Controlling the level of business process instance flexibility via rules of planning

Ilia Bider, Alexey Striy

IbisSoft, Box 19567, SE-10432 Stockholm, Sweden
ilia@ibissoft.se, a_streey@yahoo.co.uk

Abstract. When an organization decides on the level of flexibility in handling business process instances, it needs to impose this level in operational practice. The way of imposing a given level of flexibility depends on the means employed for controlling business processes. When a Business Process Support (BPS) system is used, the flexibility limits can be incorporated in it. The paper discusses how a given level of flexibility can be imposed by a BPS system built based on the state oriented view on business processes. This is done by combining different kinds of rules of planning: obligations, prohibitions, recommendations, and negative recommendations. Changing the status of a rule from obligation to recommendation, or from prohibition to negative recommendation gives more flexibility, and vice versa. The discussion is illustrated with the help of a simplified example already implemented in a BPS system called Probis.

1 Introduction

The level of flexibility when handling business process instances is decided based on the business environment, external, and internal, in which an organization functions. However, when the level is decided upon, it needs to be institutionalized so that everybody knows the limits of flexibility and follows them. The way of the institutionalization depends on how business processes are controlled in the given organization. In case of manual control, the institutionalization is done via rules included in manuals, employees books, etc. In case there is a computer based Business Process Support (BPS) system, limits of flexibility can be incorporated in the system, which will help the users to follow the rules without consulting manuals each time they need to deviate from a usual pattern.

The way of incorporating flexibility limits, certainly, depends on what principles a given BPS system has been built. We differentiate four different views on business processes that can be used when building a BPS system [1], namely: (1) input/output flow, (2) workflow, (3) agent related view, and (4) state flow. In this paper, we discuss how a desired level of flexibility can be introduced in a system built based upon the state-oriented view (state flow) [3,4]. As the underlying conceptual models for different views differs, we do not expect that the findings of this paper can be easily applied to the systems built based on other views. More probably, they will not be applicable or will need substantial modification. However, the task of proving or disproving this hypothesis is not included in the research reported in this paper.

Due to the lack of space, we do not discuss the topic of how flexibility control can be introduce in BPS built on other views on business processes. We concentrate solely on the systems built on the state-oriented view. In such a system, control over process instances is realized via so called rules of planning. As was suggested in [2], the natural way of introducing flexibility in these circumstances is by introducing several kinds of rules. In the work reported in this paper, we ensure various levels of flexibility through differentiating four types of rules, which is one more than in [2]:

1. Obligations
2. Recommendations
3. Prohibitions
4. Negative recommendations

By combining rules of different types, we can obtain various levels of flexibility. For example, "obligation + prohibition to do otherwise" constitutes a strict rule which does not allow any flexibility, while "recommendation + negative recommendation not to do otherwise", allows full flexibility.

From the taxonomy of flexibility point of view [4], the material discussed in this paper can be positioned as follows: *Abstract level of change = Instance*, *Subject of change = Operational perspective*.

From a broader perspective, our rules of planning represent a kind of business rules (BR). The literature on BR and its application to software design is vast, see for example [5], and its list of references. Nevertheless, we found no theoretical, or practically oriented papers that suggested an approach that could be applied to rules of planning. A search to find an approach to formalize the idea of recommendation gave even less promising results. We were, more or less, forced to work out our own approach to dealing with rules of planning in general, and with recommendations in particular. In this paper we do not discuss the existing BR literature, neither do we present our approach in a general form. Instead, we concentrate on explaining the ideas using an example.

Though the example we use may seem a bit artificial, we consider it quite representative for the problems we face when defining flexible rules of planning. It was chosen due to its simplicity, as it allows explaining the main ideas in a paper of limited size. The paper is written according to the following plan. In section 2, we shortly review the main ideas of the state oriented view on business processes. In section 3, we show how various level of flexibility can be achieved via different kinds of rules of planning. In section 4, we shortly review current implementation of such rules in a BPS system. Section 5 contains concluding remarks and plans for the future.

2 State oriented view on business processes

The main concept of the state-oriented view on business processes is the process's *state* [2,3]. The process's state is aimed to show how much has been done to achieve the operational goal of the process instance, and how much is still to be done. A state of a process is represented by a complex structure that includes attributes, and references to various active and passive participants of the process, such as process

owner, documents, etc, see Fig. 1 for en example. A state of a given process instance does not show what activities have been executed to reach it, it only shows the results achieved so far.

