Controlled Flexibility and Lifecycle Management of Business Processes through Extensibility

Sören Balko[1], Arthur H.M. ter Hofstede[2], Alistair Barros[3], Marcello La Rosa[2], and Michael Adams[2]

[1] SAP AG, Walldorf, Germany
Soeren.Balko@sap.com
[2] Queensland University of Technology, Brisbane, Australia
{a.terhofstede,m.larosa,mj.adams}@qut.edu.au
[3] SAP Research, Brisbane, Australia
Alistair.Barros@sap.com

Abstract. Vendors provide reference process models as consolidated, off-the-shelf solutions to capture best practices in a given industry domain. Customers can then adapt these models to suit their specific requirements. Traditional process flexibility approaches facilitate this operation, but do not fully address it as they do not sufficiently take controlled change guided by vendors' reference models into account. This tension between the customer's freedom of adapting reference models, and the ability to incorporate with relatively low effort vendor-initiated reference model changes, thus needs to be carefully balanced. This paper introduces *process extensibility* as a new paradigm for customizing reference processes and managing their evolution over time. Process extensibility mandates a clear recognition of the different responsibilities and interests of reference model vendors and consumers, and is concerned with keeping the effort of customer-side reference model adaptations low while allowing sufficient room for model change.

1 Introduction

In many industries, a company's environment, such as customer demand, technological innovations and regulatory conditions tend to change frequently and sometimes rapidly. By being able to flexibly adapt their processes to changes, agile businesses set themselves apart from their competitors. Naturally, Business process management suites (BPMS) offerings need to facilitate flexibility at low costs. At the same time, companies still wish to benefit from standardized best practices, represented through vendor-provided reference processes. The business process community has come up with numerous flexibility techniques to incorporate change into business processes, e.g. [RA07,GAJVL08,RRKD05,AWG05,AHEA06,EKR95]. These approaches cover both design time and runtime changes and provide formal frameworks for how to constrain changes. However, many established process flexibility approaches suffer from shortcomings with respect to process lifecycle management in general, and to the costs associated with changing business processes, specifically. Some techniques propose that BPMS customers alter reference processes "in place" in order to customize them to their

needs (*patching* use-case) [FLZ06]. Others suggest to use reference processes merely as templates for developing company-specific processes (*blueprinting* use-case) [SN00].

Neither of these approaches can realistically succeed in large-scale software roll-outs, involving hundreds of reference processes with an even higher number of customer adaptations on top. This is because making changes to reference processes goes along with substantial costs for carrying out these changes and later maintaining the resulting processes. Whenever a BPMS vendor ships a new reference process version to incorporate corrections or to address new requirements, existing customer adaptations have to be re-applied at great cost. Similarly, multiple independently defined adaptations have to be consolidated within a single process, for instance when two customer departments change a cross-departmental reference process independently at different points in time.

This paper introduces the concept of *process extensibility* as a new paradigm for customizing reference processes and managing their evolution over time. Process Extensibility mandates a clear recognition of the different responsibilities and interests of reference model vendors and reference model consumers, and is concerned with keeping the effort of reference model adaptations at the customer side low while allowing sufficient room for model adaptation. BPMS vendors *own* (i.e. define and maintain) reference process models, while BPMS customers own and run extensions thereof. These extensions constitute separate customer-defined "delta improvements" which hook up to a reference process through late binding mechanisms. When adhering to some plain compatibility rules, both reference processes and extensions can be patched (i.e. maintained) by their respective owners without ever having to be "re-wired". The vendor remains in the "driver seat" to update reference content, letting customers easily benefit from state-of-the-art best practices. By automatically applying existing customer extensions to patched referenced processes, the cost of rolling out new BPMS releases is greatly reduced.

The rest of this paper is organized as follows. Section 2 outlines the extensibility approach and its benefits over existing flexibility approaches. Next, Section 3 provides a taxonomy to classify flexibility approaches. Section 4 identifies extensibility patterns that occur along a process' control flow and data perspective, while Section 5 sets an agenda for future research in this area. The paper concludes with a section on related work and a summary.

2 Extensibility

The general concept of making processes more flexible by allowing deviation from their hard-wired business semantics has been around for some time. Requirements like customization, exception handling, re-use, etc. have led to different technological approaches, namely *From-Scratch Design*, *Patching*, *Blueprinting*, *Ad-Hoc Changing*, and *Runtime Settings*.

