Characterizing Declarativity across Business Process Formalisms

Jeroen van Grondelle

Be Informed, Wapenrustlaan 11-31, 7321 DL Apeldoorn, The Netherlands
j.vangrondelle@beinformed.com

Martijn Zoet

HU University of Applied Sciences Utrecht, Nijenoord 1, 3552 AS Utrecht, The Netherlands
martijn.zoet@hu.nl

Floor Vermeer

Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands
floorvermeer@gmail.com

ABSTRACT

Organizations are struggling to choose from or combine the different business process management paradigms offered in today's BPM landscape, such as workflow management, dynamic case management and straight through processing. The field of declarative processes seems to be able to address this challenge by offering a unified approach to business process modeling, providing variable amounts of flow at execution time and different levels of autonomy to the actors based on models using a single formalism. The notion of declarativity in business processes seems to be ill defined and is often treated as a black and white distinction. However, a number of quite different formalisms have been developed that are broadly agreed to be declarative. This paper proposes a number of qualitative characteristics to characterize the declarative nature of process modeling formalisms. The characteristics are evaluated by applying them to a number of relevant process modeling formalisms, both imperative and declarative, and we discuss how these characteristics can be utilized to create business processes that offer activity flows that are known up front where needed, and allow ad hoc approaches to offer experts freedom and to support impediment driven approaches in an STP context.

Key words: Declarativity, characteristics, business process formalisms, procedural.

INTRODUCTION

The field of Business Process Management (BPM) offers a number of paradigms, that each have their own process modeling formalisms, tools and techniques. As a consequence, organizations that are, for instance, selecting tools and techniques end up making big choices up front between these paradigms and having to apply those choices to processes categorically.

Current business process management research differentiates between business process paradigms by applying various characteristics. Examples of characteristics are: 1) process

structure; 2) process driver; 3) route determination at runtime; 4) actor autonomy at runtime. Although no standard classification of paradigms exist, in general, literature (Huang, Chen, & Hee, 2006; van der Aalst, Weske, & Grünbauer, 2005; Weske, 2007) distinguishes between three paradigms of business processes: *Workflow Management* (WFM), *Adaptive Case Management* (ACM) and *Straight Through Processing* (STP), see Figure 1.

WFM as defined by the Workflow Management Coalition is "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules" (Workflow Management Coalition, 1997). WFM tries to capture and support the work that is needed to process a case in terms of activities, the order they are performed in and the actors that have to do so. The choice to design these flows up front makes that they inherently struggle in offering flexibility in complex cases where experts may want or need to deviate from the default flow.

ACM as defined by Reijers, Rigter and Van der Aalst (2003) are business process in which "various activities in the process definition are distinguished but only a preferred or normal control flow between these activities is modeled. By default at runtime an actor is able to execute, redo, or bypass these activities, possible diverging from the normal flow." ACM is an approach where the actor is allowed to a high degree to choose the activities and their order. Thus allowing maximal flexibility, this approach often fails to provide its users support in mainstream cases, in which they do not need all that flexibility, and it fails to guarantee consistency across such similar cases.

STP as defined by Huang, Chen and Hee (2006) is the "the electronic movement through a process from initiation through post-execution and final settlement without manual intervention." STP is applied in cases where the majority of cases are expected to be processed in a single transaction, and all cases that fail to do so are treated as exception cases and require manual intervention. Often, intervention on a single aspect leads to a complete case being treated in exception mode, also on aspects that could be regarded as within STP parameters, instead of dealing with the impediment locally and treating the case as a regular case otherwise.

In practice, the choice between these process paradigms cannot be made for complete business processes or for the complete caseload handled in specific business processes. For example, insurance claim handling, patent applications, accident investigation, legal cases and crime investigations require a combination of straight through processing, workflow management and adaptive case management. These types of processes have to meet mixed requirements: 1) Providing autonomy to experts, while making sure all relevant regulations are complied with; 2) Support for 80/20 caseloads, which combine mainstream cases, that can be processed using standard approaches, with complex cases; 3) Support for actors of different expertise and authorization levels, where the experts have a higher degree of freedom to deviate from standard processing practices than their less experienced colleagues. Thus, the nature of business processes is not single type oriented and organizations can no longer focus on creating effective business processes by only applying one paradigm. Instead, business processes need to combine the features of the different paradigms and differentiate between process fragments.

