Sören Balko, Arthur H. M. ter Hofstede, Alistair Barros, Marcello La Rosa, and Michael Adams

Business Process Extensibility

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 customising 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. Take the global financial crisis and the resulting economic downturn as an example. Suddenly, enterprises face a shift in customer focus towards products and services that can immediately generate revenue, banks must apply more stringent accounting rules, and companies must break new ground to find financing for investments. Beyond doubt, in each of these cases the core processes of the affected businesses would have been severely impacted. Thus, these business processes are required to adjust to cope with these changes. By being able to flexibly adapt their processes to change, agile businesses set themselves apart from their competitors.

Naturally, business process management suite (BPMS) offerings need to facilitate flexibility at low costs. At the same time, companies still wish to benefit from standardised 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., Rosemann and Aalst (2007), Gottschalk et al. (2008), Reichert et al. (2005), Aalst et al. (2005), Adams et al. (2006), Ellis et al. (1995). These approaches cover both design time and runtime changes and provide formal frameworks to constrain changes. However, many established process flexibility approaches suffer from shortcomings with respect to process lifecycle management in general, and specifically to the costs associated with changing business processes. Some techniques propose that BPMS customers alter reference processes 'in place' in order to customise them to their needs (*patching* use-case) (Fettke et al. 2006). Others suggest to use reference processes merely as templates for developing company-specific processes (*blueprinting* use-case) (Scheer and Nüttgens 2000).

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 customising 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. Moreover, by automatically applying existing customer extensions to patched referenced processes, the cost of rolling out new BPMS releases can be greatly reduced.

The rest of this paper is organised as follows. Section 2 outlines the extensibility approach and its benefits over existing flexibility approaches. Next, Sect. 3 provides a taxonomy to classify flexibility approaches. Section 4 identifies different extensibility types along the control flow, data flow and resource perspectives of a process. Section 5 discusses how an extensibility framework

can be operationalised within a BPMS server architecture, while Sect. 6 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 customisation, 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 customisation of reference content. Unlike existing approaches, extensibility clearly designates responsibilities for the process and extensions thereof. Reference processes may be patched (bug-fixed, 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 customise 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 customised 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 organisational 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 generalised best practices. The vendor 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 'customised' to a customer's needs. From a vendor perspective, it is vital to ensure that reference process change is controlled.

Customer   Customers engage their IT team to customise 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 perform 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 customisation requirements. For instance, a legal department could ask for fine-grained logging within audit-sensitive processes, whereas the IT department may be interested in being notified of technical process failures.

Extensibility offers *controlled flexibility* for the different parties that design, customise, 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. Also, the vendor will want to disallow arbitrary changes to this content to avoid inconsistencies by the customer 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 customisations are guaranteed not to distort the reference functionality.
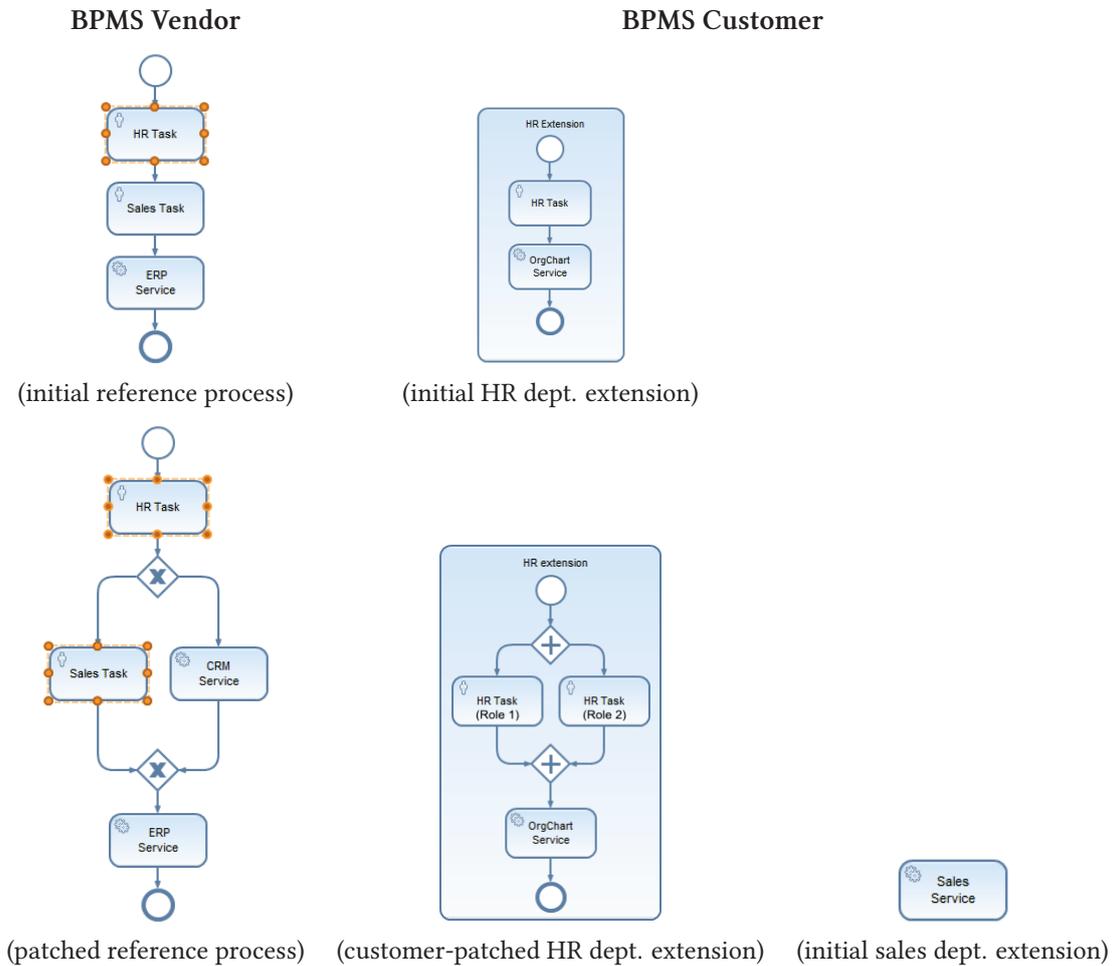
Figure 1: Extensibility Example

