

UNIVERSIDAD DE LA RIOJA

APPENDIX
TO
DOCTORAL THESIS

Towards a framework for making applications
provenance-aware

Specification of the UML to PROV patterns

Author:

Carlos Sáenz Adán

Supervisors:

Dr. Beatriz Pérez Valle

Dr. Francisco José García Izquierdo

October 2019



**UNIVERSIDAD
DE LA RIOJA**

1 Introduction

This specification defines in detail the patterns for translating UML into PROV. The main objective of this specification is to provide a tool for implementing systems that include provenance capabilities during their design phase.

This document has been organized into three main parts:

- Section 2 provides an insight into the information required for the accurate understanding of this specification such as the notational conventions used throughout the document (Section 2.1), or the description of the structure for the pattern's explanations (Section 2.2).
- Section 3 shows a table that associates each pattern's identifier with the page where it is explained.
- Sections from 4 to 6 provide a systematic explanation of each pattern classified by the addressed UML diagram.

2 Before reading

As we see this document as the reference specification for UML2PROV, each pattern has been written in a self-contained way. A reader who reads all the patterns sequentially from the first to the last will find similar explanations, even repeated ones, in several patterns. We have preferred to make the reader suffer this small inconvenience, instead of running the risk that an occasional reader of a particular pattern loses part of the explanations that are discussed elsewhere.

We assume that the reader is familiar with the following UML diagrams: UML Sequence Diagrams (SqDs), UML State Machine Diagrams (SMDs), and UML Class diagrams (CDs). Readers unfamiliar with these diagrams are encouraged to read the UML specification [1]. Additionally, due to the fact that transformations referring to CDs make use of concrete UML stereotypes used to classify UML Class's operations, we refer to Appendix A for an overview about them.

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

2.1 Notational conventions

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

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

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

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

prefix	namespace IRI	definition
var	http://openprovenance.org/var#	The namespace for template variables
prov	http://www.w3.org/ns/prov#	The PROV namespace
xsd	http://www.w3.org/2000/10/XMLSchema#	XML Schema namespace
u2p	http://uml2prov.unirioja.es/ns/u2p#	UML2PROV namespace

Table 1. Prefix and Namespaces used in this specification

2.2 Structure of the patterns

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

2.2.1 Identifier

Unique identifier of the transformation pattern. It is an acronym that refers to the type of UML diagram together with a numeric identifier. The UML *Sequence* diagram Patterns are referred to as *SeqP<N>*, where *N* is the numeric identifier. Likewise, *StP<N>* corresponds to the UML *State Machine* Patterns, and *CIP<N>* to the UML *Class Diagram* Patterns.


2.2.2 Context

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

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

2.2.3 UML Diagram



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

Key Element	UML	Rationale
<i>Name of the element</i>	UML element 	The fundamental reasons serving to account for the use of the <i>UML element</i> for modelling the <i>key element</i> .


2.2.4 Mapping to PROV

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

PROV elements


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

Attributes

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

Note: Throughout this specification, we have included the attributes `tmpl:startTime` and `tmpl:endTime` associated with *Activities* because we consider such an information very useful from a provenance point of view. Nevertheless, both attributes are optional and the user is free to include them.

PROV relations

PROV Relation	Description
PROV relation 	Description of the relation.

Note: In PROV, two relationships of the form (B, `prov:used`, A) and (C, `prov:wasGeneratedBy`, B) may be enriched with (C, `prov:wasDerivedFrom`, A) to express the dependency of C on A. This structure is a provenance construction

called *use-generate-derive triangle* [3] which explicitly connects a *generated* `prov:Entity` to a *used* `prov:Entity`. In the realm of this work, it may be applied in those templates in which a `prov:Entity` is *used* by a `prov:Activity`, and such a `prov:Activity` *generated* another `prov:Entity`. However, aiming at avoiding the overburden of the PROV template explanations with information that can be inferred, we have decided to include the relation `prov:wasDerivedFrom` only when the context of the pattern explicitly refers to such a derivation.

2.2.5 Discussion

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

3 Index of patterns

Modelled by means of UML Sequence Diagrams

Pattern identifier	Context	Page
SeqP1	A participant (the sender) interacts with another participant (the recipient) by calling an operation in the recipient, and then, it continues immediately. The call causes the recipient to execute the operation.	6
SeqP2	A participant (the sender) interacts with another participant (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.	9
SeqP3	During the execution of an operation (main operation), a nested operation call is made. After this call, the execution of the main operation can either continue immediately or wait for the response of that nested operation call. This way, this pattern complements SeqP1 and SeqP2 .	12
SeqP4	During the execution of an operation (main operation), a response of a previously issued nested operation call is received. The main operation's execution uses this response to complete its behaviour. This way, this pattern complements SeqP1 and SeqP2 with additional information regarding the response to the <i>nested operation call</i> (addressed by SeqP3).	15

Modelled by means of UML State Machine Diagrams

Pattern identifier	Context	Page
StP1	As a consequence of the execution of an operation, an object is created in its first state. This operation is usually the constructor of the object.	19
StP2	As a consequence of the execution of an operation, the behaviour of an object is completed.	23
StP3	As a consequence of the execution of an operation, an object changes its state.	27

Modelled by means of UML Class Diagrams

Pattern identifier	Context	Page
CIP1	The execution of an operation provokes the creation of a new object.	33
CIP2	The execution of an operation provokes the destruction of an object.	36
CIP3	The execution of an operation on an object returns values of concrete object's attributes. The values are returned as they are, without any further processing. This execution does not provoke the change of the object's status.	38
CIP4	The execution of an operation on an object returns values that are computed based on the object's status as a whole (the values of concrete attributes involved in the computation are unknown or irrelevant). This execution does not provoke the change of the object's status.	41
CIP5	The execution of an operation on an object returns values that are computed based on values of concrete object's attributes. This execution does not provoke the change of the object's status.	44
CIP6	The execution of an operation on an object changes the object's status as a whole (the concrete modified attributes are unknown or irrelevant).	48
CIP7	The execution of an operation on an object directly sets the information passed to the operation as values of concrete object's attributes, thus provoking a change in the object's status.	53
CIP8	The execution of an operation on an object changes the values of concrete object's attributes, thus provoking a change in the object's status.	58
CIP9	The execution of an operation on an object removes element(s) from a concrete object's collection attribute, thus provoking a change in the object's status.	63
CIP10	The execution of an operation on an object directly adds the information passed to the operation as new element(s) of a concrete object's collection attribute, thus provoking a change in the object's status.	68

4 UML Sequence Diagrams

Pattern identifier	Context	Page
SeqP1	A participant (the sender) interacts with another participant (the recipient) by calling an operation in the recipient, and then, it continues immediately. The call causes the recipient to execute the operation.	6
SeqP2	A participant (the sender) interacts with another participant (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.	9
SeqP3	During the execution of an operation (main operation), a nested operation call is made. After this call, the execution of the main operation can either continue immediately or wait for the response of that nested operation call. This way, this pattern complements SeqP1 and SeqP2 .	12
SeqP4	During the execution of an operation (main operation), a response of a previously issued nested operation call is received. The main operation's execution uses this response to complete its behaviour. This way, this pattern complements SeqP1 and SeqP2 with additional information regarding the response to the <i>nested operation call</i> (addressed by SeqP3).	15

Identifier *Sequence diagram Pattern 1 (SeqP1)*

Context

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

Key elements

<i>Sender</i>	The participant that makes the operation call.
<i>Operation call</i>	The call that starts the execution of the operation.
<i>Input data</i>	The information (if any) passed to the operation through the <i>Operation call</i> .
<i>Operation execution</i>	The execution of the operation.

UML Diagram

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

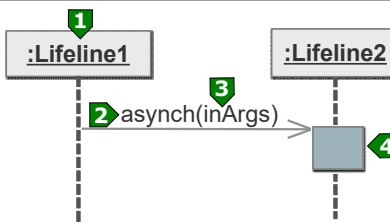


Figure 1. UML representation that models the context given by *SeqP1*

Mapping to PROV

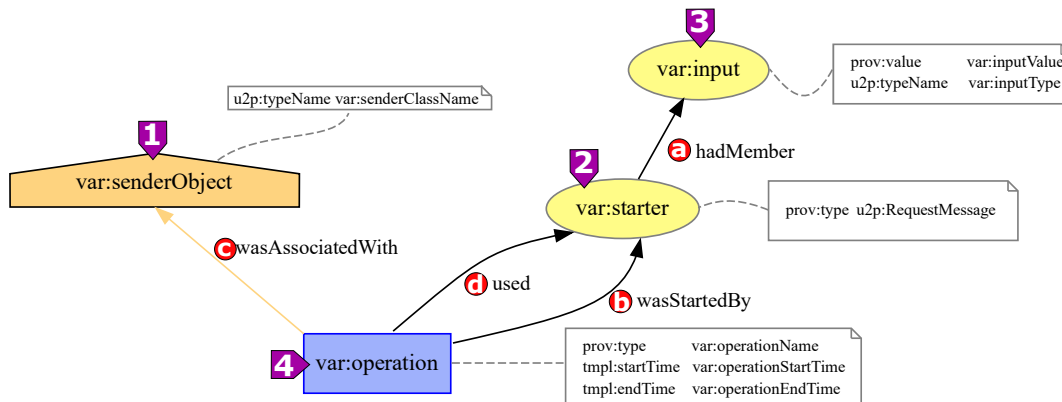


Figure 2. PROV template generated from the UML representation used in *SeqP1* (Figure 1)

PROV elements

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

Attributes





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

PROV relations

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

Discussion

- Figure 2 depicts the responsibility of the *Sender* lifeline (`var:senderObject`) for the recipient lifeline to execute the operation (`var:operation`). However, the recipient lifeline is not modelled in this PROV template, even though it is the participant that executes the operation. This decision is based on other patterns' better ability to both (1) identify the participant responsible for executing that operation, and (2) give a more detailed information about the implications that the execution of that operation has in the recipient participant. More specifically, these patterns are: *StP1-StP3*, which mainly focus on representing possible changes in an object's state caused by an *Operation execution*; and patterns *CIP1-CIP10*, which put more stress on how the execution affects the status of the object responsible for performing such an execution.
- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have considered this circumstance with the aim of covering a wider spectrum of cases. When the *Operation call* lacks *Input*

data, the UML representation in Figure 1 will not include Input Arguments . As a consequence, the resulting PROV template in Figure 2 will also lack `var:input`  and its associated PROV relations. Finally, we remark that the resulting PROV template does not reflect the *usage* of `var:input`  by `var:operation`  because SqDs stick to the flow of information, not its usage. Patterns addressing CDs (*CIP1-CIP10*) are better suited for this purpose.

Identifier Sequence diagram Pattern 2 (SeqP2)

Context

A participant (the sender) interacts with another participant (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.

Key elements

<i>Sender</i>	The participant that makes the operation call.
<i>Operation call</i>	The call that starts the execution of the operation.
<i>Input data</i>	The information (if any) passed to the operation through the <i>Operation call</i> .
<i>Operation execution</i>	The execution of the operation.
<i>Response</i>	The recipient's response to the <i>Operation call</i> .
<i>Output data</i>	The information contained in the <i>Response</i> .

UML Diagram

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

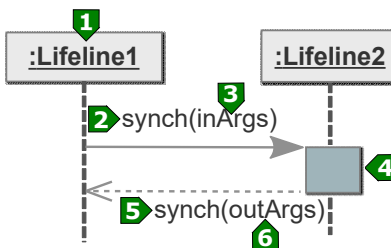


Figure 3. UML representation that models the context given by SeqP2

Mapping to PROV

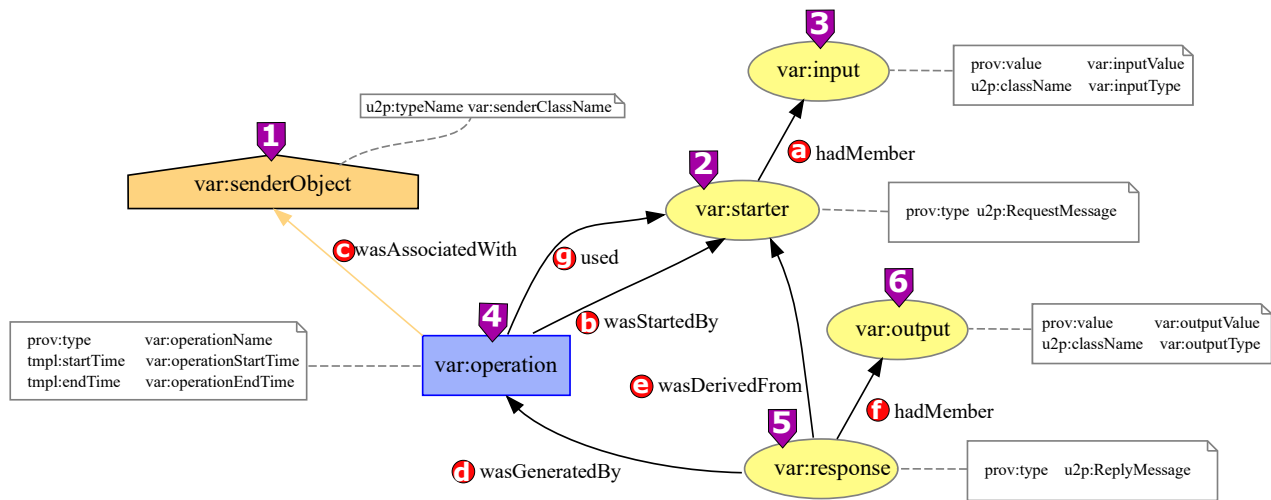


















Figure 4. PROV template generated from the UML representation used in *SeqP2* (Figure 3)








PROV elements

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





Attributes

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

PROV relations

PROV Relation	Description
 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.
 prov:used	It is the beginning of utilizing var:starter by var:operation.

Discussion

- Figure 4 depicts the responsibility of the *Sender* lifeline (var:senderObject) for executing the operation (var:operation) in a recipient lifeline. However, the recipient lifeline is not modelled in this PROV template, even though it is the participant that executes the operation. This decision is based on other patterns' better ability to both (1) identify the participant responsible for executing that operation, and (2) give a more detailed information about the implications that the execution of that operation has in the recipient participant. More specifically, these patterns are: *StP1-StP3*, which mainly focus on representing possible changes in an object's state caused by an *Operation execution*; and patterns *CIP1-CIP10*, which put more stress on how the execution affects the status of the object responsible for performing such an execution.
- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have considered this circumstance with the aim of covering a wider spectrum of cases. When the *Operation call* lacks *Input data*, the UML representation in Figure 3 will not include *Input Arguments* . As a consequence, the resulting PROV template in Figure 4 will also lack var:input  and its associated PROV relations. Finally, we remark that the resulting PROV template does not reflect the *usage* of var:input  by var:operation  because SqDs stick to the flow of information, not its usage. Patterns addressing CDs (*CIP1-CIP10*) are better suited for this purpose.