2.1 Proposal

Extensibility is a new approach to support process flexibility which specifically addresses customization of reference content. Unlike existing approaches, extensibility

clearly designates responsibilities for the process and extensions thereof. Reference processes may be patched (bugfixed, updated) by the vendor only. Customers receive reference processes as read-only shipped content which is only updated as part of a software release.

Customers then customize reference processes to their needs by independently defining and deploying extensions which solely constitute "deltas" (process fragments). Extensions exist alongside the (reference) processes. At runtime, an extensibility framework dynamically invokes the defined extension(s) for a process. Multiple extensions to a single process can be independently defined (e.g. by different customer departments) to be deployed in isolation (i.e. at different points in time). As the extensibility framework automatically controls the interplay between multiple different extensions and their target (reference) process, there is no need to statically integrate all extensions upfront.

Both reference processes and extensions can be "patched", thereby spawning new versions. Patches to a reference process should adhere to some compatibility rules that allow the new version to support, wherever possible, existing customer extensions. The extensibility framework takes care of automatically incorporating all customer extensions that were defined atop any old version of the patched reference process. Similarly, extensions need to follow some compatibility rules. These rules constrain to what extent the business semantics of a reference process can be deviated from. Apart from "safe" implicit compatibility rules, the BPMS vendor may define more relaxed explicit constraints.

Figure 1 illustrates a vendor-shipped reference process (left) that is customized with extensions of the customer's HR and Sales departments. The initial reference process is a sequence of three activities: "HR Task", "Sales Task", and "ERP Service". The first extension replaces "HR Task" with a subflow comprising the existing "HR Task" followed by some organizational chart lookup ("OrgChart Service").

As part of a new release, the vendor ships a patched reference process that conditionally performs an automated "CRM Service" instead of the (manual) "Sales Task". The patched reference process is compatible with any extensions defined on its predecessor version. The extension framework needs to automatically route the patched reference process to existing extensions, where applicable (here "HR Task" → "HR Extension"). Independently to the vendor shipment, the customer may have replaced the manual "Sales Task" with an automated "Sales Service". The earlier defined "HR Extension" was also refined to introduce a four-eyes principle. That said, patches may be applied to both the original reference process and a customer extension.

Extensibility is a prerequisite for proper process lifecycle management where the reference content vendor and the customer represent distinct parties having different requirements and obligations:

Vendor The vendor is responsible for (1) delivering correct reference processes ("shipped content") that represent generalized best practices. He also needs to (2) maintain that content, i.e. ship patches when bugs are detected or requirements change. Finally, (3) the vendor should provide the means to have its content "customized" to a customer's needs. From his perspective, it is vital to ensure that reference process change is controlled.

