SOCA (2016) 10:253–271
DOI 10.1007/s11761-015-0185-y

ORIGINAL RESEARCH PAPER

Patterns and tools for business process monitoring customization

Marco Comuzzi[1] · **Samuil Angelov**[2]

Received: 5 September 2014 / Revised: 6 October 2015 / Accepted: 14 October 2015 / Published online: 16 November 2015

Abstract In a cross-organizational service-based process provisioning scenario, one provider is likely to execute a given business process to serve several customers. Each customer may hold different expectations about the way this process can be monitored. We present a solution allowing the provider to support the requirements of different customers on the monitoring of a given process, i.e., offering them the opportunity to customize the way a process will be monitored. We propose a multi-dimensional classification model of patterns for process monitoring and rules to compose the patterns to design customized monitoring infrastructures. The fit for purpose of the patterns is evaluated empirically, whereas the feasibility of our solution is demonstrated by a tool supporting process monitoring customization adhering to our pattern design and composition methodology.

Keywords Business process · Monitoring · Customization

1 Introduction

Only agile businesses, which can flexibly redesign or reconfigure their operations and processes, can survive in the currently rapidly evolving social and economic ecosystem. Organizations often implement agility and flexibility through simplification, focusing on their core business, and engaging in collaborations with partners to maintain and possibly improve the required level of quality and cost-effectiveness in their business processes [35].

In such a scenario, customization plays an important role. Individually tailored collaborations between providers and clients strengthen business relationships, improving customer satisfaction and productivity by limiting the amount of adaptation required by the customer organization to engage in the collaboration. Customization is also likely to increase the degree of control perceived by customers over outsourced processes, reducing their *opaqueness* [21].

Several aspects of business processes can be customized, such as QoS, resource allocation, control flow, or monitoring. Monitoring, in particular, can be leveraged to reduce the perceived opaqueness of outsourcing relationships, since it allows gathering information that could be used to exert control over the monitored entity. As enabler of control over externalized business processes, customers may require monitoring for a variety of internal business concerns, from assessing the satisfaction of established contracts and SLAs [31], to synchronizing their own internal business processes [36,40] or predicting the quality of the collaboration with a given partner in the future [30].

As an example, we can consider a retail customer and a trader outsourcing their electronic portfolio management to a financial institution. Both customers are interested in the same processes, e.g., buying/selling specific financial products on the stock exchange. Traders may have very strict requirements on the performance (QoS) of the process, since they have to trade large quantities in a short amount of time, and may want to monitor closely the QoS of the process over time. Retail customers, while not particularly interested in the performance of the process, may be interested in the level of security of the financial products, and may require

✉ Marco Comuzzi
marco.comuzzi.1@city.ac.uk

Samuil Angelov
s.angelov@fontys.nl

[1] Centre on Adaptive Computing Systems, City University London, London, UK

[2] Software Engineering Team, Fontys Hogeschool ICT, Eindhoven, The Netherlands

the financial institution to add additional security controls to satisfy this requirement.

According to the service-oriented computing paradigm, collaborations are implemented in the form of business process provisioning, whereby a provider organization executes one or more (service-based) business processes for one or more customer organizations [4,6,14]. Customization of monitoring, in this context, requires focus on both control flow aspects, specifying the way to capture monitoring information and make it available, and resources, specifying what has to be monitored and at which stage of the process.

Currently available Process-Aware Information Systems (PAIS) offer limited capabilities for process monitoring customization. PAIS may provide standard consoles or monitoring APIs to access process information, which require customers to develop their own specific monitoring application logic [14,30,36]. This has several drawbacks for both the provider and the customer. Besides the need to develop and maintain monitoring applications, customers may also need to change such implementations whenever the providers' monitoring API changes, no longer complying with the customers' internal needs. Providers may be prevented to smoothly run any change or upgrade of their own internal process enactment technology, since this may lead to unsatisfied customers, no longer able to monitor the execution of the service provider processes.

In this regard, we argue for the need of an analysis of the business process monitoring concern that separates the monitoring requirements of the customer organizations, i.e., *what* has to be monitored, from the way process providers configure or extend their business process technology to satisfy these requirements, i.e., *how* monitoring is implemented. This enables the provisioning of process monitoring *as-a-Service*, that is, customer organizations will access the monitoring data that they need without having to deal with predefined sets of API made available by the process provider. While this may limit the control of monitoring clients over the monitoring options made available by the provider, clients massively gain in terms of cost reduction, reduced time to get the monitoring infrastructure up and running, and always up to date monitoring services, with internal APIs and software upgrades run in the background by the monitoring provider.

In this paper, we devise a coherent framework to support business process providers in the provisioning of customized business process monitoring infrastructures to their customers. The contributions of this paper are:

- the definition of a set of patterns capturing the variability of customer requirements about process monitoring;
- a methodology to compose patterns to design customized monitoring infrastructures;
- a proof-of-concept implementation of our pattern design and composition methodology.

The patterns support the provider in identifying the solution space for process monitoring customization. They are positioned in a multi-dimensional classification model, constructed by reasoning on the literature about software programs, Web service, and business process monitoring. Patterns are specified using colored Petri net (CPN), and the composition of patterns occurs through a set of formal rules, exploiting the concepts of node fusion in Petri nets.

We assess the appropriateness, or fit for purpose, of the patterns to capture process monitoring requirements using an empirical study involving process designers, whereas we devise a tool to demonstrate the feasibility of our solutions. The tool, in particular, demonstrates the automated generation of customized monitoring infrastructures starting from customer requirements specified using our classification model. The implementation exploits the principles of customization in software product lines and, more specifically, Feature-Oriented Software Development (FOSD).

The paper is organized as follows. A model of the possible set of options for process monitoring customization is presented in Sect. 2. Section 3 introduces a running example taken from a real-world application scenario. Patterns for the identified options are specified in Sect. 4. Section 5 presents the composition rules to design monitoring infrastructures, whereas Sect. 6 presents the empirical evaluation and the implemented tool. Section 7 discusses the related work, and eventually, we draw our conclusions and discuss future work in Sect. 8.

2 A model of process monitoring customization

In this section, we first discuss a generic conceptual model of the business process monitoring life cycle. Then, we propose a multi-dimensional model of the possible options characterizing process monitoring that can be offered by process providers to customers.

By conceptually separating the monitoring options model from the implementation patterns, we decouple the choices available to the customer to customize monitoring infrastructures from their implementation. In this way, as long as the monitoring options model does not change, providers can revise or upgrade the options implementation transparently in respect of their customers and, in particular, the way they use the provided monitoring information.

2.1 Business process monitoring life cycle

Figure 1 depicts the general outline of a monitoring architecture. In the business process provider domain, a monitoring infrastructure (also called sensor or observer [15,46]) cap-

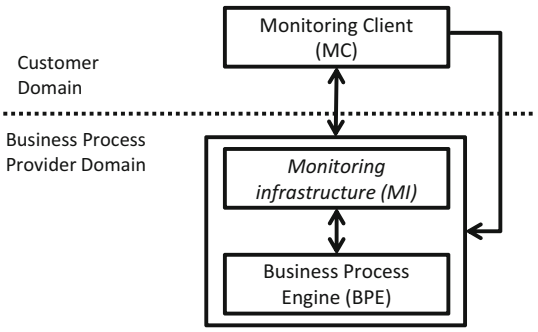


Fig. 1 Business process management architecture

turing relevant information is built around a (service-based) business process engine. A business process engine is meant in a broad sense—it is a component producing process information (activity or process states, data values, etc.). As we will clarify later while presenting the tool implementation, the restrictions imposed by our approach on the underlying *baseline* process engine are satisfied by current business process management technology.

Although the monitoring infrastructure can be built in (or even be an integral part of) the business process engine [15], conceptually, it is a separate entity [46]. Considering it as a separate entity promotes separation of concerns with respect to the underlying process engine, leading to a more focused and context-independent analysis of the monitoring infrastructure component and its possible customization. In the customer (also called controller [46]) domain, a monitoring client obtains the information captured by the monitoring infrastructure. It processes the information obtained and if desired exerts control over the business process engine.

In many scenarios, the monitoring client can be an intermediary for the actual business entity that requires the monitoring information. Monitoring may take place within an organization (between independent business units) or in cross-organizational settings. Furthermore, the client of the monitoring infrastructure may be an autonomous application or a human user. As these cases do not introduce any specifics in our work, the remainder of the paper abstracts from them, considering only the general scenario of Fig. 1.

2.2 Conceptual model of process monitoring options

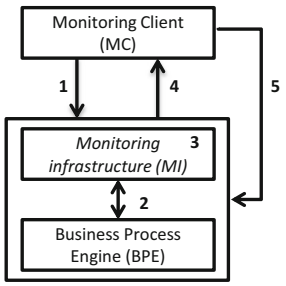
The dimensions of our framework identify what parts of the scenario in Fig. 1 can be customized by the monitoring client, i.e., they define the monitoring solution space for the customer of the business process. The first two dimensions concern the context in which monitoring has to occur. In particular, we consider the monitoring variable and the anchoring points, defined as follows.

Monitoring variable (**mv**) The monitoring variable mv specifies the object of monitoring, i.e., the process information that the monitoring infrastructure (MI) obtains from the business process engine (BPE) and makes available to the Monitoring Client (MC). It has a domain, which specifies the (range of) values that it can assume, and a unit of measure, if needed. In the world of software programs monitoring, the monitoring variable is the target, for instance, of a watch for debugging. In business process monitoring, the monitoring variable may range from infrastructure-level data, such as timestamps of service calls [6], to application-level process data, such as the status of an activity or domain-specific data produced by an activity [42].

Note that in this paper we consider instance-level business process monitoring. Given a process, monitoring is restricted to variables related to a specific instance started by customers. For example, the status of a particular activity (in a given process instance) is a monitoring variable, which can assume values such as *waiting*, *executing*, *terminated*, and *faulted*. A value of the monitoring variable represents a single data element captured by MI during the execution of one instance of the business process, e.g., the timestamp of an order, the warehouse level at a specific point in time,or the unique id of a user executing a specific activity. Captured values can be stored by MI during the execution of a business process and made available in batches to MC. Instance-level monitoring is usually opposed to cross-instance monitoring, where monitoring values are defined across a set of instances, e.g., the average execution time of an activity across all instances started by the same customer [12].

Anchoring point (**ap**) The MI is enabled within a specific scope of the process to be monitored. Anchoring points specify the scope of the process within which the MI is enabled. The notion of anchoring point derives from the literature on software program monitoring, where running the monitoring program in the same memory space of the monitored program can be costly, and therefore, the monitoring program has to be enabled only when strictly necessary [16]. In a business setting, while in many cases it can be assumed that monitoring is permanently enabled, defining anchoring points is helpful when capturing and making available the values of the monitoring variable to the client is very costly. Intrusive process monitoring [6] is an example of this scenario, since the execution of intrusive monitoring statements blocks and, therefore, delays, the execution of the process. In such a scenario, the MC may want to enable monitoring only when strictly necessary. In the remainder, we refer to apStart and apEnd as the anchoring points enabling and disabling the monitoring infrastructure (MI), respectively.