A goal of a business process can be defined as a set of conditions that must be fulfilled before a process instance can be considered as finished. A process state that satisfies these conditions is called *final state* of the process.

The process is driven forward through activities executed either automatically or with a human assistance. An activity can be viewed as an *action* aimed at changing the process state in a special way. Activities can be planned first and executed later. A *planned activity* records such information as type of action (goods shipment, compiling a program, sending a letter), planned date and time, deadline, name of a person responsible for an action, etc.

All activities currently planned for a process instance make up its *operational plan* or to-do list, see Fig. 2 for en example. The plan lists activities the execution of which diminishes the *distance* between the current state of the process instance and the *nearest* final state.

The plan together with the "passive" state (attributes and references) constitutes a so called *generalized state* of the process, the plan being an "active" part of it. When an activity is executed, a process changes its generalized state. Changes may concern the passive and/or active parts of the state. At the minimum, the executed activity disappears from the plan. In addition, changes are introduced in attributes and references and/or new activities are planned to drive the process forward.

With regards to the generalized state, the notion of a *valid* state can be defined in addition to the notion of *final state*. To be valid, the generalized state should include all activities required for moving the process to the next stipulated state. A business process type can be defined as a set of valid generalized states. This definition can be converted into an operational procedure called *rules of planning*. The rules specify what activities could/should be added to an invalid generalized state to make it valid. Using these rules, the process instance is driven forward in the following manner. First, an activity from the operative plan is executed and the state of the process is changed. Then, an operative plan is corrected to make the generalized state valid.

3 Defining flexibility in terms of rules of planning

The main ideas of our approach are demonstrated and explained on an example of a process of organizing a meeting. The state of such process can be represented as a screen capture in Fig. 1. Each meeting has a number of so called core participants (see "meeting participants" in Fig. 1), meeting date and place. Fig. 2 represents the list of activities currently planned for the meeting process from Fig. 1. In this list, each core participant has an activity *Meeting* planned for him/her that indicates that he/she should attend a meeting at the specified date and time. The list on Fig. 2 represents the "normal" correspondence between the sate and the plan. Below we consider several scenarios that ensure that such correspondence is imposed strictly or with some deviations allowed.

Scenario 1 - Strict regulation. Core participants must attend, and only them are allowed to attend. This scenario can be described by a combination of obligations and prohibitions as follows:

- *Obligation*. If a person belongs to the core participants, there should be an activity "Meeting" assigned to him/her in the plan.
- *Obligation*. Date and time of a *Meeting* activity must be the same as prescribed by the process state.
- *Prohibition*: A *Meeting* activity is allowed to be in the plan only if it is assigned to a core participant.
- *Prohibition*: Only one *Meeting* activity per person is allowed in the plan.

The rules are applied in the following manner. If a person is added to the participants list, a new *Meeting* activity assigned to him/her is automatically added to the plan. If this activity is later manually deleted, it will appear once more. If a person is deleted from the participants list, his/her *Meeting* activity is also deleted. If a *Meeting* activity is manually added to the plan and assigned to a person who currently is not on the participants list, the activity will be deleted. In addition, date and time parameters are always corrected to the actual date and time from the process state. Multiple *Meeting* assignments to the same person are reduced to one.