Identifier *Sequence diagram Pattern 3 (SeqP3)*

Context

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

Key elements

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

UML Diagram

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

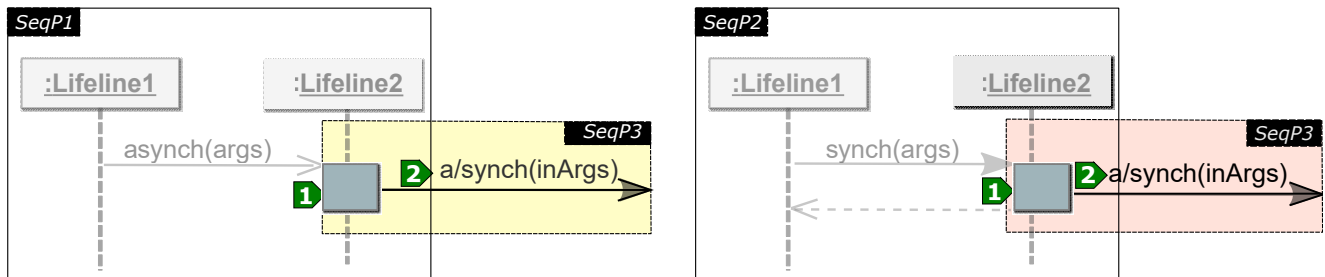


Figure 5. The left hand side is the UML representation of *SeqP1* complemented by *SeqP3*, whereas the right hand side is the UML representation of *SeqP2* complemented by *SeqP3*. Only the shaded areas correspond to the UML elements contributed by this pattern.

Mapping to PROV

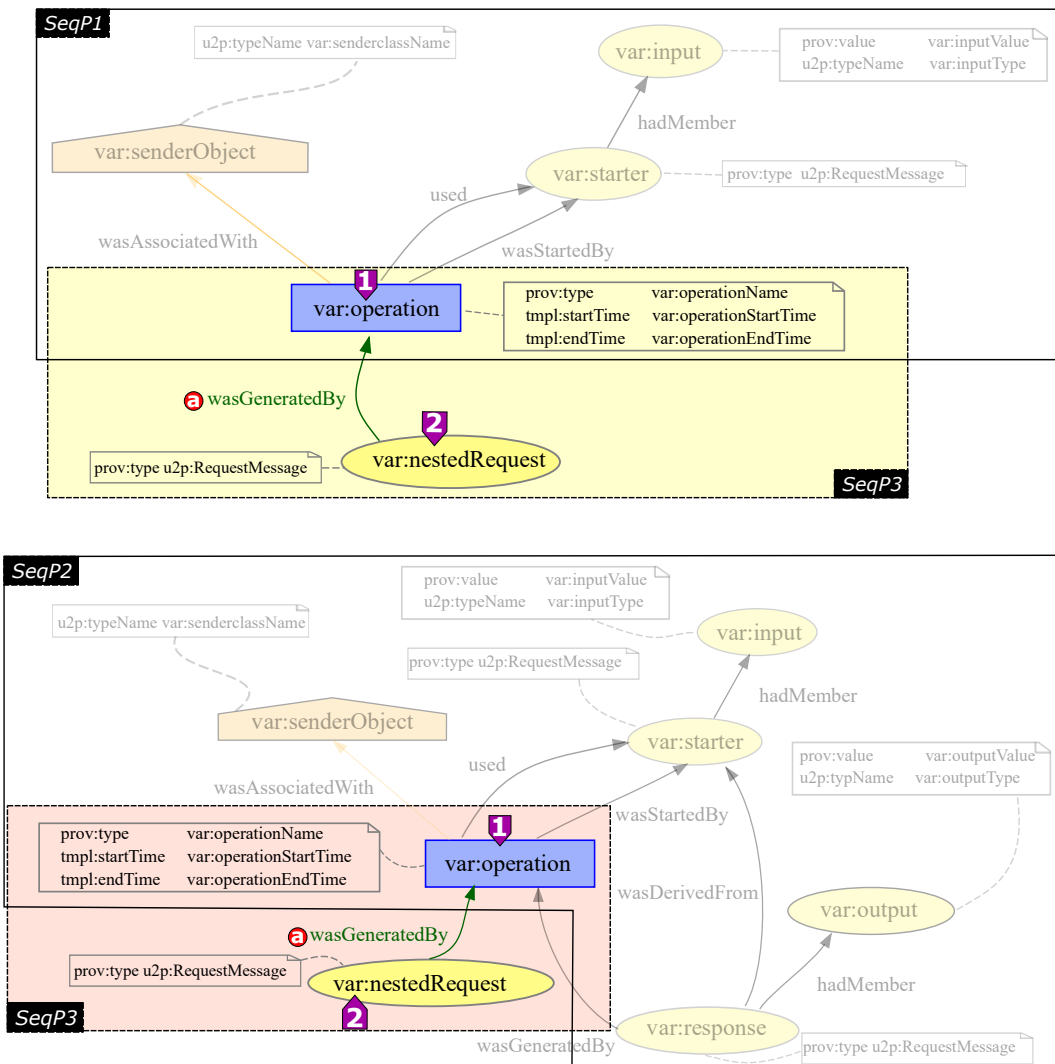


Figure 6. At the top, it is a PROV template generated from the UML representation in the left side of Figure 5. At the bottom, it is a PROV template generated from the UML representation in the right side of Figure 5. Only the shaded areas correspond to the PROV elements contributed by this pattern.

PROV elements

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

Attributes

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

PROV relations

PROV Relation	Description
a prov:wasGeneratedBy	It is the completion of production of var:nestedRequest by var:operation.

Discussion

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

Identifier *Sequence diagram Pattern 4 (SeqP4)*

Context

During the execution of an operation (main operation), a response of a previously issued nested operation call is received. The main operation’s execution uses this response to complete its behaviour. This way, this pattern complements *SeqP1* and *SeqP2* with additional information regarding the response to the *nested operation call* (addressed by *SeqP3*).

Key elements

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

UML Diagram

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

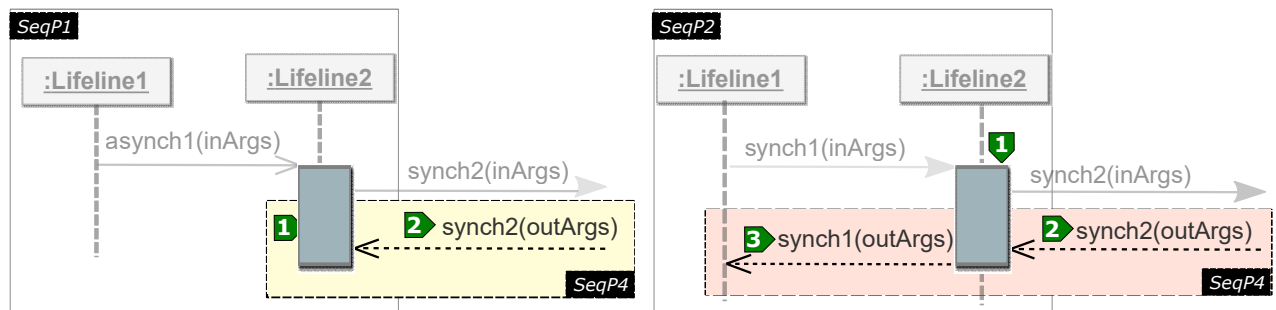


Figure 7. The left hand side is the UML representation that models the context given by *SeqP1* complemented by *SeqP4*, whereas the right hand side is the UML representation that models the context given by *SeqP2* complemented by *SeqP4*. Only the shaded areas correspond to the UML elements contributed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

Mapping to PROV

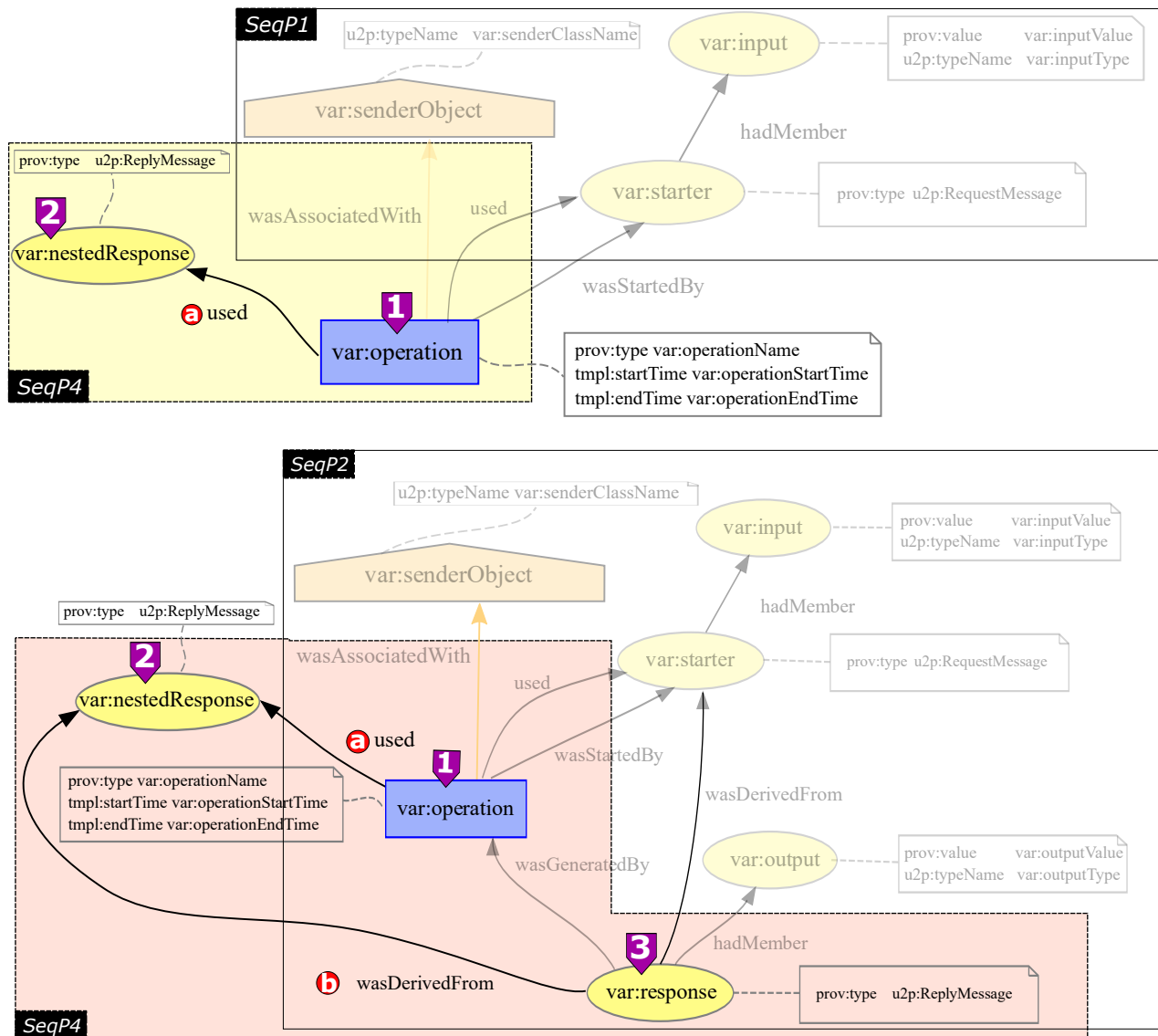


Figure 8. At the top, it is the PROV template generated from the UML representation in the left side of Figure 7. At the bottom, it is a PROV template generated from the UML representation in the right side of Figure 7. Only the shaded areas correspond to the PROV elements contributed by this pattern.

PROV elements

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

Attributes

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

PROV relations

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

Discussion

- As it could be inferred from the context, a requirement for this pattern to be applied is that the *Main operation execution* uses the *Nested response* during its execution. This causes the relations a prov:used and b prov:wasDerivedFrom to appear in the template; the former showing that when the ExecutionSpecification 1 receives the nested Reply Message 2, it *utilises* that Reply Message 2 to complete its behaviour; and the latter showing that the main Reply Message 3 is *influenced* by the nested Reply Message 2 (this last one can only be applied if the *Main operation execution* is triggered by a synchronous message, i.e. when SeqP4 complements SeqP2). If a specific scenario does not meet the aforementioned requirement, i.e., the *Main operation execution* does not use the *Nested response* or it is not worth recording such a dependency, this pattern should not be applied. Even in this case, the provenance about the nested operation call and its corresponding response would be captured thanks to SeqP1 and SeqP2, respectively.

5 UML State Machine Diagrams

Pattern identifier	Context	Page
StP1	As a consequence of the execution of an operation, an object is created in its first state. This operation is usually the constructor of the object.	19
StP2	As a consequence of the execution of an operation, the behaviour of an object is completed.	23
StP3	As a consequence of the execution of an operation, an object changes its state.	27

Identifier *State machine diagram Pattern 1 (StPI)*





Context

As a consequence of the execution of an operation, an object is created in its first state. This operation is usually the constructor of the object.

Key elements

- Object* The object created as a consequence of the execution of the operation.
- First object's state* The first state after the object creation. This is the first state the object may undergo during its lifetime.
- Object creation* The execution of the operation that creates the object.

UML Diagram

Key Element	UML	Rationale
<i>Object</i>	Object 	It represents the created object. <i>Note:</i> since <i>Object</i> lacks a graphical representation in UML State Machine diagrams, Figure 9 does not depict this element.
	StateMachine 	In UML, a <i>StateMachine</i> represents the set of states an <i>Object</i> can go through during its lifetime in response to events.
<i>Object creation</i>	Initial Pseudostate 	It refers to the execution of the operation that creates the <i>Object</i> , leading it to its first state.
<i>First object's state</i>	State 	It models the first state of the <i>Object</i> .

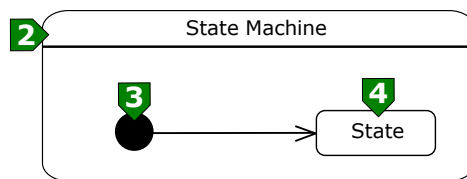


Figure 9. UML representation that models the context given by *StPI*

Mapping to PROV

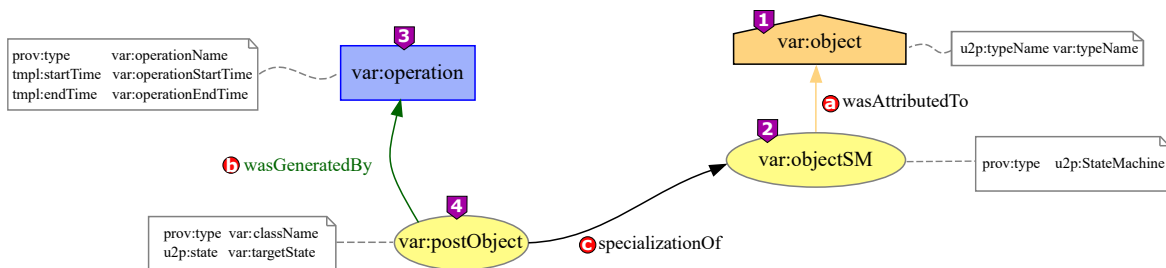


Figure 10. PROV template generated from the UML representation used in *StPI* (Figure 9)

PROV elements

UML	PROV / id	Rationale
Object 1	<code>prov:Agent 1 /</code> <code>var:object</code>	The Object 1 bears some form of responsibility for the existence of the StateMachine 2, since the existence of StateMachine 2 does not make sense without an Object 1. To reflect this fact, the Object 1 is mapped to a <code>prov:Agent</code> identified by <code>var:object</code> .
StateMachine 2	<code>prov:Entity 2 /</code> <code>var:objectSM</code>	The StateMachine 2 is a <code>prov:Entity</code> identified by <code>var:objectSM</code> . It reflects the abstraction of the object's states, which will be specialized by each concrete state the object goes through.
Initial Pseudostate 3	<code>prov:Activity 3 /</code> <code>var:operation</code>	The Initial Pseudostate 3, referring to the execution of the operation that creates the Object 1, is a <code>prov:Activity</code> with the identifier <code>var:operation</code> .
State 4	<code>prov:Entity 4 /</code> <code>var:postObject</code>	The State 4 is a <code>prov:Entity</code> identified by <code>var:postObject</code> . We use this name for this identifier because it corresponds to the state of the Object 1 after (post) the object creation.

