Controlled Flexibility and Lifecycle Management of Business Processes through Extensibility

Sören Balko[1], Arthur H.M. ter Hofstede[2], and Alistair Barros[3]

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

Abstract. Companies employ business process management suites to model, run, and maintain their processes. These processes are required to comply with requirements originating from standards or policies and follow industry best practices. At the same time, business processes must be sufficiently flexible to incorporate company-specific customizations. This paper introduces extensibility as a new process flexibility approach. Extensibility addresses the issue of customizing reference processes. Reference processes are "shipped content" which is maintained by the BPMS vendor, keeping the total cost of ownership (TCO) down at the customer side. At the same time, customers may flexibly define process extensions as "deltas" on top of reference processes. Extensibility is for the most part motivated through shortcomings of other process flexibility techniques with respect to a separation of (maintenance) concerns, and excessive upgrade costs in large-scale software rollouts. With process extensibility, we actually hope to establish a whole new area of future research. To guide in there, we pinpoint some directions to explore in detail.

1 Introduction

Companies define business processes to analyze, document, execute, monitor, and govern complex repetitive tasks that are part of a value chain. Business process management suites (BPMS) provide an integrated tooling support for these use-cases and add lifecycle management capabilities on top [20]. In many industries, a company's surroundings such as customer demand, technological innovations, and regulatory conditions tend to change frequently and sometimes rapidly. Take the current financial crisis and the resulting economic downturn as an example. Suddenly, enterprises face a shift in customer focus towards products and services that immediately generate revenue, banks must apply more stringent accounting rules, and companies must break new grounds to find financing for investments. By implication, core processes of the affected businesses would be required to be adjusted to cope with these changes.

By being able to flexibly adapt their processes to changes, agile businesses set themselves apart from their competition. Naturally, 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 dynamically incorporate change into business processes [13, 3–5]. 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 lifecycle management in general, and the costs that come with changing business processes, specifically. Some existing techniques suggest that BPMS customers alter reference processes [6] "in place" in order to customize them to their needs (*patching* use-case). Alternatively, reference processes act as templates [17] for newly developed processes (*blueprinting* use-case).

Neither of these approaches can realistically succeed in large-scale software rollouts, 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 (1) carrying out the changes and (2) later maintaining the resulting processes. Any time the BPMS vendor ships a new 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, two customer departments may wish to change a cross-departemental reference process independently at different points in time.

Business process extensibility introduces a flexibility approach which specifically aims at the parties which are involved with defining, using, and maintaining business processes. As part of that, BPMS vendors *own* (i.e., define and maintain) reference processes whereas BPMS customers own and run extensions thereof. Process extensions constitute separate customer-defined "deltas" 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 owners without ever having to be "re-wired".

It is a benefit of extensibility to designate the responsibilities for maintaining processes and extensions. 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.

In this paper, we introduce the concept of extensibility aiming at low-cost customization and a clear separation of concerns. Section 2 outlines its benefits over existing flexibility approaches. In Section 3, we provide a taxonomy to classify these approaches. Section 4 presents use-cases and proposes an extensibility framework architecture. Section 5 formulates an agenda for potential follow-up research in this area and Section 6 surveys related work.

2 Extensibility

The general concept of making processes more flexible to let them deviate 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*.