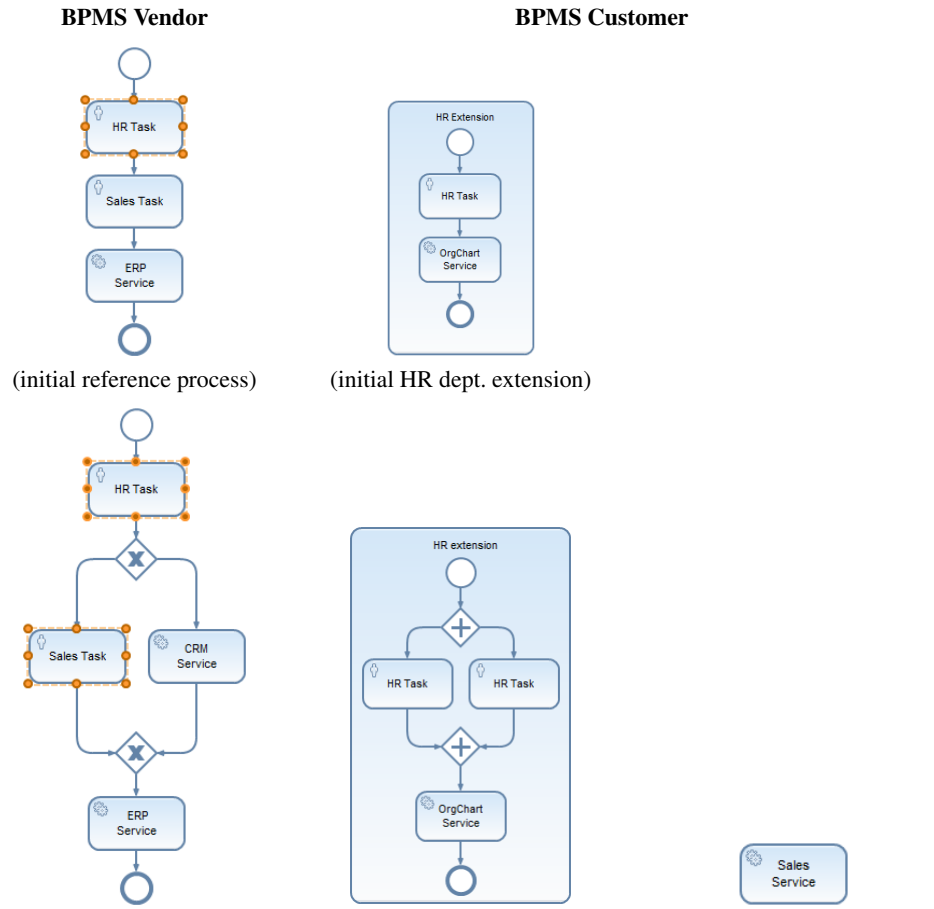

Fig. 1. Extensibility Example

Customer Customers engage their IT team to customize a BPMS release (they may also hire contractors to do so). In regard to reference processes, that includes (1) changing settings which deviate in the customer's landscape, (2) reducing complexity by removing functionality that is not needed, and (3) adding new functionality for requirements which are not yet covered.
End users essentially run processes (i.e. start new instances or are involved in process activities). There are often multiple end user roles that (1) interact with the same process but (2) have their distinct customization requirements. For instance, a legal department could ask for fine-grained logging within audit-sensitive processes, whereas the IT department may be interested in getting notified of technical process failures.

Extensibility offers *controlled flexibility* for the different parties that design, customize, and run processes and is motivated by the specific concerns these parties typically have.

For instance, the vendor must be able to easily patch shipped content without introducing extra, per-customer development costs or significantly increasing the cost of ownership at the customer. Last but not least, the vendor will want to disallow arbitrary changes to this content to avoid mistakes on the customer that which are very difficult to support. In turn, customers are essentially concerned with running their businesses while keeping IT costs down. While flexibility does have its merits, customers also want to build their business on best practices. Besides, customers have a vital interest in correct, law-conforming processes where customizations are guaranteed not to distort the basic functionality.

2.2 Benefits

Technically, the vendor ships reference processes that incorporate "extension points" which are pre-planned artifacts where customers can incorporate their extensions. Extension points apply to almost any dimension of modeling business processes, including control flow, data flow, resources, rules, security, etc. We will give extensibility examples for some perspectives below. Extensibility comes with a number of significant advantages over existing flexibility approaches, notably:

Extensibility (1) offers a lifecycle model for controlled flexibility taking into account obligations and concerns of different parties involved in designing, customizing and using business processes. It helps avoiding errors at the customer side and reduces maintenance costs (*Controlled Change*). Customers automatically (2) benefit from best practices within shipped reference processes. In particular, the vendor can set extension points in a way that the basic business objective of the reference process cannot be tampered with by customer-defined extensions (*Best Practices Adoption*). Reference processes may be (3) subject to patching. Extensions defined on an old version of some process transparently apply to any new version. Reference processes can thus be fixed without losing (or having to manually re-apply) their extensions (*Supportability*).

Instead of using reference processes as templates for newly created processes, (4) extensions consume fewer resources at runtime. This is because an extension solely constitutes a small "delta". As a side effect, this model is ideally suited for process outsourcing where reference processes are remotely run at *SaaS* providers (*Resource Consumption*). Extensibility allows multiple people (at the customer side) to (5) independently define "additive" extensions to the same reference process. This greatly improves separation of concerns between different business departments. As a result, multiple extensions can be independently defined at different points in time (*Multiple Extensions*).

If desired, vendors may (6) ship their processes as "black boxes", only exposing interfaces and extension points. This may be desirable if details in the reference content constitute significant intellectual property which is not to be disclosed (*Intellectual Property*). Reference processes may also be purely documentary models that are not executed in a proper BPMS runtime but rather as a coded application. The customer (7) may still want to extend these "application processes" with proper process models. With some application instrumentation to add extension points, extensibility may even help in bridging these platform and paradigm differences (*Application Extension*).