Attributes

PROV Element	Attribute / Value	Description
<code>var:object 1</code>	<code>u2p:typeName /</code> <code>var:className</code>	The value <code>var:className</code> is the string with the name of the class to which <code>var:object 1</code> belongs.
<code>var:objectSM 2</code>	<code>prov:type /</code> <code>u2p:StateMachine</code>	The value <code>u2p:StateMachine</code> shows that <code>var:objectSM 2</code> is a state machine.
<code>var:operation 3</code>	<code>prov:type /</code> <code>var:operationName</code>	The value <code>var:operationName</code> is the name of the operation <code>var:operation 3</code> .
	<code>tmpl:startTime /</code> <code>var:operationStartTime</code>	The <code>var:operationStartTime</code> is an <code>xsd:dateTime</code> value for the start of <code>var:operation 3</code> .
	<code>tmpl:endTime /</code> <code>var:operationEndTime</code>	The <code>var:operationEndTime</code> is an <code>xsd:dateTime</code> value for the end of <code>var:operation 3</code> .
<code>var:postObject 4</code>	<code>prov:type /</code> <code>var:className</code>	The value <code>var:className</code> is the name of the class to which the object in the state <code>var:postObject 4</code> belongs.
	<code>u2p:state /</code> <code>var:targetState</code>	The value <code>var:targetState</code> is the string with the name of the state <code>var:postObject 4</code> .

PROV relations

PROV Relation	Description
a <code>prov:wasAttributedTo</code>	It is the assignment of responsibility to <code>var:object</code> for <code>var:objectSM</code> .
b <code>prov:wasGeneratedBy</code>	It is the completion of production of <code>var:postObject</code> by <code>var:operation</code> .
c <code>prov:specializationOf</code>	<code>var:postObject</code> is a specialization of <code>var:objectSM</code> .

Discussion

- Note that Figure 9 only contains simple states. We do not deal with composite or submachine states, and focus only on simple states, because the former may be transformed into the latter by resorting to a flattening process consisting of removing composite states as well as submachine states. In fact, to flatten State Machine diagrams is a very common approach in contexts such as model checking and code generation [6]. However, the user might be interested in representing composite states directly into the PROV templates, perhaps because she/he is interested in collecting information about them, or just because she/he does not want to flatten the State Machine diagram. We can give an insight into how composite states can be mapped to PROV by placing the elements from Figure 9 inside a Composite State 5 (see Figure 11). A reader familiar with the UML specification will realize that the semantics of the Initial Pseudostate 3 in Figures 9 and 11

are different, but these semantic nuances would have no effect on the PROV transformation. The transformation of Figure 11 is shown in Figure 12. Both Figure 11 and 12 highlight the added elements by blurring the elements coming from Figure 9 and Figure 10, respectively. Briefly speaking, the new Composite State 5 is translated into a `prov:Entity` identified by `var:compState` 5, which is associated with `var:objectSM` 2 and `var:targetState` 4 by means of the relations 1 `prov:specializationOf` and 3 `prov:hadMember`, respectively. At this point, it is also worth remarking that for this example we have used a *simple composite state* (i.e., Composite State 5), which means that only one substate is active at a given time within such a state; but we could have used *orthogonal composite states* instead, which means that within such a state several substates are active at the same time. Note that both types of *composite states* would be translated into the same PROV template (see Figure 12); nevertheless, the generated bindings would be different. In case of a *simple composite state*, as there can be only one active substate at the same time, there would be only one value associated with the variable `var:postObject` 4. Conversely, in case of an *orthogonal composite state*, `var:postObject` 4 will be associated with several values (as many as active states).

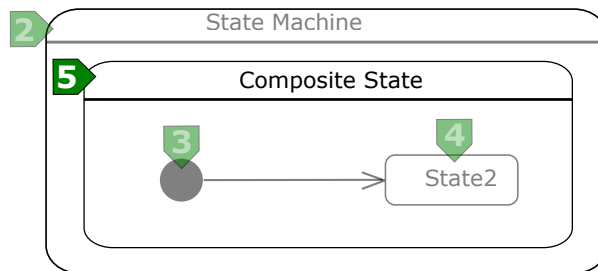


Figure 11. Excerpt of a UML State Machine diagram locating the UML elements from *StP1* in a *simple composite state*.

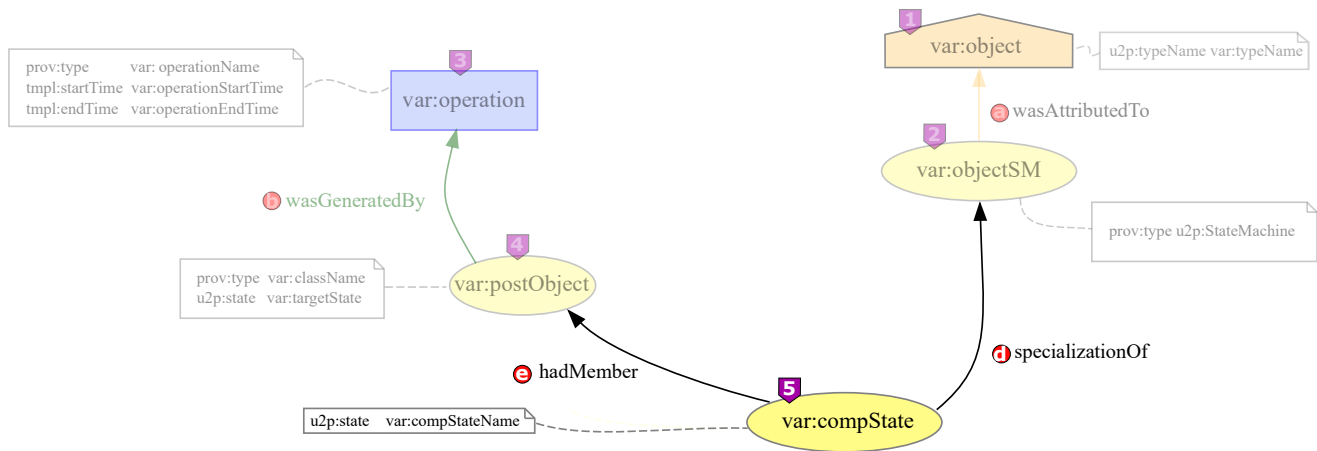


Figure 12. PROV template generated from the UML diagram in Figure 11

PROV elements

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

Attributes

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

PROV relations

PROV Relation	Description
i prov:specializationOf	<code>var:compState</code> is a specialization of <code>var:objectSM</code> .
e prov:hadMember	It states that <code>var:postObject</code> is one of the elements in <code>var:compState</code> .

Identifier State machine diagram Pattern 2 (StP2)

Context

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

Key elements

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

UML Diagram

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

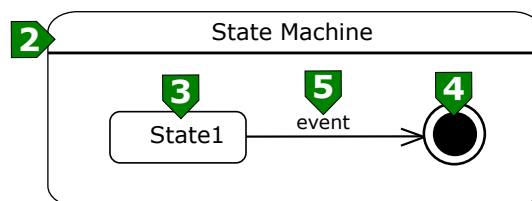


Figure 13. UML representation that models the context given by StP2

Mapping to PROV

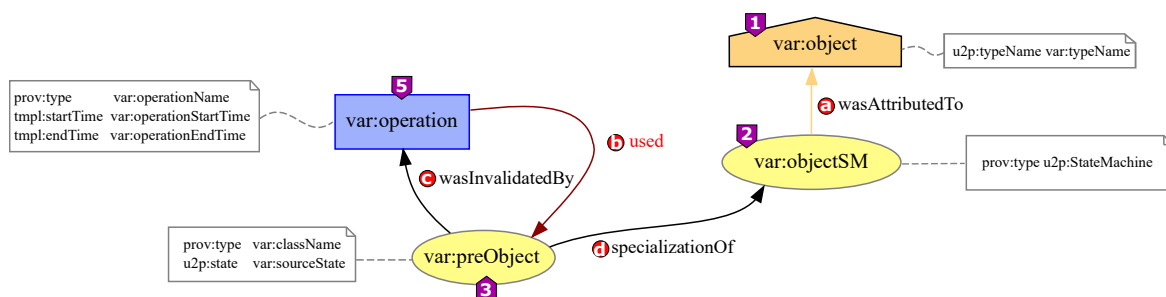


Figure 14. PROV template generated from the UML representation used in StP2 (Figure 13)

PROV elements

UML	PROV / id	Rationale
Object 1	<code>prov:Agent 1 /</code> <code>var:object</code>	The Object 1 bears some form of responsibility for the existence of the StateMachine 2, since the existence of StateMachine 2 does not make sense without an Object 1. To reflect this fact, the Object 1 is mapped to a <code>prov:Agent</code> identified by <code>var:object</code> .
StateMachine 2	<code>prov:Entity 2 /</code> <code>var:objectSM</code>	The StateMachine 2 is a <code>prov:Entity</code> identified by <code>var:objectSM</code> . It reflects the abstraction of the object's states, which will be specialized by each state the object goes through.
State 3	<code>prov:Entity 3 /</code> <code>var:preObject</code>	The State 3 is a <code>prov:Entity</code> identified by <code>var:preObject</code> . We use this name for this identifier because it corresponds to the state of the Object 1 before (pre) the execution of the operation.
FinalState 4	None /	Without mapping (see the discussion block for an explanation about this decision).
Event 5	<code>prov:Activity 5 /</code> <code>var:operation</code>	The Event 5 represents that the execution of an operation has taken place. Such an execution is a <code>prov:Activity</code> with the identifier <code>var:operation</code> .

Attributes

PROV Element	Attribute / Value	Description
<code>var:object 1</code>	<code>u2p:typeName /</code> <code>var:className</code>	The value <code>var:className</code> is the string with the name of the class to which <code>var:object 1</code> belongs.
<code>var:objectSM 2</code>	<code>prov:type /</code> <code>u2p:StateMachine</code>	The value <code>u2p:StateMachine</code> shows that <code>var:objectSM 2</code> is a state machine.
<code>var:preObject 3</code>	<code>prov:type /</code> <code>var:className</code>	The value <code>var:className</code> is the string with the name of the class to which the object in the state <code>var:preObject 3</code> belongs.
	<code>u2p:state /</code> <code>var:sourceState</code>	The value <code>var:sourceState</code> is the string with the name of the state <code>var:preObject 3</code> .
<code>var:operation 5</code>	<code>prov:type /</code> <code>var:operationName</code>	The value <code>var:operationName</code> is the name of the operation <code>var:operation 5</code> .
	<code>tmpl:startTime /</code> <code>var:operationStartTime</code>	The <code>var:operationStartTime</code> is an <code>xsd:dateTime</code> value for the start of <code>var:operation 5</code> .
	<code>tmpl:endTime /</code> <code>var:operationEndTime</code>	The <code>var:operationEndTime</code> is an <code>xsd:dateTime</code> value for the end of <code>var:operation 5</code> .

PROV relations

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

Discussion

- This pattern is consistent with *CIP2* because the completion of the object's behaviour usually involves its destruction. Among the different reasons why an *Object* can complete its behaviour, we can distinguish its destruction, from the remainder cases. In order to be consistent with *CIP2* (that addresses the execution of an operation which provokes the destruction of an object), we have decided not to explicitly map the `FinalState 4` in the PROV template but including its semantics (the completion of the object's behaviour) by the relation **c** `prov:wasInvalidatedBy` showing that `var:preObject 3` is not longer

available. However, if the user is interested in explicitly representing the `FinalState` into the PROV templates, we refer him/her to *StP3*, where the state of the *Object* before and after an *Operation execution* is included.

- Figure 13 only contains simple states. We do not deal with composite or submachine states, and focus only on simple states, because the former may be transformed into the latter by resorting to a flattening process consisting of removing composite states as well as submachine states. In fact, to flatten State Machine diagrams is a very common approach in contexts such as model checking and code generation [6]. However, the user might be interested in representing composite states directly into the PROV templates, perhaps because she/he is interested in collecting information about them, or just because she/he does not want to flatten the State Machine diagram. We can give an insight into how composite states can be mapped to PROV by placing the elements from Figure 13 inside a Composite State (see Figure 15). A reader familiar with the UML specification will realize that the semantics of the `FinalState` in Figures 13 and 15 are different, but these semantic nuances would have no effect on the PROV transformation. The transformation of Figure 15 is shown in Figure 16. Both Figure 15 and 16 highlight the added elements by blurring the elements coming from Figure 13 and Figure 14, respectively. Briefly speaking, the new Composite State is translated into a `prov:Entity` identified by `var:compState`, which is associated with `var:objectSM` and `var:preObject` by means of the relations `prov:specializationOf` and `prov:hadMember`, respectively. At this point, it is also worth remarking that for this example we have used a *simple composite state* (i.e., Composite State), which means that only one substate is active at a given time within such a state; but we could have used *orthogonal composite states* instead, which means that within such a state several substates are active at the same time. Note that both types of *composite states* would be translated into the same PROV template (see Figure 16); nevertheless, the generated bindings would be different. In case of a *simple composite state*, as there can be only one active substate at the same time, there would be only one value associated with the variable `var:preObject`. Conversely, in case of an *orthogonal composite state*, `var:preObject` will be associated with several values (as many as active states).

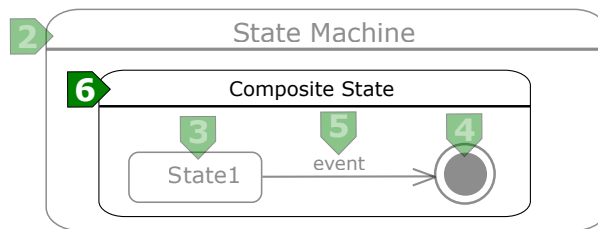


Figure 15. Excerpt of a UML State Machine diagram locating the UML elements from *StP2* in a *simple composite state*.