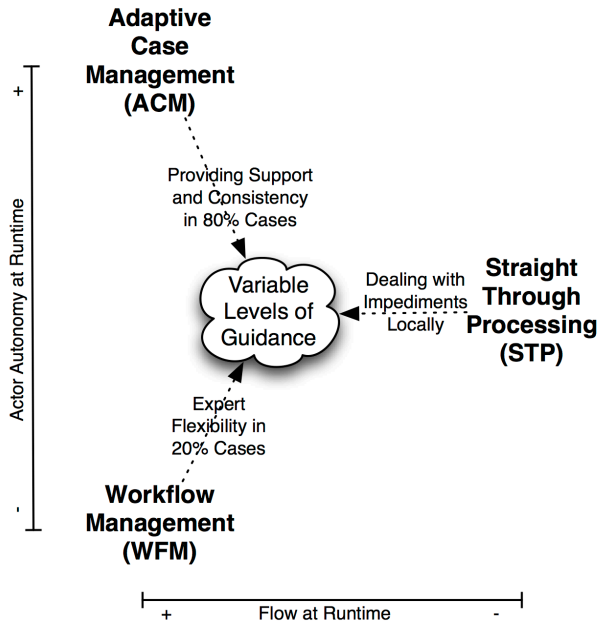

Figure 1: BPM Paradigms and their Inherent Challenges

Current business process modeling standards such as BPMN (Recker, 2010) are based on imperative formalisms. Imperative formalisms do not allow any deviation at runtime from the prescribed model without changing the model. Therefore, scholars as well as practitioners conclude that imperative formalisms are not the most suitable modeling languages to model ACM process instantiations (Goedertier & Vanthienen, 2007; van der Aalst et al., 2005; Van Grondelle & Gülpers, 2011).

The main reason identified is that procedural modeling languages state how, what and when work must be done. On the other hand, declarative modeling languages only state what must be done (Goedertier & Vanthienen, 2007). Declarative modeling languages leave as much freedom as permissible to determine an execution scenario. However, the notation of procedural and declarative, in scientific literature, is treated as a black and white distinction (Goedertier & Vanthienen, 2007). This makes it hard to determine whether and how declarative formalisms meet the requirements for mixed support mentioned.

This article extends the understanding of the declarative nature of different process modeling formalisms. Similar to previous research, we will consider qualitative characteristics as the means to characterize business processes. Dissimilar to previous research, we do not focus on the distinction between declarative and procedural, but start by analyzing the declarative nature of process modeling formalisms and define our characteristics from this. With these premises, the specific research question addressed is: *"how can declarativity be characterized across process formalisms?"*

The remainder of this paper proceeds as follows. The next section provides a context by describing different viewpoints on procedural versus declarative formalisms. The third section describes the four characteristics of *declarativity* identified. Section four presents the results of

an experiment based on case study data. The final section summarizes the study's core findings, contributions as well as it limitations.

RELATED WORK

The notion of declarative formalisms to model business processes has recently attracted increased interest, from scientist as well as professionals, due to limitations of traditional procedural formalism (Recker et al, 2005). To overcome limitations of procedural formalism researchers have introduced new declarative process modeling languages.