Finally, the (8) meta-process of defining extensions is of interest itself, as it reveals how a customer deals with business change. Mining the logs of a meta-process could help the customer optimizing its business by getting answers to questions like: Which line of business is most often subject to change? Which user roles require most change to reference processes? (*Flexibility Mining*)

3 Taxonomy

Common process flexibility approaches can be classified with respect to a number of dimensions, the most important being (1) the *primary use-case* which outlines the main purpose and most frequent usage, (2) the *parties* (vendor, customizer, end user) that are affected, (3) the functional *role* descriptions of each participant, (4) the *lifecycle* stages (design time, runtime) of the process, (5) the *constraints* that restrict what can be done, and (6) the *scope* (process type, instance, version) within which the flexibility technique operates. Existing flexibility techniques can be classified with this taxonomy, which helps in understanding their differences. It also outlines the contribution of extensibility to the overall picture. We specifically discuss the differences between *from-scratch modeling* of new processes, *patching* existing processes, re-using a vendor-provided template to develop a new business process (*blueprinting*), performing *ad-hoc changes* of process instances at runtime, modifying (technical) *runtime settings*, and *extending* reference processes:

From-Scratch Modeling Modeling a business process "from scratch" is typically the result of analyzing and documenting existing processes. Most importantly, there is no pre-existing reference process to build upon. Instead, a new process is modeled and then successively refined, following a top-down approach. Bottom-up approaches start with modeling detailed process fragments which are later aggregated into larger end-to-end business processes.

Strictly speaking, this approach traditionally does not constitute a flexibility use-case. However, modeling a (reference) process from scratch is a prerequisite for any other flexibility technique. It is usually business analysts who start modeling from scratch. Both the vendor and its customers may perform this use-case (for reference processes and customer processes, respectively). Newly modeled processes are not subject to any constraints, except for the inherent restrictions of the chosen modeling standard.

Patching Occasionally, process models that have been previously deployed to a BPMS runtime engine, may need to be altered. The vendor may have to patch reference processes to fix bugs or simply to address new requirements. Customers may want to patch their processes to incorporate various changes in their business. Patching is closely related to versioning where the affected process will be labeled with a new version number.

Both IT (process developer) and business (domain expert) users may want to patch a process. Patching is a design time operation but will only take effect after deploying the patched process version into the BPMS runtime engine. There are some constraints that limit what can be changed when patching a process. Firstly, interface compatibility must be preserved such that client processes do not have to be

adapted to cope with the change. Secondly, existing extension points must be retained in the patched version such that extensions transparently apply to the patch.

Blueprinting Vendor-delivered reference processes often constitute best practices rather than ready-to-run processes. Blueprinting uses reference processes as a "master" for newly modeled processes. Technically, the reference process is physically copied to a blank process model where it is further refined. While being fully flexible in what changes can be done from there on, BPMS vendors will not be able to support those changes. That is, customers will have to manually apply all changes in a new reference process version in their derived processes (copies). Altogether, blueprinting is a design time operation where customers adapt vendor-delivered reference processes to their needs (as opposed to extensibility which relies on late binding mechanisms). Unlike patching, customers perform modifications on physically separate copies of the template and rather create new variations that are independent from (and do not overwrite) the original process.

Ad-hoc Changing Sometimes, end users have to deviate from the behavior of the process instances they are involved in. Actually, human-driven processes often run into exceptional situations. End users then need to (implicitly) alter the process model for their specific instance, thus deviating from its original business semantics.

There are some constraints around ad-hoc changes, mostly affected with role-related restrictions and instance migration issues. That is, ad-hoc changes alter the models of running process instances. Consequently, ad-hoc changes must allow for automatically migrating the instance state to the altered model. Typically ad-hoc changes affect only a single process instance. The altered process model is kept temporarily, i.e. for the life time of that instance.

Runtime Settings Some environmental settings globally hold for all processes and, when changed, need to immediately apply to both all running processes and newly started instances. Those settings include modifications to organizational charts, security settings and other technical configurations. In most cases, these settings are not even part of any process model such that there is essentially no design time aspect here. Those changes are typically done by system administrators.

Extensibility constitutes a separate flexibility approach where customers define process extensions as deltas (process fragments) on top of reference processes. The primary use-case behind extensibility is customization where the customer adapts a given reference process to its business needs. Various customer roles may define process extensions, each with different objectives. Domain experts from specific organizational lines (e.g. Sales, Procurement, Manufacturing, etc.) may independently define extensions to adjust a cross-organizational process to their needs. A customer typically defines extensions in a design time environment, even though that does not rule out the option of having a runtime user interface to let end users specify extensions in an ad-hoc fashion. Extensibility is subject to some constraints, either originating from implicit compatibility rules or explicitly from modelled extension points within reference processes. Table 1 classifies existing flexibility techniques according to the dimensions introduced and positions extensibility as a new approach.