Figure 2 refines the conceptual outline of the monitoring architecture by showing the life cycle of communications among the different elements of the architecture. We use the phases of this life cycle to derive the next three dimensions of our framework. In particular, the first phase concerns commanding the acquisition of monitoring data. This can be done



Lifecycle Phase	*Customizable Monitoring Dimension*
1. Monitoring Trigger.	**tri**
2. Obtain data from BPE.	(not customizable)
3. Manage data.	**man**
4. Supply data to MC.	**sup**
5. Exert control.	(out of scope)

Fig. 2 Business process management life cycle

either by MC or by MI (Phase 1). For instance, the client may specify that the monitoring infrastructure must acquire values of a specific monitoring variable periodically or may want to be allowed to command this acquisition proactively. The second phase concerns MI obtaining the required data from BPE (Phase 2). Since this phase is internal to the business process provider domain and cannot be customized, it is not considered further in this paper. Then, the monitoring infrastructure may have to manage, e.g., store in batches, the monitoring data obtained by BPE (Phase 3) before supplying them to the client (Phase 4). Eventually, the client processes the monitoring data and may decide to exert control on the monitored process executed by the business process engine (Phase 5). This last phase is out of scope in this paper.

Phases 1, 3, and 4 of the life cycle in Fig. 2 are characterized by one respective monitoring dimension in our framework, since they involve aspects that are customizable by the client. Each dimension has a set of options. Options represent the solution space for the customization of process monitoring, that is, a customized monitoring infrastructure can be built once the customer has chosen one option for each possible monitoring dimension.

In the remainder of this section, we present the monitoring dimensions and their possible options.

Monitoring trigger The monitoring trigger phase is characterized by one dimension identifying the entity commanding the acquisition of values of mv (tri). The acquisition of a monitoring value can be triggered by MC or by MI. In the former case (tri=mcTrg), MC makes a request to the provider for commanding the acquisition of the value of MV. In the latter case (tri=miTrg), MI acquires values of MV proactively. Note that in this case MC should be able to customize the way in which MI acquires values, for instance periodically at a certain frequency or on change. The third option (tri=mixTrg) fits cases in which acquisition is triggered by MI, e.g., periodically, but also the client MC wants to be allowed to request acquisition. Once acquisition is triggered, MI will obtain monitoring data from BPE.

Manage data For this life cycle phase, we define one monitoring dimension (man), which captures the monitoring infrastructure's logic in managing the values obtained from the process engine before supplying them to the client. New values obtained from BPE can rewrite old values captured for the same mv or the obtained values can be stored (persisted), e.g., to produce historical series of values of mv. When supplied to the client, values can be consumed, i.e., they will not be available in the future to the client, or they can just be read, remaining available also in the future. Thus, we identify four options for man, i.e., (1) rewrite-consume (man=manRc), (2) rewrite-read (manRr), (3) persist-consume (manPc), and (iv) persist-read (manPr). Figure 3 exemplifies the behavior resulting from the four options available for this pattern in the case of a generic monitoring value with integer values.

The monitoring infrastructure can also be instructed to keep only those monitoring values satisfying a specific constraint, such as being above/below a given threshold, or being included in a given set of specific values. The application of these constraints is orthogonal to the man dimension and, therefore, can be applied to any of the four options identified above.

Supply data to MC This phase is characterized by the dimension sup establishing the direction of the interaction between the monitoring infrastructure and the client. For sup, we define the options push, pull, and mix. The option push models cases in which the monitoring infrastructure pushes monitoring values to the client, whereas pull models cases in which the monitoring infrastructure sends values of mv only after having received a request from the client. Similarly to the tri dimension, the option mix combines the push and pull options: monitoring values are pushed to the client according to a certain policy, but the client is still allowed to pull monitoring values when needed.

Note that this dimension is conceptually clearly separated from the monitoring trigger dimension discussed before. For example, a client may require to specify explicitly when the value of a monitoring variable must be collected by the business process engine, i.e., tri=mcTrg, and may either be notified of the batch of collected values at a specified frequency from the process engine (sup=push) or on demand (sup=pull).

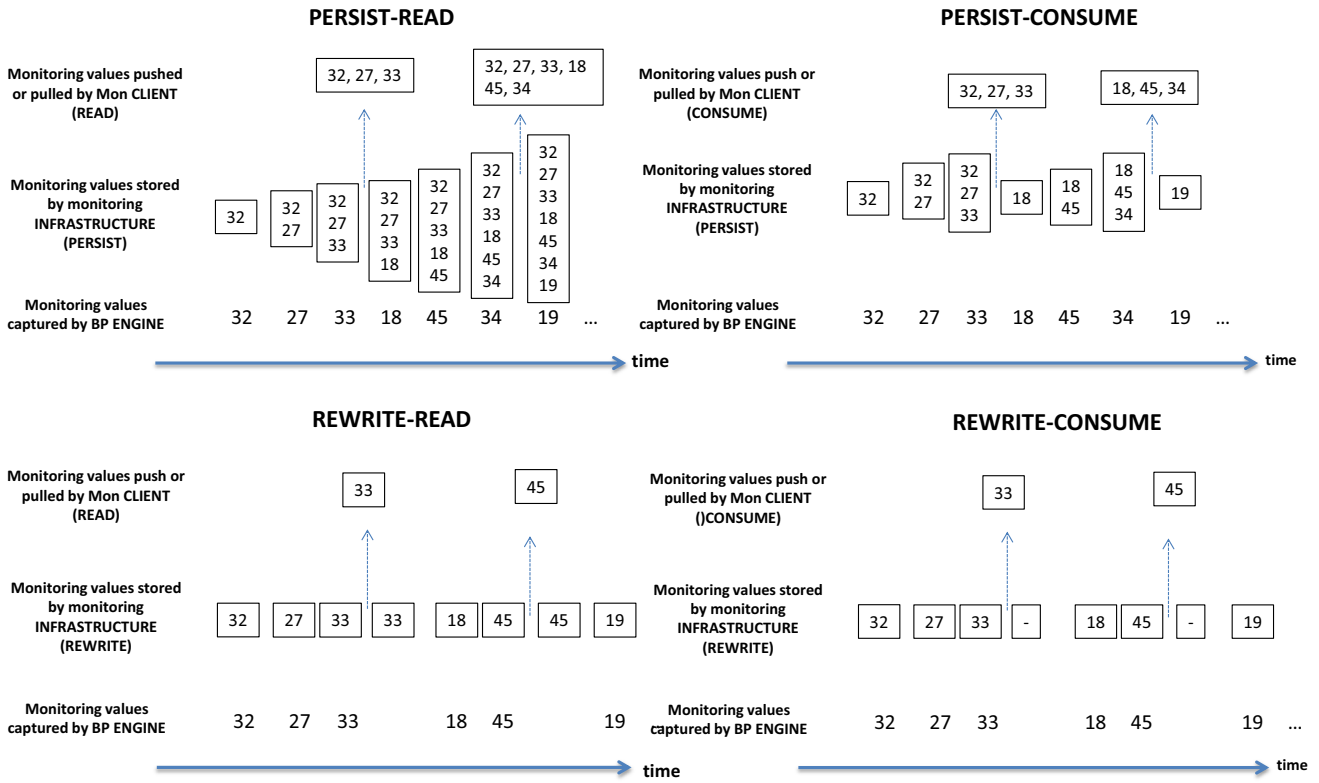


Fig. 3 Options for the manage data dimension—example

Generally, communication between the monitoring infrastructure and the client is a distributed systems communication issue and its customization may require the definition of additional dimensions, such as space and time decoupling options [1,17]. However, since those aspects are not monitoring specific, we do not further discuss them here.

3 Running example

In this section, we present an example of an online advertising scenario, which is adapted from a real-world business case described in greater detail in [2].

Figure 4 shows the excerpt of the process considered in this paper. The advertising provider (a newspaper's Web site) offers advertising space to companies (customers). The customer sends the advertisement to the provider and when the time agreed to start the campaign comes, the provider starts the advertising campaign by publishing the ad in the newspaper. When the budget of the customer is depleted and/or the number of agreed appearances of the ad is reached, the campaign ends. The customer may ask to change the campaign if they observes that the ad is not reaching the target audience, the campaign has little impact, etc.

In a traditional advertising setting, the provider offers a fixed set of tools to all customers for monitoring their campaigns. For example, the customers can monitor the IP addresses of the readers seeing their ad, the number of shows of their ad, number of clicks on their ad, the spots where the ad has been published (banner, column, pop-up, in-text). Typically, customers have to access their account on a management system made available by the provider to access this information. The provider builds such system using a standard monitoring console.

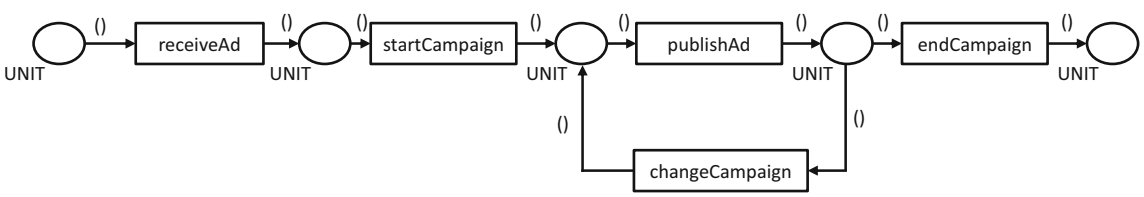


Fig. 4 Online advertising process

This monitoring advertising scheme does not address the individual preferences of the customers. Customers may be interested in receiving the monitoring information directly instead of having to query it from the provider; they may be interested in obtaining the information in an aggregated format at the end of the campaign or be constantly updated to be able to adapt their campaign, or they may be interested, motivated by a cheaper price, in obtaining only some of the monitoring information instead of monitoring all possible variables.

For brevity, we focus here on the construction of monitoring infrastructures for only one monitoring variable, i.e., the number of clicks on an ad, which is the most straightforward indicator of the ad success and profitability over time. To apply the framework for the customization of a mv, the provider has to consider the possible options from the monitoring dimensions presented in Sect. 2 to offer to the customers. The mv in this case is the current value of number of clicks on the ad of the customer. We assume that the provider (newspaper), through an advertising engine, keeps track continuously of this value. The customer wants to monitor mv along the campaign. Therefore, the anchoring point enabling and disabling monitoring are the start and the end of the campaign, respectively. Note that, in principle, each customer may choose a different anchoring point, for instance enabling monitoring only after the campaign has been changed a first time.

Having the framework as a guiding tool in the set of possible monitoring options, the provider can define all possible monitoring options for the number of clicks mv. A customer may like to be pushed monitoring information or to pull it (thus, sup=push or sup=pull). The customer may prefer to monitor the mv only at a specific point in time that cannot be revealed (e.g., it is not known) to the provider (tri=mcTrg), but may also like to delegate the acquisition of the monitoring values to the provider at a pre-agreed time, e.g., periodically (tri=miTrg). Typically, customers would prefer to have all the monitored values stored by the provider, so that these can be queried any time later on and used to analyze the number of clicks trend over time (man=manPr, persist-read). If storage space is crucial for the provider, they may offer some incentives (e.g., financial) to the customers to choose also man=manPc (persist-consume).

