Component-based process

For modeling language evaluation

Khaoula Sayeb, Oualid Khayati, Naoufel Kraeim
RIAD-ENSI
University of Manouba
Manouba, Tunisia
firstname.lastname@gmail.com

***Abstract*—This paper presents a component-based approach to build a modeling language evaluation process. We firstly define a process components repository that capitalizes and implements solutions for modeling language evaluation. We describe the process component model. Then we describe how we reuse the process components repository to define a process for modeling language evaluation based on pre-built components.**

Keywords-process component; modeling language; evaluation; component model; reuse; components repository.

I. Introduction

One of the major evolutions in software development is the component-oriented programming approach. It facilitates the construction of complex applications, their deployment, their administration and their evolution control. The rise of software components produces two types of complementary processes:

- The design for reuse: components engineering needed to create, enrich and maintain a repository of reusable components. It implements the identifying features, specification, development, validation and organization of components.
- The design by reuse: components based engineering consists on the reuse of pre-built components to define new products.

Components-based approach is applied to software engineering, method engineering and also process engineering. In our approach, we propose a process components repository that we use later to define an evaluation process. Each process component describes a modeling language (ML) evaluation solution. It can be then composed and connected with other components to build a ML evaluation process. The aim of this paper is to present the process component model. This later describes process components structure, relationships and composition rules. We present the process components repository and how we reuse it to define a component-based process for ML evaluation.

This paper is structured as following: In the second section, we give an overview about ML evaluation. We introduce our process component model in section three. Section four is dedicated to describe the components-based process for ML evaluation and to give a demonstration example that explains our approach. We finish this paper by the conclusion.

II. Modeling language evaluation

A ML has a key role in software modeling process. It affects directly the quality of software design (models) and so the final software quality. A ML is defined as the models expression language [1]. It is determined by a semantic (that is a set of concepts and rules that specify the field), a concrete syntax (which is a set of symbols that represent concepts) and an abstract syntax (expressed by meta-model) [2].

Although the ML evaluation domain is relatively recent, there are a huge variety of approaches and frameworks in this field. They propose solutions, instructions and features to qualify or quantify the ML quality. We propose to unify and organize terms on this domain. Then, each quality approach is called quality framework. For instance, the physics of notations (Moody's framework [3]) is a framework for concrete syntax evaluation, sequal (krogstie's framework [4]) proposes a generic framework for the whole language evaluation, etc. A quality framework is composed of a set of quality attributes. The concept of quality attributes unifies existing concepts in the literature such as dimensions, attributes, features or sub-features, criteria, factors, etc. The physics of notations proposes nine quality attributes to define the cognitive effectiveness of a visual notations in general and specifically graphical concrete syntax. Perceptual discriminability is a quality attribute that determines the ease and accuracy with which graphical symbols can be differentiated from each other. It uses other quality attributes to decompose the solution (perceptual discriminability uses visual distance and perceptual popout). To measure quality attributes, we use evaluation techniques that can be a metric or a qualitative or quantitative protocol. In addition, an evaluation technique offers a concrete outcome that estimates a quality attribute.

We use process components to model knowledge in the ML evaluation domain. Our first purpose is to provide a structured documentation about this field. Secondly, we aim to provide tools for the implementation and the built of a components-based process. Process components are defined following a model that we describe in the next section.

III. design for reuse: Process Component

In our approach, a process component is a component that provides a solution for ML. In this section, we present the process component model. A component model consists of a set of conventions to be followed in the construction and use of

DOI reference number: 10.18293/SEKE2016-159

components. It has to define the component structure, relationships and component reuse techniques.

A process component can be a conceptual component (a design pattern that describes a framework, a quality attribute or an evaluation technique) or a software component (that implements a conceptual component). A design pattern describes the context of the framework, the solution provided and the problem resolved by the framework. In addition, we use design pattern to capitalize knowledge about ML evaluation. A software component implements the solution offered by conceptual component. For instance, a conceptual component describes the solution to assess the visual distance between concrete syntax symbols. An associated software component takes in entry the list of concrete syntax symbols and measures the visual distance between them. The result of the conceptual component is a solution approach. The result of the software component is a significant value that represent the visual distance. The next subsections detail conceptual and software component models.