	Use-Case	Party	Role	Lifecycle	Constraints	Scope
Designing from Scratch	Business process analysis	Vendor, Customer	Business analyst Process architect	Design Time	–	new process
Patching	Changing requirements, Bugfixing	Vendor, Customer	Process developer, Domain expert	Design Time	Extension/ interface compatibility	new version
Blueprinting	Customization, Adoption of best practices	Customer (Vendor)	Process developer, Domain expert	Design Time	–	new process
Ad-Hoc Changing	Handling of exceptional cases	Customer	Task owner, Process administrator	Runtime	Instance migration, Role	single instance
Runtime Settings	System-wide settings	Customer	System administrator	Runtime	–	all running instances
Extensi-bility	Customization, Adoption of best practices	Customer	Domain expert, IT department	Design Time (Runtime)	Extension point	all future versions

Table 1. Process Flexibility Taxonomy

4 Extensibility Patterns

Conceptually, extensibility is open to different process perspectives. This section identifies some frequent extensibility patterns in the control flow and data flow perspectives. Without loss of generality, we use a BPMN-like notation to illustrate these use-cases.

4.1 Control Flow Perspective

Many extensibility use-cases do in some way alter the control flow by adding or replacing process fragments by customer extensions. Extensions may also skip or even re-arrange existing reference process branches. Multiple variants exist, most notably for how to spawn (conditionally, (a)synchronously, etc.) and merge back extension flow (with or without synchronization).

In this paper, we solely consider *Usage Extensibility* which is the most straightforward way of creating control flow extensions. Usage extensibility applies to activities, denoting atomic tasks (either performed automatically or by a human actor) or referencing nested subflows. The idea is to have an extension *replacing* an activity A of the reference flow by another activity A'. Technically, the to-be-replaced and replacing activities A and A' need to expose compatible interfaces (for the data flow) to have the extensibility framework seamlessly perform the replacement without human intervention at runtime.

Figure 2 depicts a "Make to Order" reference process derived from a public SAP *Solution Composer*[4] business scenario map. Make to Order specifies a vendor-side pro-

[4] http://www.sap.com/solutions/businessmaps/composer/index.epx

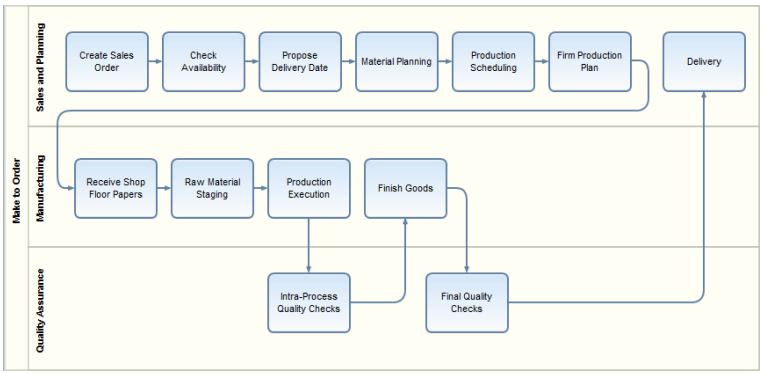

Fig. 2. "Make to Order" Reference Process

cess in discrete industries where a good is manufactured upon an incoming order from a customer. On the vendor side, activities are performed by three different roles: (1) sales department, (2) manufacturing, and (3) quality assurance. After negotiating delivery dates and completing the production planning, manufacturing ultimately produces the good with interleaved quality checks for the production process and final checks for the good itself. At customization, this process is extended to optionally modify those quality gates depending on the order volume. That is, for high-volume orders a four-eyes quality check applies as part of the final checks. For this purpose, the extension replaces the "Final Quality Checks" task by the subflow depicted in Figure 3 (left).

Usage Extensibility captures a wide range of customization use-cases and can be applied in a straightforward way. In fact, by substituting atomic activities through subflows, it allows the incorporation of structurally complex customer extensions into reference flows.

4.2 Data Flow Perspective

Unlike control flow, data flow is implicitly incorporated into process models. It affects the process' data context, activity interfaces, data mappings, decision gateways, and message correlations. One frequently observed requirement revolves around *Field Extensibility* which deals with (compatibly) complementing data interfaces both from a service provisioning and consumption perspective. That is, customers may wish to customize the reference process in a way that it receives (passes on) additional parameters from inbound (outbound) messages (services). New clients may interact with the process through the field-extended interface. In turn, compatibility to existing clients (provisioning) and services (consumption) must be preserved.

