Towards Aspect Oriented Adaptive Case Management

Authors:

Amin Jalali
Ilia Bider

Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), 2014 IEEE 18th International

Towards Aspect Oriented Adaptive Case Management

Amin Jalali
Department of Computer and Systems Sciences (DSV)
Stockholm University
Email: aj(at)dsv.su.se

Ilia Bider
Department of Computer and Systems Sciences (DSV)
Stockholm University
Email: ilia(at){dsv.su—ibissoft}.se

***Abstract*—Separation of concerns has long been an important strategy in the software systems development to cope with the complexity embedded in such systems. The same type of concerns, like security concerns, is often repeated in many modules of a system, which hinders the consistency, re-usability, change and maintenance of the system. Aspect orientation aims to separate and encapsulate these concerns to solve the complexity problem. This paper introduces the use of aspect orientation for case and adaptive case management through changing the rules that govern business processes on the fly. It introduces a taxonomy of such rules based on the declarative workflows approach. It also shows how so-called form-based case management systems could be extended to support aspect orientation to reduce the complexity problem.**

Keywords*—*Adaptive Case Management, Aspect Oriented, Business Process, Declarative, State Space

I. Introduction

Business Process Management (BPM) is an important area which aims to support operational activities in business processes to achieve business goals. The operational activities can vary in different instances of a business process of the same type, which makes it necessary to support the flexibility in business processes to reflect the deviations. However, the amount of deviations can be significantly high in many situations that challenge supporting business processes in an effective way. To support such processes, a new type of business process support systems is needed to enable on the fly adaptability.

Adaptive Case Management (ACM) is a paradigm to support adaptability of processes in volatile environments. Unlike BPM systems, which consider the control-flow as a dominant dimension for supporting business processes, ACM considers data as the primary core around which processes are to be supported (see Fig. 1) [29]. ACM aims to support business processes that are driven by knowledge workers; therefore, it allows a high degree of flexibility since such workers need more freedom to complete their work. Indeed, the important part in ACM is to store and trace the results of actions completed in the frame of a process instance/case, which are recorded as data. This data can be considered as defining the state of the process instance, while actions can be considered as an aim to achieve the desirable changes in the state.

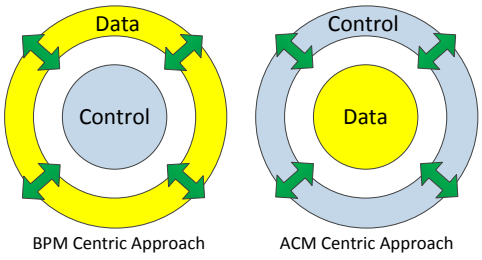


Fig. 1. BPM vs. ACM approaches, adapted from [29]

ACM and BPM aim to achieve different goals through different ways of separation of concerns, e.g., data, control-flow and human resources. Separation of concerns has long been considered as an approach to deal with the complexity in system development. Concerns can be defined in many different ways. For example, in BPM, business processes can be defined based on different perspectives, such as control-flow, data, resources. Each perspective encapsulates different constraints regarding different concerns. Although the constraints become separated through perspectives of a process model, models can still become very complex due to the number of constraints that they should incorporate in the main concern/perspective. Therefore, other separation techniques are introduced to deal with the complexity of the process models, like horizontal, vertical and orthogonal modularization techniques [20].

One sort of concerns, e.g. security, can be applicable for many business processes. Without separation that encapsulates each concern in one place, these concerns scatter over the process models, which introduces the scattering problem. Separated concerns can be modeled once and be called by different process models, yet this approach introduces another problem known as tangling. The rules of application of the concerns are determined by each concern on its own. This means that if the rule is changed, all process models which can be affected should be found and un-tangled or tangled. The scattering and tangling problems challenge the maintenance of the system, decrease the re-usability of modules that implement the concerns, and can result in violation of consistency of the system. The concerns that introduce the scattering and tangling problems are called cross-cutting concerns.

To deal with scattering and tangling problem, the so-called aspect oriented principle is followed in many disciplines to separate cross-cutting concerns from the main models. This principle is usually applied in deterministic areas, where the order of operations can be predicted like programming, service composition and workflow based business processes. Unlike

these areas, the order of activities cannot be always determined in ACM System. Thus, it is interesting to investigate how these concerns can be separated in such systems. In this paper, we investigate how the aspect oriented principle could be applied in ACM Systems to enable separation of cross-cutting concerns. More specifically, the paper exploits the usage of declarative rules for enabling this principle in ACM Systems.

The rest of this paper is structured as follows. First, an overview of aspect orientation and declarative rules is given in section II. Then, a proposal for how aspect orientation could be introduced to ACM using declarative rules is given in section III. Then, a short description of how a form-based case management system could be extended to support aspect orientation is given in section IV. Finally, a short summary of the finding and directions for the future work is presented in section V.

II. Background

This section gives a summary of applying aspect orientation in programming, service composition and business process management domains. It also gives a short summary of declarative rules that will be used later to propose an aspect oriented approach for ACM.

A. Aspect Oriented Programming

It is a common problem in the software systems development to support the realization of so-called cross-cutting concerns. These concerns are scattered in many modules, and their applicability for each module is not known by the module. Thus, even if the concerns of a given type are all encapsulated in one module, other modules will be tangled to the encapsulating module, which is called the tangling problem. The concerns that cause scattering and tangling problems are called cross-cutting concerns. In the domain of programming, Aspect Oriented Programming (AOP) aims at solving these problems [19].