Each customer interested in monitoring information on the cumulative number of clicks is presented with a set of possible options. Table 1 presents two possible sets of choices for customers A and B. Customer A delegates the acquisition of monitoring values to the provider at the beginning of the campaign, e.g., every 6 h, wants to be able to pull monitoring information when needed, and also requires that the information is kept after it is read to be able, for instance, to show the trend of this mv over time as soon as the information is pulled from the monitoring infrastructure. Thus, each time customer A pulls monitoring information, they will get a list of sampled (one every 6 h) values of the number of clicks on their ad since the publication of the ad. Customer B wants to be able to specify when the values of number of clicks have to be acquired (tri=mcTrg) and to automatically receive the monitoring information, for instance as soon as this is acquired by the provider or at the end of each day (sup=push, where the trigger for the push is the availability of a new value for number of clicks mv or the end of the day). Customer B also does not require the provider to locally store the acquired values (man=pc). This is a reasonable assumption, for instance, when data are directly pushed to the client as soon as they are acquired by the provider.

Table 1 Monitoring options selected by customers A and B

Monitoring option	Customer A	Customer B
Monitoring variable	No. of clicks on Ad banner	
Anchoring points	startCampaign, endCampaign	
Monitoring trigger	tri=miTrg	tri=mcTrg
Manage data	man=manPr	man=manPc
Supply data	sup=pull	sup=push

As different customers may choose different values in each dimension (as demonstrated in Table 1), the provider has to be prepared to support each possible combination. Having the patterns for each monitoring dimension values predefined in the framework, the provider can now directly apply them for these specific mv and anchoring points.

4 Patterns for process monitoring options implementation

In this section, we present the patterns for the implementation of the options identified in Sect. 2. Note that patterns, similarly to the more general case of workflow patterns [44], in this context must be intended as an analysis of various options that need to be supported by business process technology to enable the customization of the monitoring dimension. This is in contrast with the definition of traditional software design patterns, which aim at providing a general reusable solution to a commonly occurring problem in software design [19].

Our patterns capture the design of internal data and control flows of the monitoring infrastructure (MI). This is required by the provider to offer customization monitoring capabilities to the customers. Following common principles in the design of distributed systems [46], we first define the interfaces between MI and MC, and MI and BPE, respectively, to support the monitoring dimensions. Then, we propose a pattern for each option characterizing the monitoring dimensions.

Patterns specify the data and control flow of MIs internal implementation of the interfaces.

For business process modeling purposes, we use colored Petri nets (CPNs) [1,26]. CPNs have been chosen because they have a graphical representation, they have mature and freely available tool support, e.g., CPN Tools, and they have a precise semantic that can be translated to or directly implemented in other languages, e.g., Event-driven Process Chains [38] or YAWL [42], used by commercial and non-commercial workflow engines.

We use two token colors in our patterns, i.e., UNIT and MV. UNIT represents the default color of black tokens, and it is used to model the control flow in our modeling patterns. Tokens of color MV represent a single value of the monitoring variable MV. The precise definition of the color MV, e.g., possible values and unit of measure, is part of the definition of the monitoring context, and we do not further discuss it in this paper.

Figure 5 shows the interfaces between the elements of the monitoring solution MI (monitoring infrastructure), MC (monitoring client), and BPE (business process engine). Note that, although aspects related to the interaction between MC and BPE are internal to the provider domain and therefore not subject to possible customization, considering the interfaces between MI and BPE allows separation of concerns between business process execution and monitoring and, therefore, for more robust patterns.

MI requires one interface to supply data to MC (supply). Optionally, MI may require also an interface to receive supply requests from MC (supplyReq) and to receive acquisition requests (acquireReq). This interface is necessary only when MC decides to pull monitoring data from MI. The interface between MI and BPE is constituted by a generic interface for receiving monitoring values (obtain), by the two anchoring points, and by the interface bpeAcquireReq, allowing MI to command acquisition of values of MV by BPE. The interfaces monStart and monEnd of BPE represent the activities in the process enabling and disabling monitoring, respectively. Note that Fig. 5 does not show the color of tokens in places connecting interfaces. As discussed later, the color of these places is determined by the modeling pattern realizing the MI's internal implementation of the corresponding interface(s).

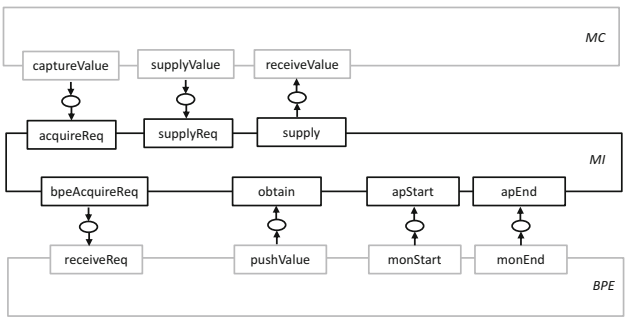


Fig. 5 Interfaces for process monitoring

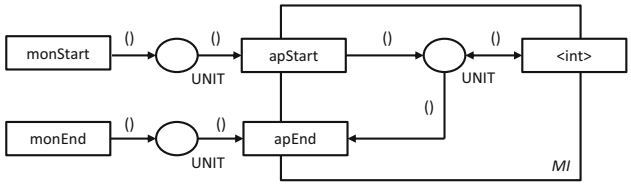


Fig. 6 Pattern for anchoring points

Concerning the context, MI is required to expose interfaces to define the anchoring points of monitoring to the business process executed by BPE. The tokens required to fire the transitions apStart and apEnd are produced by BPE when the process execution reaches the activities enabling and disabling the monitoring, respectively, labeled monStart and monEnd in Fig. 5.

Figure 6 shows the pattern for the implementation within MI of the anchoring point business logic. Specifically, the generic interface < int>, which can be any of the ones defined in Fig. 5, becomes enabled after the apStart transition has fired and is no longer enabled after the apEnd transition has fired. The remainder of this section presents the monitoring patterns for the remaining monitoring dimensions. Each pattern models one option of one monitoring dimension identified in Sect. 2.

Monitoring trigger The patterns modeling the options of dimension tri implement the acquireRec and bpeAcquireReq interfaces of MI. The patterns corresponding to the three values miTrg, mcTrg, and mixTrg are shown in Fig. 7. In Fig. 7a, the transition intTrigger captures MIs internal business logic to trigger acquisition, e.g., periodically or on change, which can be customized by MC. In Fig. 7b, the acquisition is commanded by MC, by firing the transition acquireReq, whereas Fig. 7c shows the mix case, i.e., MC or MI can both trigger an acquisition request.

Supply data to MC The patterns for the push, pull, and mix options of the dimension sup are shown in Fig. 8a, b, and c, respectively. The push option requires MI to only expose the supply interface. The transition supTrigger fires accordingly to the logic chosen by the customer to receive monitoring values. The customer, for instance, may require monitoring values periodically. For the pull option, the supply interface can fire, i.e., supplying a monitoring value, only after a request is received. The mix option combines the logic of the other two options.

Manage data The patterns for the man dimension implement the connection between the interfaces supply and obtain exposed by MI. Figure 9 shows the patterns for the four options. Note that the place bpe stores values of MV ready to be obtained by MI, whereas the place mc stores the

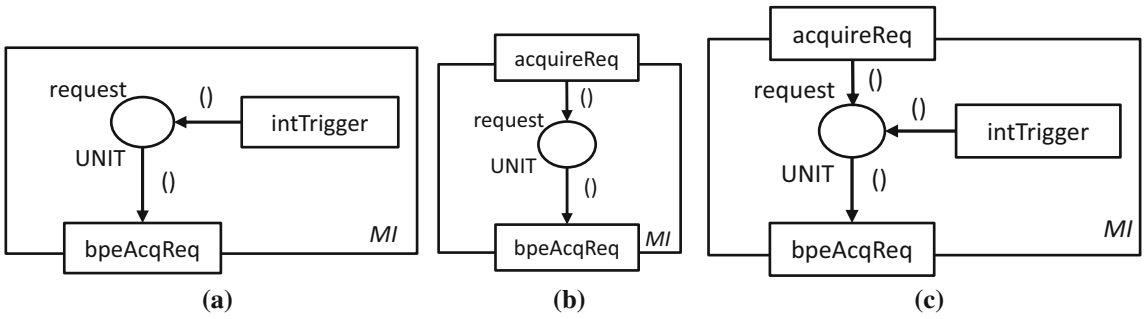


Fig. 7 Patterns for tri options. **a** miTrg, **b** mcTrg, **c** mixTrg

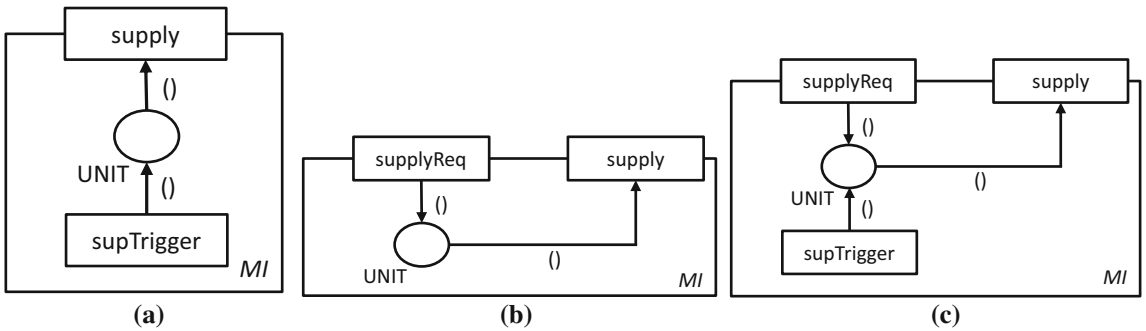


Fig. 8 Patterns for sup options. **a** push, **b** pull, and **c** mix

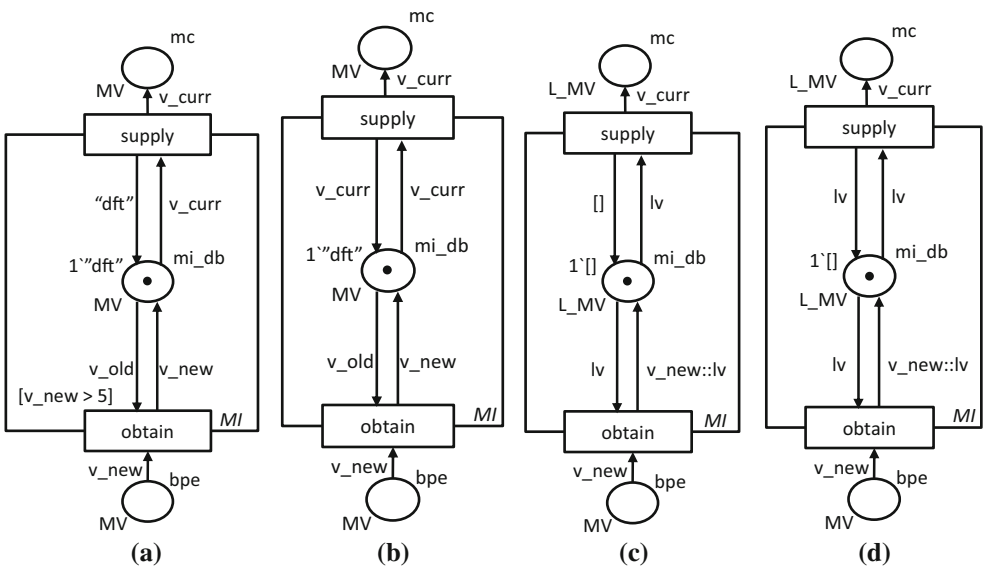


Fig. 9 Patterns for man options. **a** Rewrite-consume (manRc), **b** rewrite-read (manRr), **c** persist-consume (manPc), **d** persist-read (manPr)

value or set of values supplied to the monitoring client MC. Also, note that the color LMV represents a list of tokens of color MV.