Figure 3 (right) depicts a plain BPMN flow where the start/end events represent the inbound case, providing the process as a service has a well-defined interface. A new process instance is spawned upon receiving an inbound "request" message on that interface. In turn, the end event terminates the instance and crafts the corresponding outbound "response" message. When compatibly extending that interface to accommodate additional fields, clients (including "parent" processes) may pass on extra data to

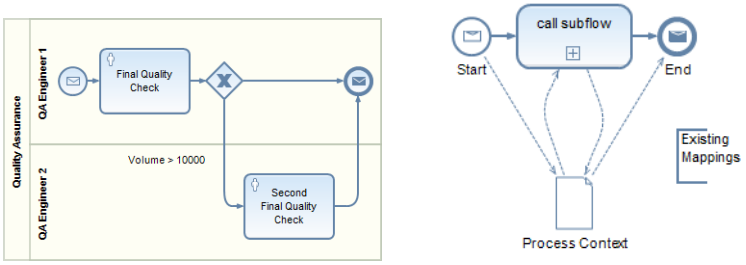

Fig. 3. Usage Extensibility (left) and Field Extensibility (right) Examples

the process. The process may then make use of this data in usage-extended activities. Existing clients remain unaffected, thus, passing (receiving) their inbound (outbound) messages to (from) an extensibility framework which adds (strips off) the extra fields.

Vice versa, the subflow activity constitutes the consumption case where the activity's interface may be field extended in the same manner. Altogether, Field Extensibility is concerned with preserving compatibility despite interface changes. When used in isolation, it does not specify the means to take advantage of additional data fields.

5 Open Research Challenges

In this paper, we introduce the idea of process extensibility but do not yet cover the whole topic exhaustively. In fact, we believe extensibility constitutes a whole new area of BPM research. In this section, we present a research agenda that gives indications for future research on conceptual and technical follow-up topics. Most topics revolve around (1) fully understanding the applicability and limitations of process extensibility and (2) laying its formal and technical foundations:

Extensibility Patterns To set the scene for follow-up research, it is important to gain a comprehensive overview on relevant extensibility use-cases. These use-cases should preferably constitute real-world customization requirements which need to be classified and mapped to extensibility patterns.

Reference Process Conformance Extensions alter the behavior of reference processes which are, in turn, supposed to represent best practices. It is thus necessary to preserve some core characteristics of an extended process. Future work in this area could result in an explicit constraint model for defining extensions for reference processes.

Reference Process Patchability After shipment, a reference process p is solely maintained through patching (cf. Fig. 4, left). The vendor may ship a new version p' that all existing extensions transparently apply. Hence, existing extensions (e_1) implicitly impose compatibility rules which constrain to what extent a patched reference process p' can differ from the predecessor version p.

Future research should formulate compatibility rules for reference process patching. That includes providing migration instructions to autmatically handle "dangling extensions" that no longer match a patched reference process.

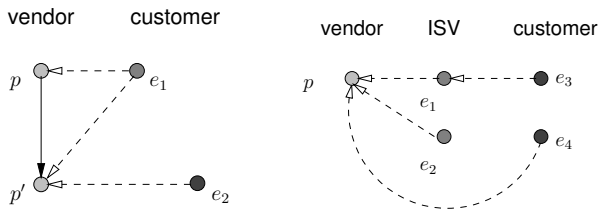

Fig. 4. Reference Process Patchability (left) and Stacked Extensions (right)

Extension Mining Deviations from reference processes may initially not be specified as proper extensions. Instead, end users may also make use of costly ad-hoc changes to gain the required flexibility. To liberate end users from tedious ad-hoc changes, and thus, essentially saving costs, process log mining may be employed to (1) detect "manual" deviations from a reference process original behavior and (2) automatically derive extension definitions.

Extension Point Extension points are part of a reference process and expose its extensible aspects. Future work should develop a concept for specifying extension points, capturing all extension patterns. That may include additional constraints on the extensions that are "plugged in". Finally, extension points should be self-sufficient such that reference processes could also be shipped as "black box" content, omitting implementation details.

Stacked Extensions In large software rollouts, 3rd party contractors may be involved. For instance, a contractor may be responsible for customizing reference processes through some baseline extensions. The customer itself may further refine these contractor-defined extensions by providing other extensions on top of it. In this way, a transitive extension chain may emerge. Figure 4 (right) depicts a scenario where both a contractor and the customer define extensions atop a reference process p. Customer extensions (e_3 and e_4) can both refer to a contractor extension (e_1) or the reference process directly. Future work needs to devise an extensibility framework architecture that supports these scenarios.