Technically, the vendor ships reference processes that incorporate 'extension points', which are pre-planned artifacts where customers can incorporate their extensions. Extension points virtually apply to any dimension of modelling business processes, including control flow, data flow, resources, rules, security, etc. We will describe different extensibility types in Sect. 4.

2.2 Expected Benefits

Extensibility comes with a number of expected advantages over existing flexibility approaches. It (1) offers a lifecycle model for controlled flexibility taking into account obligations and concerns of different parties involved in designing, customising and using business processes. It

helps to avoid 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 (3) be 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, extensions (4) consume fewer resources at runtime. This is because

an extension solely constitutes a small 'delta'. As a side effect, this approach 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, for example, details in the reference content constitute significant intellectual property that is not to be disclosed (*Intellectual Property*). Reference processes may also be purely documentary models that are not directly executed in a proper BPMS runtime but rather serve as a blueprint to implement applications, i.e., the processes are hard-coded in applications. 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 optimising 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, customiser, 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 aids in understanding their differences. It also outlines the contribution of extensibility to the overall picture. We specifically discuss the differences between *from-scratch modelling* of new processes, *patching* existing processes, re-using a vendor-provided template to develop a new business process (*blueprinting*), configuring a reference model to customers' specific settings (*configuration*), performing *ad-hoc changes* of process instances at runtime, modifying (technical) *runtime settings*, and *extending* reference processes:

From-Scratch Modelling Modelling 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. Alternately, bottom-up approaches start with modelling detailed process fragments which are later aggregated into larger end-to-end business processes.
Strictly speaking, this approach is not traditionally regarded as a flexibility approach, because there is no initial (reference) process model. However, modelling a (reference) process from scratch is a prerequisite for any other flexibility technique. It is usually business analysts who start modelling 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 modelling language.

Patching This is a design time approach where a model undergoes changes, typically to address new or changed requirements or to fix bugs. The vendor may have to patch reference processes for these reasons. 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 change. Secondly, existing extension points must be retained in the patched version such that extensions transparently apply to the patch.

Blueprinting Vendor-supplied 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.

Configuration This is an evolution of blueprinting because it allows a controlled derivation of a customised model (called an *individualised model*) from a configurable reference model. A configurable process model allows certain types of refinements and these refinements can only restrict the reference model behaviour. By enforcing these constraints, it is possible to preserve the correctness of the reference model (e.g., with respect to soundness) during configuration, such that all individualised models are correct. It is most suited when the reference model originates from the consolidation of a number of process variants for a specific industry domain. In this way the variation points can be automatically identified with the points in the model where multiple variants exist. In this approach vendors ship reference models with variation points, and customers use these points to configure the model to their specific needs.

Ad-hoc Changing Sometimes end users have to deviate from the behaviour of the process instances they are involved in. This is often the case of human-driven processes, which end up running into exceptional situations, such as a designated task processor being absent or a completion deadline overdue. End users need to (implicitly) alter the process model for their specific instance, thus deviating from its original business semantics. Those changes will typically be done in a lightweight modelling environment, targeted to end users. Typical changes include modifying task assignments, setting back processes (to re-do a task), introducing tasks to have additional people work on an issue, and so on.

Constraints for ad-hoc changes relate to role-related restrictions and instance migration. 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 a single process instance only. The altered process model

is kept temporarily, i.e., for the lifetime of that instance. Only optionally, an ad-hoc change may be applied to more than a single instance, making it necessary to permanently persist the changed model to the process repository. Ad-hoc changes will mostly be conducted by a process administrator. In selected cases, process end-users (e.g., task owners) may be offered some limited ad-hoc change operations (e.g., altering a task assignment, re-doing a task, etc.).

Runtime Settings Some environmental settings hold globally for all processes and, when changed, need to immediately apply to both running processes and newly starting instances. Those settings include modifications to organisational charts, security policies and other non functional aspects. 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. Similar to blueprinting and configuration, the primary use-case is adoption of best practices. In doing so, the customer may decide to adapt a given set of best practices to suit their specific business needs. Various customer roles may define process extensions, each with different objectives. Domain experts from specific organisational lines (e.g., Sales, Procurement, Manufacturing, etc.) may independently define extensions to adjust a cross-organisational process to their needs. In turn, the IT department may want to incorporate extensions which allow for better monitoring of running processes. 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 applies to all future model versions, and is subject to some

constraints, either originating from implicit compatibility rules or explicitly from modeled extension points within reference processes.

Alongside other restrictions, the vendor may predefine places and types of extensions within the reference model. Most importantly, extended reference models are fully supportable in a sense that the reference model can be patched (by the vendor) to incorporate bug fixes, etc. Any customer extension defined on top of that process will transparently apply to the patched version, thus dramatically reducing costs of ownership for the customer who does not have to re-do all the afore-defined extensions on top of the patched reference model version (as opposed to template re-use).