AOP has been considered as a solution to modularize cross-cutting concerns in programs [19]. For example, a logging mechanism can be considered as a cross-cutting concern candidate, which is scattered in the system code. To change scattered code, a programmer should find all pieces used for logging and change them. Even if the programmer encapsulates them in a separate module, the changes in the condition of the usage of this module require finding and updating all reference points. These problems increase the cost of maintenance, reduce the re-usability, and hinder the traceability of the concerns. AOP proposes implementing concerns as individual program modules and linking them to the core concern in order to solve the problems of code tangling and code scattering. It also defines some basic terminology, listed below:

- *Join points*. Join points are defined as identifiable points in the application [21]. These points enable the addition of one or several cross-cutting concerns to a program module. There are three different types of join points, i.e. field, method and type. These types are also called signatures.
- *Pointcut*. An expression in a formal language that defines relevant join points in the software system. For example, suppose logging needs to be executed when more than 500 EUR is withdrawn from an account. An expression that specifies this condition constitutes a pointcut. An appropriate pointcut language should be able to express a condition of the above kind in a systematic manner applicable to all similar conditions.
- *Advice*. A construct that enables adding or altering behavior in the program module; these constructs represent cross-cutting concerns, like the security concern. An advice could be executed before, after, or around a so-called advised join point. An *advised join point* designates the join point for which the advice should be executed.
- *Weaving*. A mechanism to alter the structure of a core concern using the execution of cross-cutting concerns. The weaving mechanism executes advices for advised join points that are determined with the help of the pointcut expressions.
- *Aspect*. A module which contains one or more cross-cutting modules is called an aspect. An aspect contains pointcuts and advices. For example, the security concern can be implemented in a module called the security aspect. This aspect can contain a pointcut and an advice. The pointcut specifies the advised join points on which the advice should be executed. The advice specifies the security code that should be executed before, after, or around the advised join point.

B. Aspect Oriented Service Composition

The investigation of aspect orientation in the programming area inspired a new direction of research in service composition where cross-cutting concerns could be separated from the service specifications. Thus, aspect orientation is also investigated in service decomposition to enable separation of cross-cutting concerns. AO4BPEL [7], [8] represents one of the examples of using aspect orientation in the service science. AO4BPEL is an extension to the Business Process Execution Language (BPEL) to support aspect orientation. This extension is defined based on the soap message life-cycle in which Charfi defines the order of advices as: "before $\rightarrow$ around $\rightarrow$ before soapmessageout $\rightarrow$ around soapmessageout $\rightarrow$ around soapmessagein $\rightarrow$ after soapmessagein $\rightarrow$ after" [6].

C. Aspect Oriented Business Process Management

This section gives a short overview of the existing attempts to introduce aspect orientation in the BPM domain.

In business process modeling, some works introduce elements in process models which call the aspects to solve scattering problems, e.g. [30], [28], [10]. Although these approaches solve the scattering problem through using aspects, they fail to solve the tangling problem [14]. Other works introduce rules that connect process models and aspects, e.g. [9], [5]. For example, Charfi et al. propose an aspect oriented extension to Business Process Model and Notation (BPMN) [24] called AO4BPMN [9]. Cappelli et al. [5] propose another aspect oriented extension to BPMN.

In Business Process Enactment area, Jalali et al. [17] propose operational semantics for aspect oriented business process enactment using Coloured Petri Nets [18]. The semantics is implemented as a service in Yet Another Workflow Language (YAWL) [1]. YAWL is a workflow managemeent systems following the Worklfow Reference Model [11]. It is designed and implemented based on Service Oriented Architecture, and it has formal definition for syntax and semantics. This service is tested on a banking case that shows how separation of cross-cutting concerns can add value to BPM.

Furthermore, there are works aimed at investigating different applications of aspect orientation in BPM domain. For example:

- How aspect oriented business process modeling can help managing process variability is investigated in [22].
- How goals can help in identifying aspects in process models is investigated in [27].
- How heuristics methods can help in identifying aspects is explained in [26].
- How aspect oriented approaches can be assessed in the BPM area is defined in [14].
- How aspect oriented business process models can be discovered from event logs is explained in [13].

Despite differences in the approaches of introducing aspect orientation in BPM, we can differentiate some common basic constructs:

- *Join points*. To avoid the tangling problem, the relation between modules and realization of concerns should not be explicitly defined in models. Instead, the system should monitor actions that try to change the state, and it should evaluate if some concerns should be considered for such actions. A point that a system uses to monitor an action is called a join point. In BPM area, these points are activities in process models, since the flow of activities is the main focus of BPM.
- *Pointcuts*. Pointcut is a rule (e.g., an expression) that defines joint points of a certain kind; this rule is used by the system for monitoring joint points. A pointcut rule defines whether a specific concern is applicable in a particular point of a process instance execution. Pointcuts are defined based on information (data) used in actions intended at changing the state of the system.
- *Advices*. The realization of a cross-cutting concern is called an advice. Advices are actions of the same kind as other actions defined in the system. For example, a module of code is considered as an advice in AOP, while a process model can be considered as an advice in BPM [16].
- *Aspects*. A group of advices with similar goals is called an aspect. Charfi et al, define security, privacy, logging and some other groups of cross-cutting concerns as aspects in BPM area [9].

Furthermore, there are two steps that an aspect oriented system should support in order to solve both scattering and tangling problems: join point selection, and advice injection.

In *join point selection*, the system monitors join points based on the pointcuts rules. This step is very important since it increases the level of support for separation of cross-cutting concerns [14]. In AO4BPMN [9], the join point selection is based on control-flow. Therefore, this approach is not able to select join points based on other perspectives like data or resources. In the extension proposed by Cappelli et al. [5], the selection is based on data and control-flow perspectives. Thus, it provides a better level of separation in comparison to AO4BPMN [14]. Jalali et al. explain how the multi-perspective selection can increase the level of support for separating cross-cutting concerns in [15].

