TAXONOMY OF CLASS' OPERATIONS

Carlos Sáenz-Adán[1]*, Beatriz Pérez[1], Francisco J. García-Izquierdo[1], Luc Moreau[2]

[1]*Dept. of Mathematics and Computer Science, Univ. of La Rioja, La Rioja, Spain,*

{carlos.saenz,beatriz.perez,francisco.garcia}@unirioja.es

[2]*Dept. of Informatics, King's College London, London, UK,*

luc.moreau@kcl.ac.uk

1 Introduction

Depending on their nature, operations have specific semantics which can also produce information of interest for provenance capture. For instance, the key factors involved in the execution of an operation such as *getName* (accessing an object's attribute) are different from the ones related to *setName* (modifying an object's attribute), and consequently, the provenance information captured from the execution of both operations must be different. Taking this into account, we are interested in identifying a taxonomy of operations which covers the vast varied range of operation's types of interest for provenance capture. Based on this taxonomy, we can define our CDs to PROV template patterns so that the generated templates can provide concrete provenance information depending on the operation's semantics.

2 A taxonomy of operations

To define our taxonomy, we undertook a literature search looking for different categorizations of operations based on their behaviours. We distinguished the approaches that present a more general classification of operations such as [3], from those that provide a more fine grained taxonomy of operations such as [1] or, more remarkably, the work presented by Dragan et al. in [2], which is one of the most complete. Dragan et al.'s taxonomy is based both on how an operation accesses data (i.e., an operation changes the object's status or leaves it unchanged), and on its behavioural characteristics (i.e., creational, structural...). Such a taxonomy is expressed as a classification of operations' `Stereotypes`. These UML `Stereotypes` are extension mechanisms that allow us to complement each CD's `operation` with specific semantic information regarding its category within the taxonomy, thus linking the `operation` with its corresponding semantics. The notation for a `Stereotype` is a string with the stereotype name between a pair of guillemets (e.g., «`add`»).

This taxonomy define five categories: (1) *Creational* refers to operations responsible for creating or destroying objects of the class. (2) *Structural Accessor* refers to operations that return information regarding the attributes of the object to which it belongs, without changing the state of the object. (3) *Structural Mutator* corresponds to operations that change the state of the object to which it belongs. (4) *Collaborational* which helps define the communication between objects and how objects are controlled in the system. Finally, (5) *Degenerate* corresponds to operations which give us little information about.

Our proposal (see Table 1), which is inspired from the Dragan et al.'s one [2], includes additional stereotypes (marked with an asterisk) not initially considered in the original taxonomy: the *search*, *add* and *remove* stereotypes, which cover operations for the management of collection attributes (such as search, addition or removal, respectively); the *process* stereotype, for operations returning information based on the whole object's internal structure; and *modify*, for operations that modify a specific attribute without setting an input value directly. We have not considered the categories *collaborational* and *degenerate* since they represent behaviours already modelled by SqDs (such as the communication between objects, given by *collaborational*), or reflect aspects that cannot be tackled without checking the source code (e.g., *degenerate* category). Below, we explain the behaviour represented by each stereotype.

Table 1: Extension of the taxonomy given in [2] showing the categories of UML Class' 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.
	predicate	The operation returns boolean values that are computed based on concrete attributes of an object.
	property	The operation returns values (of any type) that are computed based on concrete attributes of an object.
	void-accessor	The operation, by means of a parameter, returns values (of any type) that are computed based on concrete attributes of an object.
Structural *Mutator*	command	The operation changes the status of an object as a whole (the modified attributes are unknown or irrelevant). It does not return information.
	non-void-command	The operation changes the status of an object as a whole (the modified attributes are unknown or irrelevant). It does return information.
	set	The operation directly sets the information passed to the operation as values of concrete attributes of an object.
	modify*	The operation modifies concrete attributes of an object.
	remove*	The operation removes an element from a concrete collection attribute of an object.
	add*	The operation adds an element on a concrete collection attribute of an object.

References

[1] P. Clarke, B. Malloy, and P. Gibson. Using a taxonomy tool to identify changes in OO software. In *Proceedings of the 7th European Conference on Software Maintenance and Reengineering, (CSMR'03)*, pages 213–222, 2003.

[2] 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*, pages 1–10, 2010.

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