Towards Quantifying the Adaptability of Executable BPMN Processes

Jörg Lenhard

Distributed Systems Group, University of Bamberg, Germany
joerg.lenhard@uni-bamberg.de

Abstract. Process languages such as the *Business Process Model and Notation* 2.0 or the *Web Services Business Process Execution Language* promise the portability of executable artifacts among different runtime environments, given these artifacts conform to the respective specification. However, due to the natural imperfectness and differing priorities of runtime environments, actual portability of process code is often hard to achieve. A first step towards tackling this problem is the quantification of the actual *degree of portability* of process code using software metrics. The ISO/IEC 25010 software quality model defines portability as a main software quality characteristic with several sub-characteristics. One of these is *adaptability*, the degree to which a piece of software can be adapted in order to be executed in a different environment. In this paper, we propose a mechanism for quantifying the degree of adaptability of BPMN 2.0 processes and demonstrate its computation.

Keywords: Adaptability, ISO/IEC 25010, BPMN, Metrics

1 Motivation

A central part of process-aware applications is the runtime platform for executable process models. To run on such a platform, processes need to be tailored to it, thus often locking them into that particular platform. This lock-in effect is undesirable. Application *portability* addresses this issue.

A way to improve the portability of an application is by programming it according to an open specification that promises the portability of the outcome. This is the route taken for various process languages [10], such as the *Business Process Model and Notation* (BPMN) 2.0 [15] or the *Web Services Business Process Execution Language* (BPEL) 2.0 [14]. Being open international standards, these languages name portability as a central goal. The problem is that these standards are just specifications and portability of code ultimately depends on their implementations. These, however, rarely implement the complete specification and naturally inhibit faults or mandate the usage of non-standard extensions that limit the portability of a process. Huge differences in standard conformance have been demonstrated for BPEL runtimes [4, 5]. It can be expected that the situation is the same in the case of BPMN, as indicated by recent studies showing

compliance issues for its serialization format [3]. This implies that processes implemented in these languages, despite being conformant to an open international standard, cannot be considered as portable per se.

A first step towards tackling this problem is the ability to quantify it: to be able to compute a degree of portability, or adaptability, for a given process implemented in a particular language using software metrics. This could yield several benefits, as for instance:

1. When integrated into a metrics suite, developers could continuously inspect the portability or adaptability of the process during development. This would allow them to get direct feedback for changes they introduce and make them aware of changes that limit portability or adaptability [13]. Raising their awareness has the potential to result in more portable code.
2. When having to port a process, metrics can be used as a basis for decision making. A high degree of portability or adaptability translates to a high likelihood that the process can actually be ported or adapted. In contrast to this, a low degree can support the decision to rewrite the process from scratch for the new platform.
3. When selecting one among a set of alternative processes for execution, metrics can serve as a means for quality comparison and ranking the alternatives.

As a basis for such a quantification, the ISO/IEC 25010 software quality model [7] can be of help. This new revision of the widely-accepted ISO/IEC 9126 quality model lists portability as a main quality attribute of software, consisting of several sub-attributes: *Adaptability*, *installability*, and *replaceability*. Each of these characteristics should be measurable to compute the degree of portability and it is our goal to build a measurement framework that achieves this for process-aware and service-oriented systems. Since we addressed direct code portability [12] and installability [11] in previous work, we now try to tackle the quantification of adaptability for this type of software. In this paper, we try to quantify the adaptability of BPMN processes using structural code metrics. To meet this end, we propose a mechanism for computing such metrics and demonstrate its application in a use case. We are trying to compute a quantitative representation of the likelihood that the code of a process can be adapted to a different form that results in the same runtime behavior. We are *not* trying to provide a metric that states if a process can be modified to run on a particular engine.

We have to emphasize that the purpose of this paper is the proposal and description of the mechanism and metrics. Due to this scope and the page limit, we defer the important aspect of the validation of the metrics, for instance in terms of measurement theory [2] or construct validity [9], to future work.