In the rewrite-options, the new value of MV v_new always replaces the old value v_old. In the persist-options, the values of MV are stored by MI in a list lv (the list in this case represents a generic data structure); a new monitoring value v_new is simply inserted into lv. In the *consume* options, the supply transition, when fired, replaces the current monitoring value stored by MI with the default, whereas in the *read* options the monitoring value is put back in the place mi_db after being read and can be read again in the future by MC. Note that the default monitoring value is the empty list [] for the persist-options and the token with value dft of color MV for the rewrite-options.

Orthogonal constraints on the monitoring values obtained by MI are modeled as guards on the transition obtain. Such constraints are modeled as guards on the obtain transition. Figure 9a, in particular, shows the case in which MI obtains only the values of v_new greater than 5.

5 Composing patterns for monitoring infrastructures design

The patterns described in the previous sections must be properly composed by the provider to design a customized monitoring infrastructure for a customer organization. Composition refers to both coupling patterns to the process to be monitored and coupling patterns among each other to support the customer monitoring requirements. The former refers to filling in the parameterized transition <INT> of Fig. 5 to enable and disable the monitoring infrastructure. The latter refers to appropriately *merging* transitions with the same label in different patterns to obtain a coherent monitoring infrastructure.

In this section, we describe the set of rules for combining the patterns and how these are applied in our running example (see Sect. 5.2).

5.1 Composition rules

Since we model the operational implementation of our patterns using CPNs, we rely on the literature about Petri net composition and structuring to derive our composition rules. Composition and structuring in Petri nets are a prominent research issue, mainly owing to the fact that Petri net models tend to become very complex, in terms of number of elements and their connections, even for simple modeling tasks [22]. There are two approaches to Petri net composition, i.e., fusion and folding [7,22]. The former aims at simplifying models by allowing the interconnection of several Petri nets, similarly to the concepts of module linking and class composition in programming languages. The latter focuses on hierarchical decompositions of Petri nets, similarly to the concepts of macros and procedures in programming languages. In this paper, we propose rules following the fusion paradigm, as our patterns do not embed hierarchical structure, but they must rather be interconnected, or *fused*, to design a complete customized monitoring infrastructure. Net composition can occur through both transition and place fusion. For expressing rules, we use the following syntax, defined in [22]:

$$\begin{aligned}\mathsf{NewNet} = (&\mathsf{OperandNet}_1 \oplus \cdots \oplus \mathsf{OperandNet}_P)\\ &(\mathsf{node}_{1_1}/\mathsf{node}_{1_2}/\cdots/\mathsf{node}_{1_n} \mapsto \mathsf{newNode}_1,\\ &\ldots, \mathsf{node}_{p_1}/\mathsf{node}_{p_2}/\cdots/\mathsf{node}_{p_m} \mapsto \mathsf{newNode}_k).\end{aligned}$$

where $\mathsf{OperandNet}_p$, with $p = 1, \ldots, P$ are the nets to be composed and node can be places or transitions. We will use a dotted notation, e.g., $\mathsf{node}_{1_p} = = \mathsf{OperandNet}_1.\mathsf{node\_name}$, to avoid ambiguity on places and transition labeling when required.

Pattern composition occurs in three steps, leading to three different set of rules:

1. Anchor MI to the process to be monitored (which leads to the subnet ProAp);
2. Compose the patterns for the dimensions tri, man, and sup with the anchoring points pattern (subnets TriAp, ManAp, SupAp, respectively);
3. Compose the subnets obtained at steps one and two into a complete monitoring infrastructure properly anchored to the process to be monitored (subnet MI).

Note that the above steps are not commutative, that is, the composition process as specified above leads to the correct design of customized monitoring infrastructures only when the steps are applied in the specified order.

Step 1 concerns defining the rule to anchor the MI to the process to be monitored. For defining this rule, the process is abstracted to the transitions monStart and monEnd, that is, the transitions starting and ending the monitoring activity, respectively, as specified by the customer. The rule is reported in Eq. 1, and the process and the anchoring point patterns are fused using transition fusion. Note that the subnet obtained through the application of the rule is labeled ProAp.

Figure 10 exemplifies the application of the rule in our running example, where the user specified to start and terminate the monitoring at the activities startCampaign and endCampaign, respectively.

$$\begin{aligned}\mathsf{ProAp} = (&\mathsf{process} \oplus \mathsf{ap})\\ &(\mathsf{process.start/ap.monStart} \mapsto \mathsf{ProAp.apStart},\\ &\mathsf{process.end/ap.monEnd} \mapsto \mathsf{ProAp.apEnd}).\end{aligned} \quad (1)$$

Step 2 requires a set of rules specifying the composition of anchoring points with all possible patterns identified for the dimensions tri, man, and sup.

Equation 2 shows the rule for composing the anchoring point pattern and the mcTrg option of the tri dimension. Note that in our rules we label the subnet representing patterns using the same names adopted in Sect. 4.

The application of the rule is exemplified by Fig. 11. Note that one anchoring pattern ap[i] is required to be fused for

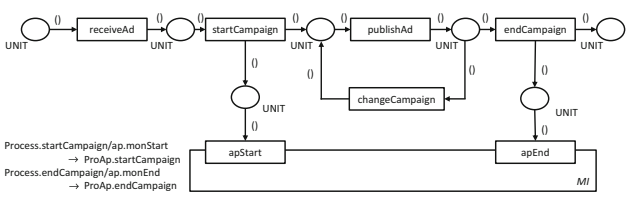


Fig. 10 Fusion of process and anchoring points

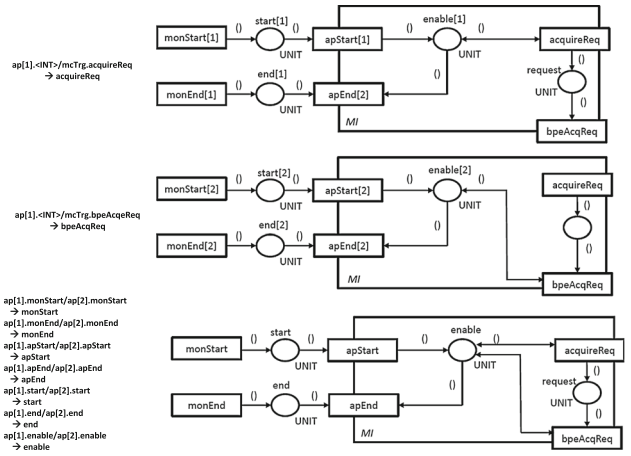


Fig. 11 Step-by-step execution of the anchoring of the tri=mcTrg pattern to the monitored process

each of the transitions controlled by the anchoring point, i.e., acquireReq (i=1) and bpeAcquireReq (i=2) in this case. As a result of the composition, a single anchoring point will control the enabling and disabling of all relevant transitions. The subnet obtained is labeled TriAp. The rule for the miTrg option is shown in Eq. 3. Note that this rule is simpler as only the interface bpeAcquireReq has to be enabled. The rule for the pattern mixTrg is the same reported in Eq. 2, as this pattern also involves the two interfaces acquireReq and bpeAcquireReq.

$$\begin{aligned}\text{TriAp} = (&\text{ap}[1] \oplus \text{ap}[2] \oplus \text{mcTrg})\\ &(\text{ap}[1].\ <\text{int}>/\text{mcTrg.acquireReq} \mapsto \text{TriAp.acquireReq},\\ &\text{ap}[2].\ <\text{int}>/\text{mcTrg.bpeAcquireReq}\\ &\mapsto \text{TriAp.bpeAcquireReq},\\ &\text{ap}[1].\text{apStart}/\text{ap}[2].\text{apStart} \mapsto \text{TriAp.apStart},\\ &\text{ap}[1].\text{apEnd}/\text{ap}[2].\text{apEnd} \mapsto \text{TriAp.apEnd},\\ &\text{ap}[1].\text{monStart}/\text{ap}[2].\text{monStart} \mapsto \text{TriAp.monStart},\\ &\text{ap}[1].\text{monEnd}/\text{ap}[2].\text{monEnd} \mapsto \text{TriAp.monEnd},\\ &\text{ap}[1].\text{start}/\text{ap}[2].\text{start} \mapsto \text{TriAp.start},\\ &\text{ap}[1].\text{end}/\text{ap}[2].\text{end} \mapsto \text{TriAp.end},\\ &\text{ap}[1].\text{enable}/\text{ap}[2].\text{enable} \mapsto \text{TriAp.enable}.) \qquad (2)\end{aligned}$$

$$\begin{aligned}\text{TriAp} = (&\text{ap} \oplus \text{miTrg})\\ &(\text{ap}.\ <\text{int}>/\text{mcTrg.bpeAcquireReq}\\ &\mapsto \text{TriAp.bpeAcquireReq}.) \qquad (3)\end{aligned}$$

Equation 4 shows the rule for composing the anchoring point pattern and the sup=push pattern. This rule composes the anchoring point the supply pattern by fusing the transitions supply and supTrigger. The rule for the pattern mix is shown in Eq. 5. This has the same structure as the one in Eq. 4, using the transition supReq instead of supply. Eventually, Eq. 6 shows the rule to compose the anchoring point and the pull pattern, in which only the transition used by the customer to pull monitoring results has to be fused with the anchoring point.