The first type of declarative modeling languages, is state driven and can be classified as Petri Net based declarative modeling languages. The Case Handling Paradigm was introduced by Van der Aalst et. al (2005). Main focus of this formalism is on the case and case data, instead of the control flow. Based on this paradigm the process languages Condec and Declare have been developed (Pesic & Van der Aalst, 2006). Both are based on Petri Nets and use transitions connected to each other by means of constraints. The constraints are based on linear temporal logic (Clarke, Grumberg, & Peled, 1999). EM-BRA^{2}CE is a formalism based on the Semantics for Business Vocabulary and Rules (Goedertier, Haesen, & Vanthienen, 2007). The formalism offers 16 types of business rules, such as activity pre- and post conditions, to create a declarative model. This model is inferred to colored Petri Nets, which are used to execute the model. Another formalism is the Declarative Process Modeling Notation (DPMN) (Van Grondelle & Gülpers, 2011). Similar to Condec, Declare and EM-BRA^{2}CE, DPMN is also based on activities and pre- and post-conditions. In addition to the development of declarative formalisms, research also focused on evaluating different aspects such as understandability and easy of change (Fahland, Lübke, et al., 2009; Fahland, Mendling, et al., 2009; Pichler, Weber, Zugal, Pinggera, & Reijers, 2012; Schonenberg, Mans, Russell, Mulyar, & van der Aalst, 2008; Zugal, Soffer, Pinggera, & Weber, 2012).

The business process management research domain as a discipline is relatively young. In comparison the computer system development domain is a mature research domain. The notation of declarativity has been extensively studied with regards to programming languages and numerous declarative programming languages have been developed. Examples of such languages are Prolog (Colmerauer & Roussel, 1996), Lisp (Steele, 1990) and more recent, Scala (Odersky, Spoon, & Venners, 2008). Advantages contributed to previously mentioned programming languages in specific and declarative programming languages in general include: 1) compactness, 2) high expressivity, 3) modularity and, 4) (automatic) concurrency. Previous research has focused on examining parallels between declarative programming languages and declarative process languages (Fahland, Lübke, et al., 2009; Fahland, Mendling, et al., 2009) at the level of formalisms. We argue that this view is limited and use the field of declarative programming languages by studying how declarative programming compilers or runtimes use the underspecification in declarative programs to make behavioral optimization choices at runtime, by for instance choosing between alternative execution models, executing code in parallel or skipping unnecessary code altogether. These are relevant choices in business process context as well, based on a similar inference span between the specification model and the eventual execution sequences, based on declarativity.

CHARACTERISTICS OF DECLARATIVITY

In order to define declarativity in business process formalisms better and study how declarativity can help combine the strong features of the different BPM paradigms in one business process, we propose four characteristics that process modeling formalisms may or may not possess. They, in an abstract way, represent ways for formalisms to reach declarativity in process modeling, by decoupling, to different degrees, the type of statements allowed when specifying processes from the concepts typically used to represent the resulting process instances at execution times, in terms of activities and the order they are performed in. We present these characteristics in the order of increasing inference span between specification statements and execution flow.

Rule-oriented Formalisms

Supporting rule-oriented statements is potentially the most intuitive criterion for a formalism to qualify as declarative.

Characteristic 1 (Rule Orientation): The formalism allows if/then statements.

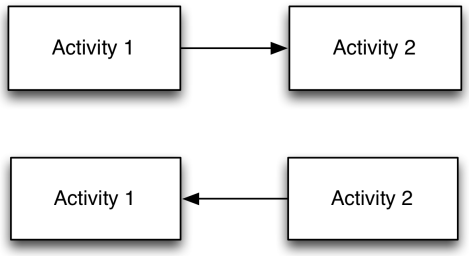

Figure 2: Inverse Direction of Specification

This characteristic is very close to the notion of rule based processes, and is from a formal perspective probably enough to qualify as a declarative formalism.

Adopting rule oriented aspects in a business process formalisms generally introduces the opportunity to reason about processes and infer the order in which activities are performed from the rules, without that order being encoded explicitly.

At the same time, rule orientation can be used in specification formalisms that stay very close to the activity flows used at execution time and as such do not bring the benefits associated with declarative business processes per se. A worst case example of this is the statement "if *X* is finished, then *Y* performs *Z*". In that case, it essentially encodes ordered flow and only trivial inference is required to construct flow. Obviously, a rule based approach becomes more useful when other modalities are used in the rules, such as when activities may, could, must or need to be performed.