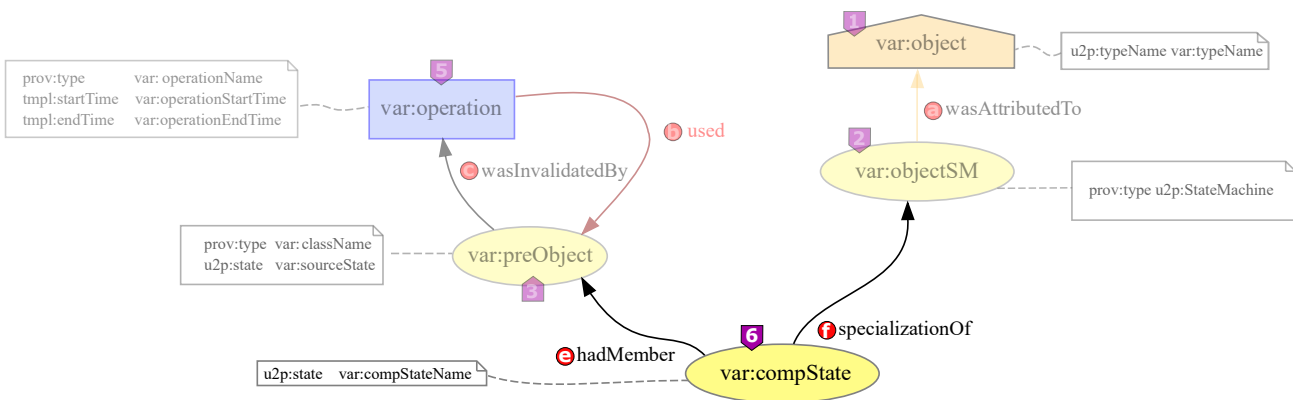




Figure 16. PROV template generated from the UML diagram in Figure 15



PROV elements

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

Attributes

PROV Element	Attribute / Value	Description
<code>var:compState</code> 	<code>u2p:state /</code> <code>var:compStateName</code>	The value <code>var:compStateName</code> is the string with the name of the state <code>var:compState</code> 

PROV relations

PROV Relation	Description
 <code>prov:hadMember</code>	It states that <code>var:preObject</code> is one of the elements in <code>var:compState</code> .
 <code>prov:specializationOf</code>	<code>var:compState</code> is a specialization of <code>var:objectSM</code> .

Identifier State machine diagram Pattern 3 (StP3)

Context

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

Key elements

Object The object that changes its state.

Pre-operation object's state The state of the object before the execution of the operation. This is one of the states the object may undergo during its lifetime.

Post-operation object's state The state of the object after the execution of the operation. This is one of the states the object may undergo during its lifetime.

Operation execution The execution of the operation that leads a change in the *Object's* state.

UML Diagram

Key Element	UML	Rationale
<i>Object</i>	Object 1	It represents the object that changes its state. <i>Note:</i> since <i>Object</i> lacks a graphical representation in UML State Machine diagrams, Figure 17 does not depict this element.
	StateMachine 2	In UML, a <i>StateMachine</i> can be used to express the set of object's states through which the <i>Object</i> goes during its lifetime in response to events.
<i>Pre-operation object's state</i>	State 3	It models the state of the <i>Object</i> before the <i>Operation execution</i> .
<i>Post-operation object's state</i>	State 4	It models the state of the <i>Object</i> after the <i>Operation execution</i> .
<i>Operation execution</i>	Event 5	It specifies that the <i>Operation execution</i> that triggers the change in the <i>Object's</i> state has taken place.

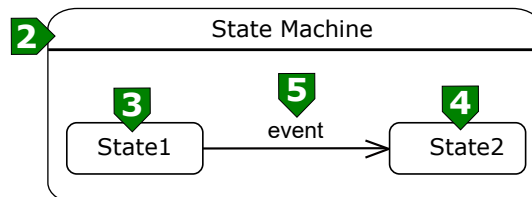


Figure 17. UML representation that models the context given by StP3

Mapping to PROV

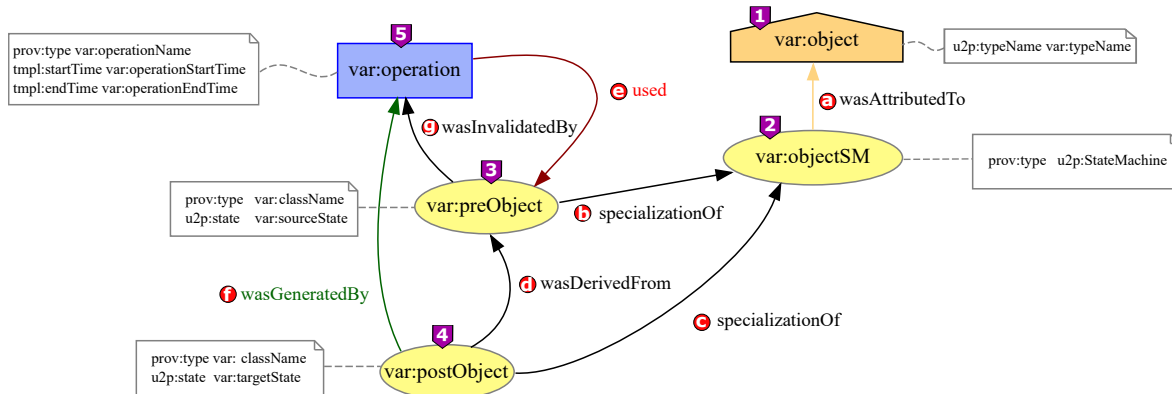


Figure 18. PROV template generated from the UML representation used in *StP3* (Figure 17)

PROV elements

UML	PROV / id	Rationale
Object 1	<code>prov:Agent 1 /</code> <code>var:object</code>	The Object 1 bears some form of responsibility for the existence of the StateMachine 2, since the existence of StateMachine 2 does not make sense without an Object 1. To reflect this fact, the Object 1 is mapped to a <code>prov:Agent</code> identified by <code>var:object</code> .
StateMachine 2	<code>prov:Entity 2 /</code> <code>var:objectSM</code>	The StateMachine 2 is a <code>prov:Entity</code> identified by <code>var:objectSM</code> . It reflects the abstraction of the object's states, which will be specialized by each state the object goes through.
State 3	<code>prov:Entity 3 /</code> <code>var:preObject</code>	The State 3 is a <code>prov:Entity</code> identified by <code>var:preObject</code> . We use this name for this identifier because it corresponds to the state of the Object 1 before (pre) the execution of the operation.
State 4	<code>prov:Entity 4 /</code> <code>var:postObject</code>	The State 4 is a <code>prov:Entity</code> identified by <code>var:postObject</code> . We use this name for this identifier because it corresponds to the state of the Object 1 after (post) the execution of the operation.
Event 5	<code>prov:Activity 5 /</code> <code>var:operation</code>	The Event 5 represents that the execution of an operation has taken place. Such an execution is a <code>prov:Activity</code> with the identifier <code>var:operation</code> .

Attributes

PROV Element	Attribute / Value	Description
var:object 1	u2p:typeName / var:className	The value var:className is the string with the name of the class to which var:object 1 belongs.
var:objectSM 2	prov:type / u2p:StateMachine	The value u2p:StateMachine shows that var:objectSM 2 is a state machine.
var:preObject 3	prov:type / var:className	The value var:className is the name of the class to which the object in the state var:preObject 3 belongs.
	u2p:state / var:sourceState	The value var:sourceState is the string with the name of the state var:preObject 3 .
var:postObject 4	prov:type / var:className	The value var:className is the name of the class to which the object in the state var:postObject 4 belongs.
	u2p:state / var:targetState	The value var:targetState is the string with the name of the state var:postObject 4 .
var:operation 5	prov:type / var:operationName	The value var:operationName is the name of the operation var:operation 5 .
	tmpl:startTime / var:operationStartTime	The var:operationStartTime is an xsd:dateTime value for the start of var:operation 5 .
	tmpl:endTime / var:operationEndTime	The var:operationEndTime is an xsd:dateTime value for the end of var:operation 5 .

PROV relations

PROV Relation	Description
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

- Figure 17 only contains simple states. We do not deal with composite or submachine states, and focus only on simple states, because the former may be transformed into the latter by resorting to a flattening process consisting of removing composite states as well as submachine states. In fact, to flatten State Machine diagrams is a very common approach in contexts such as model checking and code generation [6]. However, the user might be interested in representing composite states directly into the PROV templates, perhaps because she/he is interested in collecting information about them, or just because she/he does not want to flatten the State Machine diagram. We can give an insight into how composite states can be mapped to PROV by placing the elements from Figure 17 inside a Composite State **5** (see Figure 19). A reader familiar with the UML specification will realize that the semantics of the UML representation in Figures 17 and 19 are different, but these semantic nuances would have no effect on the PROV transformation. The transformation of Figure 19 is shown in Figure 20. Both Figure 19 and 20 highlight the added elements by blurring the elements coming from Figure 17 and Figure 18, respectively. Briefly speaking, the new Composite State **5** is translated into a prov:Entity identified by var:compState **5**, which is associated with var:objectSM **2**, var:preObject **3**, and var:postObject **4** by means of the relations **f** prov:specializationOf, **h** prov:hadMember, and **i** prov:hadMember, respectively. At this point, it is also worth remarking that for this example we have used a *simple composite state* (i.e., Composite State **5**), which means that only one substate is active at a given time within such a state; but we could have used *orthogonal composite states* instead, which means that within such a state several substates are active at the same time. Note that both types of *composite states* would be translated into the same PROV template (see Figure 20); nevertheless, the generated bindings would be

different. In case of a *simple composite state*, as there can be only one active substate at the same time, there would be only one value associated with the variable `var:preObject` ❸ and another value with `var:postObject` ❹. Conversely, in case of an *orthogonal composite state*, `var:preObject` ❸ and `var:postObject` ❹ will be associated with several values (as many as active states).

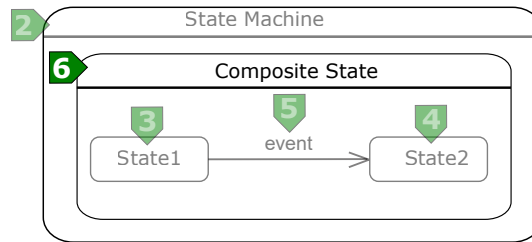


Figure 19. Excerpt of a UML State Machine diagram locating the UML elements from *StP3* in a *simple composite state*.

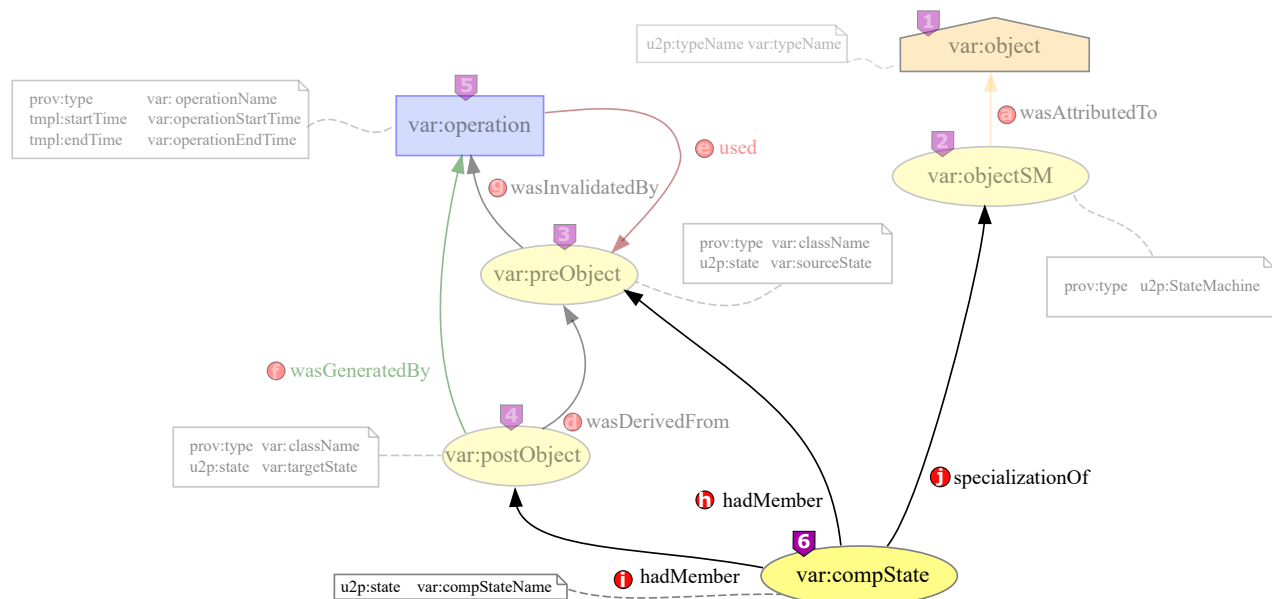


Figure 20. PROV template generated from the UML diagram in Figure 19




PROV elements

UML	PROV / id	Rationale
Composite State ❸	<code>prov:Entity</code> ❸ / <code>var:compState</code>	The Composite State ❸ is a <code>prov:Entity</code> identified by <code>var:compState</code> .

Attributes

PROV Element	Attribute / Value	Description
<code>var:compState</code> ❸	<code>u2p:state</code> / <code>var:compStateName</code>	The value <code>var:compStateName</code> is the string with the name of the state <code>var:compState</code> ❸

PROV relations

PROV Relation	Description
 prov:hadMember	It states that <code>var:preObject</code> is one of the elements in <code>var:compState</code> .
 prov:hadMember	It states that <code>var:postObject</code> is one of the elements in <code>var:compState</code> .
 prov:specializationOf	<code>var:compState</code> is a specialization of <code>var:objectSM</code> .

6 UML Class Diagrams

Pattern identifier	Context	Page
CIP1	The execution of an operation provokes the creation of a new object.	33
CIP2	The execution of an operation provokes the destruction of an object.	36
CIP3	The execution of an operation on an object returns values of concrete object's attributes. The values are returned as they are, without any further processing. This execution does not provoke the change of the object's status.	38
CIP4	The execution of an operation on an object returns values that are computed based on the object's status as a whole (the values of concrete attributes involved in the computation are unknown or irrelevant). This execution does not provoke the change of the object's status.	41
CIP5	The execution of an operation on an object returns values that are computed based on values of concrete object's attributes. This execution does not provoke the change of the object's status.	44
CIP6	The execution of an operation on an object changes the object's status as a whole (the concrete modified attributes are unknown or irrelevant).	48
CIP7	The execution of an operation on an object directly sets the information passed to the operation as values of concrete object's attributes, thus provoking a change in the object's status.	53
CIP8	The execution of an operation on an object changes the values of concrete object's attributes, thus provoking a change in the object's status.	58
CIP9	The execution of an operation on an object removes element(s) from a concrete object's collection attribute, thus provoking a change in the object's status.	63
CIP10	The execution of an operation on an object directly adds the information passed to the operation as new element(s) of a concrete object's collection attribute, thus provoking a change in the object's status.	68

Identifier Class diagram Pattern 1 (CIP1)

Context

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

Key elements

Object The object created as a consequence of the execution of the operation.

Operation execution The execution of the operation.

Input data The information (if any) passed into the *Operation execution*.

Object's attributes The characteristics of the *Object*.