A. *Conceptual component model*

We use design pattern to describe our conceptual component. A design pattern is defined as a solution of a recurring problem in a context. The design pattern model specifies the structure adopted by the designer to represent patterns. It is composed of a set of rubrics. To define our process design pattern, we use the P-SIGMA [5] model that we adapt to take into account the capitalization needs. Moreover, we add some rubrics and customize others. P-Sigma is composed of three parts: Interface, Realization and Relation. **Interface** part contains all elements allowing pattern's selection. **Realization** part gives the pattern solution. Finally, the **relationship** part describe links between patterns. Fig.1 describes our adapted version of the P-SIGMA formalism (Adapted rubrics are gray). We add the *reference* rubric which gives the source of the approach described by the conceptual component. The *realize* rubric is added to define a new relationship. The *classification* rubric is customized to deal with our classification approach.

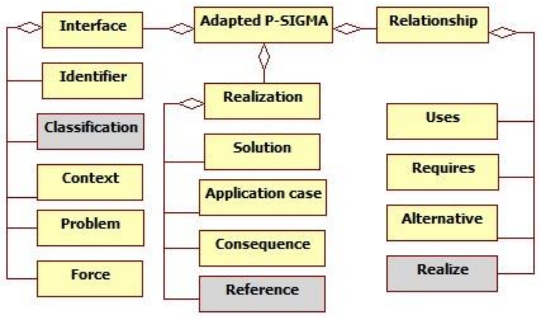


Figure 1. Adapted P-SIGMA

Conceptual components are described in more details with examples in a further work [6]. The following table (TABLE1) presents the conceptual component that describes the quality attribute: **perceptual discriminability.**

TABLE 1. CONCEPTUAL COMPONENT: PERCEPTUAL DISCRIMINABILITY

Interface
Identifier: perceptual discriminability
Classification: Graphical concrete syntax, semiotic framework, evaluation. **Problem**: Are symbols distinguishable between each other? How to define ML graphic elements that are perceptually different? **Context**: The construction of a new ML. The evaluation of the graphical concrete syntax.
Realization
Solution: we describe how to determine the perceptual discriminability [1]. It is too long to express here. **References:** [1] D.L. Moody. The 'physics' of notations: Toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on Software Engineering, 2009.
Relationship
Use: Visual distance, perceptual popout.

B. *Software component Model*

The widely accepted definition of software components is that of [7]: "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties". A software component model is a definition of the semantics of components (that is, what components are meant to be), the syntax of components (that is, how they are defined, constructed, and represented), and the composition rules of components (that is, how they are composed or assembled) [8].

In this subsection, we present our software components model for ML evaluation. This model defines software components structure, relationships and its composition rules.

1) *Software component structure*

The component structure is presented in Fig.2. Depending of the component's architecture, we distinguish two component's types. *The primitive component* is a "basic" component. *The composite component* is an aggregation of several primitive or composite components.

Each component is a part of an evaluation process. It has to realize a processing that measures quality (a solution). It requires, for this aim, parameter of the evaluation and then it provides results. Therefore, a component is composed of a solution and a set of ports. The ports realize provided and required interfaces either to acquire parameters necessary for the assessment or to provide descriptions and results of the evaluation. Then a component can have a required interface "*Evaluation parameters*" for acquiring the necessary parameters of the evaluation and a "*Primary result*" interface for gathering primary results needed to calculate the result of the concerned component. "*Primary result*" interface is not mandatory for all components. For provided interface, a component must have *An evaluation purpose* and a *Final result* interfaces. The first one to describe the purpose for which this component is used and integrated into an evaluation process. The second one to provide the evaluation result. In the case of a primitive component, it will be a result of the evaluation. In the case of a composite component, this will be a combination of other components results. The combination is done using a formula defined in the component solution.

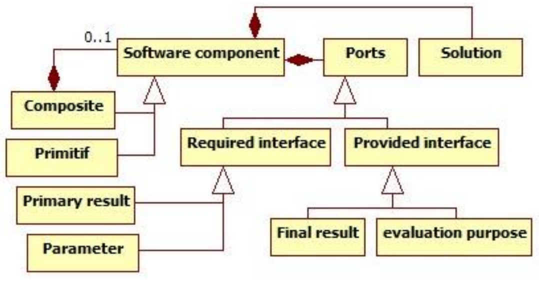


Figure 2. Software component structure

In this subsection, we defined the software components structure. The next step is the definition of relationships.

2) Relationship

As mentioned bellow, a software component implements a quality framework, a validation, a quality attribute or an evaluation technique. We have to define relationships between components. Furthermore, we have four types of relationships.

The *composition* relationship: one component is composed of other components. Therefore, its treatment is delegated to its composite components. Specifically in our approach, quality framework components and validation components must be composed of at least one quality attribute component.

The *realization* relationship: it connects two components where the solution of the former is more refined than that of the second. This relationship connects a quality attribute component and an evaluation technique component. Indeed an evaluation technique component realizes a quality attribute component by providing the tools to calculate it and enhance its solution.