In this paper, we restrict ourselves to formalisms that aim to represent business processes. This excludes a class of production rule systems that infer sequences of behavior based on domain rules only.

Indirection between Activities

In flow-based formalisms, each activity is connected to other activities to encode the order in which they are performed. This strong coupling limits the ability to reconfigure that order later, including at runtime.

Characteristic 2 (Activity Indirection): The formalism supports statements that refer only to a single activity, but that do impact the order of execution at runtime.

This property introduces meaningful inferences, as activities do need to be related at execution time in terms of order (restrictions) between them. When the process is specified using statements that do not relate activities, inferring that order from other types of constraints is by definition non-trivial.

A typical example of this is indirection through artifacts or data, as in Figure 3. In that example, the producer and consumer of the data or artifact are conventionally modeled as having an order relation, to guarantee that the artifact is produced before it is consumed. Modeling the production and consumption instead allows manipulation of the flow, especially with respect to other activities that do not interact with the artifact at all and can be mixed with these activities in a multitude of ways.

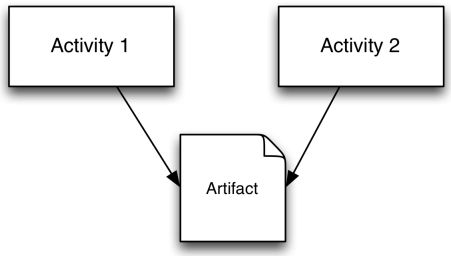

Figure 3: Example of Indirection between Activities across Artifacts

From an implementation perspective, this characteristic allows a whole new class of process modeling methodologies. As no statement needs to refer or relate an activity to another, resulting specifications can be unconsolidated to a high degree. Domain experts and process owners can express their constraints independent of other participants and the activities they perform. A flow that meets the constraints is inferred, potentially at runtime.

Furthermore, this indirection provides implicit override flows. Typically, this characteristic is implemented by replacing order by another notion, such as causality or availability. In the example, the artifact may for instance already exist from a previous process instance or the production of the artifact could be overridden with alternative ways to provide the artifact.

Goal Orientation

A deeper notion, potentially spanning an arbitrary number of activities, is the notion of goal orientation. Instead of defining a process instance by capturing how it came to its current state, the main emphasis shifts to what is needed to complete the case.

Characteristic 3 (Goal Orientation): The formalism allows expressing a desired end state and allows making inferences on which activities would contribute or are needed to reach that end state.

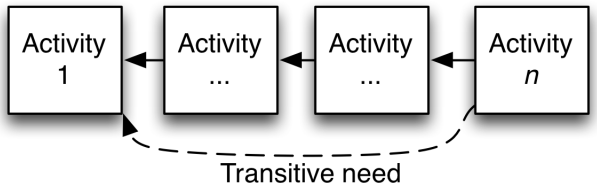

Figure 4: A Transitive Notion of Need

This characteristic is related to the idea of lazy evaluation from the field of functional programming: functional compilers for instance have the ability to defer function execution, from the moment that all required arguments are present and the function *can* be executed to the moment that can be determined that the result of the function is *needed*, to so prevent unnecessary work.

This is one of the strong features of goal oriented processes: By computing what activities are needed to reach completion from the current state, and, in turn and recursively, what activities are needed to perform those activities, flows can be computed that contain the least possible activities to complete a process instance from any possible state. Even if the process instance state is the result of an ad hoc intervention by an expert, leading to an unanticipated flow, a follow up process to complete the case is available, without the need to model all the exception flows up front.

Implicit Activities

A central notion in any process formalism typically is the concept of activity. It encodes a unit of work, to be performed by someone at a time to some extent prescribed by the process. However, many exceptions in daily operation require work to be performed in a way not anticipated by the process, requiring the units of work to be rearranged.