UML Diagram

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

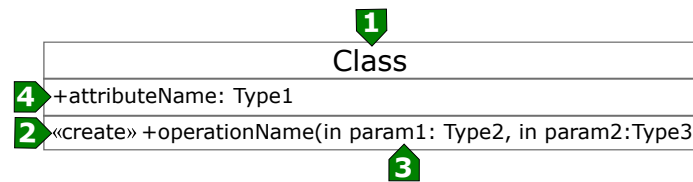


Figure 21. UML representation that models the context given by CIP1

Mapping to PROV

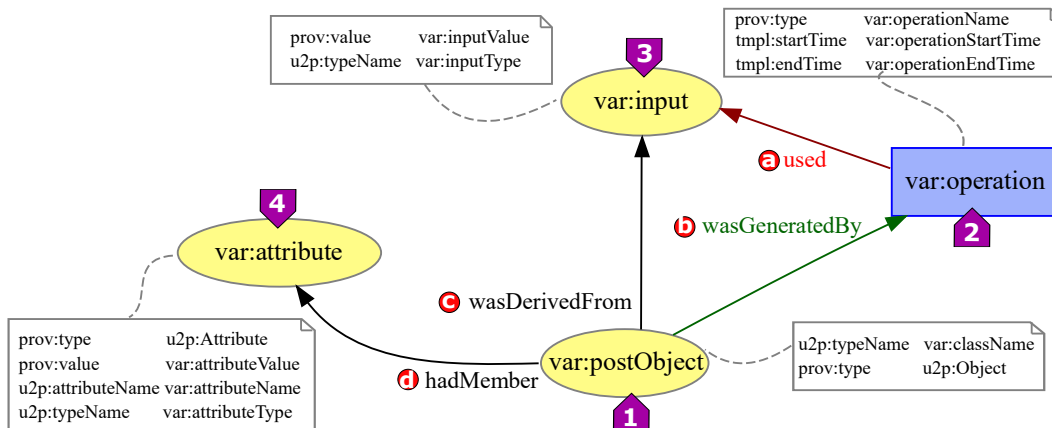


Figure 22. PROV template generated from the UML representation used in CIP1 (Figure 21)

PROV elements

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

Attributes



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

PROV relations

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

Discussion

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

lacks *Input data*, the UML representation in Figure 21 will not include `Input Parameters` . As a consequence, the resulting PROV template in Figure 22 will also lack `var:input`  and its associated PROV relations.

Context

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

Key elements

Object The object destroyed as a consequence of the execution of the operation.

Operation execution The execution of the operation.

UML Diagram

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

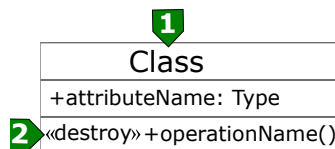


Figure 23. UML representation that models the context given by CIP2

Mapping to PROV

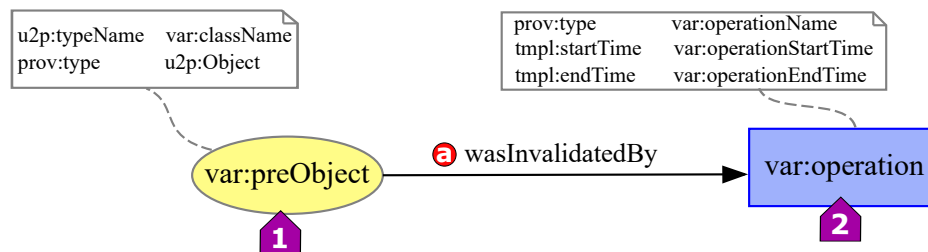


Figure 24. PROV template generated from the UML representation used in CIP2 (Figure 23)

PROV elements

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

Attributes

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

PROV relations

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

Discussion

- This pattern is consistent with *CIP2* because the completion of the object's behaviour usually involves its destruction.

Identifier Class diagram Pattern 3 (CIP3)

Context

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

Key elements

<i>Object</i>	The object on which the operation is executed.
<i>Operation execution</i>	The execution of the operation.
<i>Input data</i>	The information (if any) passed into the <i>Operation execution</i> .
<i>Output data</i>	The information obtained from the <i>Operation execution</i> .

UML Diagram

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

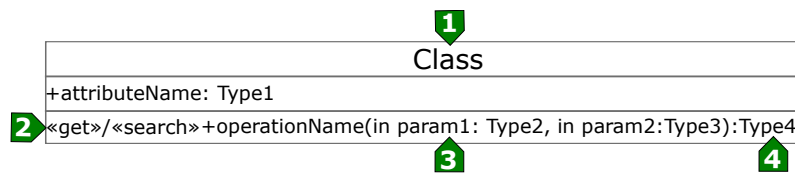


Figure 25. UML representation that models the context given by CIP3

Mapping to PROV

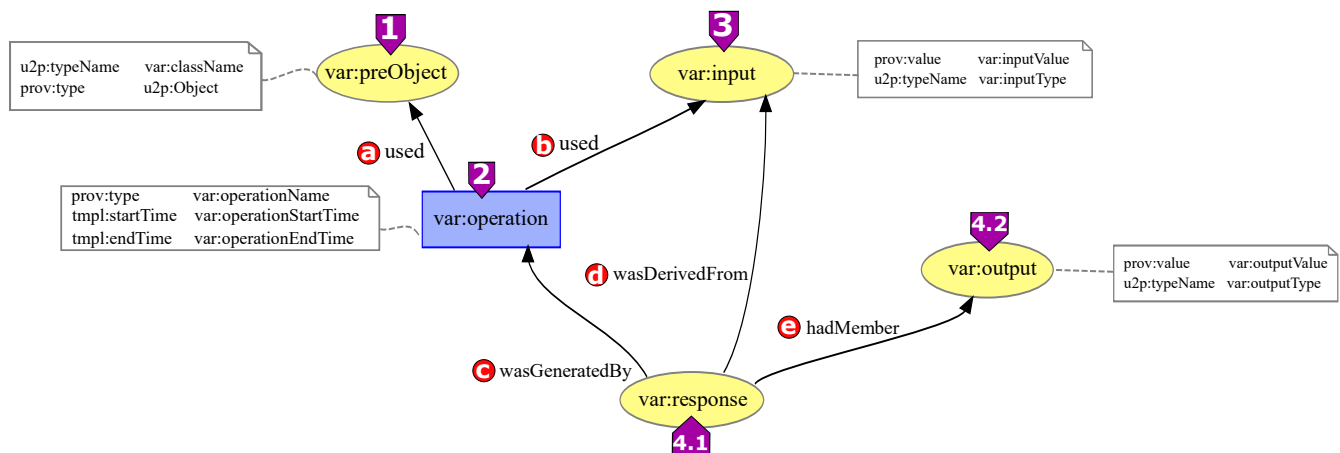


Figure 26. PROV template generated from the UML representation used in CIP3 (Figure 25)

PROV elements

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








Attributes

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

PROV relations

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

Discussion

- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have considered this circumstance with the aim of covering a wider spectrum of cases. When the executed operation lacks *Input data*, the UML representation in Figure 25 will not include `Input Parameters` . As a consequence, the resulting PROV template in Figure 26 will also lack `var:input`  and its associated PROV relations.
- In order to homogenise the UML Class representations in *CIP1-CIP10*, *Output data* have been specified by `Output Parameters` with *return* direction. Nevertheless, these `Output Parameters` could have been modelled with either *inout* or *out* directions, having no effect in their transformation to PROV.
- A concrete nuance in this pattern is that the *Output data* (`var:output`) are not computed by the *Operation execution* (`var:operation`); that is, these data already existed before the *Operation execution*. Thus, a `prov:wasGeneratedBy` relation between `var:output` and `var:operation` does not make sense in this pattern (in contrast to *CIP4-CIP6*, and *CIP7-CIP10* when they consider *Output data*). To reflect this pattern's nuance in the PROV template and taking into account the consistency between the different kinds of patterns, we have taken inspiration from how UML sequence diagram patterns (e.g., *SeqP2*) address the *Output data*. We have made this decision because the semantics of the sequence diagrams patterns (in terms of *Output data*) bears a strong resemblance with this pattern. Thus, we have included a `prov:Entity` identified by `var:response`  related to (1) `var:operation` , by means of  `prov:wasGeneratedBy` (to represent the fact that it is the response who is generated by the *Operation execution*), and (2) `var:output` , through  `prov:hadMember` (to show that such a response is composed by the concrete output values).

Identifier Class diagram Pattern 4 (CIP4)

Context

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

Key elements

<i>Object</i>	The object on which the operation is executed.
<i>Operation execution</i>	The execution of the operation.
<i>Input data</i>	The information (if any) passed into the <i>Operation execution</i> .
<i>Output data</i>	The information obtained from the <i>Operation execution</i> .

UML Diagram

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

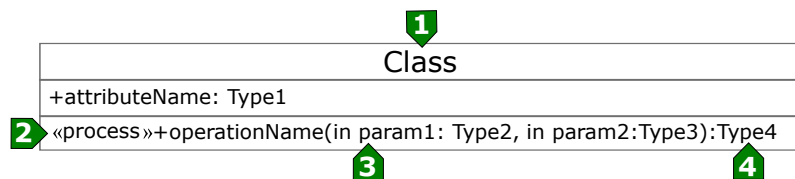


Figure 27. UML representation that models the context given by CIP4

Mapping to PROV

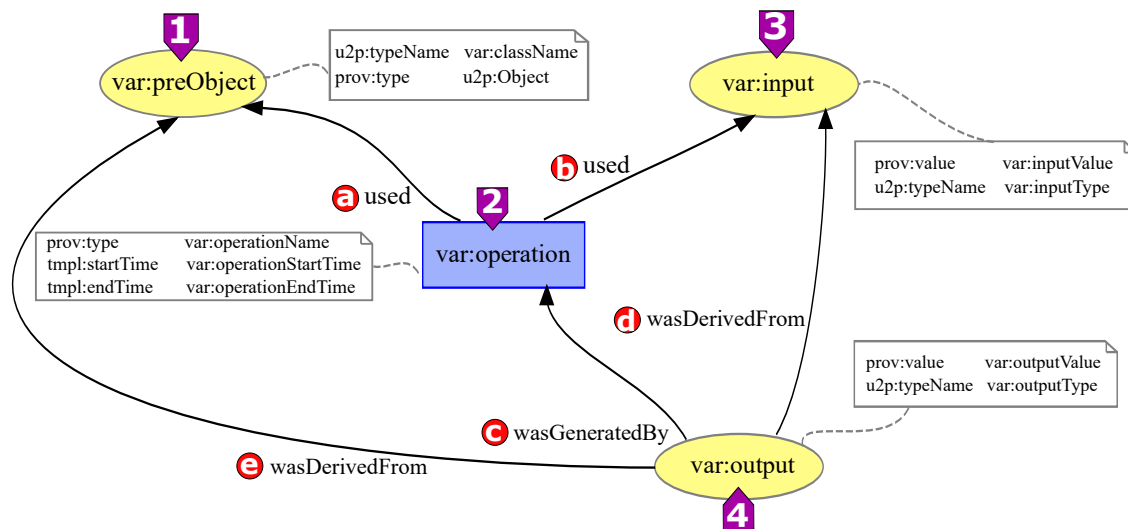


Figure 28. PROV template generated from the UML representation used in *CIP4* (Figure 27)

PROV elements

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

Attributes

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

PROV relations

PROV Relation	Description
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

- Among the Class Diagrams patterns, both *CIP4* and *CIP5* address the execution of an operation that returns values computed based on information included on an object. While *CIP4* considers the object's status as a whole (without taking into account the concrete attributes' values considered for its computation), *CIP5* identifies the concrete attributes used to compute such information. Thus, *CIP4* gives a coarser grained provenance than *CIP5*.
- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have considered this circumstance with the aim of covering a wider spectrum of cases. When the executed operation lacks *Input data*, the UML representation in Figure 27 will not include *Input Parameters* 3▶. As a consequence, the resulting PROV template in Figure 28 will also lack var:input 3▶ and its associated PROV relations.
- In order to homogenise the UML Class representations in *CIP1-CIP10*, *Output data* have been specified by *Output Parameters* with *return* direction. Nevertheless, these *Output Parameters* could have been modelled with either *inout* or *out* directions, having no effect in their transformation to PROV.

Identifier Class diagram Pattern 5 (CIP5)

Context

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

Key elements

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

UML Diagram

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

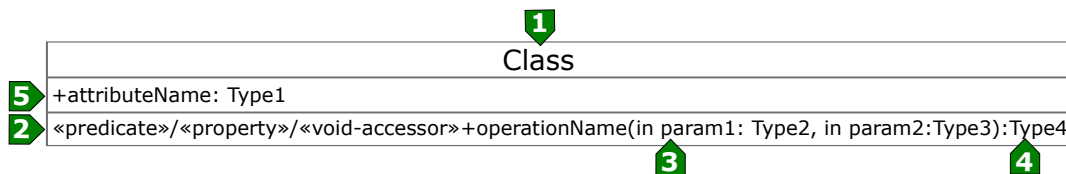


Figure 29. UML representation that models the context given by CIP5

Mapping to PROV

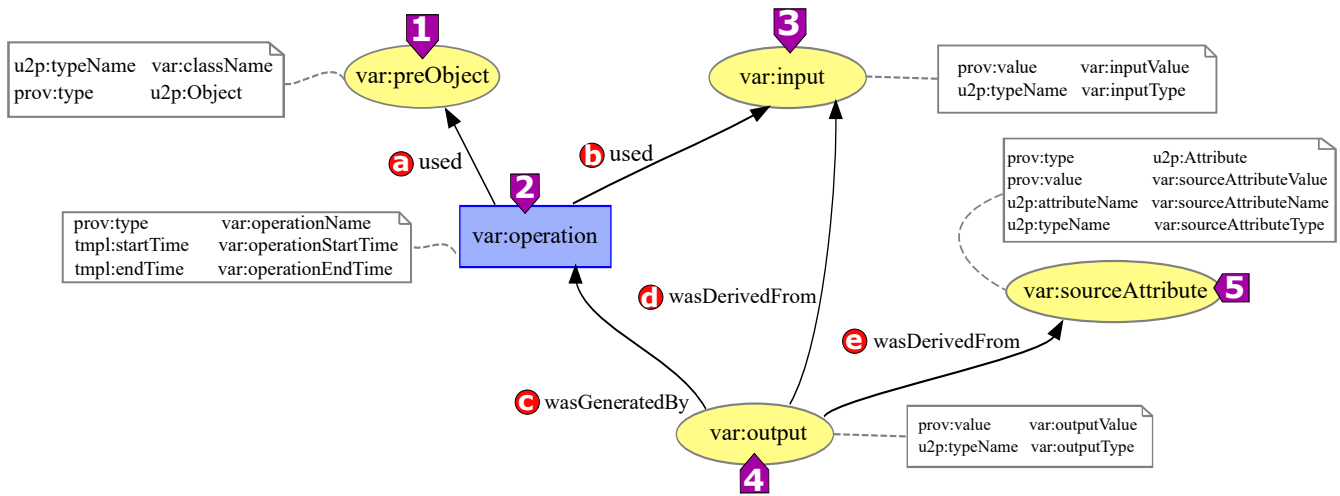


Figure 30. PROV template generated from the UML representation used in *CIP5* (Figure 29)

PROV elements

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

Attributes

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




PROV relations

PROV Relation	Description
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

- Among the Class Diagrams patterns, both *CIP5* and *CIP4* address the execution of an operation that returns values computed based on information included on an object. While *CIP5* identifies the concrete attributes used to compute such information, *CIP4* considers the object's status as a whole (without taking into account the concrete attributes' values considered for its computation). Thus, *CIP5* gives a finer grained provenance than *CIP4*.
- A question that might arise is why in Figure 30 var:sourceAttribute 5 is not associated with var:preObject 1 (which represents the object with the status before the execution of the operation) by means of a prov:hadMember relation, whether var:sourceAttribute 5 is an attribute of var:preObject 1. We have made this decision because an object that acts as a var:preObject in an operation execution, was a var:postObject (which represents the object with the

status after the execution of the operation) in a previous operation execution. Thus, the attributes associated to such an object in a `var:preObject` were registered when it previously played the role of `var:postObject`.