The remainder of the paper is structured as follows: In the next section, we discuss related notions of adaptability and adaptability metrics. In section 3 we introduce our approach and explain our proposed metrics. Thereafter, we evaluate the approach by computing the adaptability degree for a use case process. Finally, we draw a conclusion and point to future work.

2 Notions of Adaptability and Related Work

The ISO/IEC quality model defines adaptability as the "*degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments*" [7, p. 15]. Here, we focus on adaptions to the software environment only. The scenario we have in mind is a required change of a set of processes to a different runtime engine. If the processes cannot be ported directly, they have to be adapted to preserve their executability. This is different to other views of adaptability, as for instance in autonomous systems, where adaptability refers to the ability of the system to automatically cope with changing situations, such as an increased load, at runtime [16] or adapter synthesis, where adaptability refers to whether an adapter for a pair of services can be created [18].

In this paper, we try to quantify adaptability in an abstract, runtime-independent fashion. Hence, we base the following metrics on the BPMN specification [15] only. Nevertheless, it might be worthwhile to consider actual process runtimes in the computation of adaptability metrics, as for instance done in [12], since the adaptability of a particular process to a particular runtime ultimately depends on the runtime to which it should be ported. However, our aim is to allow for the measurement of adaptability already at a point in time, where no new runtime has been selected yet. Still, we plan to evaluate the usage of data on language support in runtimes to see if it can enhance the metrics proposed here.

The metrics for adaptability of the new ISO/IEC quality model [8] are not yet publicly available, but will likely be similar to that of previous versions, e.g. [6]. These metrics are based on counting the number of program functions that seem to be adaptable to different contexts. This number is contrasted with the number of functions that are required to be adaptated in the current situation, which is typically all program functions that need to be available in the new environment after porting. By relating these two numbers, one can obtain the percentage of program functions that can be adapted and thus provide a basic notion of adaptability for the complete program. However, such a measure is very coarse and there is no description of how to actually determine if a function is adaptable or not. Here, we try to provide a mechanism to determine if a program element is adaptable. Other studies that evaluate adaptability [1] rely on surveys of stakeholders. Metrics based on human judgment can have limitations in terms of reproducibility and reliability, which is why we aim to provide structural code metrics that can be computed automatically and are reproducible instead.

Such structural metrics do primarily exist for the architectural layer of a software product and not the concrete source code [16, 17]. There, adaptability is first quantified in a binary or weighted fashion for an atomic element of the respective system, such as a component in the software architecture. These element adaptability scores are then subsequently aggregated using different adaptability indices at different layers of abstraction to arrive at a global value of adaptability for the complete software architecture. This way of computing adaptability should also work when looking at code artifacts and not architectural elements of a program. Here, we focus on executable service-based processes and

try to reproduce the adaptability computation in the above sense. Thus, our idea is to quantify adaptability at the level of an atomic process element, such as an activity, and to aggregate this to a global degree for the complete process.

3 Measuring Structural Adaptability

In the terms of BPMN, an atomic process element is an activity, task, or gateway. Using the above approach, we need to assign an adaptability score to each of those elements. Our idea is to count the number of alternative representations for the functionality provided by the element that result in the same runtime behavior. In BPMN, there are typically multiple alternatives for each process element that can result in identical process behavior at runtime. The more alternatives exist for a given process element, the easier it is to replace this element with such an alternative, and hence the more adaptable the resulting code actually is.

A simple example for multiple alternative implementations of the same functionality in BPMN is repetitive execution of a task through a *Loop* marker for the task. Any of the following language constructs can be used to define repetitive execution of a task and hence can be used as an alternative to a *Loop* marker:

1. A combination of an *Exclusive Gateway* and *Sequence Flows*
2. Enclosing the task in a *Loop Sub-Process*
3. Enclosing the task in an *Ad-Hoc Sub-Process*
4. Enclosing the task in an *Event Sub-Process*

It is likely that a BPMN engine will only support a subset of these options. For instance the Activiti engine[1], currently does not support normal *Loop* markers (`standardLoopCharacteristics`). It does support the combination of *Exclusive Gateways* and *Sequence Flows*, as well as *Event Sub-Processes*, but no *Loop* or *Ad-Hoc Sub-Processes*. Given a process with a task that uses a *Loop* marker needs to be ported to the Activiti engine, the code needs be adapted to one of the versions Activiti supports. To summarize the above discussion, the adaptability score of a task with a *Loop* marker is equal to four.