$$\begin{aligned}\text{SupAp} = (&\text{ap}[1] \oplus \text{ap}[2] \oplus \text{supPush})\\ &(\text{ap}[1].\ <\text{int}>/\text{supPush.supply} \mapsto \text{SupAp.supply},\\ &\text{ap}[2].\ <\text{int}>/\text{supPush.supTrigger} \mapsto \text{SupAp.supTrigger},\\ &\text{ap}[1].\text{apStart}/\text{ap}[2].\text{apStart} \mapsto \text{SupAp.apStart},\\ &\text{ap}[1].\text{apEnd}/\text{ap}[2].\text{apEnd} \mapsto \text{SupAp.end},\\ &\text{ap}[1].\text{monStart}/\text{ap}[2].\text{monStart} \mapsto \text{SupAp.monStart},\\ &\text{ap}[1].\text{monEnd}/\text{ap}[2].\text{monEnd} \mapsto \text{SupAp.end},\\ &\text{ap}[1].\text{start}/\text{ap}[2].\text{start} \mapsto \text{SupAp.start},\\ &\text{ap}[1].\text{end}/\text{ap}[2].\text{End} \mapsto \text{SupAp.end},\\ &\text{ap}[1].\text{enable}/\text{ap}[2].\text{enable} \mapsto \text{SupAp.enable}.) \qquad (4)\end{aligned}$$

$$\begin{aligned}\text{SupAp} = (&\text{ap}[1] \oplus \text{ap}[2] \oplus \text{supMix})\\ &(\text{ap}[1].\ <\text{int}>/\text{supMix.supplyReq} \mapsto \text{SupAp.supplyReq},\\ &\text{ap}[2].\ <\text{int}>/\text{supMix.supTrigger} \mapsto \text{SupAp.supTrigger},\\ &\text{ap}[1].\text{apStart}/\text{ap}[2].\text{apStart} \mapsto \text{SupAp.apStart},\\ &\text{ap}[1].\text{apEnd}/\text{ap}[2].\text{apEnd} \mapsto \text{SupAp.end},\\ &\text{ap}[1].\text{monStart}/\text{ap}[2].\text{monStart} \mapsto \text{SupAp.monStart},\\ &\text{ap}[1].\text{monEnd}/\text{ap}[2].\text{monEnd} \mapsto \text{SupAp.end},\\ &\text{ap}[1].\text{start}/\text{ap}[2].\text{start} \mapsto \text{SupAp.start},\\ &\text{ap}[1].\text{end}/\text{ap}[2].\text{End} \mapsto \text{SupAp.end},\\ &\text{ap}[1].\text{enable}/\text{ap}[2].\text{enable} \mapsto \text{SupAp.enable}.) \qquad (5)\end{aligned}$$

$$\begin{aligned}\text{SupAp} = (&\text{ap} \oplus \text{supPull})\\ &(\text{ap}.\ <\text{INT}>/\text{supPull.supplyReq} \mapsto \text{SupAp.supplyReq}.)\\ &\qquad (6)\end{aligned}$$

Equation 7 shows the rules for composing the anchoring point pattern and the man patterns. Note that, since the supply interface is already composed with the anchoring points through rules for the sup dimension, this rule involves only the obtain interface of MI and it is the same for all the patterns (for the sake of illustration, Eq. 7 considers the rewrite-read rr pattern). The new obtained net is labeled ManAp.

$$\begin{aligned}\text{ManAp} = (&\text{ap} \oplus \text{manRr})\\ &(\text{ap}.\ <\text{int}>/\text{manRr.obtain} \mapsto \text{ManAp.obtain}.) \qquad (7)\end{aligned}$$

Eventually, in Step 3 we specify a rule for the composition of the subnets ProAp, TriAp, ManAp, and SupAp obtained in Steps 1 and 2 in a complete monitoring infrastructure. Equation 8 shows the composition rule, which relies on the fusion of the transitions and places of anchoring points and on the fusion, as previously anticipated, of the supply transition of patterns monitoring data supply and management (man and sup, respectively).

Table 2 Average user correct choices in the experiment

	Anchoring	Trigger	Management	Supply
Simple scenario—correct choices (%)	70.7	43.2	78.4	85.2
Complex scenario—correct choices (%)	79.6	56.2	53.7	79.0

$$
\begin{aligned}
&\mathsf{MI} = (\mathsf{ProAp} \oplus \mathsf{TriAp} \oplus \mathsf{ManAp} \oplus \mathsf{SupAp})\\
&\quad \mathsf{ProAp.start/TriAp.monStart/ManAp.monStart/}\\
&\qquad \mathsf{SupAp.monStart} \mapsto \mathsf{MI.start}),\\
&\quad \mathsf{ProAp.end/TriAp.monEnd/ManAp.monEnd/}\\
&\qquad \mathsf{SupAp.monEnd} \mapsto \mathsf{MI.end},\\
&\quad \mathsf{ProAp.apStart/TriAp.apStart/ManAp.apStart/}\\
&\qquad \mathsf{SupAp.apStart} \mapsto \mathsf{MI.apStart},\\
&\quad \mathsf{ProAp.apEnd/TriAp.apEnd/ManAp.apEnd/}\\
&\qquad \mathsf{SupAp.apEnd} \mapsto \mathsf{MI.apEnd},\\
&\quad \mathsf{ProAp.enable/TriAp.enable/ManAp.enable/}\\
&\qquad \mathsf{SupAp.enable} \mapsto \mathsf{MI.enable},\\
&\quad \mathsf{ManAp.supply/SupAp.supply} \mapsto \mathsf{MI.supply}.) \qquad (8)
\end{aligned}
$$

5.2 Monitoring infrastructure design in running example

In this section, we demonstrate the application of patterns and composition rules to the design of a complete customized process monitoring infrastructure.

The customized monitoring infrastructures for customers A and B (see Table 1 for the options chosen) are shown in Fig. 17a, b in the "Appendix", respectively. Both MIs have been designed by applying sequentially the composition rules for the chosen patterns defined previously.

The complete list of compositions rules applied for designing the monitoring infrastructures of Fig. 17 is also reported in the "Appendix". In particular, the activities start and end of rule in Eq. 1 are instantiated into the activities Start Campaign and End Campaign of the advertising process of Fig. 4, respectively.

6 Evaluation and tool implementation

In this section, we evaluate our framework for process monitoring customization. In particular, we first evaluate the appropriateness (or fit for purpose) of our framework in capturing the monitoring requirements of a given process using an empirical study. Then, we show the feasibility of our framework by discussing the proof-of- concept implementation of a tool supporting the synthesis of customized monitoring infrastructures starting from requirements expressed using our process monitoring patterns.

6.1 Evaluation of monitoring patterns

We designed an empirical study to assess whether experts in process modeling were able to use our patterns to capture monitoring requirements on given business process specifications. The study involved graduate and postgraduate students in business computing subjects in the Netherlands and UK enrolled in modules on business process management covering business process modeling with Petri nets and BPMN. A total of 30 students participated to the experiment.

After a 15-min training session in which the monitoring patterns were explained using the same examples used in this paper, users were given two scenarios. Each scenario included (1) the description of a process, (2) a Petri net and BPMN process model representing the process, and (3) a set of three monitoring requirements of the process considered by the scenario. The *simple* scenario involved a process with six sequential tasks, whereas the process in the *complex* scenario involved 12 tasks, conditional and parallel execution, and one loop. Users were asked to capture the monitoring requirements in each scenario by applying the patterns defined in Sect. 2 in a 35-min session. To avoid cognitive bias of the authors, process descriptions were selected from the set of process modeling exercises developed by the BPMN Academic Initiative (bpmnai.org) and monitoring requirements were specified by staff members other then the authors. Correct solutions to the scenarios were developed by the authors before running the experiment.

We collected two types of evidence:

- the number of time patterns was applied correctly to capture monitoring requirements;
- a subjective assessment of the mental and time-related workload of the exercise, using the standard NASA Task Load Index (TLX) survey.[1]

A pilot study involving two doctoral students as users was conducted to ensure the effectiveness of the training session and the understandability of the task specification.

Table 2 shows the correct number of answers provided by users when applying the patterns, and Table 3 shows the descriptive statistics of the subjective workload assessment survey.

Overall, the concept and options available for *anchoring* and the *data supply* dimensions are well understood by users and applied correctly on more than 70 % of the cases. The *data management* dimension is relatively poorly understood in the complex scenario, which can be due to the complexity of the scenario and its monitoring requirements. The *mon-*

[1] http://humansystems.arc.nasa.gov/groups/tlx/.

Table 3 Descriptive statistics of TLX survey

TLX survey items	Avg	StDev
How mentally demanding was the task?	13.7	4.7
How hurried or rushed was the pace of the task?	10.3	4.7
How successful were you in accomplishing what you were asked to do?	10.6	3.8
How hard did you have to work to accomplish your level of performance?	12.6	9.1
How insecure, irritated, discouraged, stressed, and annoyed were you?	9.5	5.6

All items evaluated on a scale from 0 to 21

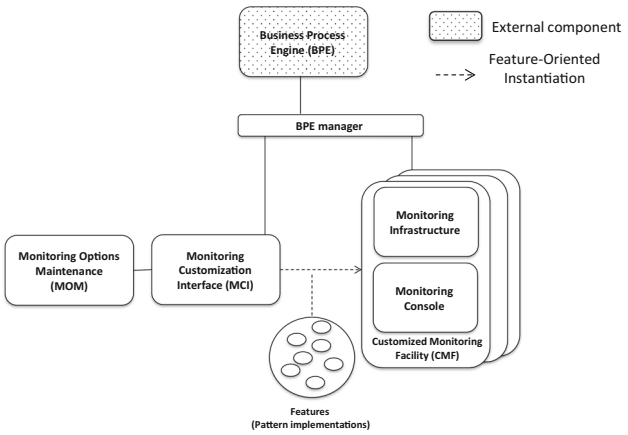


Fig. 12 Architecture of the tool for process monitoring customization

itoring trigger is relatively poorly understood and applied correctly in no more than 56 % of the cases. This may be due to the fact that common workflow technology implements only one of the available monitoring options, that is, monitoring triggered by the monitoring infrastructure. Workflow technology has often to be extended ad hoc to enable monitoring clients to trigger the acquisition of monitoring data directly. This may have represented a bias for student users with limited classroom experience on real-world workflow systems.

Regarding the workload subjective assessment (see Table 3), the task did not appear to be particularly frustrating and users found enough time to complete it. The mental effort required by the task is relatively high. This is consistent with the fact that we involved students in our experiment. We argue that the subjective mental effort will be lower for more experienced professional process designers.

6.2 A tool for process monitoring mass customization

In this section, we describe the implementation of a proof-of-concept tool to support customization of monitoring infrastructures. We consider the case of monitoring infrastructures executing on top of a generic service-oriented process engine. Our objective is to show the feasibility of the approach discussed thus far in the paper and to set a foundation for its future test implementation in real business settings.

Generally, software customization is the focus of software product lines [11,29], in which a software product is decomposed into a set of variant artifacts. Users (customers) choose the set of artifacts satisfying their requirements and the software vendor will instantiate, possibly automatically, the software program from the artifacts and deliver it to the customer. Among the techniques to implement and manage software product lines, our tool exploits specifically Feature-Oriented Software Development (FOSD) [3]. FOSD comprises a set of techniques and specifications to support the automated generation of customized software programs. It predicates to decompose software programs into features and to provide configuration options for users/designers. Customized software programs are then generated based on a selection of features, i.e., the variant artifacts, exploiting one or more generative programming techniques [9]. Generative programming (GP) techniques automate the synthesis of software programs from their building blocks, i.e., the features. By limiting the variability of software programs, GP techniques increase programmer' productivity and, generally, the maintainability of software artifacts [8,9]. FOSD is supported by FeatureIDE, a freely available Eclipse plugin supporting feature modeling, code artifacts implementation, and related code generation [43].