- The stereotypes `«predicate»`, `«property»`, and `«void-accessor»` denote behaviours with specific nuances. Nevertheless, these nuances do not have impact in the translation into PROV since all of them compute *Output data* based on concrete object's attributes without modifying the object's status. Concretely, `«predicate»` denotes that the operation returns boolean values, `«property»` does not restrict the type of the returned values, and `«void-accessor»` returns the information through a parameter. As we said previously, there is no distinction in the transformation into PROV; however, some of the nuances given by the stereotypes will be included in the generated provenance through the values assigned to the template's variables. For instance, `«predicate»` defines the *Output data* as boolean, fact that is included in the provenance through the value assigned to `var:outputType` in `var:output` .
- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have considered this circumstance with the aim of covering a wider spectrum of cases. When the executed operation lacks *Input data*, the UML representation in Figure 29 will not include *Input Parameters* . As a consequence, the resulting PROV template in Figure 30 will also lack `var:input`  and its associated PROV relations.

Identifier Class diagram Pattern 6 (CIP6)

Context

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

Key elements

- Object* The object on which the operation is executed.
 - Pre-operation object* The object with the status before the execution of the operation.
 - Post-operation object* The object with the status after the execution of the operation.
- Operation execution* The execution of the operation.
- Input data* The information (if any) passed into the *Operation execution*.
- Object’s attributes* All the characteristics of the *Object*.

UML Diagram

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

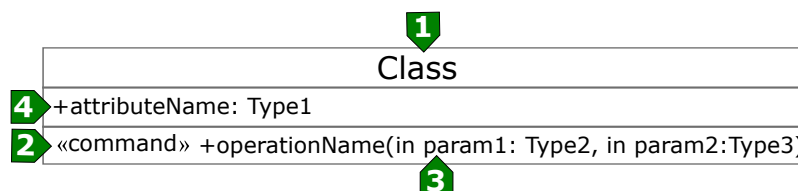


Figure 31. UML representation that models the context given by CIP6

Mapping to PROV

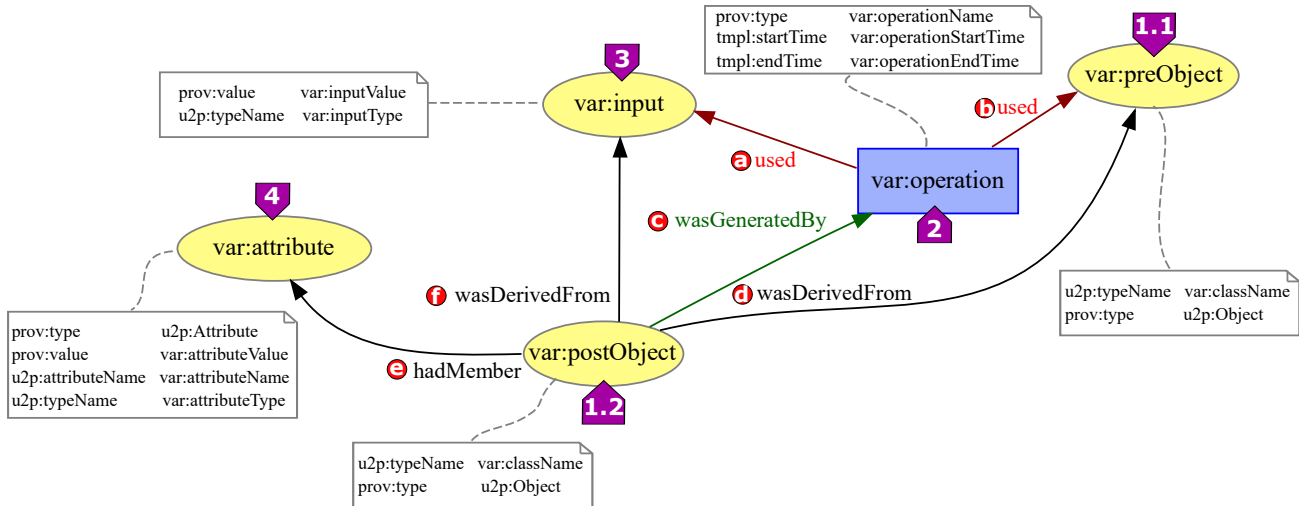


Figure 32. PROV template generated from the UML representation used in *CIP6* (Figure 31)

PROV elements

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

Attributes

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

PROV relations

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

Discussion

- Among the Class Diagrams patterns, patterns from [CIP6](#) to [CIP10](#) address the execution of operations that change an object's status. While, [CIP6](#) changes the object's status as a whole (being the concrete modified attributes unknown or irrelevant), in patterns [CIP7-CIP10](#) the concrete attributes modified by the *Operation execution* are explicitly known. In contrast to [CIP7](#) which directly sets the information passed into the *Operation execution* as values of concrete object's attributes, the other mentioned patterns use such information to change the object's status as a whole or the values of concrete object's attributes. It must also be noted that patterns [CIP9](#) and [CIP10](#) address the execution of operations which remove or add elements from/into an object's collection attribute, while patterns [CIP7](#) and [CIP8](#) affect either a univalued attribute or a collection attribute as a whole.
- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we do not consider this circumstance with the aim of covering a wider spectrum of cases. When the executed operation lacks *Input*

data, the UML representation in Figure 31 will not include Input Parameters 5. As a consequence, the resulting PROV template in Figure 32 will also lack var:input 5 and its associated PROV relations.

- A question that might arise is why in Figure 32 var:attribute 4 is associated with var:postObject 1.2 (which represents the object with the status after the execution of the operation), but it is not associated with var:preObject 1.1 (the object with the status before the execution). We have made this decision because an object that acts as a var:preObject in an operation execution, was a var:postObject in a previous operation execution. Thus, the attributes associated to such an object in a var:preObject were registered when it previously played the role of var:postObject.
- Stereotypes «command» and «non-void-command» denote that the operation performs a change to the object's status as a whole. They differ in that an operation stereotyped by «non-void-command» returns information, while a «command» stereotyped operation does not. Due to the fact that the context of this pattern does not explicitly state that output data are obtained from the Operation execution, we represented this context in UML using the stereotype «command» (see Figure 31).

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

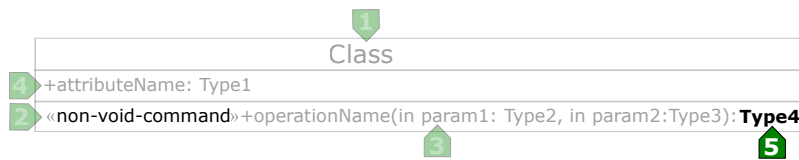


Figure 33. UML representation that models the context given by CIP6, including Output Parameters.

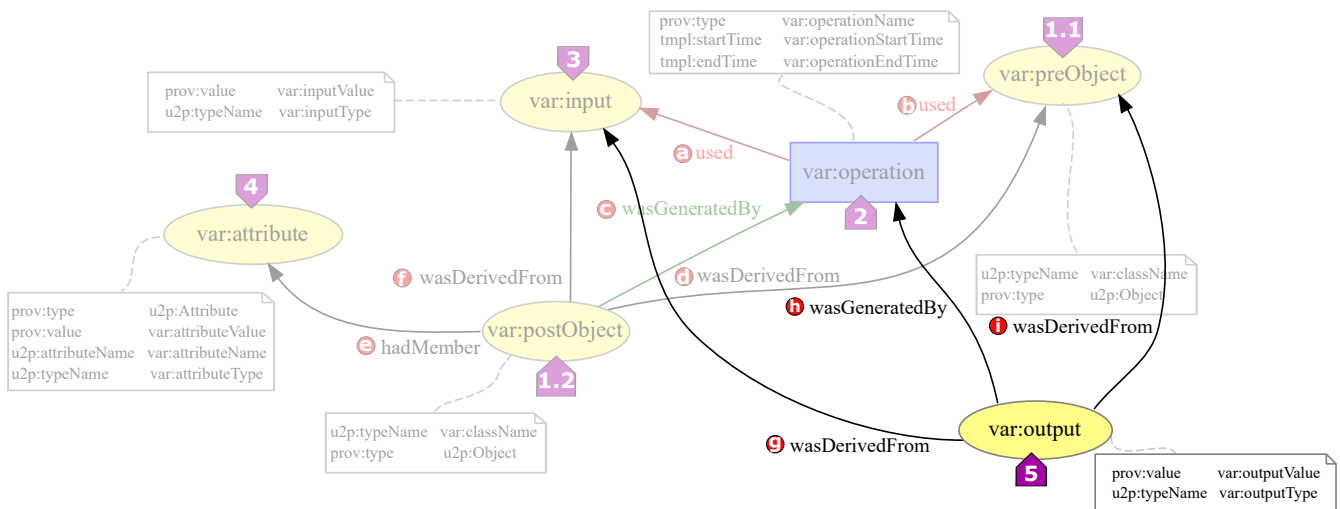





Figure 34. PROV template generated from the UML representation used in CIP6, including Output Parameters (Figure 33)




PROV elements

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

PROV relations

PROV Relation	Description
 prov:wasDerivedFrom	It is the construction of <code>var:output</code> based on <code>var:input</code> .
 prov:wasGeneratedBy	It is the completion of production of <code>var:output</code> by <code>var:operation</code> .
 prov:wasDerivedFrom	It is the construction of <code>var:output</code> based on <code>var:preObject</code> .

Attributes

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

Identifier Class diagram Pattern 7 (CIP7)

Context

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

Key elements

<i>Object</i>	The object on which the operation is executed.	
	<i>Pre-operation object</i>	The object with the status before the execution of the operation.
	<i>Post-operation object</i>	The object with the status after the execution of the operation.
<i>Operation execution</i>	The execution of the operation.	
<i>Input data</i>	The information passed into the <i>Operation execution</i> .	
<i>Object’s attributes</i>	All the characteristics of the <i>Object</i> . Since, as a consequence of the <i>Operation execution</i> , the values of some attributes change, we have identified:	
	<i>Modified attributes</i>	The modified <i>Object’s attributes</i> .
	<i>Unmodified attributes</i>	The not modified <i>Object’s attributes</i> .

UML Diagram

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

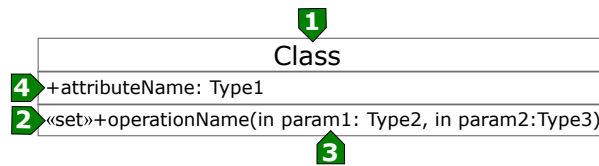


Figure 35. UML representation that models the context given by CIP7

Mapping to PROV

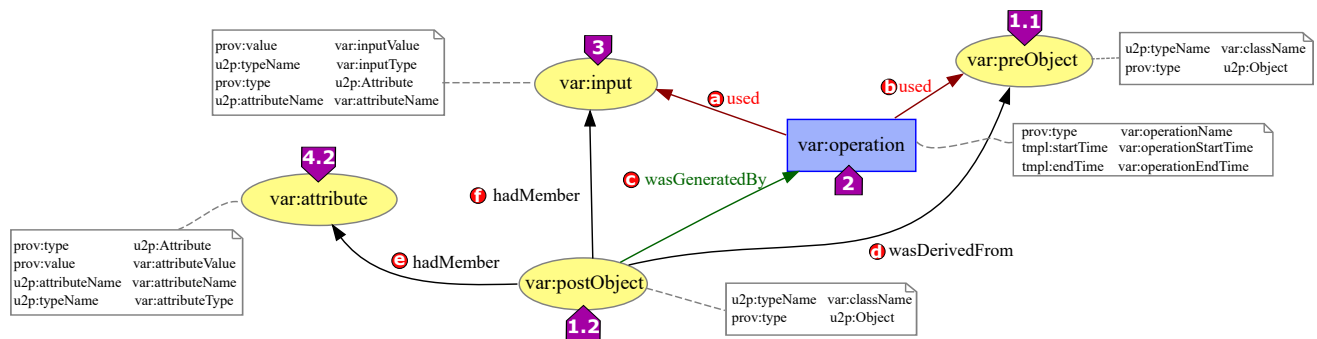


Figure 36. PROV template generated from the UML representation used in CIP7 (Figure 35)

PROV elements

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

Attributes

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

PROV relations

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

Discussion

- Among the Class Diagrams patterns, patterns from *CIP6* to *CIP10* address the execution of operations that change an object's status. While, *CIP6* changes the object's status as a whole (being the concrete modified attributes unknown or irrelevant), in patterns *CIP7-CIP10* the concrete attributes modified by the *Operation execution* are explicitly known. In contrast to *CIP7* which directly sets the information passed into the *Operation execution* as values of concrete object's attributes, the

other mentioned patterns use such information to change the object's status as a whole or the values of concrete object's attributes. It must also be noted that patterns *CIP9* and *CIP10* address the execution of operations which remove or add elements from/into an object's collection attribute, while patterns *CIP7* and *CIP8* affect either a univalued attribute or a collection attribute as a whole.

- A question that might arise is why in Figure 36 `var:attribute` 4.2 is associated with `var:postObject` 1.2 (which represents the object with the status after the execution of the operation), but it is not associated with `var:preObject` 1.1 (the object with the status before the execution). We have made this decision because an object that acts as a `var:preObject` in an operation execution, was a `var:postObject` in a previous operation execution. Thus, the attributes associated to such an object in a `var:preObject` were registered when it previously played the role of `var:postObject`.
- This context states that the *Input data* are directly set as values of certain object's attributes, which means that the *Input data* correspond directly to the *Modified attributes*. This fact is represented in the PROV template by means of the pair attribute/value `prov:type/u2p:Attribute` of `var:input` 3, and the relation `prov:hadMember` between `var:postObject` 1.2 and `var:input` 3. Additionally, `var:input` 3 has the attribute `u2p:attributeName` whose value `var:attributeName` denotes the name of the modified attribute.
- Although the *context* of this pattern does not explicitly state that output data should be obtained from the *Operation execution*, this could be the case. However, we do not include this output data in this pattern description to avoid overburden both the UML and PROV explanations with information out of the scope of the *context*.

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

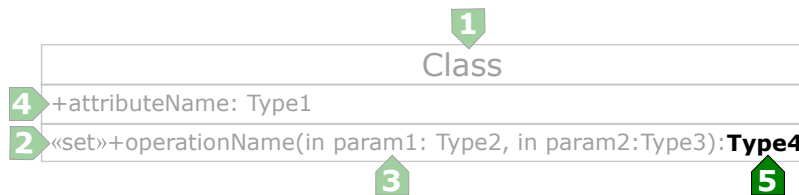


Figure 37. UML representation that models the context given by *CIP7*, including *Output Parameters*.