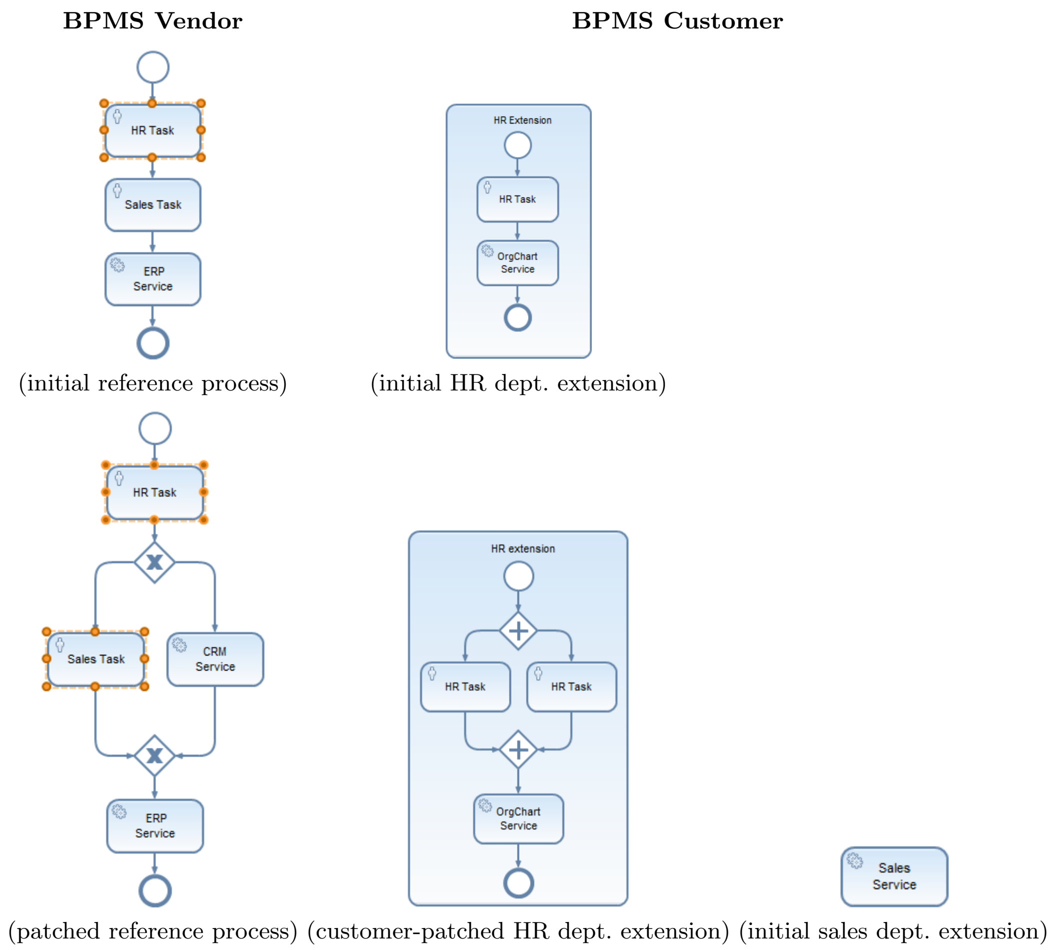
2.1 Proposal

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 customize reference processes to their needs by independently defining and deploying extensions which solely constitute "deltas" (process fragments). Extensions exist alongside the (reference) processes. Only when running a process, an extensibility framework dynamically invokes its extension(s). 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 fix bugs or simply override outdated functionality which makes patching a crucial means of maintaining processes. Patches to a reference process must adhere to some compatibility rules which require the new version to support any existing customer extension. The extensibility framework takes care of automatically incorporating all customer extensions that were defined atop any old version of the patched reference process. Just as well, 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) which is customized with extensions of the customer's HR and Sales departments. The initial reference process is a plain 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 which 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



(initial reference process) (initial HR dept. extension)

(patched reference process) (customer-patched HR dept. extension) (initial sales dept. extension)

Fig. 1. Extensibility Example

(here: “HR Task” → “HR Extension”). Independently of 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.

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 from 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 which (1) interact with the same process but (2) have their distinct customization requirements. For instance, a legal department could ask for fine-granular logging in 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 side 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 to not 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 further 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 loosing (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 desireable if details in the reference content consitute 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" by proper process models. With some application instrumentation to add extension points, extensibility may even help in bridging this sort of 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 this 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 actual people's functional *role* descriptions, (4) the *lifecycle* stages (design time, runtime) it is used, (5) the *constraints* that restrict what can be done, and (6) the *scope* (process type, instance, version) the flexibility technique operates on. Existing flexibility techniques can be fit into this classification 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 modelled 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 does not traditionally constitute a flexibility use-case. However, modeling a (reference) process from scratch is a prerequisite for any other flexibility technique. It is mostly business analysts 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 (which were deployed to a BPMS runtime before) may need to be altered. The vendor may have to patch reference processes to fix bugs or to simply 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 labelled 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 deployment of the new (patched) process version into the BPMS runtime. There are some constraints that limit what can be changed when patching a process. At first, interface compatibility must be preserved such that client (processes) do not have to be adapted to cope with the change. Secondly, existing extensions 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 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 (configuration & individualization) 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 that template and rather create new variations that are independent from (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 models of running process instances. Consequently, ad-hoc changes must allow for automatically migrating the instance state to the altered model.

Ad-hoc changes typically affect a single process instance, only. The altered process model is kept temporarily 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 configuration. 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.

	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
Template Reuse	Reuse	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
Extensibility	Customization, Adoption of best practices	Customer	Domain expert, IT department	Design Time (Runtime)	Extension point	all future versions

Table 1. Process Flexibility Taxonomy

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 line organization (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 even 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 before and positions extensibility as a new approach.