In *advice injection*, the system incorporates advices appropriate for the points determined in the join point selection step. These advices should be enforced in appropriate places either in the process model itself or during the process instances execution, i.e. on the fly. Injecting advices in a process specification is called static weaving, while injecting advices in the process instances execution is called dynamic weaving. In BPM area, the appropriate places are defined as 'before', 'after', or 'around' in relation to the advised join point.

Both join point selection and advice injection steps are depended on pointcut rules. This paper introduces the use of declarative rules to enable aspect orientation in ACM. A summary of the declarative rules defined for workflow management systems is given in the next section. These rules serve as a basis for defining pointcuts for the ACM domain in III.

D. Declarative Workflows

Classical workflow systems lack support for flexibility and change management, which constitutes a problem in the dynamic business environment of today. Business processes need to be flexible in both managing deviations in process instances and adapting to more permanent changes in the environment. Therefore, it is very important to support the flexibility along providing support for running processes. Fig. 2 shows the relation between providing flexibility and support for business processes. On one hand, there are very structured processes, which needs a high degree of support like production workflows. On the other hand, there are unstructured processes, which require a high degree of flexibility like knowledge intensive processes such as health care processes. Declarative workflow aims to make balance between these two requirements. It provides some degree of flexibility while providing enough support at the same time, which suits knowledge intensive processes more than classical workflow systems.

Fig. 3 shows different sort of constraint, which exists in a business process, i.e. forbidden, optional, or allowed. Forbidden constraints are those which should not happen. Optional Constraints are those which can be violated; however, the system shows a warning if violation happens. Allowed constraint cannot be violated in a business process. In classical workflow systems, everything is forbidden except what is explicitly specified as allowed [2], while the declarative workflow covers all types of constraints, including the optional ones.

Fig. 3b shows a comparison between these two approaches, where the traditional approaches only support allowed constraints using control-flow specification. The workflow speci-

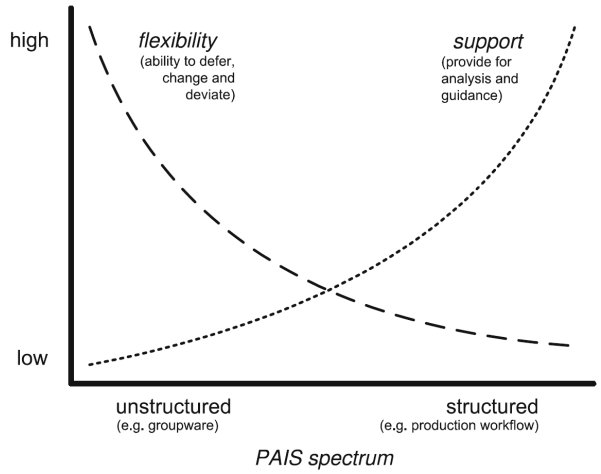


Fig. 2. Process Aware Information Systems (PAIS) Spectrum: Flexibility and Support [2]

fication can cover many allowed constraints while considering many scenarios in the model. However, this makes the model very complex. The declarative approach aims to support both allowed and optional constraint by preventing the actions defined by forbidden constraint to happen. Five constraint types can be defined in declarative workflows [25]:

- *Existence constraints* specify how often an action should be taken in a trace.
- *Choice constraints* allow specification of n-out-of-m choices in a trace.
- *Relation constraints* restrict the orders between actions through enforcing some relations among them.
- *Negation constraints* define the prohibition paths in a trace.
- *Branching constraints* specify relation and negation constraints involving more than two actions.

In workflow systems, the above constraints are used to control the order of execution of tasks. In this paper, we adapt these constraints to enable advice injection in case management systems. The details regarding these categories

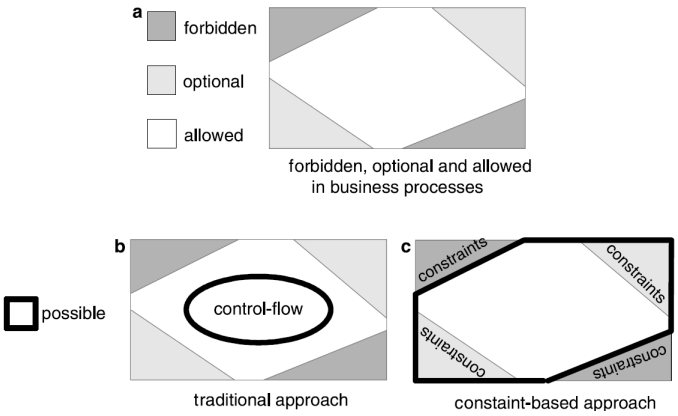


Fig. 3. Constraints in Business Processes [2]

are explained while discussing their application to aspect oriented ACM in the next section.

III. Approach

This section proposes an approach to enable aspect orientation in ACM systems. It explains how join point selection and advice injection could be applied to the ACM domain.

To select a join point, we need to track changes in the state space of a process. The change in data recorded in the system moves the ACM system from a state to a new state [3]. ACM systems need to control the movement between states in the system in a way to produce information history, capture the results of tasks execution, notify people about available tasks, etc. [29]. The movement from one state to another can be considered as potential paths which can be obligatory, optional, or prohibited. Fig. 4 shows a system which is moved from state *s* to *s'*. The state of the system can be changed to one of the potential states through potential obligatory or optional paths. These alternatives can be defined through four categories of rules of movement in the state space: *obligations*, *recommendations*, *negative recommendations* and *prohibitions* as proposed in [4]. Obligations can be considered as obligatory paths which should be followed. Recommendations and negative-recommendations can be considered as optional paths, and prohibitions can be considered as prohibited paths.

