university of groningen

umcg:

University of Groningen

Requirements and Tools for Variability Management

Aiello, Marco; Bulanov, Pavel; Groefsema, Heerko

Published in:
Computer Software and Applications Conference Workshops (COMPSACW)

DOI:
10.1109/COMPSACW.2010.50

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2010

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
Aiello, M., Bulanov, P., & Groefsema, H. (2010). Requirements and Tools for Variability Management. In *Computer Software and Applications Conference Workshops (COMPSACW)* (pp. 245-250). IEEE (The Institute of Electrical and Electronics Engineers). https://doi.org/10.1109/COMPSACW.2010.50

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license. More information can be found on the University of Groningen website: https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment.

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Download date: 05-10-2026

2010 34th Annual IEEE Computer Software and Applications Conference Workshops

Requirements and Tools for Variability Management

Marco Aiello, Pavel Bulanov, Heerko Groefsema
Distributed Systems Group
University of Groningen
The Netherlands
Email: {m.aiello, p.bulanov, h.groefsema}@rug.nl

***Abstract*—Explicit and software-supported Business Process Management has become the core infrastructure of any medium and large organization that has a need to be efficient and effective. The number of processes of a single organization can be very high, furthermore, they might be very similar, be in need of momentary change, or evolve frequently. If the ad-hoc adaptation and customization of processes is currently the dominant way, it clearly is not the best. In fact, providing tools for supporting the explicit management of variation in processes (due to customization or evolution needs) has a profound impact on the overall life-cycle of processes in organizations. Additionally, with the increasing adoption of Service-Oriented Architectures, the infrastructure to support automatic reconfiguration and adaptation of business process is solid.**

In this paper, after defining variability in business process management, we consider the requirements for explicit variation handling for (service based) business process systems. eGovernment serves as an illustrative example of reuse. In this case study, all local municipalities need to implement the same general legal process while adapting it to the local business practices and IT infrastructure needs. Finally, an evaluation of existing tools for explicit variability management is provided with respect to the requirements identified.

I. Introduction

Tools for Business Process Management (BPM) are becoming more and more a utility of any medium and large scale enterprise. The emergence of Service-Oriented Architectures and standards such as Web Services has accelerated the trend and opened a wide range of automation and integration possibilities. Though, with the adoption of new tools, new needs emerge [1]. Designed to support rigid and repetitive units of work, like production processes, business process models offer little in the area of flexibility [2], [3]. Consider, for instance, the issue of customizing a process for a given product or to implement changes for a set of similar products. The current common practice is to branch a process model into many very similar instances. As a result, redundancy is introduced and maintainability is set at mostly unacceptable levels.

The recurrence of the need to adapt processes to instances and changes become concrete with the notion of *variability*, which first emerged in software engineering. Variability refers to the possibility of changes in software products and models. The changes are normally explicitly defined via appropriate variability modelling languages and are managed with semi-automatic tools. An example is the COVAMOF framework [4] used in product families to manage the different requirements and constraints of similar, but not equivalent products. In [5], the example of the Intrada product family of intelligent traffic systems (from Dacolian B.V.) is used as the running example, being a code base of approximately 11 million lines of code adapted to products which will be deployed in different traffic conditions.

Product lines are a prototypical example of software variability, but not the only one. In [6], two examples from health care and the automotive industry are proposed. The case of moving from a nationally defined law to hundreds of implementations in local government bodies is the object of the SAS-LeG project [7]. Broadening the view, flexibility has received a wider attention. Aalst et al. [8] describe the set of patterns to cover requirements of business process models. Weber et al. [6] discuss patterns related to change in business process models. In [9], a number of approaches to flexibility in process models are surveyed. A different approach is taken in [10], [11] by focusing on what needs to be done instead of how things are done. Finally, a number of tools have been proposed for flexibility in BPM and supporting SOA techniques.

This paper addresses the subject of variability in BPM by providing a definition of variability and its key related concepts (Section II). Using an example from variability in eGovernment (Section III), the set of requirements for variability management is presented in Section IV. Tools and frameworks are compared based on the requirements in Section V.