4 Implementation

Conceptually, extensibility is open to different process perspectives. In this Section, we identify some frequent extensibility patterns in the control flow and data flow perspectives. Without loss of generality, we use BPMN-like notations to illustrate these use-cases. We also propose a high-level BPMS server architecture, incorporating an "Extensibility Framework" component.

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 (to hierarchically structure processes and re-use existing functionality). 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 process 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) a 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 even allows for incorporating structurally complex customer extensions into reference flows.

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

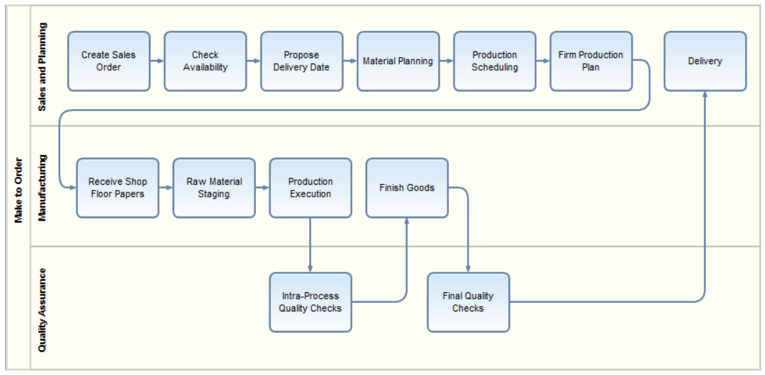


Fig. 2. "Make to Order" Reference Process

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 (to 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

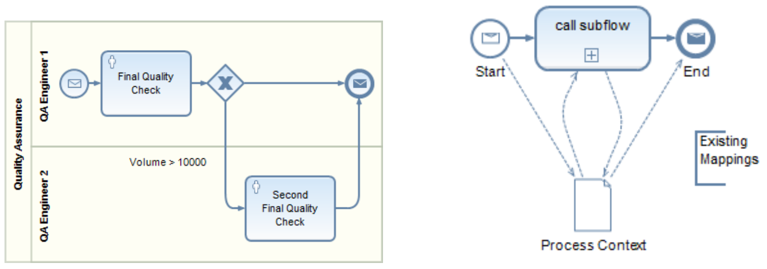


Fig. 3. Usage Extensibility (left) and Field Extensibility (right) Examples

where the start/end events represent the inbound case, providing the process as a service having 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 accomodate additional fields, clients (including “parent” processes) may pass on extra data to 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 means to take advantage of additional data fields.

4.3 Architecture

Extensions are incorporated into processes at runtime using late binding mechanisms. A corresponding BPMS runtime needs provide an “Extensibility Framework” to dynamically invoke extensions. Figure 4 illustrates a high-level BPMS server architecture where the process execution engine interacts with an extensibility framework execute all extensions defined atop a given process. When

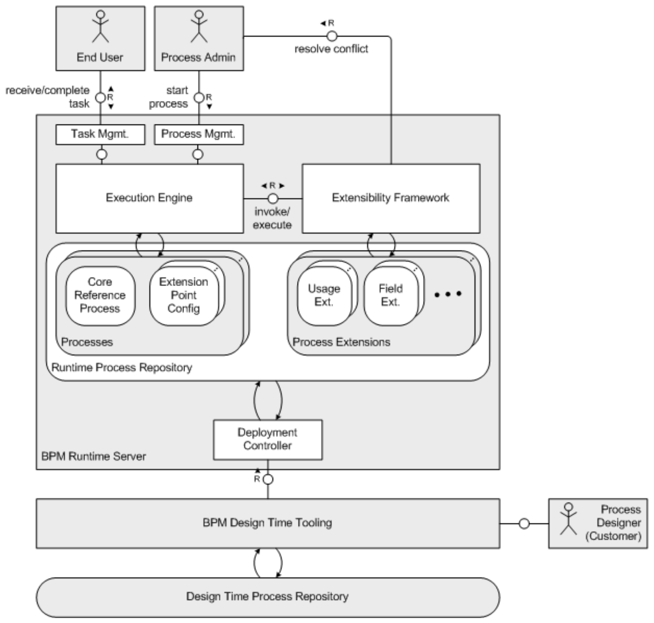


Fig. 4. BPMS Server Architecture w/ Extensibility Support

executing a process instance that runs into an extension point, the process execution engine triggers the extensibility framework. The extensibility framework merely acts as a lookup facility which initially retrieves the matching extension(s) (if any) from a process extension store. It then (2) makes use of the process execution engine to actually execute the extension(s). Once completed, the extensibility framework returns focus to the reference process.