3.1 Adaptability of Atomic Process Elements

We define the adaptability score for atomic process elements as:

$$AS(e) = \mid \{alt_1^e, \ldots, alt_n^e\} \mid \tag{1}$$

The *adaptability score* AS of element e is equivalent to the cardinality of the set of alternatives $\{alt_1^e, \ldots, alt_n^e\}$ for the element that are available in the language. For the approach to work, such a score must be provided for every relevant atomic element of the BPMN specification. At the moment, we are fixing the appropriate score for every element, being activities (*Tasks*, *Sub-Processes*, *Call Activities*), *Data Items*, *Events* and *Gateways*.

[1] For more information, see the Activiti user guide: `http://www.activiti.org/userguide/index.html`.

The decision on what counts as a relevant atomic element is a design choice of the approach. It is reasonable to exclude a certain set of language elements from the computation. On the one hand, these are elements that are very basic and also very common and, as a consequence, are supported by every implementation of the standard. The inclusion of these elements in the adaptability computation would only have a distorting effect. On the other hand, certain elements are simply irrelevant to process execution and their implementation in a runtime is unimportant. *Lanes* fall into the latter category, as they have no real impact on the executability of a process, but are mainly relevant to visualization. The first category contains *Sequence Flows* and *Exclusive Gateways*. *Sequence Flows* are very basic language elements and it is hard to build processes in BPMN without using them. *Ad-Hoc Processes* in combination with *Data Inputs* and *Data Outputs* can, to a limited degree, replace *Sequence Flows*, but fail for instance when parallelism is involved. As they are typically very frequent, but cannot really be adapted anyway, we exclude them from the computation. *Exclusive Gateways* can be adapted to most other forms of gateways, such as an *Inclusive Gateway* where only one expression will evaluate to true, a *Complex Gateway*, or an *Event-Based Exclusive Gateway*. Nevertheless, they are the most basic mechanism for controlling the program flow and normally available in every implementation of the specification. Including them in the computation would introduce noise into the metric value. To decide if an element belongs to the basic subset, it could be helpful to look at its typical frequency in process models. [19] disusses this aspect for an older revision of BPMN and an updated study focused on executable processes might be worthwhile.

Finally, it is important to note that this approach rewards the availability of multiple equivalent constructs in a language. From a usability point of view, this is often considered as a drawback of the language, because its users can be confused on what syntax is best to be used. However, from the viewpoint of adaptability, it is positive, since multiple alternatives increase the likelihood of having at least one of them available in a given runtime.

3.2 Aggregation of Adaptability Scores

Based on atomic adaptability scores, we now need a mechanism for aggregating these scores to a global adaptability degree for the complete process. This is necessary to allow for the comparison of different processes in terms of their adaptability. Moreover, the aggregated degree should be normalized with respect to the size of the process, to enable the comparison of processes of different size. A straightforward way of aggregating adaptability scores is the following:

1. Normalize the score for every element.
2. Similar to [17], compute the mean score of all elements in the process.

This leads to the question of how to normalize scores on an atomic level. We propose to divide the score by a reference value. This reference value can be identified by the maximum adaptability score achieved by any of the elements in the language. That way, the most adaptable language element will have a

normalized score of one, whereas other elements will have a value between zero and one. This results in the following equation:

$$AD(p) = \overline{AD(e_1, \ldots, e_n)} = \overline{(AS(e_1)/R), \ldots, (AS(e_n)/R)} \tag{2}$$

The *adaptability degree AD* of process p, which consists of the elements $e_1, \ldots, e_n$, is equal to the arithmetic mean of the *adaptability scores AS* for every element e divided by the *reference value R*.