Characteristic 4 (Implicit Activities): The formalism includes an notion of activity to explicitly demarcate a unit of work, but also supports other activities at runtime that are not specified as activity explicitly, but are inferred from the process model in other ways.

For example, runtime activities may include activities that are equivalent to a combination of specified activities, or are part of a specified activity. The first is useful to provide straight through flows for easy cases, where a single transaction may be constructed based on the process model to complete the case without human interaction. The latter can be useful to deal with

activities that can only be performed in part, and completion has to be deferred to a later time or to an actor with higher authorization level.

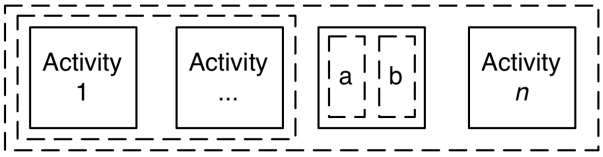

Figure 5: An Example of Implicit Activities available at Runtime

This characteristic also has a counterpart in the field of functional programming. The side effect free nature of functions in functional programming allows the compiler or runtime to reconfigure functions to optimize execution, by for instance inlining often used, small functions into their dependent functions. Another important objective of functional programming is to optimize workloads in parallel execution across cores without the programmer explicitly dealing with this in the code.

ANALYSIS

The goal of this research is to validate the four declarative characteristics using existing Business Process formalisms found in literature. The units of analysis therefore are: declarative and imperative process formalism. The languages selected as proxy values are: 1) BPMN, 2) Declare, 3) EM-BRA^{2}CE and, 4) DPMN. To ground our analysis we apply a running example: the hiring process depicted in figure 6.

Running example

In the figure below (Figure 6), the process for hiring a new employee is modeled in BPMN. The process consists of several sequential tasks. Some of the tasks are performed concurrently, because they are preceded by an AND waypoint (denoted by a diamond figure containing the "+" symbol.

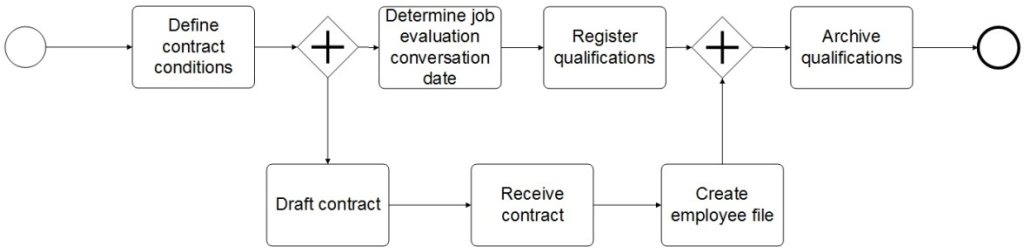

Figure 6: The hiring process, modeled in BPMN

While BPMN can be used to model this process, none of the declarative characteristics previously mentioned are featured by the formalism. The activities cause another activity to be activated, instead of requiring activities to be completed before continuing (rule orientation). Furthermore, except for routing concepts such as waypoints, all activities are followed up by another activity. This indicates that indirection between activities is not possible. There is also no

goal orientation in this model, since all activities are performed sequentially without any regard to reaching the end of the process. Lastly, activities in BPMN are the only concepts that indicate any work needs to be done, leaving out the possibility for implicit activities.

The second language analyzed is Declare. Declare utilizes restrictions on execution paths between activities to design process models. Different type of restrictions can be applied; for example no restriction between activities or if / then construction between activities. By allowing if / then constructions Declare exhibits characteristic one: rule orientation. However, because restrictions must be specified between activities Declare does not allow for activity indirection. Declare does not support implicit activities (characteristic four). All activities must be defined upfront and runtime can only invoke changes in order of activities based on predefined restriction.

In contrast, multiple characteristics are featured in the constraint-based formalisms. A formalism which is able to support the Rule Orientation, Indirection and Goal Orientation characteristics is EM-BRA^{2}CE (Goedertier et al., 2007). To demonstrate the rule orientation characteristic, an example of inversing activity relations is given:

"To start a Receive contract activity, it is necessary that a Draft contract activity has been completed and that no Receive contract activity has been started."