Table 1 classifies existing flexibility techniques according to the dimensions introduced and positions extensibility as a new approach.

4   Extensibility Types

Conceptually, extensibility is open to different process perspectives. This section identifies different extensibility types along the control flow, data flow and resource perspectives of a business process. 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 synchronisation).

In this paper, we consider two types of control flow extensibility: *Usage Extensibility* and *Structural Extensibility*. Usage extensibility is the most straightforward way of creating control flow extensions and applies to activities, denoting atomic tasks, either performed automatically

Table 1: Process Flexibility Taxonomy

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, Bug fixing	Vendor, Customer	Process developer, Domain expert	Design time	Extension/ interface compatibility	new version
Blueprinting	Best practice adoption, customisation	Customer	Process developer, Domain expert	Design time	–	new process
Configuration	Best practice adoption, customisation	Customer	Process architect, Domain expert	Design time	Correctness preservation	new process
Ad-Hoc Changing	Handling of exceptional cases	Customer	Process administrator, Task owner	Runtime	Instance migration, Role-related	single instance
Runtime Settings	System-wide settings	Customer	System administrator	Runtime	–	all running instances
Extensi-bility	Best practice, adoption, customisation	Vendor, Customer	Domain expert, IT department	Design Time, Runtime	Compatibility rules	All future versions

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 data flow interfaces, in order for the extensibility framework to 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*[1] 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) sales department, (2) manufacturing, and (3) quality assurance. After negotiating delivery dates and completing the production

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

planning, manufacturing ultimately produces the good with interleaved quality checks for the production process and final checks for the good itself. At customisation, this process is extended to optionally modify those quality gates depending on the order volume. That is, for high-volume orders a four-eye 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 Fig. 3 (left).

Usage Extensibility captures a wide range of customisation use-cases and can be applied in a straightforward way. In fact, by substituting atomic activities with subflows, it allows the incorporation of structurally complex customer extensions into reference flows. Complex structural changes that go beyond substituting activities are out of reach for Usage Extensibility. This is due to the fact that the replacing activity or subflow A' is technically bound to the interface of the activity to be replaced. In essence, A'
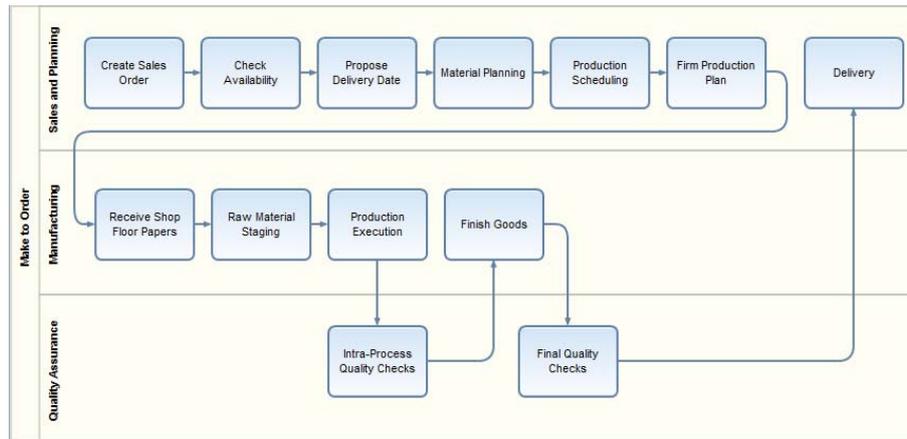
Figure 2: 'Make to Order' Reference Process

is constrained to operate and manipulate those fragments of the process data context which are passed to and retrieved from *A*. This is because customer extensions are dynamically invoked through late binding mechanisms which rely on *a-priori* fixed interfaces.

There are other cases where Usage Extensibility is simply inadequate and impractical to use. Suppose one wants to extend the 'Make to Order' process in a way that for standard goods the 'Material Planning', 'Production Scheduling', and 'Firm Production Plan' steps could be automated such that the resulting process fragment conceptually looks like the one in Fig. 3 (right).

This scenario is obviously more challenging. An existing reference process fragment (the 'Material Planning' → 'Production Scheduling' → 'Firm Production Plan' activity chain) would need to be *optionally* skipped and the alternative flow ('Automated Planning' activity) would need to be merged back into the main flow. In that sense, the structural extensions constitute process fragments which hook up to the reference flow through a branching-and-merging extension point. This is thus a case of Structural Extensibility. In this context, extension points are distinct control flow connectors of the reference process which do have some gateway semantics. That is, one can require the extension flow to run in parallel, optionally or exclusively. Correspond-

ingly, the merging behaviour can be specified as synchronising or simple merge.

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. A 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 customise the reference process in a way that it receives/passes on additional parameters from inbound/to outbound messages. New clients may interact with the process through the field-extended interface. In turn, compatibility to existing clients (provisioning) and services (consumption) must be guaranteed.