Business Process Outsourcing Both *Software-as-a-Service* and *Cloud Computing* promise significant cost savings through scaling effects. In this regard, *Business Process Outsourcing* has become the corresponding catchphrase for the BPM realm. The idea is to externalize execution of processes to 3rd party hosting providers. In terms of extensibility, one might host the reference process at the vendor side, making invocations to extensions which run on the customer side. Future work should yield an extensibility framework architecture supporting distributed execution environments that tackle challenges like performance, availability, transactionality, failover, authorization, etc.

Authorization Issues Role awareness is a key differentiator of extensibility, as opposed to other process flexibility approaches. Consequently, authorization becomes an issue inasmuch as certain operations (like view, patch, extend, run) may be constrained to certain roles. For instance, the reference process may solely be patched by the vendor, but may be extended on the customer side. More finely grained authorization schemes may be invoked to further constrain the roles that may define

extensions for specific extension points. Altogether, future research should define a comprehensive authorization concept, supporting the fore-mentioned use cases.

Design Time Usability The extensibility approach promises great cost savings over other flexibility approaches. As a prerequisite, BPMS need to include modelling tools to define extensions. These tools need to visualize relevant aspects of the to-be-extended reference process and to define and "wire up" extensions in an easy to comprehend fashion such that the impact of those changes becomes unambiguously evident.

This agenda is by no means complete: our focus is to lay the foundations for practically-oriented extensibility support as part of a BPMS.

6 Related Work

In this paper, *business process extensibility* is positioned as a new area of research in the well-explored field of *process flexibility*. A recent taxonomy in process flexibility [SMR+08] identified four approaches to achieving flexibility:

- *flexibility by design* – where a number of alternative pathways are explicitly specified in the process model at design time.
- *flexibility by deviation* – where at run-time an alternative course of action can be taken which differs from the course of action prescribed by the process model.
- *flexiblity by underspecification* – where detailed specification of (parts of) the process model is avoided. As mentioned in [SMR+08], this category covers both *late modelling* and *late binding*.
- *flexibility by change* – where a process model can be modified after deployment.

BPM systems such as ADEPT1 [RRD03], YAWL [AtH05] (including its Worklet service [AHEA06]), FLOWer [AWG05] and DECLARE [PSSA07] are classified in [SMR+08] according to this taxonomy.

Patterns are a useful means to compare the capabilities of different languages/systems and there are two pattern collections in the area of process flexibility that have recently been developed for this purpose. On the one hand, so-called *change patterns* and *change support features* are documented in [WRRM08], while on the other hand the flexibility taxonomy gave rise to a collection of *flexibility patterns* [MAR08]. In [MAR08] it is claimed that the "majority of" the change patterns can be "mapped on" the flexibility patterns. Neither pattern collection addresses the issue of managing the evolution of (reference) process models by vendors and of their counterparts by customers. However, they can be used as a mechanism to operationalize our ideas.

A well-researched problem in the area of dynamic/adaptive workflow is the migration of process instances across different versions of a process model. Consider e.g. early work by Ellis et al. [EKR95] or van der Aalst [Aal01] dealing with changed control-flow dependencies. A comparative overview of correctness criteria used by various approaches is presented in [RRD04]. More recently, Rinderle et al [RMRW08] investigated new, more relaxed, correctness criteria for process migration, taking not only the control flow perspective but also the data perspective into account. Work in

this area could be exploited and extended to deal with (controlled) changes by the vendor, the customer, or both. The last case in particular poses a challenge.

Reference models are models for targeted application domains that incorporate "best practice" [KKR06] methods in these domains. Reference process models serve to capture the procedural aspects of best practices. In the SAP R/3 environment many such models are made available using the Event-driven Process Chain (EPC) notation. As a reference process model may be quite large in order to capture all possible pathways in the various settings in which it may be used, the notion of a *configurable reference process* models was introduced [RA07]. Customizing a configurable reference process model to a particular setting may lead to a model in which many of the pathways were eliminated as they are simply not applicable. Process configuration typically is a one-off activity where there is no provision for further adaptation of the configured model. Additionally, evolution of configurable reference process models has not yet been investigated or even identified as a topic worthy of research.

An approach to tackling challenges dealing with a collection of so-called "process variants" is documented in [HBR08]. It is proposed that for a process variant the change operations that need to be applied to derive it from a base process model are explicitly stored, rather than keeping only the results of these operations. This is an idea that is valuable to the area of business process extensibility as well. The mixture of design-time and run-time considerations as well as the requirement of supporting restricted changes and the propagation of such changes, position the field of business process extensibility uniquely with respect to process flexibility and process configuration.