In our context, we need a tool allowing the composition of predefined monitoring patterns on top of the core monitoring capability, i.e., the monitoring API, provided by the underlying process engine. The monitoring options identified in Sect. 2, therefore, are implemented as features of a core process monitoring capability exploiting the native monitoring API of the process engine. In particular, we experimented with the OpenESB BPEL and YAWL process engines. Customized monitoring infrastructures are generated automatically based on the monitoring options selected by the customers of a process using a FOSD approach.

The conceptual architecture of the tool is shown in Fig. 12. It combines design-time and run-time components.

At design time, the provider maintains the models of monitoring options for its processes through the monitoring option maintenance (MOM) component. Customers have access to the monitoring customization interface (MCI), i.e., a Web interface, where they can customize the monitoring of the process instances they have started in the BPE. The MOM component is represented by the FeatureIDE model

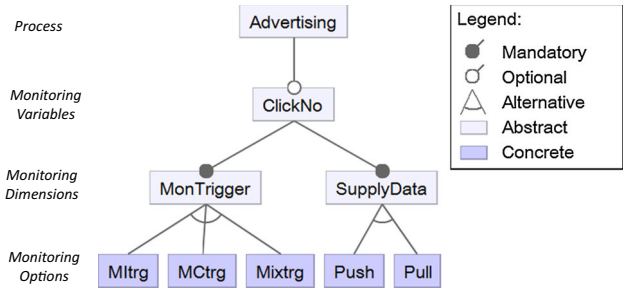


Fig. 13 Monitoring features in our running example

editor, through which process designers create and maintain a set of monitoring option models for the processes offered to customers. Figure 13 shows a snapshot of the feature model corresponding to the monitoring options offered by the provider in our online advertising scenario. Note that the feature model in Fig. 13 only includes options for the tri and sup dimensions. In particular, the root level represents the process that can be monitored, while the second level of the feature model captures the possible monitoring variables. For each monitoring variable, the third level of the feature model captures the monitoring dimensions, as defined in Sect. 2, while the leaves of the feature model represent the monitoring options available to customers. Note that customers may choose not to monitor some variables, i.e., variables monitoring is optional, as represented by the white dot, while once a variable has been chosen, customers have to specify one option for each possible monitoring dimension, i.e., monitoring dimensions are mandatory (black dot in the feature model) and options for a monitoring dimension are *alternative*. Also, note that only monitoring options are *concrete*, since they have an implementation (based on the patterns defined in Sect. 4).

The MCI shows to a logged in customer the options available for monitoring for only the instances that the customer has started. The MCI, therefore, combines information obtained from the BPE, i.e., a list of running instances, with information from the MOM, i.e., the monitoring options available for active processes, as stored in the configuration file generated by MOM. The MCI allows only feasible combinations of monitoring options, since this information is embedded in the monitoring option model and, therefore, in the configuration files. Figure 14 shows examples of a monitoring options model in the MOM, and snapshots of the related configuration XML file generated by FeatureIDE and the correspondent MCI.

At run time, for each process started by customers, a customized monitoring facility (CMF) is instantiated. A CMF comprises a customized monitoring console (MC) and a monitoring infrastructure (MI). The MC displays monitoring information and allows customer interaction about monitoring, e.g., pulling monitoring values, as specified by the options chosen by the customers at design time. The MI implements the logic to obtain, store, and made available monitoring information as specified by the patterns discussed in Sect. 4.

The generation of CMF exploits the principle of Feature-Oriented Software Development. In particular, for each monitoring option, the process monitoring designer must provide a feature implementation file, specifying the implementation of the related Monitoring Infrastructure and console. Feature implementations are then combined to create a complete CMF in response to a customer request. Such a combination exploits the theory of feature-oriented software programming AHEAD [8]. FeatureIDE provides tool support for AHEAD-based composition of features specified in Java. Figure 15 shows an example of a feature configuration file in FeatureIDE and the correspondent customized monitoring console, in which only two of the possible monitoring variables are selected for monitoring with specific monitoring options.

Each monitoring option feature extends a base feature constituted by an empty monitoring GUI and an empty data structure to collect monitoring values. A monitoring option feature extends the base feature, providing a specification for the GUI and for the application logic required by the monitoring infrastructure.

6.2.1 Required capabilities of business process

In service-oriented business process monitoring, at the conceptual level customization is built on top of a baseline constituted by a set of standard monitoring capabilities of the BPE. In other words, in our approach the BPE needs to implement a standard monitoring interface that the BPE manager component will use (see Fig. 16). Based on the architecture of our tool and the patterns of Sect. 4, at design time (DT) the standard interface should allow:

DT1. retrieving the instances started by a given customer (see method `getInstanceList()` in Fig. 16);
DT2. retrieving the list of variables defined for a process (`getVarList()`).

DT capabilities are required by the MCI to show the list of instances available to the user.

At run time (RT), the BPE manager should allow the CMFs:

RT1. retrieving values of monitoring variables (`getMonValue()`);
RT2. commanding the acquisition of monitoring values (`acquireMonValue()`);
RT3. retrieving the status of activities in a given process instance (`getActivityStatus()`).

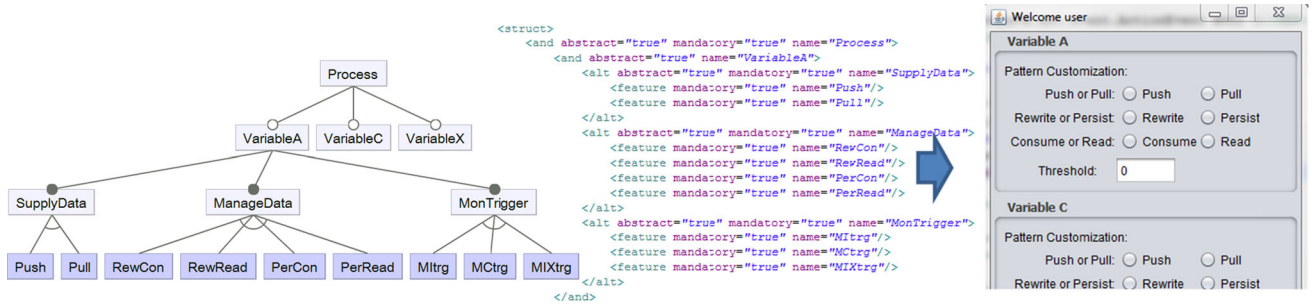


Fig. 14 From feature modeling to monitoring customization interface

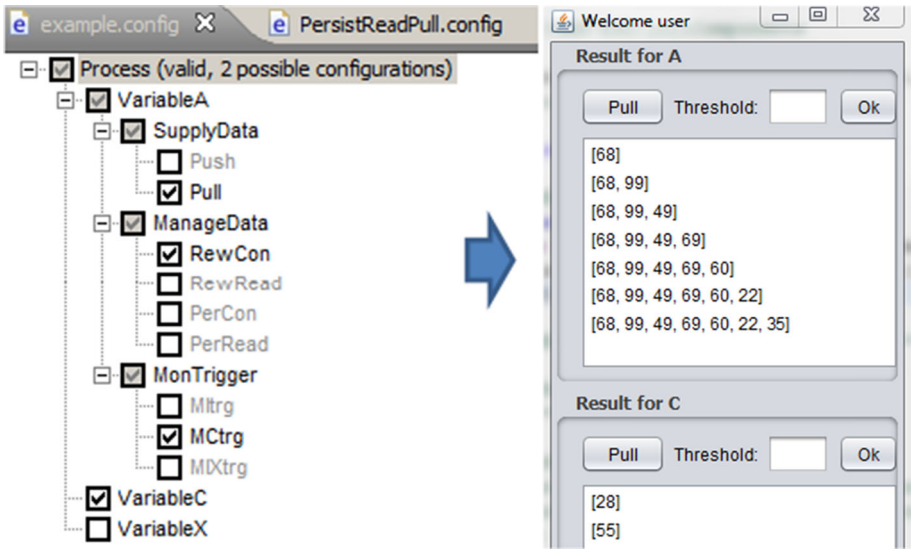


Fig. 15 From feature configuration to customized monitoring console

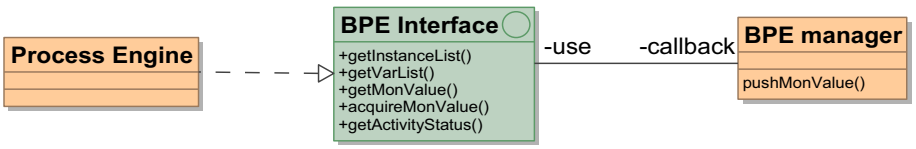


Fig. 16 The BPE interface

RT capabilities RT1 and RT2 are required for obtaining monitoring variables values, whereas RT3 is required for the implementation of anchoring points. Specifically, anchoring points can be implemented within the CMF by polling the status of the monStart and monEnd activities of the process and enabling or disabling monitoring consequently.

Although a thorough review of business process management technology is out of scope in this paper, we argue that the functionality of the BPE manager can be implemented on top of currently available process management technology. As a partial demonstration of this statement, we have successfully integrated our tool with the OpenESB BPEL engine and the YAWL workflow engine as BPEs.

Both engines provide a basic API to retrieve the list of running instances (or cases, in YAWL) and related variables. This satisfies the needs of the BPE manager at design time. Also, the same API provides methods to retrieve the status of activities of processes and running instances. These are used for the implementation of anchoring points (see capability RT3 defined before).

About run time, the OpenESB engine stores monitoring information, e.g., variable and activity status values, in a database and provides a basic API to retrieve values from such a database [34]. This API is used to implement the capability RT1 discussed before in pull mode. For allowing the engine to push monitoring values to the BPE manager, the BPE manager exposes a callback interface, which is called by a custom trigger implemented in the BPE native monitoring component database, e.g., when a value of monitored value changes or exceeds a given threshold (see `pushMonValue()` in Fig. 16). Note that a similar mechanism can be implemented using the *sensors* of the Oracle BPEL Process Manager. Sensors are used to declare interest in particular events of interest within process execution, e.g., values of process variables.

Sensors can be extended by defining a custom *action*, e.g., to send the values of monitoring variables to the BPE manager callback interface.

The YAWL engine implements a more advanced native monitoring component by providing a native observer interface that external applications can implement to be pushed or to pull monitoring values as required [14,42]. Custom developed observers are used for the implementation of the BPE run-time capabilities discussed above.

Both process engines log regularly the values of process variables and status activities. Hence, the run-time capability RT2 does not need to be explicitly implemented, since updated monitoring values are always available in the monitoring component database.

7 Related work

Customization is a paramount activity in the implementation of complex enterprise systems, such as ERP [25]. Traditional customization in enterprise systems is a design-time concern, which aims at designing standard processes across the implementing organization. While standardization across the enterprise promotes uniformity and interoperability [25], it is also seen as a constricting institutional factor for large companies with diverse business units limiting the flexibility of the company [20]. In this paper, we take a much more dynamic perspective on customization, allowing individual customers to specify their own monitoring requirements and supporting the derivation of a monitoring infrastructure to satisfy those.