Figure 4 shows an example of Field Extensibility. It depicts a plain BPMN flow where the start and end events identify the boundaries of an inbound case, providing the process as a service has a well-defined interface. A new process instance is spawned upon receiving an inbound message on that interface. In turn, the end event terminates the instance and prepares the corresponding outbound message. When compatibly extending that interface to accommodate additional fields, customers (including parent processes) may pass
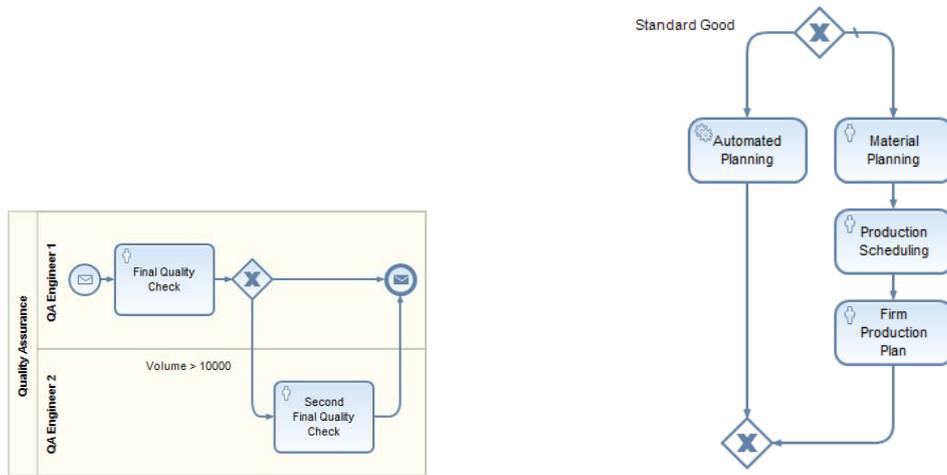
Figure 3: Usage Extensibility (left) and Structural Extensibility (right)

on extra data to the process. The process may then make use of this data in usage-extended activities. Existing customers remain unaffected, as they pass on their inbound messages to an extensibility framework which adds the extra fields, or receive their outbound messages from an extensibility framework which strips off the extra fields.

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.
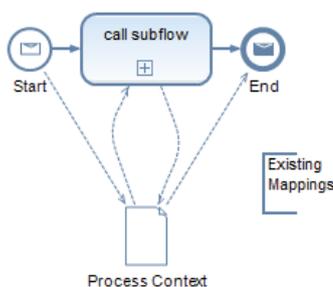

Figure 4: Field Extensibility

4.3 Resource Perspective

In the resource perspective extensions are predominately concerned with user roles and other organisational entities. To some extent, this type of change will be dealt with as part of the *Runtime Settings* flexibility approach. Nevertheless, customisations may still want to constrain these settings for a concrete process.

Organisational charts including role assignments are valid for the entire organisation and do not need to be maintained at the process level. Customisation is only required to impose additional restrictions that support the specific semantics of the given process. Consider the 'Make to Order' reference process (Fig. 2) where two quality checks are in place: one for the manufacturing process (Q_1) and one for the produced good itself (Q_2).

From an organisational perspective, Q_1 and Q_2 are conducted by quality assurance (QA) engineers subsumed under the 'Quality Control' role. For reasons such as obtaining independent views in these checks and also avoiding workload peaks and bottleneck situations it may be desirable to explicitly require that Q_1 and Q_2 are not conducted by one and the same person. Technically,

the owners of both tasks are in the 'Quality Control' role, yet Q_2 must not be carried out by the owner of Q_1. *Role Extensibility* describes the customisation use-case of adding more constraints to the task role assignments. The extensibility framework can easily support this use-case by offering a filter operation which computes a subset of the role members according to the concrete extension semantics. In this example, it takes the owner of Q_1 as a parameter and excludes it from the potential processors of Q_2.

5   Extensibility Framework: Technical Considerations

In this section we describe how the extensibility framework can be operationalised via an *Extensibility Service* integrated into a BPMS server architecture. As a proof of concept, an Extensibility Service may be implemented as a Custom Service within the YAWL workflow system (Hofstede et al. 2010). YAWL has been chosen as the exemplary platform since it provides an expressive workflow language based on the workflow patterns,[2] together with a formal semantics (Aalst and Hofstede 2005). It also provides a fully interfaced workflow enactment engine, and a process design tool for process model and extension creation. The YAWL environment is opensource and follows the service-oriented architecture paradigm, allowing extensibility support to be provided by means of a complementary service independent to the core engine. However, while this example illustrates the Extensibility Service within the YAWL architecture, it should be seen as in no way limited to that environment.

Custom YAWL services interact with the YAWL engine through XML/HTTP messages via certain interface endpoints, some located on the engine side and others on the service side. Specifically, custom services may elect to be notified by the engine when certain events occur in the life-cycle of nominated process instantiations (i.e., when a work item becomes enabled, when a

---

[2]www.workflowpatterns.com

work item is canceled, when a case completes), to signal the creation and completion of process instances and work items, or to notify of certain events or changes in the status of existing work items and cases.

Figure 5 presents a high-level architecture that illustrates how the process execution engine interacts with an extensibility service that manages and executes all extensions defined atop a given process.

Extensions are dynamically incorporated into processes at runtime using late binding mechanisms managed by the Extensibility Service. When a process instance reaches an extension point, the process execution engine notifies the Extensibility Service of an extension point event, passing certain process descriptors (e.g., process and extension point identifiers, instance data, resourcing information and so on) along with the notification.

The Extensibility Service then retrieves the matching extension(s) (if any) from the customer's process repository. If there are multiple *entry* extensions defined for the extension point notified, the Extensibility Service makes an 'intelligent' choice using the context of the particular process instance against a set of pre-defined business rules. Once a matching extension has been chosen, it is passed to the engine for execution; once the extension completes, the Extensibility Service notifies the engine to return focus to the original reference process. If no extension is found for the extension point notified, or if no appropriate extension is found amongst the matches given the instance context, the Extensibility Service sends a *decline* message to the engine, allowing the reference process to continue as originally defined.