II. Variability Engineering

In this section the area of variability Engineering is explored. First, the general idea of variability and how it applies to business processes is discussed. Next, the impact on the BPM lifecycle is addressed, and finally a number of techniques to support the variability are given.

A. Variability

In software engineering, *variability* refers to the possibility of changes in software products and models [4]. In the context of BPM, variability indicates that parts of a business process remain variable, or not fully defined, in order to support different versions of the same process depending on the intended use or execution context. Such variability is often included through the introduction of so-called *variation points*, that is, elements of a business process where change may occur. A process in which variability is included is called a *reference* or *generic process*. Processes where choices have been made deriving from the reference process are called

978-0-7695-4105-1/10 $26.00 © 2010 IEEE
DOI 10.1109/COMPSACW.2010.50

IEEE computer society

variants. The strategies in order to come to a reference process can differ greatly, and ranges from drafting a simple template based on commonalities between variants to using a single variant as a reference process. *Variability management* is the set of activities aimed to cover the creation and support of differences in versions of reference processes.

The advantage of explicit variability treatment to modelling and managing business processes resides in the reuse of the reference processes and in the handling of evolutions and customizations. Instead of creating entirely different processes for each variant and thus introducing redundancy and the possibility of errors, variability offers the introduction of these variants through explicit traversal choices, thus eliminating duplication of work and a source of inconsistencies across versions. Another paramount advantage lies in the readability and maintainability of the processes. Instead of using BP constructs within a process to model the variations in circumstances explicitly, and thus introducing readability and maintainability problems, variability offers a clean solution through the introduction of variants.

Finally, we remark that *variability* is very closely related to *flexibility*. Flexibility offers adaptation and change of a process, whereas variability deals with different versions of a process. Of course, in order to support different versions, a certain amount of flexibility and change management is required. Two approaches to flexibility exist: imperative and declarative flexibility. The former of these focuses on how a task is performed, where the latter focuses on what tasks should be performed. The last of these two approaches is generally considered to be more flexible to changes but less strictly defined [9].

B. Variability management

Variability management is an extension of the typical activities involved in business process management. We give a general depiction in Figure 1. On the left can be seen how requirements drive the definition of the design processes. Once the designer has a first design of the process, then it moves on to the deployment and run-time phases. In case of errors, unpredictable situations and changes in requirements, the whole procedure must be repeated, starting from the design stage. During execution, information on normal and exceptional execution is collected (Monitoring and Diagnose phases), thus allowing for more flexible process support and feedback for process evolution. Variability management complements these general BPM phases by introducing a set of parallel stages, on the right in the figure. First, a decision is made on the types of variability, and at which stage of the lifecycle they should be introduced. In this context, there are two main categories: design-time and run-time variability.

Design–time variability deals with the definition of variations in processes at design–time. Similar processes are implemented using a reference process and applying different variations in order to support all variants. Often this means finding the commonalities between similar processes being implemented and introducing variations where differences

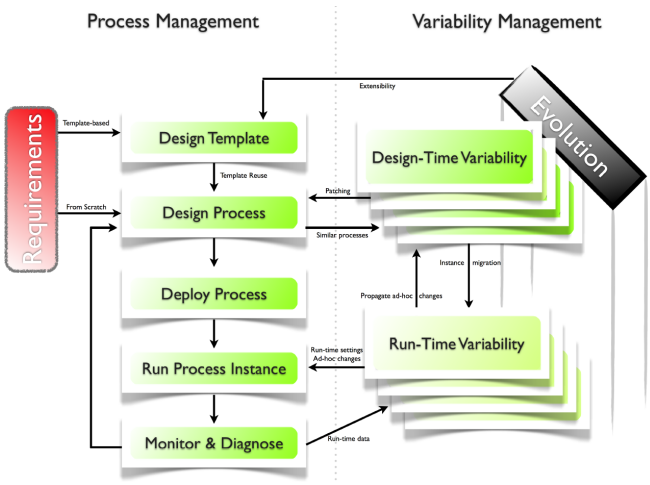

Fig. 1. Process lifecycle and variability management.

occur. Dually, it may mean taking a process and foreseeing all possible customization and changes that different context may require. The next step is designing the variations for the generic process in such a way that they cover all variants identified in the requirements through the flexibility offered by the variability. New variants are added by reuse of the existing reference process or patching existing variants. The source of such changes might be either changes in requirements or propagation of a change at a different level.