The approach presented in [23] aims at proposing a monitoring solution which, as our work, decouples the monitoring specification from the details of the underlying workflow system, through the definition of monitoring aspects. This approach, however, does not consider explicitly customization and, consequently, fails to identify a solution space for monitoring to allow customers to define their specific monitoring requirements.

The paper in [10] stresses that, while clearly reducing implementation efforts, information systems configuration is not likely to anticipate and capture the requirements of all possible users. Therefore, information systems require a certain degree of adaptation to fully capture user requirements. To support adaptation, the authors propose a conceptual model that, as we do in this paper, clearly separates the modeling of possible adaptation aspects from the implementation of related adaptation mechanisms.

In the SOA literature, service adaptation can be applied to align service specifications in service compositions [28], in reaction to detected SLA violations [30], or to restore QoS guarantees [33]. In all these cases, however, adaptation is not customer specific, since the changes entailed by adaptation mechanism are reflected on all the customers using the service. Examples of customer-specific service adaptation can be found in the mobile services literature, where the service presentation layer and, to a lesser extent, the service functionality can be adapted on the basis of the user access conditions [47,48], e.g., type of device, screen resolution, quality of Internet connection.

The literature on monitoring of service-based business processes has focused extensively on languages and tools to capture monitoring requirements of interested stakeholders, e.g., [5,41]. In this case, however, the objective is to give the user a means to control the instrumentation of the process engine to capture the appropriate monitoring variables. While these approaches can enrich the specification of the monitoring variable mv, they do not consider the design of the monitoring infrastructure to store and communicate monitoring data.

Customization of service-based systems monitoring infrastructure is also considered in [13,18], where different components can be optimally selected to monitor the terms of SLAs, or in [24], where monitoring components at different architectural levels can be combined to provide an integrated monitoring facility to external client. In these works, customization is not considered at the fine-grained level considered in this paper. Monitoring components are, in fact, monolithic services that cannot be modified to better adhere to the monitoring stakeholders' requirements.

Customization of workflow monitoring can be also achieved through the use of agent-based frameworks [27,39,45]. In [45], in particular, the authors propose a framework to extend the traditional workflow reference architecture with agents for monitoring. The language discussed to specify the monitoring plan shares some commonalities with our monitoring dimensions, such as the monitoring scope that defines when monitoring agents are activated to capture monitoring data. Agents to support semantic alignment of monitoring information are considered by [27]. Semantic alignment could be applied in our case to align the definition of monitoring data between the business process engine and different monitoring clients.

Regarding the possible behaviors of the monitoring infrastructure and its communication with the business process and the client application, we analyzed the literature on traditional software program monitoring, Web service-based monitoring, and business process monitoring.

A survey on software programs and software requirements monitoring can be found in [16]. From this survey, we take the notion of monitoring points. Monitoring points define the anchors of the monitoring program to the monitored program. Similarly, in our model we define anchor points for monitoring options to the monitored process. The survey in [16] is used by [6] to classify approaches to Web service-based process monitoring. In particular, [49] consid-

ers the modality of notification of monitoring information as a classification criterion. Monitoring information, usually captured by an instrumentation of the Web service container, can be either pushed to the monitor or pulled by it. Web service monitoring usually takes an event source-listener approach [17], where the instrumented Web service container is the source that pushes monitoring-related event to the monitor (listener) [32]. When monitoring information is pushed, the work in [6] also considers the multiplicity and frequency with which monitoring information is made available to the monitor. Still in the context of Web service monitoring, the model in [37] considers the concept of monitoring socket, i.e., a generic component which is responsible for the generation of monitoring data, which can then be pushed to or pulled by the monitor.

8 Conclusions and outlook

This paper presents a solution for the design and implementation of customized monitoring infrastructures of service-based business processes. We first tackled the problem at a conceptual level, by proposing a multi-dimensional model of possible monitoring options and a set of conceptually defined patterns for their implementation. By separating the monitoring option models from the definition of patterns, we conceptually separated the offer that the provider can make to the customers from the actual implementation of such an offer.

Then, we provided a set of rules to compose the patterns into the design of a complete monitoring infrastructure. We then presented a proof-of-concept implementation exploiting the principles of Feature-Based Software development. Our tool allows designing and maintaining monitoring options model at design time and the generation of customized monitoring infrastructures at run time. Monitoring infrastructures, in particular, are generated by the combination of software features implementing the business logic of monitoring options as specified by the patterns and the composition rules.

The work presented here can be extended along several lines. As far as the monitoring options model is concerned, we are planning to extend our work with cross-instance monitoring options. Cross-instance monitoring should allow customers to monitor variables defined across the set of process instances that they have started. These can be defined for all instances of a given processes, e.g., the average value of a given process variable, or across instances belonging to different processes, i.e., the average response time of a service used within different processes. In the longer term, our work on customization can be extended also to other process management aspects, such as QoS or resource allocation. Further extensions of our monitoring patterns may concern the visualization of monitoring data and higher-level concerns such as compliance to given regulatory requirements.

As far as the implementation is concerned, we are planning to evaluate our proof-of-concept tool in a real-world process monitoring customization scenario. The objective in this case would be to show the scalability of the tool with the numbers of users and the number of monitoring requests. We argue that the cloud computing paradigm has the potential to support our approach on a large scale. Through our monitoring customization framework, the service provider can identify the monitoring options requested by a high number of customers and the ones involved only in specific customer requests. The former can be served by a common dynamic (multi-tenant) computing infrastructure, whereas computing resources to provision the latter can be allocated on demand.

A further challenge will be to address the relationship between our patterns and big data. When the amount of data generated by the monitoring infrastructure becomes extremely large, we may need to define new patterns, particularly regarding data management, to capture for instance that only a limited amount of data can be stored at any given time or that data must be processed on line, perhaps in memory, as soon as they are generated. We argue that such issues do not concern traditional workflows involving human actors, but may become relevant in the context of the Internet of Things, where processes involve sensors capturing and processing large amount of data.

Appendix 1: List of composition rules applied to design the MI for Customer A

Step 1
Anchoring MI to process:

$$\begin{aligned}&\mathsf{ProAp} = (\mathsf{process} \oplus \mathsf{ap})\\ &\quad(\mathsf{process.startCampaign/ap.monStart}\\ &\quad\mapsto \mathsf{ProAp.StartCampaign},\\ &\quad\mathsf{process.endCampaign/ap.monEnd}\\ &\quad\mapsto \mathsf{ProAp.endCampaign}).\end{aligned}$$

Step 2
Anchoring pattern tri=miTrg:

$$\begin{aligned}&\mathsf{TriAp} = (\mathsf{ap} \oplus \mathsf{miTrg})\\ &\quad(\mathsf{ap.} < \mathsf{int} > \mathsf{/mcTrg.bpeAcquireReq}\\ &\quad\mapsto \mathsf{TriAp.bpeAcquireReq.})\end{aligned}$$

Anchoring pattern sup=pull:

SupAp = (ap ⊕ supPull)
(ap. < int >/supPull.supplyReq ↦ SupAp.supplyReq.)

Anchoring pattern man=pr:

ManAp = (ap ⊕ manPr)
(ap. < int >/manPr.obtain ↦ ManAp.obtain.)

Step 3
Composing MI:

MI = (ProAp ⊕ TriAp ⊕ ManAp ⊕ SupAp)
ProAp.startCampaign/TriAp.monStart/ManAp.monStart/
SupAp.monStart ↦ MI.startCampaign),
ProAp.endCampaign/TriAp.monEnd/ManAp.monEnd/
SupAp.monEnd ↦ MI.endCampaign,
ProAp.apStart/TriAp.start/ManAp.start/
SupAp.start ↦ MI.apStart,
ProAp.apEnd/TriAp.end/ManAp.end/
SupAp.end ↦ MI.apEnd,
ProAp.enable/TriAp.enable/ManAp.enable/
SupAp.enable ↦ MI.enable,
ManAp.supply/SupAp.supply ↦ MI.supply.)

Appendix 2: List of composition rules applied to design the MI for Customer B

Step 1
Anchoring MI to process:

ProAp = (process ⊕ ap)
(process.startCampaign/ap.monStart
↦ ProAp.StartCampaign,
process.endCampaign/ap.monEnd
↦ ProAp.endCampaign).

Step 2
Anchoring pattern tri=mcTrg:

TriAp = (ap[1] ⊕ ap[2] ⊕ mcTrg)
(ap[1]. < int >/mcTrg.acquire ↦ TriAp.acquireReq,
ap[2]. < int >/mcTrg.bpeAcquireReq
↦ TriAp.bpeAcquireReq,
ap[1].apStart/ap[2].apStart ↦ TriAp.apStart,
ap[1].apEnd/ap[2].apEnd ↦ TriAp.end,
ap[1].monStart/ap[2].monStart ↦ TriAp.monStart,
ap[1].monEnd/ap[2].monEnd ↦ TriAp.end,
ap[1].start/ap[2].start ↦ TriAp.start,
ap[1].end/ap[2].End ↦ TriAp.end,
ap[1].enable/ap[2].enable ↦ TriAp.enable.)

Anchoring pattern sup=push:

SupAp = (ap[1] ⊕ ap[2] ⊕ supPush)
(ap[1]. < int >/supPush.supply ↦ SupAp.supply,
ap[2]. < int >/supPush.supTrigger ↦ SupAp.supTrigger,
ap[1].apStart/ap[2].apStart ↦ SupAp.apStart,
ap[1].apEnd/ap[2].apEnd ↦ SupAp.end,
ap[1].monStart/ap[2].monStart ↦ SupAp.monStart,
ap[1].monEnd/ap[2].monEnd ↦ SupAp.end,
ap[1].start/ap[2].start ↦ SupAp.start,
ap[1].end/ap[2].End ↦ SupAp.end,
ap[1].enable/ap[2].enable ↦ SupAp.enable.)

Anchoring pattern man=pc:

ManAp = (ap ⊕ manPc)
(ap. < INT >/manPc.obtain ↦ ManAp.obtain.)

Step 3
Composing MI:

MI = (ProAp ⊕ TriAp ⊕ ManAp ⊕ SupAp)
ProAp.startCampaign/TriAp.monStart/ManAp.monStart/
SupAp.monStart ↦ MI.startCampaign),
ProAp.endCampaign/TriAp.monEnd/ManAp.monEnd/
SupAp.monEnd ↦ MI.endCampaign,
ProAp.apStart/TriAp.start/ManAp.start/
SupAp.start ↦ MI.apStart,
ProAp.apEnd/TriAp.end/ManAp.end/
SupAp.end ↦ MI.apEnd,
ProAp.enable/TriAp.enable/ManAp.enable/
SupAp.enable ↦ MI.enable,
ManAp.supply/SupAp.supply ↦ MI.supply.)

See Fig. 17.