The Extensibility Service takes responsibility for the proper execution of an extension within the scope of an extension point of a reference process. If an extension is specified to merge synchronously after completion, the parent process is temporarily suspended while the extension is
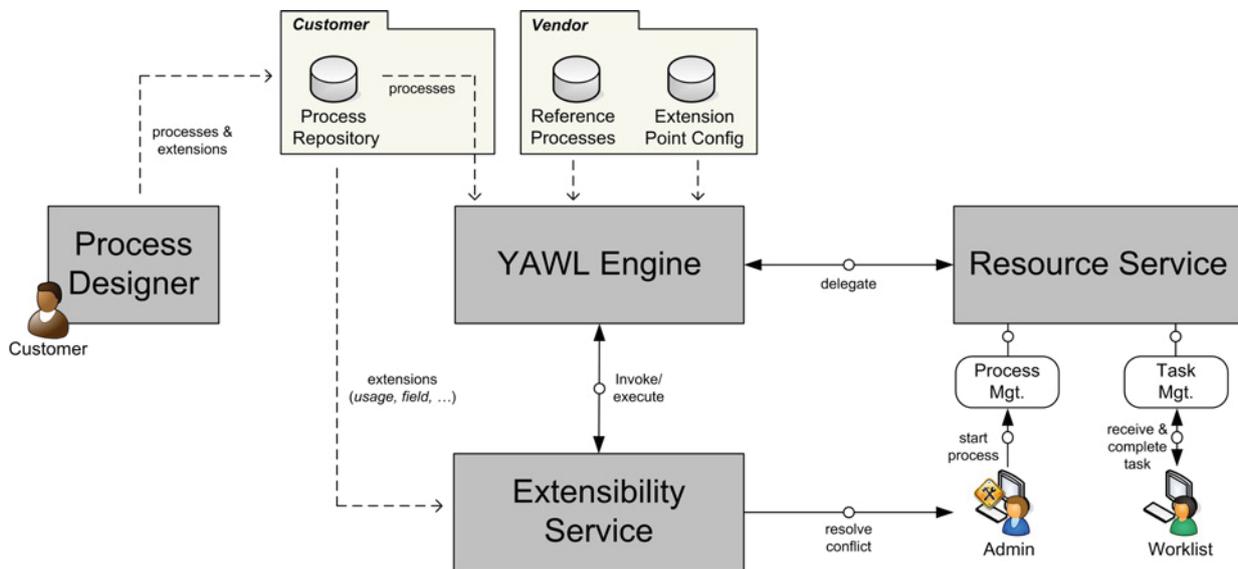
Figure 5: YAWL Architecture with Extensibility Support

executed. The executed extension is run as a separate entity within the engine, so that, from an engine perspective, the extension and its parent reference process instance are two distinct cases. The Extensibility Service tracks the relationships, data and resource mappings, and synchronisation between the two instances, so that when the extension completes, the parent process can be seamlessly resumed or updated as required.

A customer may designate that a particular extension point requires a number of extensions to be executed. Such semantics may be specified within an extension definition, so that an array of extensions are run in parallel or sequentially or a combination of both. In such cases, the Extensibility Service handles the flow of multiple extension execution, only passing control back to the parent process once all relevant extensions have completed.

Human actors come into play at various occasions. A process designer uses the design tool to craft both process definitions and extensions, which are stored in the customer's process repository. These process definitions are loaded into the runtime engine, along with the reference

process definitions and extension point configuration data supplied by the vendor. Customer-defined extensions are loaded from the process repository by the Extensibility Service on a just-in-time basis in response to engine notifications. The launch of a new process instance in the engine is triggered by a process administrator through a administration layer provide by the Resource Service (process management). End users interact with processes through tasks allocated to them, which are cached and dispatched by a task management component of the Resource Service.

In exceptional situations, the Extensibility Service will notify the process administrator of a conflicting situation that requires resolution. For instance, a reference process may have been patched in an incompatible way such that it does not fit a certain extension definition. In such cases, the process administrator may resolve the issue by deciding to cancel the process instance or to skip the 'dangling extension'.

6   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 view of relevant extensibility use-cases. These use-cases should preferably constitute real-world customisation requirements that need to be classified and mapped to extensibility patterns, along the three main process perspectives (control flow, data flow and resources). As a result, BPM vendors can better address customer requirements with respect to recurrent extensibility use-cases.

Reference Process Conformance Any customer-side extensions will alter the behaviour of reference models in some way, which, by definition, represent best practices. Therefore, for reasons such as conformance to legal requirements or industry standards, it will be necessary to preserve certain core characteristics of the reference model within the extended process. A possible way to address this problem is by defining an explicit constraint model for extension points (i.e., places in reference models where extensions may be safely inserted), in order to prevent the core characteristics from being corrupted. An alternate approach is to develop a framework for specifying the core characteristics of reference models, for example formulated as pre- and post-conditions, thus leaving the inner detail of the process underspecified. Extensibility constraints could then be derived from the definitions of the core characteristics, providing a large degree of flexibility.

Reference Process Patchability After shipment, a reference process p is solely maintained through patching (cf. Fig. 6, 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 automatically handle 'dangling extensions' that no longer match a patched reference process.

Reference Process Versioning Once a reference process model is shipped to clients, it is fully maintained by the vendor, who will ship from time to time new versions of it (which may include bug-fixes or revised legal requirements, for example). Any customer-side extensions that were attached to the original version of the reference model would transparently attach to the new version. Therefore, client process extensions must implicitly impose compatibility rules that constrain to what extent a new reference model version can differ from the previous version, while still maintaining extension compatibility between reference model versions. Through the enforcement of these rules, vendors would then be able to detect potential constraint violations, and perform migration plans to handle violating extensions ('dangling extensions') on the client side.