Run–time variability is responsible for managing variation of processes in execution. The major issue it addresses is handling redesigns of running processes. Such redesigns can range from skipping/deleting a single activity to moving to a whole different variant. The source of changes might be either changes in requirements, responding to an erroneous situation (via analysing run–time data) or a propagation of a change at a higher variability level (design–time or evolution).

A cross-cutting element of variability management is the evolution that generic processes go through. *Evolution variability management*, in black on the right of Figure 1, represents the changes introduced not by customization but rather by changes occurring over time.

C. Variability techniques

Frameworks for variability management should support different techniques to produce and manage variants. Techniques of this kind are called *variability techniques*. The following list of techniques is reworked from the list provided in [12]. The list divided between design–time and run–time techniques. The following variability techniques are specific to design–time variability:

1) Patching of existing processes in order to support specific conditions, must be facilitated. This technique is related to arbitrary modifications which were not or could not be anticipated at design time.

2) Blueprinting or design from template, indicates the possibility to define a template and then extend it with

variation expressions.

3) Process inheritance is a special form of template reuse, where all variants are derived from the template process by extension, specification or substitution [13].

4) Late modelling requires that parts of the template process, called *placeholders*, may be left unspecified.

The following techniques are specific to run–time variability:

1) Adhoc changes requires the possibility to make a variation for an executing instance of a variant.

2) Runtime settings requires the possibility of changing the execution of a set of variants based on the setting of some preferences.

3) Late binding is closely related to *Late modelling* with the only difference being that it is done at run–time.

4) Run-time case selection: processes may allow the expression of non–deterministic choices. The variation management tool should allow the expression of cases, but also their run–time execution (e.g., by random choice or prompting the user).

III. Case Study: Variability in eGovernment

In the Netherlands there are 441 municipalities that have to implement the same national laws, though, they are different in size, business models, IT infrastructures and so on. Recently the WMO law (Wet maatschappelijke ondersteuning, Social Support Act, 2007) was approved that mandates, for instance, the rules for providing publicly subsidized wheel chairs to needing citizens. There are two roads to manage the translation from national law to "instance of giving out a wheel chair in municipality X to citizen Y." [7]. One way is to give the interpretation document and let each municipality to implement the law, as it is done today with no reuse. The other possibility is to provide a formalized and generic process including variability describing the law and let the municipalities to customize it according to their organizational and IT structure.

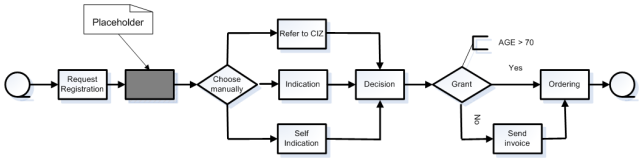

Fig. 2. Generic process for obtaining a subsidized wheel chair.

Figure 2 illustrates the second. From left to right, we have the initial activity of registering for obtaining a subsidized wheelchair. Next there is a place–holder for amongst others a "Home Visit" activity where an expert from the municipality visits the home of the requesting actor in order to asses the situation. Then there is a gate requiring a choice being made by an authorized civil servant. Based on this choice three further activities are possible. These three activities all cover some sort of indication, an assessment of the requesting party on their rights of getting the wheelchair. The three options (from the bottom up) cover the options of doing the indication themselves as the municipality, having an indication from for example a doctor, and having a national third party known as the CIZ (Centrum indicatiestelling zorg) handling the indication. These three options then rejoin in an accepting the request activity. Now a further check is performed on the age of the requesting party. At the end, one will receive a wheelchair having to pay for it himself or having it subsidized. Furthermore, in one municipality the minimum age for getting a wheelchair may be fixed at say 70, and in another one it may depend not only on the age.

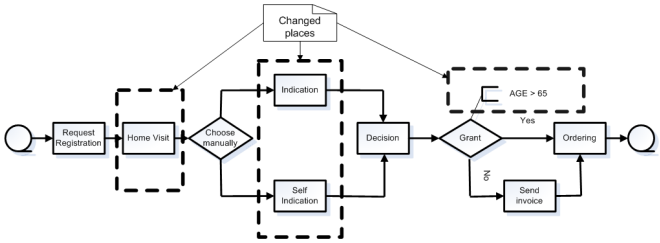

Fig. 3. Case study. A variant.