The *use* relationship: it connects two quality attributes components. It allows the decomposition of a problem described by a component more elementary components. More specifically, in our case, a quality attribute component uses the results provided by the used quality attribute components.

The *alternative* relationship: it connects two evaluation techniques components that provide two alternative solutions for the same problem. Therefore, these components realize the same quality attribute component.

3) Composition rules

Composition specifies how components are interconnected. Compositions declare instances number of components and define their configuration. Furthermore, a composition specifies how the ports of those instances are wired, i.e., which connector is used for connecting which ports. In our approach, we define the following composition rules:

Hierarchical composition and encapsulation (built components, sub-components): Composite components (quality frameworks component and validation components) encapsulate all the components involved in their achievements. In this case, each port of the composite component should be linked to one or more interfaces of its son components. Especially, the final result of a composite component is calculated based on primary results of a its sub-components.

Interconnection components throw connectors: In fact, the connector can assemble components using their provided and required interfaces. In our approach, we have two interconnections throw connectors. The first case (Case1) when a quality attribute component uses other quality attribute components (in Fig.4 perceptual discriminability uses visual distance and perceptual popout). The second case (Case2) when evaluation technique components realize a quality attribute component (in Fig.4 metric for visual distance realizes visual distance. In both cases (Case1 and Case2), the connection is made between a required interface final result and one or more provided interfaces primary result.

A demonstration of composition rules is shown in section four where we present a minimal example of application of process component to define a process for ML evaluation.

IV. Design by reuse: Component based process

Component-based software engineering aims to improve the software engineering process by providing reusable components. Following the process component model described in section three, we create a components repository that capitalizes knowledge about ML evaluation. Software components serve to build a component-based process. In addition, a ML assessor selects software components form the repository and implements an evaluation process. In this section, we firstly present an evaluation process model. Then we give a minimal example that instantiates it.

A. Evaluation process model

We propose a model for the ML evaluation process that resumes all related features (Fig.3). In addition, a ML evaluation process is applied to a subject which is the parameter of the evaluation and produces as result an execution report. It depends on the context and the needs of the assessor. The context may be a comparative study of existing MLs or an improvement and validation of a ML under construction. The subject of an evaluation process may be a ML [4], a part of the ML (i.e. concrete syntax [3]), a ML family (i.e. Business Process ML [9]) or even just a ML property (i.e. usability [10]).

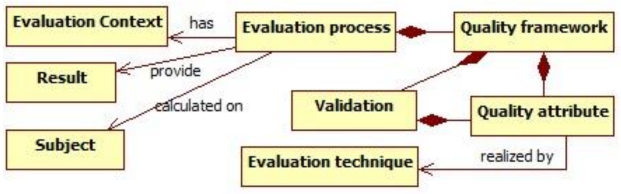


Figure 3. Evaluation process model

Besides, a ML evaluation process is composed of quality frameworks that provide solution for evaluating a ML. A quality framework is composed of validation and a set of quality attributes. A validation is an optional part in a quality framework. It provides an assessment of a ML by validating one of its parts relative to another as a set of quality attributes (for example evaluate the concrete syntax with respect to the abstract syntax). A quality attribute is realized by means of evaluation techniques (which calculate it throw metrics or throw protocols offering a concrete outcome of the evaluation).

A ML evaluation process is composed of one or many frameworks. A ML assessor builds an evaluation process by the selection of existing frameworks. We use the concept of process component to model these frameworks.

B. *Evaluation process example*

We propose a component base composed of conceptual and software components. Conceptual component capitalizes solution for ML evaluation. A ML assessor documents about the domain throw conceptual components. If he decides to implement an evaluation process, he has to define its context and its subject. Then he selects software components that compose the evaluation process. Its execution produces a report that resumes its application result. In this section, we give an evaluation process example that explains our solution and instantiates the proposed evaluation process model.

The process example:

Context: the evaluation of a graphical ML concrete syntax and its validation compared to the abstract syntax.

Subject: a graphical concrete syntax and an abstract syntax.

The evaluation process: the component diagram in Fig.4 represents software components that compose the example process. We use the framework proposed by Moody [4] for assessing graphical concrete syntax. It is composed of eight quality attributes to evaluate the cognitive effectiveness of a graphical concrete syntax and a quality attribute (semiotic clarity) that validates the concrete syntax with respect to the abstract syntax. In this example process, we just implement two of them (semiotic clarity and perceptual Discriminability).

Some informations are not represented to simplify the diagram. For instance, we had to wire each provided interface evaluation purpose (EP) of a composite component to the relative composed component. It is similar for the required interface parameter (P).

