A SOFTWARE ENGINEERING FRAMEWORK FOR MAKING APPLICATIONS PROVENANCE-AWARE

TAXONOMY OF OPERATIONS

1 Introduction

From UML CDs, we are interested in information about both (1) the object's status (i.e., values of the *attributes*) before and after an operation's execution, and (2) the object's internal changes (e.g., setting a new attribute, adding/removing an element in a collection). This information will depend on the nature of the operations, which can be viewed from a number of different perspectives, being categorized by aspects such as how they access data (i.e., a method changes the object's status or leaves it constant) and their behavioral characteristics (i.e., creational, structural, or collaborational) [2]. For instance, the key factors involved in the execution of an operation such as *getName* (accessing a a data member of an object) are different from the ones related to *setName* (modifying an object's data member), and consequently, the provenance information must be different. With the aim of identifying a set of *contexts* that covers the most varied nature of operations, we have established a taxonomy of operations for laying the definition of the contexts.

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 behaviour. The notation for a `Stereotype` is a string with the stereotype name between a pair of guillemets (e.g., «`add`»). Table 1 shows our taxonomy of operations, which extends the Dragan et al.'s one, with additional stereotypes we have defined (they are marked with an asterisk). We have not considered the rest of categories from [2] (*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). Our proposal in Table 1 includes additional stereotypes not initially considered in Dragan et al.'s taxonomy: the *search*, *add* and *remove* stereotypes, which cover operations for the management of collection data members (such as search, addition or removal, respectively); the *process* stereotype, for operations returning information based on the whole objectâĂŹs internal structure; and *modify*, for operations that modify a specific data member without setting an input value directly. Below, we explain the behaviour represented by each category together with the contexts identified for them. Each pattern is associated to one or various `stereotypes` of the category. Beneath each context we show the `stereotypes` associated to it.

Table 1: Extension of the taxonomy given in [2] showing the categories used in our proposal. Stereotypes with an asterisk denote those which extends the proposal.

Category	**Stereotype name**	**Description**
Creational	create	Creates objects
	destroy	Destroys objects
Structural *Accessor*	get	Returns a data member
	search*	Returns an element from a data member collection
	predicate	Returns Boolean value which is not a data member
	property	Returns information about data members
	void-accessor	Returns information through a parameter
	process*	Returns information derived from the whole object
Structural *Mutator*	command	Performs a complex change
	non-void-command	
	set	Sets a data member
	modify*	Modifies a data member
	add*	Adds an element within a data member collection
	remove*	Removes an element from a data member collection

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.