Now consider the process of Figure 2 to be a generic process. Departing from the precess of this generic process, one can derive a variant that explicitly deals with the wishes of various municipalities. A possible variant is shown in Figure 3. The changed regions are outlined with dotted rectangles. In the resulting variant of Figure 3 the choice has been made to include the "Home Visit" activity. One could imagine the reason for this being that most cases in the municipality are not well known beforehand and hence there is a need in home visit in order to asses the situation. The second variation made is one in the choice for the indication. In the Netherlands it is possible for a municipality to outsource the indication to an organisation known as the CIZ. This however is optional and the municipality of Figure 3 has chosen not to use their expertise. The last change was made to the age requirement, which was decreased from 70 to 65.

IV. Requirements

The case of eGovernment is not an isolated one, in many domains one wants to describe a generic process for a given service or product and then manage a possibly infinite number of variants. But what are the common traits of processes with explicit variability management? Next we provide a systematization of these features in the form of requirements. We organize the requirements in four categories: (A) those that deal with the expressive power for expressing variability, (B) requirements for service based processes with variability, (C) run–time requirements connected with consistency and fault handling, and (D) requirements stemming from the need of managing evolution of processes with variability.

A. Expressive Power

Explicit variability management means the ability to express the possible variations. The *expressive power* requirements

provide an indication of what must be possible to express for a variation in a process.

1) Structural variation: a variability system must be able to express atomic structural changes of the reference business process. Compiled from [6], the following changes are the most common ones and should be self–explanatory: (a) *Insert Process Fragment*, (b) *Delete Process Fragment*, (c) *Move Process Fragment*, (d) *Replace Process Fragment*, (e) *Swap Process Fragment*, (f) *Copy Process Fragment*, (g) *Extract Sub–process*, (h) *Inline Sub–process*, (i) *Embed Process Fragment in Loop*, (j) *Parallelize Activities*, (k) *Embed Process Fragment in Conditional Branch*, (l) *Add Control Dependency*, (m) *Remove Control Dependency*, and (n) *Update Condition*.

For example, in Figure 3 one process fragment is deleted (1b), one added (1a) , and one condition updated (1n) in order to move from the reference process to the variant. A variability management framework should allow the process designer to express the above imperative structural changes, possibly via graphic tools.

2) Constraint expressions: consists of expressing variations by declaring the borders which limit the possible process modifications. Unlike structural changes which indicate imperatively what can vary, a constraint one limits the borders of changes explicitly. Most common constraint expressions are the following ones (extended from [10], [11]): *Selection constraints:* force the selection of activities during the execution of the variation. Most notable examples are: (a) *Mandatory*: specify which activities which must be included; (b) *Prohibitive*: specify which activities which must never be included; (c) *Prerequisite*: specify the dependency between two specific activities, such as, if activity A is included then activity B must also be included in the process; (d) *Exclusion*: specify the dependency between two specific activities, such as, if activity A is included then activity B must not be included into the process; (e) *Substitution*: specify the implied substitution, such as, if activity A is not included then activity B must be included; (f) *Corequisite*: specify the tight relation between two specific activities, such as, they both must be either included or excluded; and (g) *Exclusive–Choice*: specify two activities as mutually exclusive.

Scheduling constraints: force the temporal relation among the execution of activities of the process. Most notably: (h) *Mandatory*: specify that a specific activity must be executed; (i) *Order*: specify that two specific activities must be executed in certain order; (j) *Parallel*: specify that two specific activities must be executed in parallel only; (k) *Exclusive–Execution*: specify that two activities cannot be executed in the same process; and (l) *Repetition*: specify how many times a specific activity might be executed (min, max).

For example, a local business rule stating that the process in Figure 2 must include either a home visit or a referral to the CIZ is expressed by a constraint stating that there is a bilateral *substitution (e)* relationship between the "Home Visit" and "Referral to CIZ" activities, that is, at least one of those activities must be included into the result process.