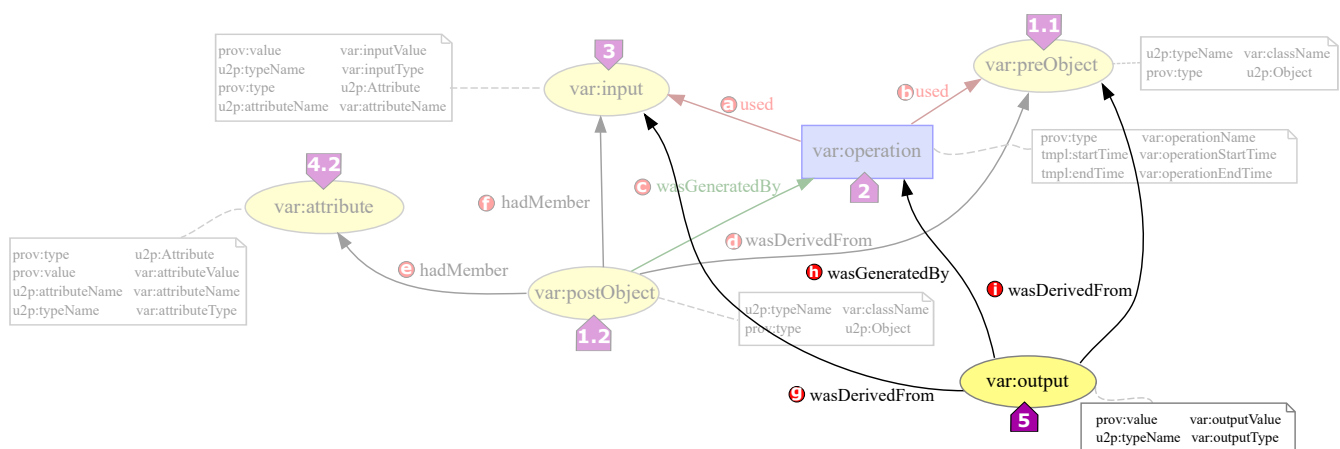








Figure 38. PROV template generated from the UML representation used in *CIP7*, including *Output Parameters* (Figure 37)




PROV elements

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

PROV relations

PROV Relation	Description
 prov:wasDerivedFrom	It is the construction of var:output 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:preObject.

Attributes

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

Identifier Class diagram Pattern 8 (CIP8)

Context

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

Key elements

<i>Object</i>	The object on which the operation is executed.
<i>Pre-operation object</i>	The object with the status before the execution of the operation.
<i>Post-operation object</i>	The object with the status after the execution of the operation.
<i>Operation execution</i>	The execution of the operation.
<i>Input data</i>	The information (if any) passed into the <i>Operation execution</i> .
<i>Object’s attributes</i>	All the characteristics of the <i>Object</i> . Since, as a consequence of the <i>Operation execution</i> , the values of some attributes change, we have identified:
<i>Modified attributes</i>	The modified <i>Object’s attributes</i> .
<i>Unmodified attributes</i>	The not modified <i>Object’s attributes</i> .

UML Diagram

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

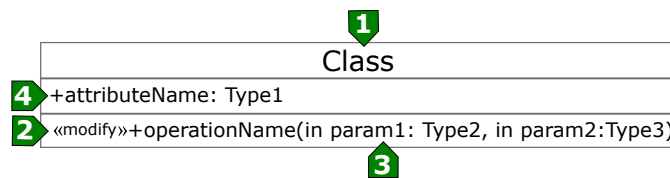


Figure 39. UML representation that models the context given by CIP8

Mapping to PROV

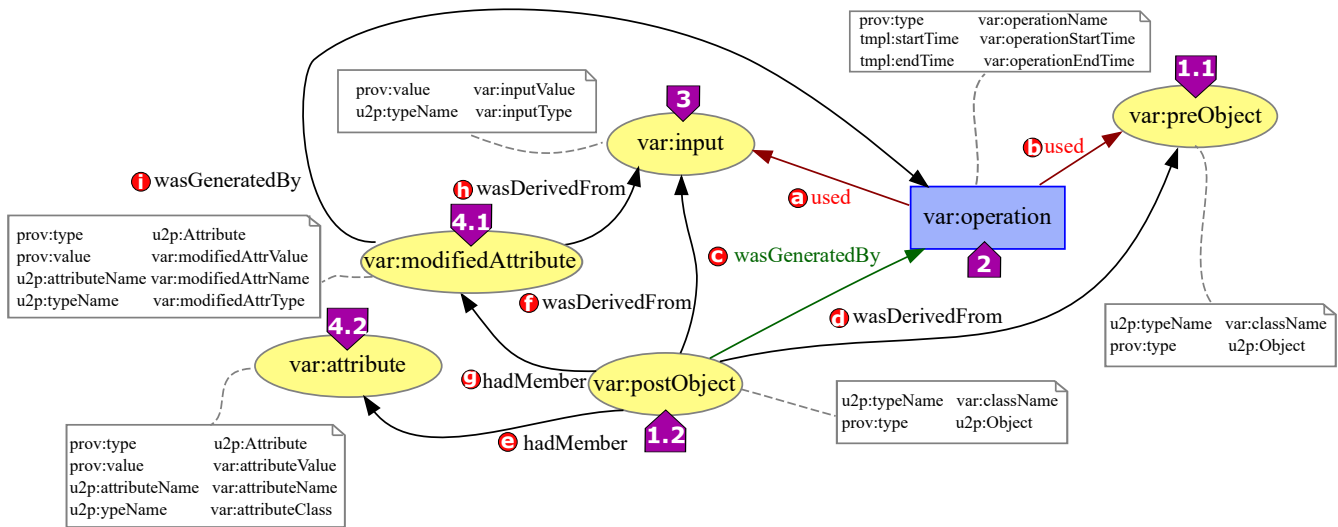


Figure 40. PROV template generated from the UML representation used in CIP8 (Figure 39)

PROV elements

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

Attributes

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

PROV relations

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

Discussion

- Among the Class Diagrams patterns, patterns from *CIP6* to *CIP10* address the execution of operations that change an object's status. While, *CIP6* changes the object's status as a whole (being the concrete modified attributes unknown or irrelevant), in patterns *CIP7-CIP10* the concrete attributes modified by the *Operation execution* are explicitly known. In contrast to *CIP7* which directly sets the information passed into the *Operation execution* as values of concrete object's attributes, the other mentioned patterns use such information to change the object's status as a whole or the values of concrete object's attributes. It must also be noted that patterns *CIP9* and *CIP10* address the execution of operations which remove or add elements from/into an object's collection attribute, while patterns *CIP7* and *CIP8* affect either a univalued attribute or a collection attribute as a whole.
- A question that might arise is why in Figure 40 `var:attribute` is associated with `var:postObject` (which represents the object with the status after the execution of the operation), but it is not associated with `var:preObject` (the object with the status before the execution). We have made this decision because an object that acts as a `var:preObject` in an operation execution, was a `var:postObject` in a previous operation execution. Thus, the attributes associated to such an object in a `var:preObject` were registered when it previously played the role of `var:postObject`.
- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we do not to consider this circumstance with the aim of covering a wider spectrum of cases. When the executed operation lacks *Input data*, the UML representation in Figure 39 will not include *Input Parameters*. As a consequence, the resulting PROV template in Figure 40 will also lack `var:input` and its associated PROV relations.
- Although the *context* of this pattern does not explicitly state that output data should be obtained from the *Operation execution*, this could be the case. However, we do not include this output data in this pattern description to avoid overburden both the UML and PROV explanations with information out of the scope of the *context*.

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

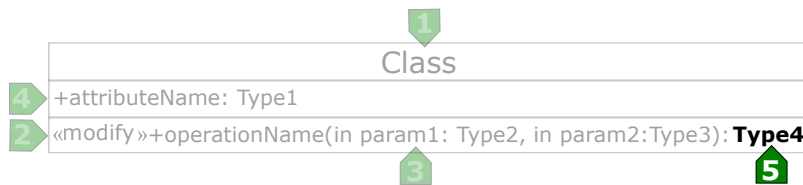


Figure 41. UML representation that models the context given by *CIP8*, including *Output Parameters*.

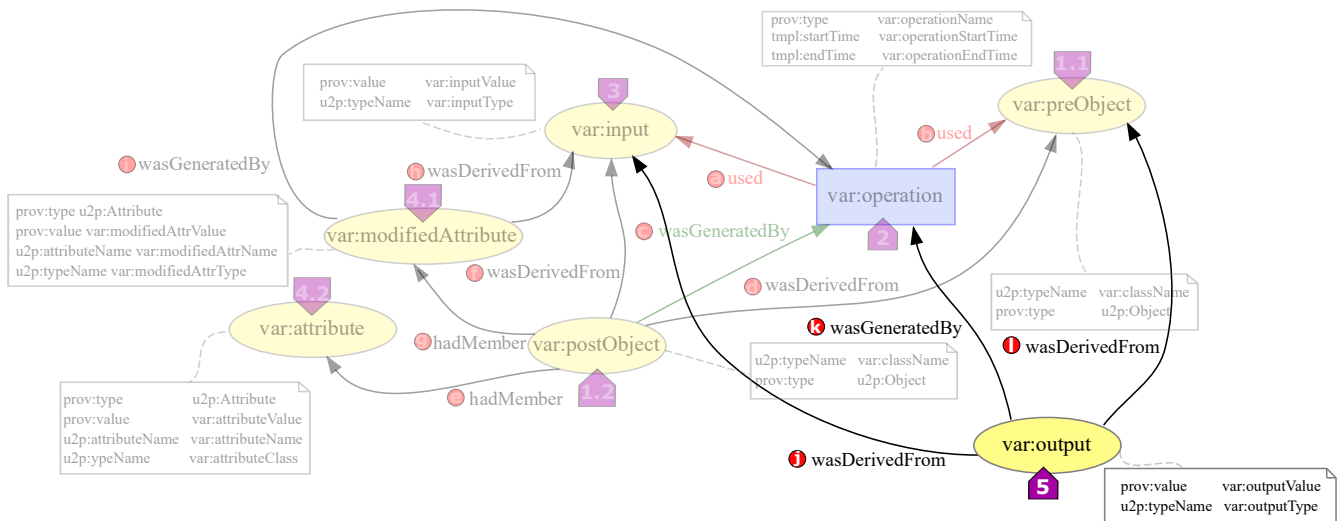


Figure 42. PROV template generated from the UML representation used in *CIP8*, including Output Parameters (Figure 41)

PROV elements

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

PROV relations

PROV Relation	Description
<code>prov:wasDerivedFrom</code>	It is the construction of <code>var:output</code> based on <code>var:input</code> .
<code>prov:wasGeneratedBy</code>	It is the completion of production of <code>var:output</code> by <code>var:operation</code> .
<code>prov:wasDerivedFrom</code>	It is the construction of <code>var:output</code> based on <code>var:preObject</code> .

Attributes

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

Identifier Class diagram Pattern 9 (CIP9)

Context

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

Key elements

<i>Object</i>	The object on which the operation is executed.
<i>Pre-operation object</i>	The object with the status before the execution of the operation.
<i>Post-operation object</i>	The object with the status after the execution of the operation.
<i>Operation execution</i>	The execution of the operation.
<i>Input data</i>	The information (if any) passed into the <i>Operation execution</i> .
<i>Object’s attributes</i>	All the characteristics of the <i>Object</i> . Since, as a consequence of the <i>Operation execution</i> , a concrete collection attribute changes, we have identified:
<i>Modified collection attribute</i>	The modified <i>Object’s attribute</i> .
<i>Unmodified attributes</i>	The not modified <i>Object’s attributes</i> .

UML Diagram

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

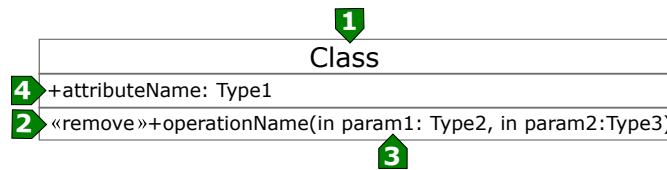


Figure 43. UML representation that models the context given by CIP9

Mapping to PROV

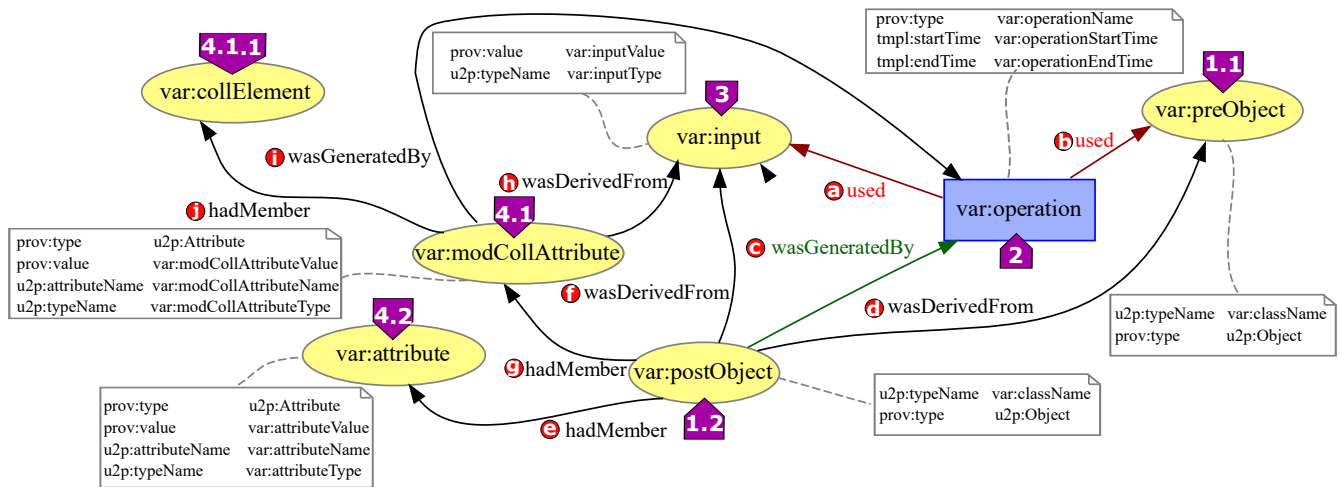


Figure 44. PROV template generated from the UML representation used in *CIP9* (Figure 43)

PROV elements

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






Attributes


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

PROV relations

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

Discussion

- Among the Class Diagrams patterns, patterns from *CIP6* to *CIP10* address the execution of operations that change an object's status. While, *CIP6* changes the object's status as a whole (being the concrete modified attributes unknown or irrelevant), in patterns *CIP7-CIP10* the concrete attributes modified by the *Operation execution* are explicitly known. In contrast to *CIP7* which directly sets the information passed into the *Operation execution* as values of concrete object's attributes, the other mentioned patterns use such information to change the object's status as a whole or the values of concrete object's attributes. It must also be noted that patterns *CIP9* and *CIP10* address the execution of operations which remove or add elements from/into an object's collection attribute, while patterns *CIP7* and *CIP8* affect either a univalued attribute or a collection attribute as a whole.
- A question that might arise is why in Figure 44 `var:attribute`  is associated with `var:postObject`  (which represents the object with the status after the execution of the operation), but it is not associated with `var:preObject`  (the object with the status before the execution). We have made this decision because an object that acts as a `var:preObject` in an operation execution, was a `var:postObject` in a previous operation execution. Thus, the attributes associated to such an object in a `var:preObject` were registered when it previously played the role of `var:postObject`.
- Although the *context* of this pattern does not explicitly state that *Input data* should be passed to the operation, we have considered this circumstance with the aim of covering a wider spectrum of cases. When the executed operation lacks *Input data*, the UML representation in Figure 43 will not include *Input Parameters* . As a consequence, the resulting PROV template in Figure 44 will also lack `var:input`  and its associated PROV relations.
- Although the *context* of this pattern does not explicitly state that output data should be obtained from the *Operation execution*, this could be the case. However, we have decided not to include this output data in this pattern description to avoid overburden both the UML and PROV explanations with information out of the scope of the *context*.