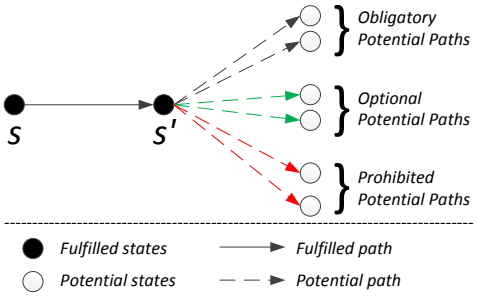


Fig. 4. Potential Paths to move in the state space

There are different ways to control movement in the state space of an ACM system. As an example, the movement between states could be controlled by form based rules, as it is done in *iPB*, which is a case-oriented business process support system [3]. *iPB* provides forms for knowledge workers to report the result of business operations, thus forms represent actions aimed at changing the state of the system. These forms are available to a knowledge worker based on certain rules that define who can change the state of the system and in which direction.

In practice, it is impossible to have all information that is needed for movement in the state space before an action is started; some information is obtained during the execution of the action. For example, if a clerk in a bank has the right to transfer money with some limitation, he/she has the right to access the *transfer money* form, but some security check should be added to the form for that user. This addition constitutes an advice from the aspect orientation point of view. The security check will require additional actions when the user signs a transaction with the amount that exceeds the limitation assigned to his/her category of users. Such actions

can include changing the nature of potential paths through the state, for example, an optional path can become obligatory.

Summarizing the deliberation above, to inject an advice, the system should be able to alter the potential paths based on the evaluation of pointcut rules. To define possible pointcut rules and advices that correspond to them, this paper proposes using trace constraints, which are defined in BPM to manage the control perspective based on occurrence of activities in a trace [25]. To adapt the rules, we, first, introduce the concept of lifecycles for actions in ACM and connect it to the movements in the state space. This allows us to interpret the ideas from [25] in the terms more acceptable for ACM.

A. Action Lifecycle in ACM

This section introduces the idea of a lifecycle for actions in ACM. Fig. 5 shows the states of the action lifecycle, and their relations to the system state. An action can be selected, which is shown by `Selected` state in the action lifecycle. Selecting of an action does not change the state of the system visible for other actions. The results achieved by the selected action are preserved in a temporary, or intermediate state of the system. As soon as the temporal state of the system is changed by the action, the selected action is considered to be in the `In Progress` state in the action lifecycle. The selected action can be canceled or committed, which is shown using `Canceled` and `Committed` states in the action lifecycle. The cancellation of an action brings the state of the system to original state, i.e. `S` in the figure. However, the commitment of an action moves the state of the system forward, i.e. `S'` in the figure. Furthermore, selecting an action already in progress can be considered as cancellation of the previous in progress action and selecting it again as a new action. This lifecycle enables the system moving between states based on commitment/cancellation of different actions.

Selecting of actions is governed by rules that allow, prohibit the movement to the targeted by the action state. According to our previous work [3], in ACM the structure of the system state may not be fixed, and selecting a certain action may mean extending the state space with new dimensions, which will be explored in the next sections.

B. Join Point Selection

Join points can be defined in ACM as actions which intend introducing changes in the state of the system. Based on this definition, we can introduce two types of joint points - action selection, and action commitment. The first one is related to the intention of changing the system state in a certain direction without defining details on how the new state would like; the second one includes these details.

C. Advice Injection

Advice injection can be defined as changing the potential paths through the system state space based on the results of join point evaluation. If the evaluation of the joint point results in the need of injecting an advice, the set of possible paths through the system space should be reconfigured. The reconfiguration can be done by changing the rules that govern the system like:

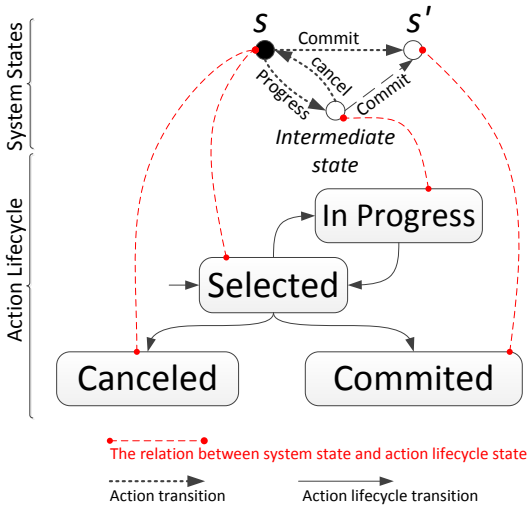


Fig. 5. Action Lifecycle

- Some actions that have been allowed become prohibited
- Some actions that were not mandatory, or even prohibited became mandatory, i.e. new obligations are added. In addition, some options that were prohibited may become allowed or even mandatory. With addition of new options and obligations, there may arise a need to extend the state space of the system, so that a path defined by new options and obligations becomes possible.

The above can be done in both cases, when the joint point is selecting an action, and when committing it. However, for the joint point of the second type, there is another possibility. The temporal state that is allowed to commit when considering it without the advice injection may become improper when the advice injection is added. In this case, the commitment is rejected, and a new targeted state for the action in progress is defined. Again, if needed, the space space of the system can be extended in order to define this new targeted action. The commitment is rejected, and the process participant who tries to commit the action gets instructions on the new rules of commitment. He/she then can choose to cancel the action, or leave it in progress until all necessary changes in the intermediate state are completed, and the action is allowed to commit under the new rules that resulted from the advice injection.