For the choice of the reference value, which we currently determined to be six, different schemes are possible. The scheme we use here has several advantages with respect to the computation:

1. The resulting metric value always ranges in the interval of $[0, \ldots, 1]$ and thus resembles a percentage value. This scale is easy to understand and interpret, which is critical for the adoption of the metric.
2. The reference value is identical for processes of the same language. Using a reference value that is specific to a concrete process might output a more meaningful adaptability degree for that process, but it would no longer be directly comparable with different processes. That way, the metric would lose one of its primary purposes.

4 Use Case

In the following, we use an example process[2], depicted in Figure 1, to demonstrate the computation of the adaptability degree. The process consists of a *Lane*, two

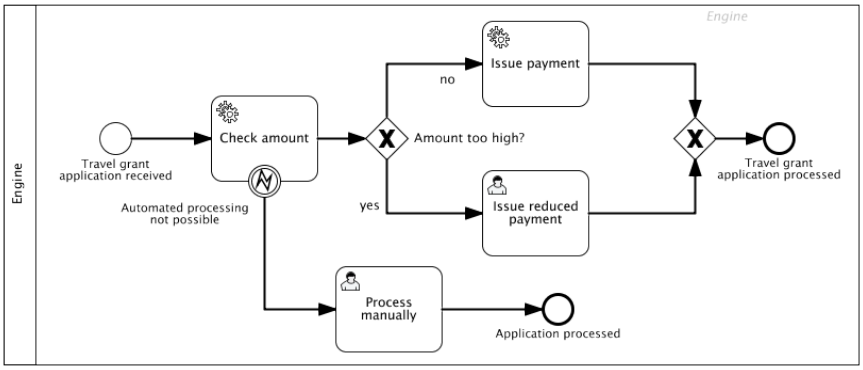

Fig. 1. Travel grant application process

User Tasks and two *Service Tasks*, one of which has an *Interrupting Error Boundary Event*, two *Exclusive Gateways*, several *Sequence Flows*, as well as two *End Events* and a *Start Event*. Table 1 shows the adaptability scores we

[2] The process is executable on Camunda BPM 7.0.0. The code and instructions on how to execute it are available at `https://github.com/uniba-dsg/zeus2014`.

Table 1. Adaptability scores of process elements rounded to two decimal places

Element	*None Start Event*	*None End Event*	*Task*	*Error Boundary Event*
$AS(e)$	5	4	5	6
$AD(e)$, $R = 6$	0.83	0.67	0.83	1

determined for the respective elements of the travel grant application process. As discussed in section 3.1, *Lanes*, *Sequence Flows*, and *Exclusive Gateways* are not considered in the computation. The BPMN specification lists seven different triggers for process start, and hence seven different types of *Start Events* [15, pp. 240/241] do exist. All except for the *Timer Event* are a suitable alternative for the *None Start Event* used in the process, resulting in an adaptability score of five. For *End Events*, nine different types do exist [15, pp. 247–249], five of which, including the *None End Event*, can be used to express orderly termination, resulting in four alternatives for it. Furthermore, there are seven different types of *Tasks* in BPMN [15, pp. 158–165]. Again, the idea is that *Tasks* which are not supported by an engine can be adapted to a different type of *Task*, for instance a *Service Task* could be adapted to a *Script Task*. All tasks actively perform an action, except for the *Receive Task* which is waiting for an action, so there are five alternatives for the tasks used in the process. If the *Error Boundary Event* is not supported, it might be possible to use a different interrupting boundary event which also changes the normal flow into an exception flow. Here, a *Message*, *Escalation*, *Conditional*, *Signal*, *Multiple*, or *Multiple Parallel Interrupting Boundary Event* could achieve the same result [15, pp. 254–257]. The reference value R is six, which results in the adaptability degree values depicted in Table 1. The resulting adaptability degree for the use case is computed in the following: $AD(p) = ((1 * AD(StartEvent)) + (4 * AD(Task)) + (2 * AD(EndEvent)) + (1 * AD(ErrorEvent)))/8 = ((1 * 0.83) + (4 * 0.83) + (2 * 0.67) + (1 * 1))/8 = 0.81$.