Another example with multiple activity preconditions is given below:

"To start a Archive qualifications activity, it is necessary that a Register qualifications has been completed, a Create employee file has been completed and that no Archive qualifications activity has been started."

The notion of activity preconditions can be expanded, where the precondition is not on the preceding activity, but rather on the availability of an artifact. As such, indirection between activities can be achieved. As an example, the following rules could be used to achieve indirection:

"To complete an activity that *has type* create employee file, it is necessary that the process *has an* employee file" and "To start an Archive qualifications activity, it is necessary that the process *has* an employee file and a Register qualifications activity has been completed."

These rules state that the activity "create employee file" can only be finished when the process contains an employee file. Furthermore, to start the "Archive qualifications activity", an employee file needs to be available and the register qualifications activity needs to be completed.

A formalism that supports all of the declarative characteristics is DPMN. To further demonstrate the declarative characteristics in existing formalisms, examples of goal orientation and implicit activities are shown in this formalism.

The figure below (Figure 7) depicts a fragment of the process model in DPMN (Van Grondelle

& Gülpers, 2011). As can be seen in the figure, the main concept in the model is the case, which performs several activities. The activities have a relation to one or more artifacts as a pre- or post condition. The only mandatory activity in this model is the "archive qualifications" activity and the completion of this activity can be seen as the goal of the process model. The other activities only have to be performed if they are needed. Because the "Archive qualifications" activity requires artifacts resulting from the other activities to be available, they have to be performed as well. The inferring of activities which are required to be performed corresponds to the third characteristic described in this paper: Goal orientation. DPMN also features a weak notion of activity, allowing activities to be split, merged or combined otherwise when it is possible.

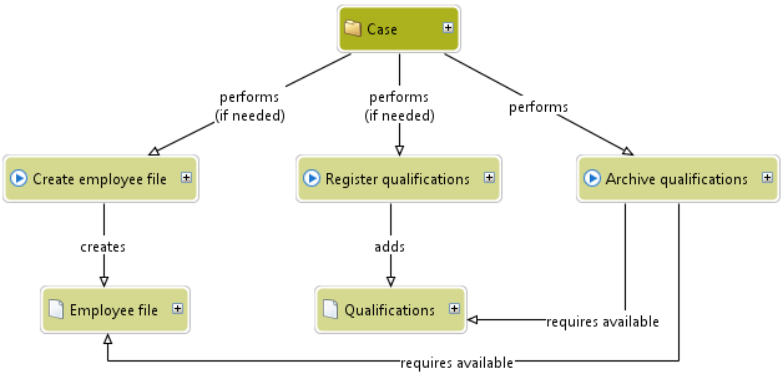

Figure 7: Fragment of hiring process in DPMN

CONCLUSION AND DISCUSSION

In this paper we have introduced four characteristics of business process formalisms that capture how such formalisms decouple specification metaphors from the activity flows supported at runtime in a formalism independent way, and the types of inferences they support. We have shown that a number of business process formalisms that are broadly regarded to be declarative do not meet the same characteristics and therefore that declarativity in business process formalisms should not be considered a binary property.

The characteristics introduced seem to map well to the challenges described in the introduction, on how to combine the strong features of the major business process management paradigms within single business processes.

A formalism that supports indirection between activities provides implicit override scenarios where experts, when meeting authorization constraints, can progress within a business process when the default activities prescribed cannot be executed. By realizing the artifact that is used for indirection in an alternative way, the normal activity flow can be joined again after an override by performing the activity that depends on it according to the process model.

Goal orientation in a process formalism facilitates dual use of activity based process models:

laymen and inexperienced users can be supported by offering activities that lead to the goal, while experts can be allowed to perform other activities too, of which the purpose is not (yet) clear. The goal orientation allows for the process model to still provide guidance for process instances in which overrides were made. Flow's forward orientation does not have this property, as it cannot provide next steps for instances that left the flow prescribed by the model.