As the case of joint point that coincide with commitment has more options, below, we only consider the joint points of this kind. The discussion will apply to the joint points that represent action selection, except of the rules that prevent committing an action.

The re-configuration of a path can be performed through definition of so-called pointcuts. This paper proposes using some of the five types of constraints, which are described in section II to define how paths should be re-configured.

D. Defining pointcuts with the help of trace constraints

The pointcuts we introduce in this paper are defined using trace constraints introduced in II. Though in II we have introduced five types of constraints; only two of them are appropriate for defining pointcuts in the frame of the approach presented above, namely relation constraints and negation constraints; both are described in the subsections below.

1) Relation constraints: These constrains restrict the ordering of actions by imposing some restrictions on a trace of states in an ACM system. Fig. 6 shows different sort of relation constraints which can be used for restricting actions paths. The first column indicates the name of constraint, and the second one defines the meaning. These relations are defined in the form accepted for declarative workflows to support flexible execution of activities [23]. In the text that follows, these constraints are explained in the form more appropriate to the case management systems domain in light of Fig. 5.

A case management system should configure the path for each constraint as follows:

1) *Responded existence*. The system can commit the join point if and only if action A has been occurred before it, or action A is defined as mandatory in the path that goes through the state to be committed. If neither is true, the system should (a) insert rules that makes A mandatory after the commitment and commit, or (b) prevent commitment of the join point to enforce action A to happened before the commitment, or (c) prevent commitment and require changes in the intermediate (temporary) state that would make A mandatory or allows insertion of the rules that makes A mandatory.
2) *Response restriction*. The system can commit the join point if and only if action A has been occurred before it, or action A is defined as mandatory in the path that goes through the state to be committed. If none of the conditions are true, the system should (a) insert rules that make A mandatory after the commitment and commit, or (b) prevent commitment of the join point to enforce action A to happen before the commitment, or (c) prevent commitment and require changes in the intermediate (temporary) state that would make A mandatory or allows insertion of the rules that makes A mandatory.
3) *Alternate response restriction*. The system should follow the response restriction scenario, except it should also ensure having a prohibition of the same action as the join point's action until action A happens. In case such prohibition does not exist, the system should add it or not commit the transaction until such prohibition becomes natural or can be added.
4) *Chain response restriction*. The system should follow the response restriction scenario, except it should also ensure having a prohibition of any action until action A happens. In case that such prohibition does not exist, the system should add it or not commit the transaction until such prohibition becomes natural or can be added.
5) *Precedence restriction*. The system can let commitment of the join point if and only if action A has been occurred before it. Otherwise, the system should prevent commitment of the join point to enforce action A to happen before the commitment.
6) *Alternate precedence restriction*. The system should follow the precedence restriction scenario, except it should also check whether the same action as the join point's action has already happened after action A. In the latter case, the system should cancel the join point's action.
7) *Chain precedence restriction*. The system can let commitment of the join point if and only if action A has been occurred exactly before it. Otherwise, the system should prevent commitment of the join point to enforce action A to happen exactly before the commitment.

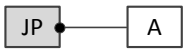
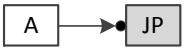
Constraint	Meaning	Graphical representation
1. responded existence	if the pointcut rule is met then A should be occur ether before or after the join point completion	
2. response	if the pointcut rule is met then eventually A occurs after the completion of the join point	
3. alternate response	if the pointcut rule is met then eventually A occurs after the completion of the join point without other occurrences of the join point in between	
4. chain response	if the pointcut rule is met then then A occurs in the next position after completion of the join point	
5. precedence	if the pointcut rule is met then A occurs before completion of the join point	
6. alternate precedence	if the pointcut rule is met then A occurs before completion of the join point without other occurrences of the join point in between	
7. chain precedence	if the pointcut rule is met then A occurs in the next position before the completion of the join point	

Fig. 6. Relation Constraints, adapted from [23]

There are other restrictions defined for declarative systems in [23], such as co-existence, succession, alternate succession and chain succession that are not applicable for our framework and therefore are excluded from the list in Fig. 6.

2) Negation constraints: The negation constraints specify negative relations between actions, so they are appropriate to define the prohibition path for the state space. Fig. 7 shows three negation constraints, which are used in declarative workflows [23]. In the text that follows, these constraints are explained in the form more appropriate to the case management systems domain in light of Fig. 5.

A case management system should configure the path for each constraint as follows:

1) *Not co-existence constraint*. The system should check

Constraint	Meaning	Graphical representation
1. not co-existence	if the pointcut rule is met then the join point and A cannot occur together	JP ⊕ A
2. not succession	if the pointcut rule is met then A cannot eventually occur after completion of the join point	JP ⊕ A
3. not chain succession	if the pointcut rule is met then A cannot occur in the next position after the completion of the join point	JP ⊕ A

Fig. 7. Negation Constraints, adapted from [23]

whether action A has been executed before or not. In case the action has been executed before, the system should cancel the action. Otherwise, the system should check if a prohibition rule for action A will be in place after commitment. If yes, the action can be committed. Otherwise, the system should add such a prohibition, or, if it is not possible, require changes in the intermediate (temporary) state that would make A prohibited or allows insertion of a rule that makes A prohibited.

2) *Not succession constraint*. The system should check if a prohibition rule for action A will be in place after commitment. If yes, the action can be committed. Otherwise, the system should add such a prohibition, or, if it is not possible, require changes in the intermediate (temporary) state that would make A prohibited or allows insertion of a rule that makes A prohibited.

