2p:className var:inputType
3
var:input
prov:type var: operationName
tmpl:endTime var:operationEndTime
tmpl:startTime var:operationStartTime
1.1
var:preObject
i wasGeneratedBy
j hadMember
h hadMember
a used
b used
4.1
var:newColl
var:operation
2
c wasGeneratedBy
prov:type u2p:Attribute
prov:value var:newCollValue
u2p:attributeName var:newCollName
u2p:className var:newCollType
f wasDerivedFrom
d wasDerivedFrom
u2p:className var: className
prov:type u2p:Object
4.2
var:attribute
g hadMember
var:postObject
u2p:className var: className
prov:type u2p:Object
prov:type u2p:Attribute
prov:value var:attributeValue
u2p:attributeName var:attributeName
u2p:className var:attributeClass
e hadMember
1.2

Figure 40. PROV template generated from the UML diagram 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 in 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 in 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 `Operation` 2, stereotyped with «add», models the execution of the behaviour given by the invoked operation. Thus, it is a `prov:Activity` identified by `var:operation`.
`Input Parameters` 3	`prov:Entiy` 3 / `var:input`	Each parameter of the `Input Parameters` 3 is a separated `prov:Entity` identified as `var:input`.
`Attributes` 4	`prov:Entity` 4.1 / `var:newColl`	The collection attribute of the `Attributes` 4 that belongs to *modified attributes values*, is a separate `prov:Entity` with identifier `var:newColl`. Additionally, each element in this collection is a `prov:Entity` identified with `var:collElement` 4.1.1
	`prov:Entity` 4.2 / `var:attribute`	Each attribute of the `Attributes` 4 that belongs to *unmodified attributes values* is a separate `prov:Entity` with identifier `var:attribute`.

Attributes

PROV Element	Attribute / Value	Description
`var:preObject` 1.1▸	`u2p:className` / `var:className`	The value `var:className` is the name of the `Class` 1▸.
	`prov:type` / `u2p:Object`	The value `u2p:Object` shows that `var:preObject` is an object.
`var:postObject` 1.2▸	`u2p:className` / `var:className`	The value `var:className` is the name of the `Class` 1▸.
	`prov:type` / `u2p:Object`	The value `u2p:Object` shows that `var:preObject` is an object.
`Operation` 2▸	`prov:type` / `var:operationName`	The value `var:operationName` is the name of the `Operation` 2▸.
	`tmpl:endTime` / `var:operationEndTime`	The `var:operationEndTime` is an `xsd:dateTime` value for the end of `var:operation` 4▸.
	`tmpl:startTime` / `var:operationStartTime`	The `var:operationStartTime` is an `xsd:dateTime` value for the start of `var:operation` 4▸.
`var:input` 3▸	`u2p:className` / `var:inputType`	The value `var:inputType` is a string with the name of the class to which the parameter belongs to.
	`prov:value` / `var:inputValue`	The value `var:inputValue` is the direct representation of `var:input` 3▸.
`var:newColl` 4.1▸	`prov:type` / `u2p:Attribute`	The value `u2p:Attribute` shows that `var:newColl` is an attribute.
	`prov:value` / `var:newCollValue`	The value `var:newCollValue` is the direct representation of the attribute.
	`u2p:attributeName` / `var:newCollName`	The value `var:newCollName` is the name of the attribute.
	`u2p:className` / `var:newCollType`	The value `var:newCollType` is a string with the name of the class to which the attribute belongs to.
`var:attribute` 4.2▸	`prov:type` / `u2p:Attribute`	The value `u2p:Attribute` shows that `var:attribute` is an attribute.
	`prov:value` / `var:attributeValue`	The value `var:attributeValue` is the direct representation of the attribute.
	`u2p:attributeName` / `var:attributeName`	The value `var:attributeName` is the name of the attribute.
	`u2p:className` / `var:attributeClass`	The value `var:attributeClass` is a string with the name of the class to which the attribute belongs to.

Relations

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

Discussion

This pattern shows the modification of a collection attribute that belongs to an object. In this case the *modified attributes values* correspond to such a collection attribute, which is translated into `var:newColl`. The remainder attributes of the object that belongs *unmodified attributes values* are translated into `var:attribute`.

Additionally, the nuance of this pattern is that the modification of the collection attribute is made by adding new element(s) (*input data*) into the collection. As the *input data* is directly added as elements into the collection, there is a `prov:hadMember` relation between `var:newColl` and `var:input`.

References

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

[2] L. Moreau, P. Missier (eds.), K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes, "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] 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.