5 Conclusion

In this paper, we proposed a mechanism for computing a degree of structural adaptability for BPMN processes that aims to quantify how easily a process can be adapted to a different form with the same runtime behavior. Furthermore, we demonstrated its computation in a use case. Such a degree can be helpful for quality assessment during development or decision support during migration.

Several aspects of the computation are still open: First, adaptability scores need to be fixed for every relevant element and they should be confirmed in peer review. Moreover, a validation of the proposed adaptability metrics is needed. On the one hand, validation should be considered from a theoretical point of view, for instance by clarifying the measurement-theoretic properties of the metrics or by confirming construct validity. On the other hand, the practical applicability of the metrics should be confirmed, for instance in an experiment with real-world processes. This could be used to evaluate if the adaptability degree can meaningfully discriminate between processes of different quality.

References

1. A. Aldris, A. Nugroho, P. Lago, and J. Visser. Measuring the Degree of Service Orientation in Proprietary SOA Systems. In *7th IEEE International Symposium on Service-Oriented System Engineering*, San Francisco Bay, USA, March 2013.
2. L. Briand, S. Morasca, and V. Basily. Property-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22(1):68–86, 1996.
3. M. Geiger and G. Wirtz. BPMN 2.0 Serialization - Standard Compliance Issues and Evaluation of Modeling Tools. In *5th Int. Workshop on Enterprise Modelling and Information Systems Architectures*, St. Gallen, Switzerland, September 2013.
4. S. Harrer, J. Lenhard, and G. Wirtz. BPEL Conformance in Open Source Engines. In *IEEE SOCA*, Taipei, Taiwan, December 17-19 2012.
5. S. Harrer, J. Lenhard, and G. Wirtz. Open Source versus Proprietary Software in Service-Orientation: The Case of BPEL Engines. In *11th International Conference on Service Oriented Computing (ICSOC)*, pages 99–113, Berlin, Germany, 2013.
6. ISO/IEC. *Software engineering – Product quality – Part 3: Internal metrics*, 2003. 9126-3:2003.
7. ISO/IEC. *Systems and software engineering – System and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*, 2011. 25010:2011.
8. ISO/IEC. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality*, 2013. 25023.
9. C. Kaner and W. Bond. Software Engineering Metrics: What Do They Measure and How Do We Know? In *10th International Software Metrics Symposium*, Chicago, USA, September 2004.
10. R. Khalaf, A. Keller, and F. Leymann. Business processes for Web Services: Principles and applications. *IBM Systems Journal*, 45(2):425–446, 2006.
11. J. Lenhard, S. Harrer, and G. Wirtz. Measuring the Installability of Service Orchestrations Using the SQuaRE Method. In *IEEE SOCA*, Kauai, Hawaii, USA, December 16-18 2013. IEEE.
12. J. Lenhard and G. Wirtz. Measuring the Portability of Service-Oriented Processes. In *17th IEEE EDOC*, Vancouver, Canada, September 2013.
13. J.-L. Letouzey and M. Ilkiewicz. Managing Technical Debt with the SQUALE Method. *IEEE Software*, 29(6):44–51, 2012.
14. OASIS. *Web Services Business Process Execution Language*, April 2007. v2.0.
15. OMG. *Business Process Model and Notation (BPMN) Version 2.0*, January 2011.
16. D. Perez-Palacin, R. Mirandola, and J. Merseguer. On the Relationships between QoS and Software Adaptability at the Architectural Level. *Journal of Systems and Software*, 87(1):1–17, 2014.
17. N. Subramanian and L. Chung. Metrics for Software Adaptability. In *Proc. Software Quality Management*, Loughborough, UK, April 2001.
18. Z. Zhou, S. Bhiri, H. Zhuge, and M. Hauswirth. Assessing Service Protocol Adaptability Based on Protocol Reduction and Graph Search. *Concurrency and Computation: Practice and Experience*, 23(9):880–904, 2011.
19. M. zur Muehlen and J. Recker. How Much Language is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In *Advanced Information Systems Engineering (CAiSE)*, Montpellier, France, June 2008.