3) Variation Relations: requires a tool to be able to express the dependencies between activities and variations as well as projection of such dependencies into domain requirements. For example, there might be a package option called "Small city" which means the removal of "Referral to the CIZ" and "Indication" activities; but this option is only available when the "Home visit" activity is included.

B. Service Requirements

BPM and service–orientation are becoming more and more synergistic in IT infrastructure. Thus, some requirements for variability are specific of service based systems.

4) Late Service Discovery by indicating interfaces or behaviour of activities to be bound to services must be supported. One can think of using registries or flooding techniques to discover the appropriate variant at run–time in a service-oriented architecture.

5) Variable Quality of Service: one could have variants of a process whose only difference is the quality of execution, e.g., the process of reserving a seat for a gold member of an airline company executes faster than that of an unregistered client.

C. Consistency & Fault Handling

A variability management framework should not only provide the modelling facilities, but also support for the run-time of process variants. Next are a set of requirements that affect the run–time.

6) Business Rules are basically constraints (Requirement 2) which are used within imperative–based framework. Such constraints help to check for the soundness of the variants as well as to introduce complementary limitations which come from law or business requirements.

7) Unreachable States must not be introduced by a variability expression. The variability should never create paths in such a way that states can not be reached. A variation which changes the guard of the process in Figure 2 to "AGE $<$ 0" should be refuted.

8) Variable Fault Handlers should be able to be designed with and attached to different variants. Fault handling should react differently with different variants. Each variation should therefore include its own fault handling mechanism.

9) Variable Roll Back Procedures should be able to be designed with and attached to different variants. Each variation should include its own roll back mechanism which is called when the variation was included and the resulting variant needs to roll back.

10) Data Flow must not be broken by variability. This means all modifications made to business process must be apprised according to the data flow in this process.

D. Evolution Requirements

From the variability management point of view, a set of changes made to a business processes is an *evolution* if such changes are (i) permanent and (ii) must be propagated to all variants. This translates into the following requirements.

11) Variants must be updated when changes to the reference process or rules (for example through law) have been made. For example, consider a change made to the reference process of Figure 2 where the option of the "Referral to CIZ" must be formally considered for each case by law. This change must then be propagated to the variant shown in Figure 3 in such a way that this task is enforced and must be included.
12) Running instances of variants should be updated when changes to the reference process or rules (for example through law) have been made. All changes to the reference process or rules should be propagated to all running instances of the modified variants. Consider again the case where the "Referral to CIZ" must be formally considered for each case. Now also assume that there exists a running instance of the variant shown in Figure 3. Such running instances should also be adopted in such a way that it includes the "Referral to CIZ" activity if possible.
13) Updates made to variants must be apprised with respect to the reference process and business rules.

V. Tools & Frameworks

Let us now turn to existing tools and frameworks proposed in the past to address explicitly issues of variability, mostly in connection to BPM. We look at these through the just provided lenses of the requirements of Section IV. Table I summarizes the satisfaction or not of a requirement by a given tool. The table is divided into tools that allow the expression of variability in an imperative way and those that use a declarative one. First, we introduce tools and frameworks and then we discuss their relation to the presented variability requirements.

Provop [14], or Process Variants by Options, is a framework designed by he researchers at the University of Ulm and Daimler AG. Developed to address issues in the domains of automotive and healthcare, it introduces variability through defining options (variations) which modify a generic process. These options can either modify the generic process through the use of element identifiers (tasks, gates, etc) or so called adjustment points, which are included in the generic process. It also includes the option to declare relationship between different adjustment points.

Variability extension to Business Process Execution Language (VxBPEL) [15] is an extension to the BPEL standard [16] that we proposed previously. It introduces a number of new keywords into BPEL that allow for variability to be included within one of its processes. VxBPEL is based upon the COVAMOF [4] variability modelling framework and as such allows the inclusion of information about variation points, variations, and realization relations into BPEL.

The ADEPT [17] project provides a framework for handling of different kinds of run–time variability. This framework has a strong support for the propagation of manual changes, thus providing evolutionary support.

The configurable workflow models [18] utilizes a blueprinting (design from template) technique to handle variability. It also supports a link between requirements and variations as well as dependencies between individual variations. As a result, links provide a flexible model which allows to create automatically an appropriate business process modification based on the system requirements.