3) *Not chain succession constraint*. The system should check if a prohibition rule for immediate execution of action A will be in place after commitment. If yes, the action can be committed. Otherwise, the system should add such a prohibition, or, if it is not possible, require changes in the intermediate (temporary) state that would make immediate execution of action A prohibited or allows insertion of a rule that makes immediate execution of action A prohibited.

IV. Implementing Aspect-Orientation in Form-Based Case Management Systems

In this section, we demonstrate how the cross-cutting concerns could be implemented in case management systems. As an example, we will consider their implementation in the *iPB* tool [12] that supports building form-based case management systems. First, we shortly describe the notions on which *iPB* is built, and then discuss how *iPB* could be augmented to implement the cross-cutting concerns.

A. Short overview of iPB

A case management system is built in *iPB* based on the following four interconnected concepts: *Process map, Process step, Process form, Form field*. The basic relationships between these concepts are as follows.

- A *process map* consists of a collection of named *process steps* arranged on a two-dimensional surface called *process layout*. The layout consists of two areas: (a) the upper row called *flow-independent* steps, and (b) a lower area, which is a two dimensional matrix called *flow-dependent* steps, see Fig. 8.
- Each *process* step in a *process map* has a *step form* attached to it.
- A *step form* consists of a collection of named *fields* arranged in a two-dimensional matrix called *form layout*, see Fig. 9. Each field belongs to a certain type. There are simple types, like, text, multi-text, date, date-time, option list, checkbox (Boolean), etc., and complex types like uploaded file, journal, person, organization. In addition, field collections that belong to different step forms can intersect. This is done by defining fields in one form, and then referring to them in other forms.

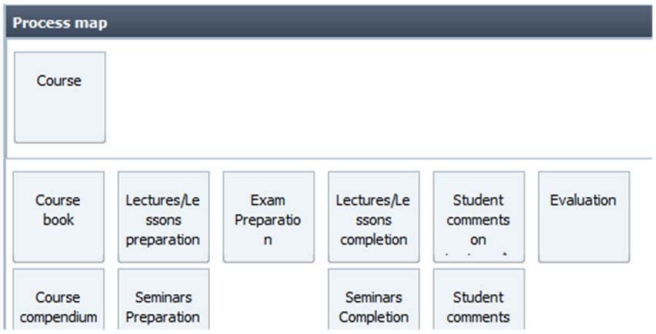


Fig. 8. Process map

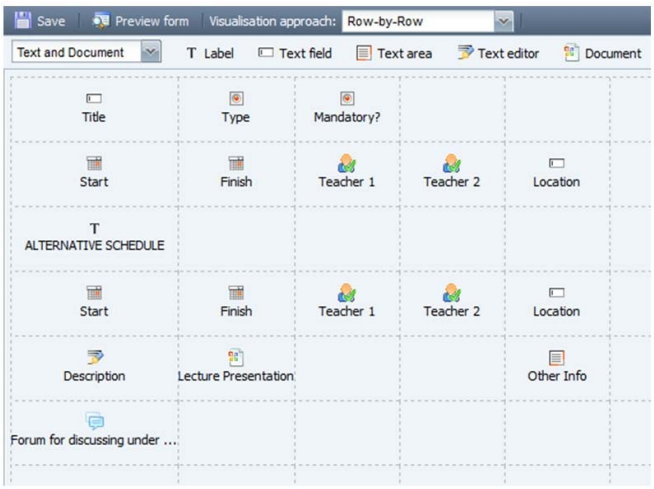


Fig. 9. Step form for step Lectures/Lesson preparation from Fig. 8

The runtime system interprets step forms as web forms for inputting, viewing and changing information, see Fig. 10. The process map constitutes a mechanism for user navigation through these forms serving as a table of content for a particular process instance, see Fig. 8. To open a web-form, the user clicks on the step in the instance map (see Fig. 8). As it can be seen from Fig. 8, some steps are allowed to have multiple forms at runtime (see the tabs for each lecture in Fig. 10).