Extension Merging Multiple users can independently define extensions of the same reference process. Those extensions may be contradictory, thus situations should be investigated in detail (i) to come up with sound execution strategies and (ii) to detect contradictory extensions, for example incompatible field extensions of the same interface.

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
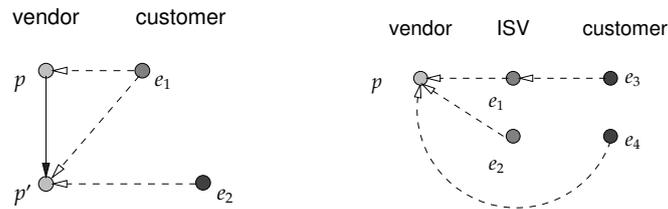
Figure 6: Reference Process Patchability (left) and Stacked Extensions (right)

the required flexibility. To liberate end users from tedious ad-hoc changes, and thus essentially saving costs, process log mining may be employed to detect 'manual' deviations from a reference process' original behaviour and to 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.

Migration Strategies Sometimes a patched reference process version will break compatibility with existing extensions. For instance, it may be required to expand an interface or to drop an activity, thus potentially invalidating customer-defined field and usage extensions atop these reference process artifacts. Consider for example the case depicted in Fig. 1 where another patch to the reference process could drop activity 'HR Task', which would implicitly invalidate 'HR extension'. Preferably, resolution strategies will automatically migrate these 'dangling extensions' without manual customer intervention. For instance, a usage extension to an activity which is dropped may simply be discarded if the extension did not have any side effects (e.g., a stateless, synchronous Web Service call). In other cases, resolution may not be as straightforward, though. Based on extensibility patterns, future research should identify automatic resolution strategies (if applicable) for 'dangling extensions'.

Stacked Extensions In large software rollouts, 3rd party contractors may be involved. For instance, a contractor may be responsible for customising 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 6 (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 externalise 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, authorisation, etc.

Authorisation Issues Role awareness is a key differentiator of extensibility, as opposed to other process flexibility approaches. Consequently, authorisation 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 authorisation schemes may be invoked to further constrain the roles that may define extensions for specific extension points. Altogether, future research should define a comprehensive authorisation 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 visualise 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.

7   Related Work

In this paper, *business process extensibility* is positioned as a new area of research within the well-explored field of *process flexibility*. A recent taxonomy in process flexibility Schonenberg et al. 2008 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 Schonenberg et al. (2008), 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 (Reichert et al. 2003a), YAWL (Aalst and Hofstede 2005) (including its Worklet service (Adams et al. 2006)), FLOWer (Aalst et al. 2005) and DECLARE (Pesic et al. 2007) are classified in Schonenberg et al. (2008) according to this taxonomy. However this taxonomy omits *configuration* as a design-time flexibility approach (configuration is discussed later on in this section).

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 Weber et al. (2008), while on the other hand the flexibility taxonomy gave rise to a collection of *flexibility patterns* (Mulyar et al. 2008). In Weber et al. (2008), the 17 change patterns refer to the ability of a system to provide high-level adaptations at the process model and the process instance levels, while the six change support features ensure performed changes are correct and consistent, traceable, and that changes are facilitated for users. However, the change patterns and features listed describe high-level change operations only, and lean towards those systems that allow for ad-hoc inclusions, exclusions and sequential changes that are applied by an administrator at runtime, for the most part manually. In Mulyar et al. (2008) it is claimed that the 'majority of' the change patterns can be 'mapped on' the flexibility patterns, although the authors do not further develop any correlation between the two pattern sets. 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 operationalise our ideas.

Issues of correctness revolve around questions of how to propagate changes without violating correctness and consistency constraints inherent in the process schemas, and how to synchronise concurrent changes. The construction of tests

that provide for the detection of potential conflicts between change at the type and instance levels is given in Rinderle et al. (2004b). While the authors concede that the formalisations are incomplete, they demonstrate the extent of the problems introduced when faced with the need to ensure correctness when propagating change. In Ly et al. (2006), the authors state it is essential to integrate semantic (or application) knowledge within any framework supporting process change in order to avoid semantic conflicts, a view that introduces problems such as how to formalise such knowledge, how to describe the notion of semantic correctness after change, how to support efficient verification and how to maintain the knowledge base. Such semantic constraints and their systemic management provide insights into how the approach proposed in this paper handles these issues.

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. (1995) or Aalst (2001) dealing with changed control-flow dependencies. A comparative overview of correctness criteria used by various approaches is presented in Rinderle et al. (2004a). Challenges inherent in the correct migration of a potentially large number of instances in various states, and possibly with various ad-hoc changes already applied, from the old to the new models, are described in Reichert et al. (2003b) and Rinderle et al. (2005). More recently, Rinderle-Ma et al. (2008) 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.

Kochut et al. (2003) present correctness criteria for concurrent process change within the scientific workflow domain. Their adaptation procedures are constrained by the evaluation of the

impact on running processes in terms of time, cost and quality. Changes are categorised as either ad-hoc (as a result of errors, rare events, special demands) or evolutionary (new strategies, reengineering efforts, alteration of external conditions).

A framework for the specification and validation of process constraints is given in Sadiq et al. (2005), with a view towards an appropriate balance between flexibility and control. Constraints are split into two main classes (*structural* and *containment*), which impose restrictions on how *fragments* (or extensions to a process model) can be composed, and on how they can be contained in the resultant process model, respectively. The approach relies on late-binding of fragments to parent model 'templates', but it is implemented by routing the pre-launching of each and every process instance to an 'expert' user to assemble the various process parts, and thus is untenable for organisations that manage large numbers of concurrent cases. However, the framework provided does present some formalisms for the deployment of the systematic approach to extensibility discussed in this paper.