Abbreviation meaning in the Fig.4 are as following:

P: Parameter; **PR**: Primary result; **FR**: Final result; **EP**: Evaluation purpose.

Result: a report that gathers all final results in the order of the process execution and its composition to get the process result.

This process example have to be calculated on a graphical concrete syntax (i.e., that of UML) to acquire concrete result.

V. CONCLUSION

In this paper, we have presented a process component model that describes the process component structure, relationships and composition rules. In addition, we use two components types: conceptual components that describe a ML evaluation approach; and software components that implement it. Conceptual components provide a structured documentation. Software components are used to build a ML evaluation process. Benefits of using component to represent our process is that 1) we favor the capitalization and the reuse of ML evaluation works; 2) we build flexible process adapted to the assessment context and needs. We also propose an evaluation process model that describes ML evaluation process.

REFERENCES

[1] Object Management Group, "Meta Object Facility (MOF) 2.0 Core Specification," 2006.

[2] A. Kleppe, "A Language Description is More than a Metamodel," *ATEM,* 2007.

[3] D. L. Moody, "The “Physics” of Notations: Toward a Scientific Basis for Constructing Visual Notations in software engineering," 2009.

[4] J. Krogstie, "Evaluating UML using a generic quality framework," chez *UML and the unified process*, USA, IGI Publishing, Hershey, PA,, 2003, pp. 1-22.

[5] A. Conte, J.-P. Giraudin, J.-C. Freire Junior, I. Hassine and D. Rieu, "A tool and a formalism to design and apply patterns," *SugarloafPLoP,* 2002.

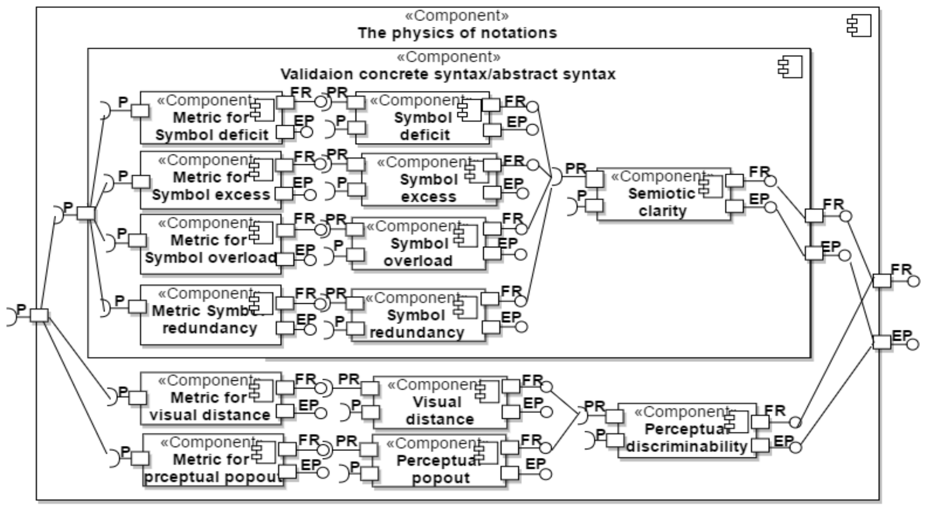
[6] K. Sayeb, D. Rieu, S. Dupuy-Chessa et N. Mandran, "Qualité des langages de modélisation et des modèles: vers un catalogue des patrons collaboratifs," *INFORSID,* 2012.

[7] C. Szyperski, "Component Software: Beyond Object-Oriented Programming", 2nd Addison-Wesley, 2002.

[8] K.-K. Lau et Z. Wang, "A survey of software component models," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,* 2007.

[9] B. List et B. Korherr, "An Evaluation of Conceptual Business Process Modelling Languages," *SAC,* 2006.

[10] K. Figly, J. Mendlingz et M. Strembecky, "Towards a Usability Assessment of Process Modeling Languages," 2009.



«Component»
The physics of notations
«Component»
Validaion concrete syntax/abstract syntax
«Component» Metric for Symbol deficit
«Component» Symbol deficit
«Component» Metric for Symbol excess
«Component» Symbol excess
«Component» Metric for Symbol overload
«Component» Symbol overload
«Component» Metric Symbol redundancy
«Component» Symbol redundancy
«Component» Semiotic clarity
«Component» Metric for visual distance
«Component» Visual distance
«Component» Metric for prceptual popout
«Component» Perceptual popout
«Component» Perceptual discriminability
P
PR
FR
EP

Figure 4. Evaluation process example