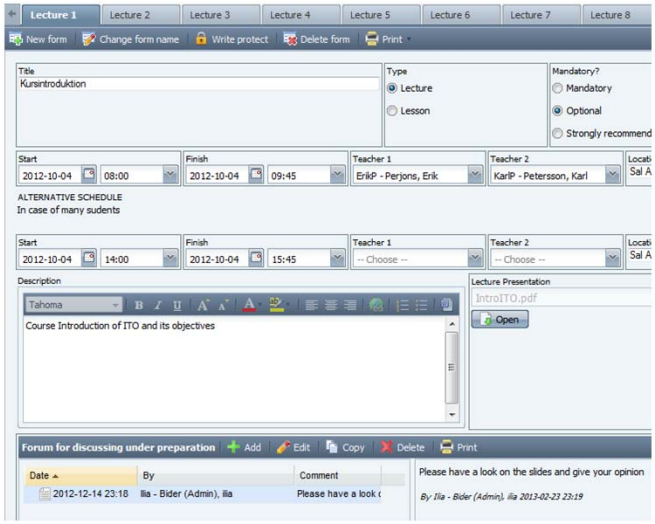


Fig. 10. Step form from Fig. 9 as a web-form

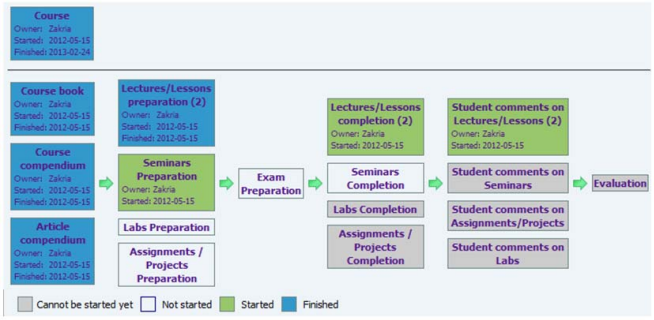


Fig. 11. Instance process map from Fig. 8 as a user navigation mechanism

The normal order of accessing the forms is from top to bottom and from left to right. However, the map itself does not prevent to skip some steps or to begin a process instance with a form that is placed in the middle of the layout. If constraints on the access order are needed, they are established by so called business rules. One type of such rules controls whether the user can open a particular step form based on the state of commitment of other forms. Steps that cannot yet be clicked on are colored gray, see Fig. 11. Such rules are specified with the help of a square matrix where both rows and columns correspond to the steps defined in the process. In this matrix, the content of a cell can determine that the row step can be started only after the column step has been commitment (blue color in Fig. 8), or started (green color in Fig. 11). Other types of rules prescribe synchronization of steps with multiple forms, e.g., steps "Lectures/Lessons preparation" and "Lectures/Lessons commitment" in Fig. 8 and 4. Yet other types of business rules establish when data in a step form can be saved or the step can be closed. Such rules are expressed via defining some fields on a form as mandatory for save, or close.

As with the access order, the map itself does not define who has rights to access web forms, and which rights, e.g. read, modify, close, etc. The rights are defined in user profiles created with the help of a special module called *Profile editor*.

B. Implementing cross-cutting concerns in iPB

As it can be seen from the description in the previous section, *iPB* already implements the idea of temporal state and its commitment. Selecting a form to fill constitutes selecting an action to complete. Filling the form constitutes creating a temporal state. Marking the form as finished, i.e. closing the form, constitutes commitment of the temporal state. *iPB* already includes one kind of post-conditions for commitment, some fields on the form can be defined as mandatory for close. If a user tries to close the form without filling these fields, an error message will be issued. This message will require the user to undertake additional efforts related to filling mandatory for close fields. Another type of conditions for committing a temporal state can be defined with the help of the Profile editor via limiting the access rights for closing a particular form to a special category of users. The user who tries to close the form for which he/she does not have access rights for close will get a message that will require him/her to ask somebody else to complete the action.

Cross-cutting concerns could be introduced in *iPB* through post-conditions rules invoked at the time of committing/closing the form. As the cross-cutting concerns are of general nature, they cannot be connected to particular forms. One way of defining a cross-cutting rule could be through attaching it to a fragment consisting of a group of fields that can be found in several forms. A rule attached to this fragment can, for example, prohibit closing a form by a particular category of users dependent on the values inputted in the fields of the fragment. Getting a notification when trying to close a form that includes the fragment to which the rule is attached, the user can decide to change the values, and then commit, or ask a user with the proper access rights to close the form (commit the temporal state). A field representing the amount of money included in a financial transaction could constitute a fragment to attach a rule that limits the amount of money processed by a junior financial officer independently of the nature of the transaction. Trying to complete a transaction with a greater than the limit value will result in a message that gives the officer two options, either diminish the value, or invite a senior officer to commit the transaction.

Another way to introduce the same limitation is by adding authorization for a junior officer. When the officer commits the transaction, a new form is added for a senior officer to authorize the transaction. The latter can be considered as extending the state space of the current process instance. In addition, an obligation is added requiring this form to be filled and committed before any other form is committed in the current instance of the process. This implementation corresponds to the *Chain response restriction* introduced in III, see Fig. 6.

Note that neither of the implementations discussed above are implemented in *iPB* at this time. The goal of this section is to show how cross-cutting concerns could be implemented in a business process support system that is built on a paradigm completely different from the workflow paradigm, of which declarative workflows are a particular case. The discussion shows that the rules introduced in III, though defined based on the works from the declarative workflows domain, have more general nature and can be applied to case management systems based on the state-oriented paradigm. In addition, the fragment

dependent rules suggested above has no correspondence to the paths rules discussed in III, thus requiring extending the taxonomy introduced in the previous sections of this paper.

V. Conclusion

This idea paper proposes an approach to changing on the fly rules that govern execution of business processes based on aspect-oriented principles and declarative constraints. It defines the lifecycle for actions and connects it to changes in the state of a business process support system. It proposes a taxonomy of aspect-oriented rules that enable changing the path in the state space on the fly. It also discusses how the proposal could be used to extend existing form-based case management system systems, such as *iPB*. The paper shows how to achieve separation of cross-cutting rules from standard actions, which could increase the re-usability of rules, and facilitate easier maintenance of the system. The approach needs further investigation, and it requires to be implemented in an case management system to be validated. By presenting the approach before such validation, we hope to inspire other researchers to investigate the usage of declarative constraints to introduce aspect orientation in Business Process Management in order to achieve a higher degree of flexibility when controlling the execution of activities.

References

[1] W. Aalst, L. Aldred, M. Dumas, and A. ter Hofstede. Design and implementation of the YAWL system. In A. Persson and J. Stirna, editors, *CAiSE*, volume 3084 of *LNCS*, pages 281–305. Springer, Springer, 2004.

[2] W. Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23(2):99–113, 2009.

[3] I. Bider, A. Jalali, and J. Ohlsson. Adaptive case management as a process of construction of and movement in a state space. In Y. Demey and H. Panetto, editors, *On the Move to Meaningful Internet Systems: OTM 2013 Workshops*, volume 8186 of *Lecture Notes in Computer Science*, pages 155–165. Springer Berlin Heidelberg, 2013.