Imperative					Declarative		
Req	Provop	VxBPEL	ADEPT	CWM	Req	DECLARE	BPCN & PVR
Expressive Power							
1a	+	+	+	+	2a	–	+
1b	+	+	+	+	2b	–	+
1c	+	–	+	+	2c	–	+
1d	+	+	+	+	2d	–	+
1e	–	–	+	–	2e	–	+
1f	–	–	–	–	2f	–	+
1g	–	–	–	–	2g	–	+
1h	–	–	–	–	2h	+	–
1i	–	–	–	–	2i	+	+
1j	–	–	–	–	2j	+	+
1k	+	–	–	+	2k	+	–
1l	–	–	–	–	2l	+	–
1m	–	–	–	–			
1n	+	–	–	+			
3	+	+	–	+	3	–	±
Service Requirements							
4	–	+	–	–	4	–	–
5	–	–	–	–	5	–	+
Consistency & Fault Handling							
6	+	–	–	+	6	+	+
7	–	–	+	+	7	+	–
8	–	+	–	–	8	–	–
9	–	+	–	–	9	–	–
10	–	–	+	+	10	–	–
Evolution							
11	–	–	–	–	11	–	+
12	–	–	+	–	12	+	–
13	+	–	–	+	13	+	+

TABLE I
Comparing tools and framework on the basis of the identified Variability requirements.

An example of declarative approach is the DECLARE framework [11], which provides variability via a special way of process definition. Instead of introducing variation points, a process is defined as a set of activities linked by constraints. These constraints act as a flexible process definition, which allow users to follow the process in an arbitrary way as long as it does not violate the process constraints.

Business Process Constraint Network (BPCN) and Process Variant Repository (PVR) [19], [10] are another examples of declarative approach. Together they define a set of activities and a set of constraints over activities that force the process to behave in a certain way.

Consider Table I, where the mentioned frameworks are represented as columns and where rows are the requirements for explicit variability management proposed in Section IV. Using the information provided in the published articles and available white papers for the tools and frameworks, we evaluate their satisfaction or non-satisfaction of the requirements. The table is divided in two parts: The left half lists the imperative–

based tools whereas the right half lists the declarative–based tools. This division affects the first section of the table, where the expressive power has a different meaning depending on which approach was used. Therefore the imperative–based tools include only the expressive power regarding to change from requirement one, and the declarative–based tools only include the constraints–based expressive power requirements. Both halves do include the third expressive power requirement describing variation relations. A plus indicates the satisfaction of the requirement, the minus the non–satisfaction and the plus/minus together indicate a borderline case.

First, we remark that none of the imperative approaches covers all the requirements of "Expressive Power" proposed by [6]. On the declarative side, the situation is similar: DECLARE lacks selection constraints (which are related to design–time and therefore not needed for a framework oriented on run–time customizations). Instead, this tool has richer scheduling constraints which makes it more powerful in terms of run–time customizations. BPCN relies on selection constraints, making it more suitable for design–time variability.

In general, both DECLARE and BPCN have better support with regard to evolution requirements than the imperative–based competitors. This is mainly because of the nature of the declarative approach which is initially more flexible and allows to sustain the link between the reference process and its variants via constraints validation. The variation relation (Req 3) is also a weak point of declarative based frameworks. This can be explained with the absence of explicitly defined variability points. In must be noted, though, that selection constraints might work as a substitution for variation relations.

The handling of data flow is neglected in most cases, and only ADEPT and CWM have means to sustain data flow consistency. Variable fault handling and rollback processing was specified only for VxBPEL. In contrast, many tools support the detection of unreachable states and rules application. Service–specific requirements are also not widely adopted. Only VxBPEL provides an implicit service discovery, since WS-BPEL does support late service discovery. As for variable QoS, only BPCN does support it by taking into account QoS attributes while searching business process variants. Finally, evolution is a difficult task to handle, though declarative ones have better possibilities. For example, change propagation can be done via constraint validation, and validation of manual changes naturally fits constraint–based process.