Features of OSGi methodologies provide some insight for approaches to implementing extensibility conformance, extension points and stacked extensions. In Gu et al. (2004), a formal, ontology-based context model is used to develop an OSGi-based infrastructure for managed services. The model uses an ontology markup language called *OWL* to define interaction points between service components. The adoption of an OSGi architecture for the Eclipse 3.0 platform is discussed in Gruber et al. (2005), taking it from a proprietary to a rich client platform supporting a plug-in architecture. The architecture features a *manager agent* that is responsible for the correct installation of (plug-in) bundles, including resolving issues of configuration, dependencies, security and equivalence.

Reference models are models for targeted application domains that incorporate 'best prac-

tice' (Küster et al. 2006) 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 (Rosemann and Aalst 2007). 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 but only identified as a topic worthy of research (La Rosa 2009).

Methods for *Reference Process Patchability* will involve issues of migration and merging. A classic treatment of model comparison, difference discovery, merging and synchronisation can be found in Lippe and Oosterom (1992), which describes an approach called 'operation-based merging'. This method goes beyond the state-based, before and after comparisons, by recording all relevant operations performed in the form of *transformations* that occur between the existing (before) and modified (after) states. While aimed primarily at data objects, the operation-based merging framework presented offers a number of primitives that may be applied to the issue of patchability, so that efficiencies of support for conflict resolution and consistency between versions can be achieved.

The problem of specifically merging an arbitrary set of process models into a configurable reference model encompassing the behaviour of all the input models, has been investigated in La Rosa et al. (2010). Accordingly, a configurable reference model is created by taking all maximum common regions between a pair of process models, and connecting these regions with those process fragments that occur in one of the two input models only. This connection is achieved via variation points while annotations are assigned to each element in the merged model, to keep track of the input model the element in question originates from. In this way one is able to trace back from which input model(s) each element originates, as well as to derive the input process models from the generated reference model, by configuring the latter. This approach can provide a basis to operationalise the merging of different vendor and customer patches and to trace back each patch's originator.

An approach to tackling challenges dealing with a collection of so-called 'process variants' is documented in Hallerbach et al. (2008). 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.

8   Summary

This paper introduced the notion of process extensibility as a new paradigm for customising 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 introductory exposition to the notion of business process extensibility, drawing from situations and examples that were sufficient to differentiate it from the related area of process flexibility. The approach to extensibility as proposed in the paper was based on allowing concurrent changes to vendor reference business process models and corresponding customer deployments through well-defined extension points in models and conflict-free model merging. The limitation of *controlled flexibility* argued 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 synchronisation with the customers' derived models. By way of illustration, we sketched how the approach applies for well-defined requirements in the form of extensibility types along the control flow, data flow and resource perspectives of a process. Moreover, discussed how such an approach could be implemented in the context of the YAWL system.

With the initial framing of extensibility in view, we listed a number of open research challenges including the need for further types, reference process model conformance and patching, mining, and extension points and stacked extensions. Future work will investigate these different challenges with the goal of proposing more comprehensive support for practitioners. A major part of this will be to understand the different situations and requirements for extensibility and to understand and predict extension needs out of a variety of sources and potentially conflicting model updates. We also envisage that an area for fruitful development to apply extension points would be through action patterns (Smirnov et al. 2009), whereby extension points can be introduced in the context of predictive model editing.

References

Aalst W. (2001) Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change. In: Information Systems Frontiers 3(3), pp. 297–317

Aalst W., ter Hofstede A. (2005) YAWL: Yet Another Workflow Language. In: Information Systems 30(4), pp. 245–275

Aalst W., Weske M., Grünbauer D. (2005) Case Handling: A New Paradigm for Business Process Support. In: Data & Knowledge Engineering 53(2), pp. 129–162

Adams M., Hofstede A., Edmond D., Aalst W. (2006) Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In: Proc. of the 14th Int. Conf. on Cooperative Information Systems (CoopIS'06). Lecture Notes in Computer Science Vol. 4275. Springer, Berlin, pp. 291–308

Ellis C. A., Keddara K., Rozenberg G. (1995) Dynamic change within workflow systems. In: Proc. of the Conf. on Organizational Computing Systems, COOCS 1995, Milpitas, California, USA, August 13-16, 1995. ACM, pp. 10–21

Fettke P., Loos P., Zwicker J. (2006) Business Process Reference Models: Survey and Classification. In: Business Process Management Workshops. Lecture Notes in Computer Science Vol. 3812. Springer, Berlin

Gottschalk F., Aalst W., Jansen-Vullers M., La Rosa M. (2008) Configurable Workflow Models. In: Int. Journal of Cooperative Information Systems 17(2), pp. 177–221

Gruber O., Hargrave B., McAffer J., Rapicault P., Watson T. (2005) The Eclipse 3.0 Platform: Adopting OSGi Technology. In: IBM Systems Journal 44(2), pp. 289–299

Gu T., Pung H., Zhang D. (2004) Toward an OSGi-Based Infrastructure for Context-Aware Applications. In: IEEE Pervasive Computing 3(4), pp. 66–74

Hallerbach A., Bauer T., Reichert M. (2008) Managing Process Variants in the Process Life Cycle. In: ICEIS 2008 - Proc. of the Tenth Int. Conf. on Enterprise Information Systems, Volume ISAS-2, pp. 154–161

Hofstede A., Aalst W., Adams M., Russell N. (2010) Modern Business Process Automation – YAWL and its Support Environment. Springer