[4] I. Bider and A. Striy. Controlling business process instance flexibility via rules of planning. *International Journal of Business Process Integration and Management*, 3(1):15–25, 2008.

[5] C. Cappelli, F. Santoro, J. do Prado Leite, T. Batista, A. Medeiros, and C. Romeiro. Reflections on the modularity of business process models: The case for introducing the aspect-oriented paradigm. *BPM Journal*, 16:662–687, 2010.

[6] A. Charfi. *Aspect-oriented workflow languages: AO4BPEL and applications*. PhD thesis, der Technischen Universitat Darmstadt, Darmstadt, November 2007.

[7] A. Charfi and M. Mezini. Aspect-Oriented Web Service Composition with AO4BPEL. In L. Zhang and M. Jeckle, editors, *Web Services*, volume 3250 of *LNCS*, pages 168–182. Springer Berlin Heidelberg, 2004.

[8] A. Charfi and M. Mezini. Ao4bpel: An aspect-oriented extension to bpel. *World Wide Web*, 10(3):309–344, 2007.

[9] A. Charfi, H. Müller, and M. Mezini. Aspect-Oriented Business Process Modeling with AO4BPMN. In T. K. et al., editor, *Modelling Foundations and Applications*, volume 6138 of *LNCS*, pages 48–61. Springer, 2010.

[10] D. C. Collell. Aspect-oriented modeling of business processes. Master's thesis, der Technischen Universitat Darmstadt, Darmstadt, 2012.

[11] D. Hollingsworth. Workflow management coalition - the workflow reference model. Technical report, Workflow Management Coalition, Jan. 1995.

[12] IbisSoft. ipb reference manual, 2010. http://docs.ibissoft.se/node/3.

[13] A. Jalali. Aspect Mining in Business Process Management. In *to appear in Proc. 13th International Conference on Perspectives In Business Informatics Research (BIR)*, 2014.

[14] A. Jalali. Assessing Aspect Oriented Approaches in Business Process Management. In *to appear in Proc. 13th International Conference on Perspectives In Business Informatics Research (BIR)*, 2014.

[15] A. Jalali and P. Johannesson. Multi-Perspective Business Process Monitoring. In S. N. et al., editor, *Enterprise, Business-Process and Information Systems Modeling*, volume 147 of *LNBIP*, pages 199–213. Springer Berlin Heidelberg, 2013.

[16] A. Jalali, P. Wohed, and C. Ouyang. Aspect oriented business process modelling with precedence. In J. M. et al., editor, *BPMN*, volume 125 of *LNCS*, pages 23–37. Springer, 2012.

[17] A. Jalali, P. Wohed, C. Ouyang, and P. Johannesson. Dynamic weaving in aspect oriented business process management. In R. Meersman, H. Panetto, T. Dillon, J. Eder, Z. Bellahsene, N. Ritter, P. Leenheer, and D. Dou, editors, *CoopIS 2013 Conferences*, volume 8185 of *Lecture Notes in Computer Science*, pages 2–20. Springer Berlin Heidelberg, 2013.

[18] K. Jensen, L. Kristensen, and L. Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007.

[19] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Akit and S. Matsuoka, editors, *ECOOP'97 Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Berlin Heidelberg, 1997.

[20] M. La Rosa, P. Wohed, J. Mendling, A. ter Hofstede, H. Reijers, and W. M. P. Van der Aalst. Managing process model complexity via abstract syntax modifications. *Industrial Informatics, IEEE Transactions on*, 7(4):614–629, 2011.

[21] R. Laddad. *AspectJ in Action*. Manning Publications Co., second edition edition, 2003.

[22] I. Machado, R. Bonifácio, V. Alves, L. Turnes, and G. Machado. Managing variability in business processes: an aspect-oriented approach. In *Proceedings of the 2011 international workshop on Early aspects*, EA '11, pages 25–30, New York, NY, USA, 2011. ACM.

[23] F. M. Maggi, R. J. C. Bose, and W. M. van der Aalst. A knowledge-based integrated approach for discovering and repairing declare maps. In *Advanced Information Systems Engineering*, pages 433–448. Springer, 2013.

[24] OMG. Business Process Model and Notation (BPMN), Version 2.0, 2011. availble on http://www.omg.org/spec/BPMN/2.0/PDF/, accessed Mar 2012.

[25] M. Reichert and B. Weber. *Enabling flexibility in process-aware information systems: challenges, methods, technologies*. Springer, 2012.

[26] F. Santos, C. Cappelli, F. Santoro, J. do Prado Leite, and T. Batista. Analysis of heuristics to identify crosscutting concerns in business process models. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1725–1726, New York, NY, USA, 2012. ACM.

[27] F. Santos, J. Sampaio do Prado Leite, C. Cappelli, T. Batista, and F. Santoro. Using goals to identify aspects in business process models. In *Proceedings of the 2011 international workshop on Early aspects*, EA '11, pages 19–23, New York, NY, USA, 2011. ACM.

[28] A. Shankardass. The dynamic adaptation of an aspect oriented business process in a service oriented architecture platform. Master's thesis, Athabasca University, Athabasca, Alberta, 2009.

[29] K. Swenson. Comparison: Acm vs. bpm', Jan. 2010. http://www.xpdl.org/nugen/p/adaptive-case-management/public.htm.

[30] J. Wang, J. Zhu, H. Liang, and K. Xu. Concern oriented business process modeling. In *e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on*, pages 355–358, 2007.