This architecture sketch merely introduces the involved components with their high-level interaction and not yet specify their detailed behavior. The latter is obviously heavily dependent on the to-be-covered extensibility use-cases.

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 giving indications for possibly future research on conceptual and technical follow-up topics. We hope those issues to be taken up by the research community and rely on their contributions to adequately cover this area.

Most topics revolve around (1) fully understanding the applicability and limitations of process extensibility and (2) to lay 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 constitue 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 that area could result in an explicit constraint model for defining extensions at reference processes.

Reference Process Patchability After shipment, a reference process p is solely maintained through patching (cf. Fig. 5, left). The vendor may ship a new version p' that all existing extensions transparently apply to. 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 do not match a patched reference process any longer.

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.

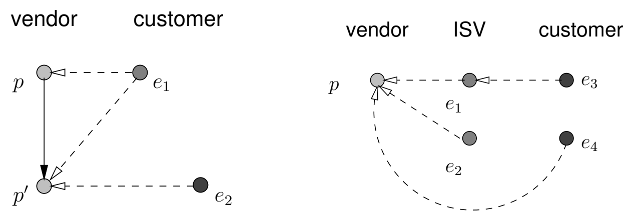


Fig. 5. Reference Process Patchability (left) and Stacked Extensions (right)

Extension Point Extension points are part of reference processes and expose its extensible aspects. Future work should come up with 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 affected with 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 5 (right) depicts a scenario where both a contractor and the end 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 which support 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 which tackles 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 fine-granular authorization schemes may be in place to further constrain the roles that may define extensions for specific extension points. Altogether, future research should define a comprehensive authorization concept, supporting the afore-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 umambiguously evident.

This agenda is by no means complete. Our focus in this agenda is to lay the foundations for a 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 [18] 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 [18], 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 [12], YAWL [2] (including the Worklet service [4]), FLOWer [3] and DECLARE [10, 11] are classified in [18] 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 [19], while on the other hand the flexibility taxonomy gave rise to a collection of *flexibility patterns* [9]. In [9] 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.

A well-researched problem in the area of dynamic/adaptive workflow is the migation of process instances across different versions of a process model. Consider e.g. early work by Ellis et al. [5] or Van der Aalst [1] dealing with changed control-flow dependencies. A comparative overview of correctness criteria used by various approaches is presented in [14]. More recently, Rinderle et al [15] 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 targetted application domains that incorporate "best practice" [8] 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 [16]. Customising 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 [7]. 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 that only the result of the application of these operations is kept. 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, positions the field of business process extensibility uniquely with respect to process flexibility and process configuration.

7 Summary

Extensibility establishes a new process flexibility technique which aims at customization of reference processes. It differs from existing approaches by offering a clear separation of Concerns between the reference process owner (vendor) and the owner of extensions thereof (customer). At the same time, extensibility enforces "controlled change" through the extension point concept and physical separation between processes and their extensions. The latter caters for both low-cost supportability of reference processes through the vendor and preservation of the reference process' core "business intent".

References

1. 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.
2. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

3. 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.
4. 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.
5. 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.
6. P. Fettke, P. Loos, and J. Zwicker. Business process reference models: Survey and classification. In *BPM Workshops*, volume 3812 of *LNCS*. Springer, 2006.
7. 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.
8. 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.
9. 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.
10. M. Pesic and W.M.P. van der Aalst. A declarative approach for flexible business processes management. In *Proc. of the First Int. Workshop on Dynamic Process Management (DPM 2006)*, volume 4103 of *LNCS*, pages 169–180. Springer, 2006.
11. 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.
12. M. Reichert, S. Rinderle, and P. Dadam. Adept workflow management system:. In *BPM 2003*, volume 2678 of *LNCS*, pages 370–379. Springer, 2003.
13. 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.
14. S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems - a survey. *DKE*, 50(1):9–34, 2004.
15. 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.
16. M. Rosemann and W.M.P. van der Aalst. A Configurable Reference Modelling Language. *Information Systems*, 32(1):1–23, 2007.
17. A.-W. Scheer and M. Nüttgens. *Business Process Management*, volume 1806 of *LNCS*, chapter ARIS Architecture and Reference Models for Business Process Management. Springer, 2000.
18. 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.
19. 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.
20. M. Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer, 2007.