Support for implicit activities in process formalisms allows for blending of manual support and STP scenarios, independent of how fine grained and explicit the work was decomposed into activities at specification time. Joining activities into bigger activities helps processing business processes that are modeled as a sequence of activities with minimal human involvement. Similarly, the tasks that need to be performed to solve impediments in an STP flow can be derived from an STP oriented process model.

REFERENCES

Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). Model Checking. *Journal of the American Statistical Association*, *962*(5), 314-320.

Colmerauer, A., & Roussel, P. (1996). The birth of Prolog. In Bergin, T. and Gibson, R. (EDS.) History of Programming Langauges II. ACM Press.

Fahland, D., Lübke, D., Mendling, J., Reijers, H. A., Weber, B., Weidlich, M., & Zugal, S. (2009). Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. In T. A. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, & R. Ukor (EDS.), *BMMDS/EMMSAD* 1(29), 353–366.

Fahland, D., Mendling, J., Reijers, H. A., Weber, B., Weidlich, M., & Zugal, S. (2009). Declarative versus Imperative Process Modeling: The Issue of Maintainability. *Proceedings of the* ER-BPM. (pp. 65-76).

Goedertier, S., Haesen, R., & Vanthienen, J. (2007). EM-BrA2CE v0. 1: A vocabulary and execution model for declarative business process modeling. *Available at SSRN 1086027*, 0–74. Retrieved from http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1086027

Goedertier, S., & Vanthienen, J. (2007). Declarative Process Modeling with Business Vocabulary and Business Rules. *Proceedings of the OTM 2007 Workshops.* (pp. 603–612).

Huang, W., Chen, Y., & Hee, J. (2006). STP Technology: an overview and a conceptual framework. *Information & Management*, *43*(3), 263–270.

Odersky, M., Spoon, L., & Venners, B. (2008). *Programming in Scala*: A Comprehensive Step-by-Step Guide. Artima Inc.

Pesic, M., & Van der Aalst, W. M. P. (2006). A Declarative Approach for Flexible Business Processes Management. *Proceedings of the Business Process Management Workshops.(pp.* 169–180).

Pichler, P., Weber, B., Zugal, S., Pinggera, J., & Reijers, H. A. (2012). Imperative versus Declarative Process Modeling Languages : An Empirical Investigation. *Proceedings of the Business Process Management Workshops*. (pp. 383–394).

Recker, J. (2010). Opportunities and Constraints : The Current Struggle with BPMN. *Business Process Management Journal*, *16*(1), 181–201.

Reijers, H. A., Rigter, J. H. M., & Van der Aalst, W. M. P. (2003). The case handling case. *International Joarnal of Cooperative Information Systems*, *12*(3), 365–391.

Schonenberg, H., Mans, R., Russell, N., Mulyar, N., & Van der Aalst, W. M. P. (2008). Process flexibility: A survey of contemporary approaches. *Advances in Enterprise engineering,* (PP. 16–30.)

Steele, G. L. (1990). *Common LISP: the language.* New York, Digital Press.

Van der Aalst, W. M. P., Weske, M., & Grünbauer, D. (2005). Case handling: a new paradigm for business process support. *Data & Knowledge Engineering*, *53*(2), 129–162.

Van Grondelle, J., & Gülpers, M. (2011). Specifying Flexible Business Processes using Pre and Post Conditions. *Proceedings of the Practice of Enterprise Modeling* (pp. 1–14).

Weske, M. (2007). *Business Process Management: Concepts, Languages, Architectures.* New York, Springer.

Workflow Management Coalition. (1997). *Workflow Handbook*. (P. Lawrence, Ed.). New York: John Wiley.

Zugal, S., Soffer, P., Pinggera, J., & Weber, B. (2012). Expressiveness and Understandability Considerations of Hierarchy in Declarative Business Process Models. Proceedings of *Enterprise, Business-Process and Information Sysyems Model*ing (pp. 167–181).