Kochut K., Arnold J., Sheth A., Miller J., Kraemer E., Cardoso J. (2003) IntelliGEN: A Distributed Workflow System for Discovering Protein-Protein Interactions. In: Distributed and Parallel Databases 13(1), pp. 43–72

Küster J. M., Koehler J., Ryndina K. (2006) Improving Business Process Models with Reference Models in Business-Driven Development. In: Business Process Management 2006 Workshops. Lecture Notes in Computer Science Vol. 4103. Springer, pp. 35–44

La Rosa M. (2009) Managing Variability in Process-Aware Information Systems. PhD Thesis, Queensland University of Technology, Brisbane, Australia

La Rosa M., Dumas M., Uba R., Dijkman R. (2010) Merging Business Process Models. In: Proceedings of the 18th International Conference on Cooperative Information Systems. LNCS Vol. 6426. Springer, Berlin

Lippe E., van Oosterom N. (1992) Operation-based merging. In: SIGSOFT Software Engineering Notes 17(5), pp. 78–87

Ly L., Rinderle S., Dadam P. (2006) Semantic Correctness in Adaptive Process Management Systems. In: Dustdar S., Fiadeiro J., Sheth A. (eds.) Proceedings of the 4th International Conference on Business Process Management. Lecture Notes in Computer Science Vol. 4102. Springer, Berlin

Mulyar N., Aalst W., Russell N. (2008) Process Flexibility Patterns. BETA Working Paper Series WP 251. Eindhoven University of Technology. Eindhoven. http://fp.tm.tue.nl/beta/publications/working\%20papers/Beta_wp251.pdf

Pesic M., Schonenberg M. H., Sidorova N., Aalst W. (2007) Constraint-Based Workflow Models: Change Made Easy. In: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated Int. Conf. Proc., Part I. Lecture Notes in Computer Science Vol. 4803. Springer, Berlin, pp. 77–94

Reichert M., Rinderle S., Dadam P. (2003a) ADEPT Workflow Management System: In: Proc. of the 1st Int. Conf. on Business Process Management 2003. Lecture Notes in Computer Science Vol. 2678. Springer, Berlin, pp. 370–379

Reichert M., Rinderle S., Dadam P. (2003b) On the Common Support of Workflow Type and Instance Changes under Correctness Constraints. In: et al. R. M. (ed.) Proceedings of the 11th International Conference on Cooperative Information Systems (CoopIS'03). Lecture Notes in Computer Science Vol. 2888. Springer, Catania, Sicily

Reichert M., Rinderle S., Kreher U., Dadam P. (2005) Adaptive process management with ADEPT2. In: Proc. of the Int. Conf. on Data Engineering Vol. 3716. IEEE Computer Society, pp. 1113–1114

Rinderle-Ma S., Reichert M., Weber B. (2008) Relaxed Compliance Notions in Adaptive Process Management Systems. In: Conceptual Modeling - ER 2008, 27th Int. Conf. on Conceptual Modeling. Lecture Notes in Computer Science Vol. 5231. Springer, Berlin, pp. 232–247

Rinderle S., Reichert M., Dadam P. (2004a) Correctness criteria for dynamic changes in workflow systems - a survey. In: Data & Knowledge Engineering 50(1), pp. 9–34

Rinderle S., Reichert M., Dadam P. (2004b) On Dealing with Structural Conflicts between Process Type and Instance Changes. In: Desel J., Pernici B., Weske M. (eds.) Proceedings of the 2nd International Conference on Business Process Management. Lecture Notes in Computer Science Vol. 3080. Springer, Berlin

Rinderle S., Weber B., Reichert M., Wild W. (2005) Integrating process learning and process evolution - a semantics based approach. In: Aalst W., Benatallah B., Casati F., Curbera F. (eds.) Proc. of the 3rd International Conference on Business Process Management. Lecture Notes in Computer Science Vol. 3649. Springer, Berlin, pp. 252–267

Rosemann M., Aalst W. (2007) A Configurable Reference Modelling Language. In: Information Systems 32(1), pp. 1–23

Sadiq S., Orlowska M., Sadiq W. (2005) Specifi-

cation and Validation of Process Constraints for Flexible Workflows. In: Information Systems 30(5), pp. 349–278

Scheer A., Nüttgens M. (2000) ARIS: Architecture and Reference Models for Business Process Management. In: van der Aalst W. M. P., Desel J., Oberweis A. (eds.) Business Process Management. Lecture Notes in Computer Science Vol. 1806. Springer, Berlin, pp. 366–379

Schonenberg H., Mans R., Russell N., Mulyar N., Aalst W. (2008) Towards a Taxonomy of Process Flexibility. In: Proc. of the CAiSE Forum, pp. 81–84

Smirnov S., Weidlich M., Mendling J., Weske M. (2009) Action Patterns in Business Process Models. In: ICSOC 2009. Lecture Notes in Computer Science Vol. 6275. Springer, Berlin, pp. 115–129

Weber B., Reichert M., Rinderle-Ma S. (2008) Change patterns and change support features - Enhancing flexibility in process-aware information systems. In: Data & Knowledge Engineering 66(3), pp. 438–466

Sören Balko

SAP AG
Walldorf
Germany
Soeren.Balko@sap.com

**Arthur H.M. ter Hofstede, Marcello La Rosa,
Michael Adams**

Queensland University of Technology
Brisbane
Australia
{a.terhofstede | m.larosa | mj.adams}@qut.edu.au

Alistair Barros

SAP Research
Brisbane
Australia
Alistair.Barros@sap.com