7 Summary

This paper introduced the notion of process extensibility as a new paradigm for customizing reference process models and managing their evolution over time. The main difference with traditional process flexibility approaches arises from the clear separation of concerns between the reference process owner (vendor) and the owner of extensions thereof (customer). The tension between customer freedom, when it comes to reference model adaptation, and the ability to incorporate with relatively low effort vendor-initiated reference model changes, needs to be carefully balanced. This paper provided an overview of, and motivation for, the notion of business process extensibility, positioned this area with respect to related areas such as process configuration and process flexibility, and identified some of the main unresolved challenges in this area. The limitation of *controlled flexibility* presented in this paper is related to the ability of the vendor to foresee where extensions to a reference model could be needed in future. In fact once extension points are set, they should not be changed to avoid losing synchronization with the customers' derived models.

References

[Aal01] W.M.P. van der Aalst. Exterminating the dynamic change bug: A concrete approach to support workflow change. *Inf. Systems Frontiers*, 3(3):297–317, 2001.

[AHEA06] M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Worklets: A service-oriented implementation of dynamic flexibility in workflows. In *Proc. of the 14th Int. Conf. on Cooperative Inf. Systems (CoopIS'06)*, volume 4275 of *LNCS*, pages 291–308. Springer, 2006.

[AtH05] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[AWG05] W.M.P. van der Aalst, M. Weske, and D. Grünbauer. Case handling: A new paradigm for business process support. *DKE*, 53(2):129–162, 2005.

[EKR95] C. A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *Proc. of the Conf. on Organizational Computing Systems, COOCS 1995, Milpitas, California, USA, August 13-16, 1995*, pages 10–21. ACM, 1995.

[FLZ06] P. Fettke, P. Loos, and J. Zwicker. Business process reference models: Survey and classification. In *BPM Workshops*, volume 3812 of *LNCS*. Springer, 2006.

[GAJVL08] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and M. La Rosa. Configurable Workflow Models. *Int. Journal of Coop. Information Systems*, 17(2):177–221, 2008.

[HBR08] A. Hallerbach, T. Bauer, and M. Reichert. Managing process variants in the process life cycle. In *ICEIS 2008 - Proc. of the Tenth Int. Conf. on Enterprise Information Systems, Volume ISAS-2*, pages 154–161, 2008.

[KKR06] J. M. Küster, J. Koehler, and K. Ryndina. Improving business process models with reference models in business-driven development. In *BPM 2006 Workshops*, volume 4103 of *LNCS*, pages 35–44. Springer, 2006.

[MAR08] N. Mulyar, W.M.P. van der Aalst, and N. Russell. Process flexibility patterns. BETA Working Paper Series, WP 251, Eindhoven University of Technology, the Netherlands, 2008. Available at http://fp.tm.tue.nl/beta/publications/working%20papers/Beta_wp251.pdf.

[PSSA07] M. Pesic, M. H. Schonenberg, N. Sidorova, and W.M.P. van der Aalst. Constraint-based workflow models: Change made easy. In *CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated Int. Conf. Proc., Part I*, volume 4803 of *LNCS*, pages 77–94. Springer, 2007.

[RA07] M. Rosemann and W.M.P. van der Aalst. A Configurable Reference Modelling Language. *Information Systems*, 32(1):1–23, 2007.

[RMRW08] S. Rinderle-Ma, M. Reichert, and B. Weber. Relaxed compliance notions in adaptive process management systems. In *Conceptual Modeling - ER 2008, 27th Int. Conf. on Conceptual Modeling*, volume 5231 of *LNCS*, pages 232–247. Springer, 2008.

[RRD03] M. Reichert, S. Rinderle, and P. Dadam. Adept workflow management system:. In *BPM 2003*, volume 2678 of *LNCS*, pages 370–379. Springer, 2003.

[RRD04] S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems - a survey. *DKE*, 50(1):9–34, 2004.

[RRKD05] M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive process management with ADEPT2. In *ICDE 2005*, volume 3716, pages 1113–1114. IEEE Computer Society, 2005.

[SMR+08] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W.M.P. van der Aalst. Towards a taxonomy of process flexibility. In *CAiSE Forum*, pages 81–84, 2008.

[SN00] A.-W. Scheer and M. Nüttgens. *Business Process Management*, volume 1806 of *LNCS*, chapter ARIS Architecture and Reference Models for BPM. Springer, 2000.

[WRRM08] B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features - Enhancing flexibility in process-aware information systems. *DKE*, 66(3):438–466, 2008.