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

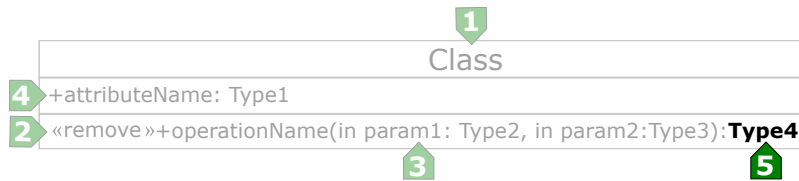


Figure 45. UML representation that models the context given by *CIP9*, including Output Parameters.

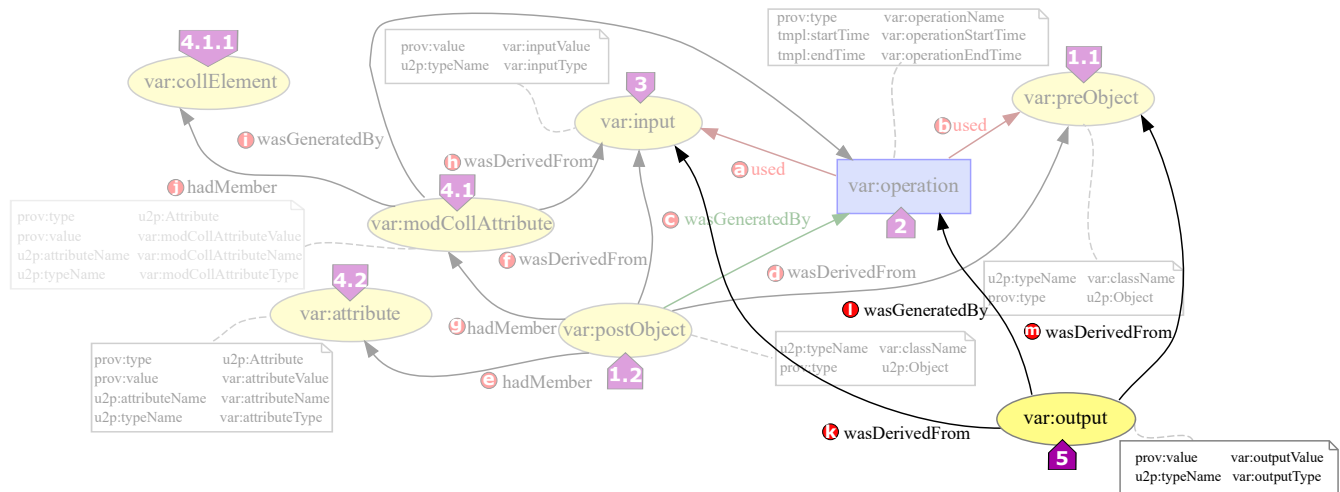


Figure 46. PROV template generated from the UML representation used in *CIP9*, including Output Parameters (Figure 45)

PROV elements

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

PROV relations

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

Attributes

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

Identifier Class diagram Pattern 10 (CIP10)

Context

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

Key elements

<i>Object</i>	The object to which the operation to be executed belongs.
<i>Pre-operation object</i>	The object with the status before the execution of the operation.
<i>Post-operation object</i>	The object with the status after the execution of the operation.
<i>Operation execution</i>	The execution of the behaviour specified by the operation.
<i>Input data</i>	The information passed into the <i>Operation execution</i> .
<i>Object's attributes</i>	All the characteristics of the <i>Object</i> . Since, as a consequence of the <i>Operation execution</i> , a concrete collection attribute changes, we have identified:
<i>Modified collection attribute</i>	The modified <i>Object's attribute</i> .
<i>Unmodified attributes</i>	The not modified <i>Object's attributes</i> .

UML Diagram

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

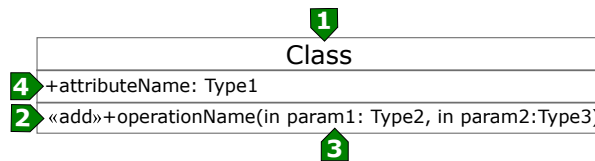


Figure 47. UML representation that models the context given by CIP10

Mapping to PROV

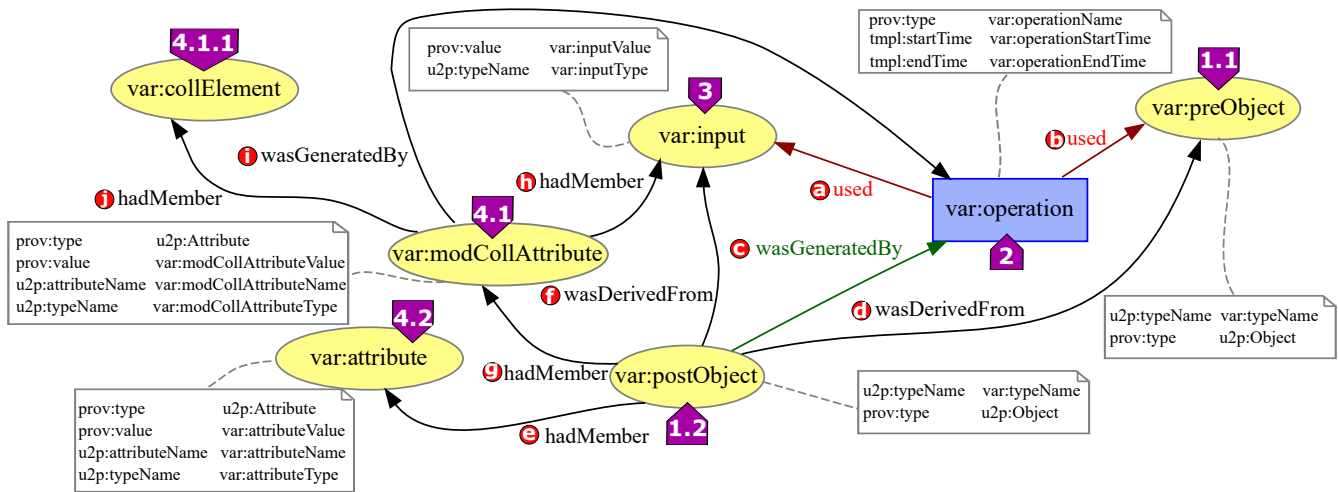


Figure 48. PROV template generated from the UML representation used in CIP10 (Figure 47)

PROV elements

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

Attributes

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

PROV relations

PROV Relation	Description
a <code>prov:used</code>	It is the beginning of utilizing <code>var:input</code> by <code>var:operation</code> .
b <code>prov:used</code>	It is the beginning of utilizing <code>var:preObject</code> by <code>var:operation</code> .
c <code>prov:wasGeneratedBy</code>	It is the completion of production of <code>var:postObject</code> by <code>var:operation</code> .
d <code>prov:wasDerivedFrom</code>	It is the update of <code>var:preObject</code> resulting in <code>var:postObject</code> .
e <code>prov:hadMember</code>	It states that <code>var:attribute</code> is one of the elements in <code>var:postObject</code> .
f <code>prov:wasDerivedFrom</code>	It is the construction of <code>var:postObject</code> based on <code>var:input</code> .
g <code>prov:hadMember</code>	It states that <code>var:modCollAttribute</code> is one of the elements in <code>var:postObject</code> .
h <code>prov:hadMember</code>	It states that <code>var:input</code> is one of the elements in <code>var:modCollAttribute</code> . This is due to the fact that in this context the input information is directly added to the object's collection attribute.
i <code>prov:wasGeneratedBy</code>	It is the completion of production of <code>var:modCollAttribute</code> by <code>var:operation</code> .
j <code>prov:hadMember</code>	It states that <code>var:collElement</code> is one of the elements in <code>var:modCollAttribute</code> .

Discussion

- Among the Class Diagrams patterns, patterns from *CIP6* to *CIP10* address the execution of operations that change an object's status. While, *CIP6* changes the object's status as a whole (being the concrete modified attributes unknown or irrelevant), in patterns *CIP7-CIP10* the concrete attributes modified by the *Operation execution* are explicitly known. In contrast to *CIP7* which directly sets the information passed into the *Operation execution* as values of concrete object's attributes, the other mentioned patterns use such information to change the object's status as a whole or the values of concrete object's attributes. It must also be noted that patterns *CIP9* and *CIP10* address the execution of operations which remove or add elements from/into an object's collection attribute, while patterns *CIP7* and *CIP8* affect either a univalued attribute or a collection attribute as a whole.
- A question that might arise is why in Figure 48 `var:attribute` **4** is associated with `var:postObject` **1** (which represents the object with the status after the execution of the operation), but it is not associated with `var:preObject` **1** (the object with the status before the execution). We have made this decision because an object that acts as a `var:preObject` in an operation execution, was a `var:postObject` in a previous operation execution. Thus, the attributes associated to such an object in a `var:preObject` were registered when it previously played the role of `var:postObject`.
- Although the *context* of this pattern does not explicitly state that output data should be obtained from the *Operation execution*, this could be the case. However, we do not include this output data in this pattern description to avoid overburden both the UML and PROV explanations with information out of the scope of the *context*.

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

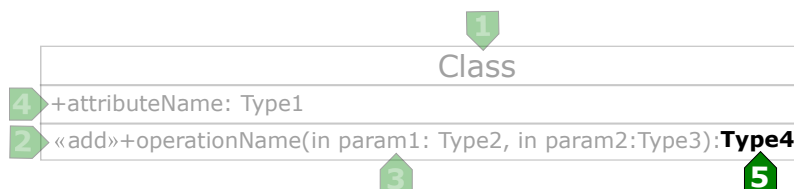


Figure 49. UML representation that models the context given by *CIP10*, including *Output Parameters*.

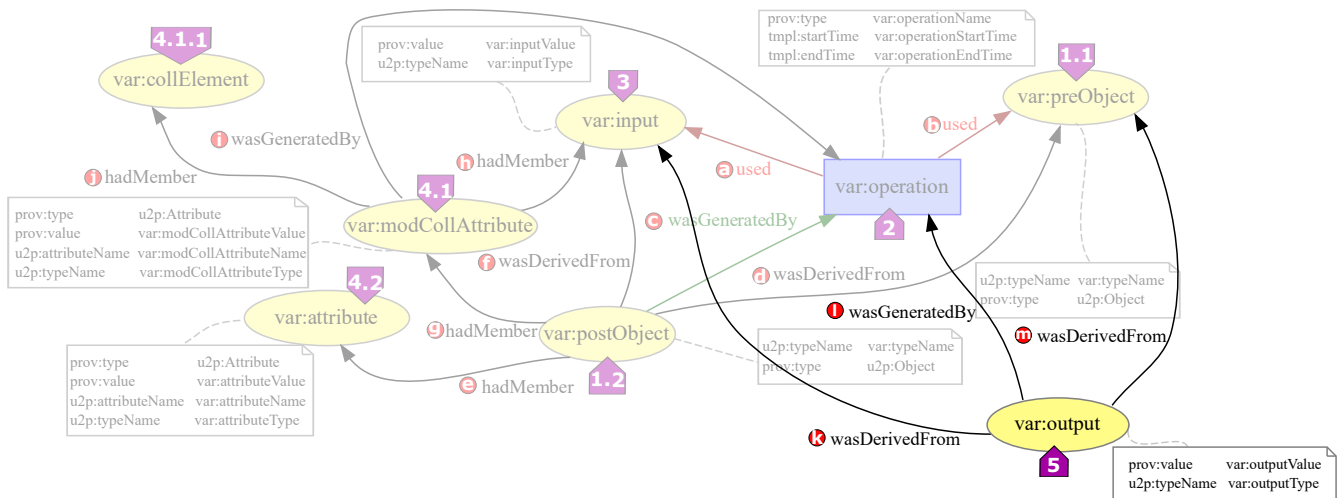


Figure 50. PROV template generated from the UML representation used in *CIP10*, including Output Parameters (Figure 49)

PROV elements

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

PROV relations

PROV Relation	Description
k <code>prov:wasDerivedFrom</code>	It is the construction of <code>var:output</code> based on <code>var:input</code> .
l <code>prov:wasGeneratedBy</code>	It is the completion of production of <code>var:output</code> by <code>var:operation</code> .
m <code>prov:wasDerivedFrom</code>	It is the construction of <code>var:output</code> based on <code>var:preObject</code> .

Attributes

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

References

- [1] OMG, “Unified Modeling Language (UML). Version 2.5,” 2015. Document formal/15-03-01, March, 2015.
- [2] L. Moreau and P. Missier (eds.), “PROV-DM: The PROV Data Model,” W3C Recommendation REC-prov-dm-20130430, World Wide Web Consortium, 2013.
- [3] N. Kwasnikowska, L. Moreau, and J. V. D. Bussche, “A formal account of the open provenance model,” *ACM Trans. Web*, vol. 9, pp. 10:1–10:44, May 2015.
- [4] PROV Graph Conventions. Available at www.w3.org/2011/prov/wiki/Diagrams. Last accessed April, 2019.
- [5] M. Dürst and M. Suignard., “Internationalized Resource Identifiers (IRIs) (RFC 3987).” January, 2005.
- [6] A. Knapp and S. Merz, “Model checking and code generation for uml state machines and collaborations,” *Proc. 5th Wsh. Tools for System Design and Verification*, pp. 59–64, 2002.
- [7] N. Dragan, M. L. Collard, and J. I. Maletic, “Automatic identification of class stereotypes,” in *Proceedings of the 26th IEEE International Conference on Software Maintenance*, pp. 1–10, 2010.

Appendix A Taxonomy of Class' operations

Depending on their nature, operations implicitly have specific semantics that can also provide information of interest for provenance capture. In order to provide UML Class diagrams with such additional semantics to be included in the generated PROV templates, we have stated a taxonomy of operations given by a set of stereotypes to be included in such diagrams. The taxonomy is based on that given by Dragan et al. [7], which has been enriched with additional stereotypes aimed at identifying extra/further operation's semantics not considered in [7] (marked with an asterisk in Table 2).

Table 2. Extension of the taxonomy given in [7] showing the categories of UML Class's operations considered in our proposal. Stereotypes with an asterisk denote those included by our proposal.

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