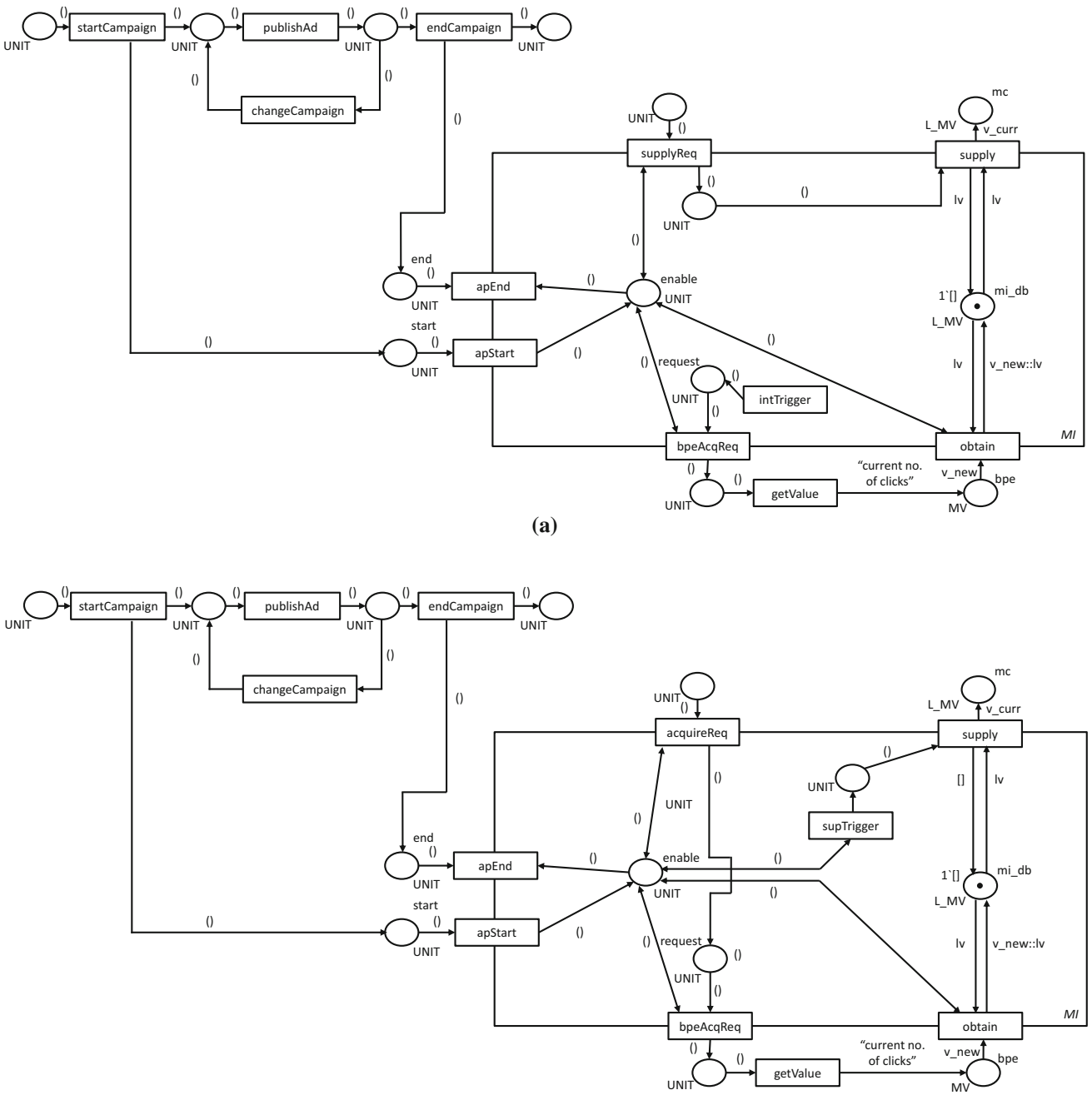


Fig. 17 Combining patterns for example customer A (**a**) and customer B (**b**)

References

1. Aldred L, van der Aalst W, Dumas M, ter Hofstede A (2009) Dimensions of coupling in middleware. Concur Comput Pract Exp 21:2233–2269
2. Angelov S, Grefen P (2008) Supporting the diversity of B2B-E-contracting processes. Int J Electron Commer 12(4):39–70
3. Apel S, Kästner C (2009) An overview of feature-oriented software development. J Object Technol 8:49–84
4. Ardagna D, Pernici B (2007) Adaptive service composition in flexible processes. IEEE Trans Softw Eng 33:369–384
5. Baresi L, Guinea S (2011) Self-supervising BPEL processes. IEEE Trans Softw Eng 37(2):247–262
6. Baresi L, Guinea S, Nano O, Spanoudakis G (2010) Comprehensive monitoring of BPEL processes. IEEE Internet Comput 14(3):50–57
7. Barros J, Gomes L (2003) Modifying petri net models by means of crosscutting operations. In: Proceedings of 3rd international conference on application of concurrency to system design, pp 177–186

8. Batory D (2006) Tutorial on Feature Oriented programming and the AHEAD tools suite. In: Proc. Generative and Transformational Techniques in Software-Engineering. pp 3–35
9. Batory D, O'Malley S (1992) The design and implementation of hierarchical software systems with reusable components. ACM Trans Softw Eng Methodol 1:355–398
10. Becker J, Delfmann P, Knackstedt R (2007) adaptive reference modeling: integrating configurative and generic adaptation techniques for information models. In: Becker J, Delfmann P (eds) Reference modeling: efficient information systems design thorugh reuse of information models. Physica-Verlag, Heidelberg
11. Clements P, Northrop L (2002) Software product lines. Addison-Wesley, Boston
12. Comuzzi M, Rafael Martinez RI (2014) Customized infrastructures for monitoring business processes. In: Proceedings of 8th IEEE symposium on service-oriented system engineering
13. Comuzzi M, Spanoudakis G (2010) Dynamic set-up of monitoring infrastructures for service based systems. In: ACM SAC, pp 2414–2421
14. Comuzzi M, Vonk J, Grefen P (2012) Measures and mechanisms for process monitoring in evolving business networks. Data Knowl Eng 71:1–28
15. Curtis G, Cobham D (2008) Business information systems: analysis, design and practice, 6th edn. Financial Times/Prentice Hall, London
16. Delgado N, Gates AQ, Roach S (2004) A taxonomy and catalog of runtime software-fault monitoring tools. IEEE Trans Softw Eng 30(12):859–872
17. Eugster PT, Felber PA, Guerraoui R, Kermarrec A-M (2003) The many faces of publish/subscribe. ACM Comput Surv 35:114–131
18. Foster H, Spanoudakis G (2011) Advanced service monitoring configurations with SLA decomposition and selection. In: Proceedings of ACM symposium on applied computing, pp 1582–1589
19. Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object-oriented software. Addison-Wesley, Reading
20. Gattiker T, Goodhue D (2005) What happens after ERP implementation: understanding the impact of interdependence and differentiation on plant-level outcomes. MIS Q 29:559–585
21. Gewald H, Dibbern J (2009) Risks and benefits of business process outsourcing: a study of transaction services in the german banking industry. Inf Manag 46:249–257
22. Gomes L, Barros J (2005) Structuring and composability issues in petri nets modeling. IEEE Trans Indus Inf 1:112–123
23. Gonzalez O, Casallas R, Deridder D (2011) Monitoring and analysis concerns in workflow applications: from conceptual specifications to concrete implementations. Int J Cooper Inf Syst 20(4):371–404
24. Guinea S, Kecskemeti G, Marconi A, Wetzstein B (2011) Multi-layered monitoring and adaptation. In: Proceedings of international conference on service-oriented computing, pp 359–373
25. Jacobs F, Whybark C (2000) Why ERP?. Irwin/McGraw Hill, New York
26. Jensen K, Kristensen LM (eds) (2009) Coloured petri nets. Springer, Berlin
27. Kang D, Lee S, Kim K, Lee JY (2009) An OWL-based semantic business process monitoring framework. Expert Syst Appl 36(4):7576–7580
28. Kongdenfha W, Saint-Paul R, Benatallah B, Casati F (2006) An aspct-oriented framework for service adaptation. In: Proceedings of ICSOC 2006, pp 15–26
29. Krueger CW (2002) Variation management for software production lines. In: Software product lines 2002, pp 37–48
30. Leitner P, Michlmayr A, Rosenberg F, Dustdar S (2010) Monitoring, prediction and prevention of SLA violations in composite services. In: Proceedings of 2010 IEEE international conference on web services, pp 369–376
31. Ludwig H, Dan A, Kearney R (2004) Cremona: an architecture and library for creation and monitoring of WS-Agreements. In: Proceedings of 2nd international conference on service oriented computing, pp 65–74
32. Mahbub K, Spanoudakis G (2005) Run-time monitoring of requirements for systems composed of web-services: Initial implementation and evaluation experience. In: ICWS 2005 Proceedings of IEEE Computer Society, pp 257–265
33. Moser O, Rosenberg F, Dustdar S (2008) Non-intrusive monitoring and service adaptation for WS-BPEL. In: Proceedings of World Wide Web conference
34. Oracle Inc. (2011) OpenESB: The open enterprise service bus
35. Robinson W (2006) A requirements monitoring framework for enterprise systems. Requir Eng 11:17–41
36. Robinson W, Purao S (2011) Monitoring service systems from a language-action perspective. IEEE Trans Serv Comput 4:17–30
37. Sadiq S, Governatori G, Namiri K (2007) Modeling control objectives for business process compliance. In: Proceedings of 5th business process management conference, pp 149–164
38. Scheer A (2000) ARIS business process modeling. Springer, Berlin
39. Shu J, Barton R (2012) Managing supply chain execution: monitoring timeliness and correctness via individualized trace data. Prod Oper Manag 21(4):715–729
40. Simmonds J, Gan Y, Chechik M, Nejati S, O'Farrell B, Litani E, Waterhouse J (2009) Runtime monitoring of Web service conversations. IEEE Trans Serv Comput 2:223–244
41. Srdic G, Juric M (2013) Model for integrated monitoring of BPEL processes. Int J Cooper Inf Syst 22(2):1–29
42. ter Hofstede AM, van der Aalst WMP, Adams M, Russell N (2010) Modern business process automation: YAWL and its support environment. Springer, Berlin
43. Thüm T, Kästner C, Benduhn F, Meinicke J, Saake G, Leich T (2013) FeatureIDE: an extensible framework for feature-oriented software development. Sci Comput Program 79:70–85
44. van der Aalst W, ter Hofstede A, Kiepuszewski B, Barros A (2003) Workflow patterns. Distrib Parallel Databases 14(3):5–51
45. Wang M, Wang H, Xu D (2005) The design of intelligent workflow monitoring with agent technology. Knowl Based Syst 18:257–266
46. Wieringa R (2003) Design methods for reactive systems: yourdon, statemate, and the UML. Morgan Kaufmann, Los Altos
47. Wu S-Y, Chang C-S, Ho S-H, Chao H (2008) Rule-based intelligent adaptation in mobile information systems. Expert Syst Appl 34:1078–1092
48. Zhang D (2007) Web content adaptation for mobile handheld devices. Commun ACM 50:70–79
49. zur Muehlen M (2005) Workflow-based process controlling. Springer, Berlin