VI. Concluding remarks

Explicit variability management is becoming an increasing need in BPM, especially with the shift towards Service–Oriented Architectures. A number of tools and frameworks have been proposed that address some of the issues. Considering these past experiences, we define and classify a fundamental set of requirements of such systems. In this way, we notice that none of the existing frameworks address all or even most of the requirements, thus leaving space for extensive research and development in the area of frameworks for the explicit management of variability.

Acknowledgements

The research is supported by the NWO **SaS-LeG** project, http://www.sas-leg.net, contract no. 638.000.000.07N07. We thank the anonymous reviewers for their suggestions and P. Avgeriou, N. van Beest, F. van Blommestein, A. Lazovik, D. Tofan, and H. Wortmann for fruitful discussion.

References

[1] W. Bandara, M. Indulska, S. Sadiq, and S. Chong, "Major issues in business process management: an expert perspective," in *European Conference on Information Systems (ECIS)*, 2007.

[2] W. van der Aalst and S. Jablonski, "Dealing with workflow change: Identification of issues and solutions," *International Journal of Computer Systems, Science, and Engineering*, vol. 15(5), pp. 267–276, 2000.

[3] S. Rinderle, M. Reichert, and P. Dadam, "Correctness criteria for dynamic changes in workflow systems: A survey," *Data and Knowledge Engineering*, vol. 50(1), pp. 9–34, 2004.

[4] M. Sinnema, S. Deelstra, and P. Hoekstra, "The covamof derivation process," in *ICSR*, 2006, pp. 101–114.

[5] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "Modeling dependencies in product families with covamof," in *IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS '06)*. IEEE Computer Society, 2006, pp. 299–307.

[6] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - enhancing flexibility in process-aware information systems," *Data Knowl. Eng.*, vol. 66, no. 3, pp. 438–466, 2008.

[7] C. Sun, R. Rossing, M. Sinnema, P. Bulanov, and M. Aiello, "Modelling and managing the variability of web service-based systems," *Journal of Systems and Software, Elsevier*, vol. 83, pp. 502–516, 2010.

[8] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14(3), pp. 5–51, July 2003.

[9] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. M. P. van der Aalst, "Process flexibility: A survey of contemporary approaches," in *CIAO! / EOMAS*, ser. LNBIP, J. L. G. Dietz, A. Albani, and J. Barjis, Eds., vol. 10. Springer, 2008, pp. 16–30.

[10] S. W. Sadiq, M. E. Orlowska, and W. Sadiq, "Specification and validation of process constraints for flexible workflows," *Inf. Syst.*, vol. 30, no. 5, pp. 349–378, 2005.

[11] M. Pesic, M. H. Schonenberg, N. Sidorova, and W. M. P. van der Aalst, "Constraint-based workflow models: Change made easy," in *OTM Conferences (1)*, 2007, pp. 77–94.

[12] S. Balko, A. H. M. ter Hofstede, A. P. Barros, and M. L. Rosa, "Controlled flexibility and lifecycle management of business processes through extensibility," in *EMISA*, 2009, pp. 97–110.

[13] W. M. P. van der Aalst and T. Basten, "Inheritance of workflows: an approach to tackling problems related to change," *Theor. Comput. Sci.*, vol. 270, no. 1-2, pp. 125–203, 2002.

[14] A. Hallerbach, T. Bauer, and M. Reichert, "Managing process variants in the process life cycle," in *ICEIS (3-2)*, 2008, pp. 154–161.

[15] C. Sun and M. Aiello, "Towards variable service compositions using VxBPEL," in *ICSR*, 2008, pp. 257–261.

[16] Oasis, "Web services business process execution language version 2.0," Oasis, Tech. Rep., April 2007.

[17] P. Dadam and M. Reichert, "The adept project: a decade of research and development for robust and flexible process support," *Computer Science - R&D*, vol. 23, no. 2, pp. 81–97, 2009.

[18] F. Gottschalk, W. M. P. van der Aalst, M. H. Jansen-Vullers, and M. L. Rosa, "Configurable workflow models," *Int. J. Cooperative Inf. Syst.*, vol. 17, no. 2, pp. 177–221, 2008.

[19] R. Lu, S. Sadiq, and G. Governatori, "On managing business processes variants," *Data Knowl. Eng.*, vol. 68, no. 7, pp. 642–664, 2009.