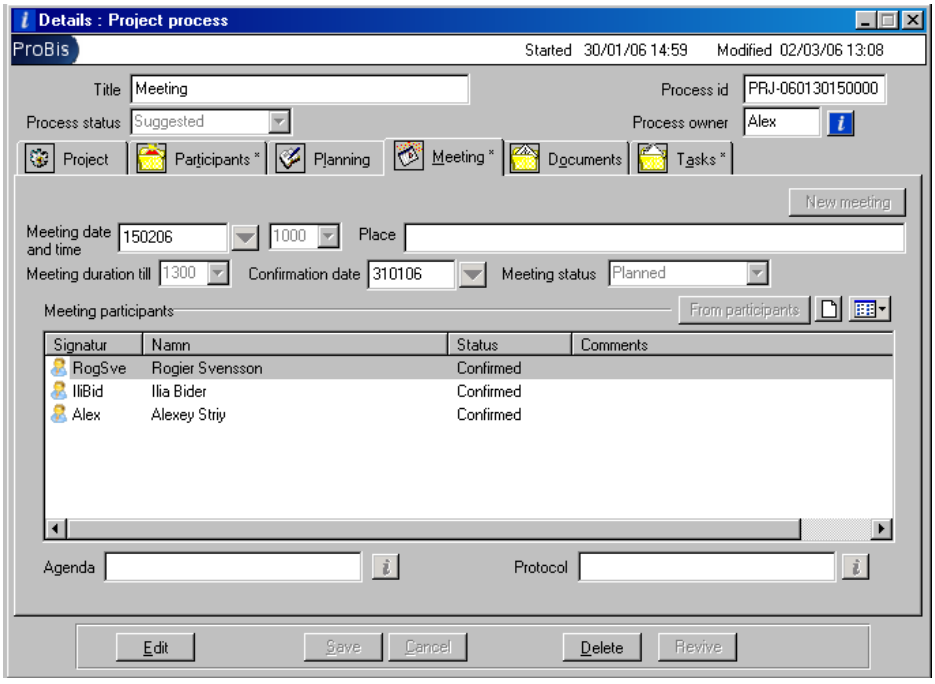

Figure 1. State of the meeting process

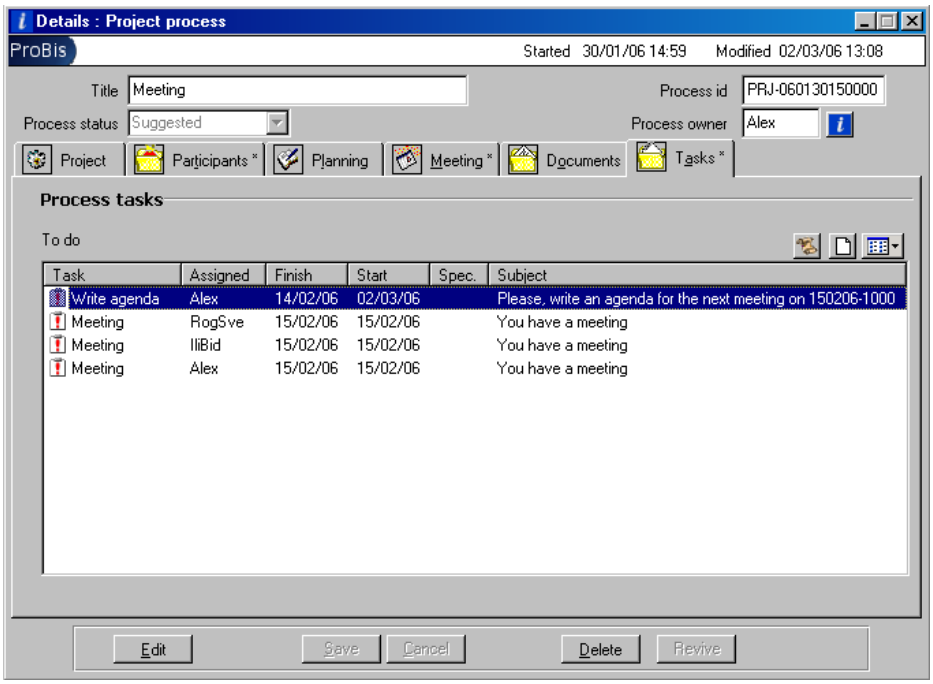

Figure 2. Activities planned in the frame of the process instance

Note. Application of the rules described above is based on the following assumption. The user is allowed to manipulate the process plan freely. It means that he is permitted to add or delete any activities he wants. Rules of planning are applied immediately after he finishes his/her job, and presses the *Save* button. The application of rules could be maid more sophisticated, so that a user is not allowed to delete a mandatory activity (task). However, this is not always possible. For example, if a mandatory planning includes two alternative activities, you need to allow the user to delete the existing one first in order to insert the alternative. The *Save* button solution adapted in this work allows us to treat all planning rules in the same, though simplified fashion. More details see in the next session.

Scenario 2 – Guests allowed. Core participants must attend but guests are allowed. This scenario can be described by a combination of obligations, negative recommendations, and prohibitions as follows:

- *Obligation*. If a person belongs to the core participants, there should be an activity *Meeting* assigned to him/her in the plan.
- *Obligation*. Date and time of a *Meeting* activity must be the same as prescribed by the process state.
- *Negative recommendation*: A *Meeting* activity is not recommended to be in the plan if it is assigned to a person who is not a core participant.
- *Prohibition*: Only one *Meeting* activity per person is allowed in the plan.

The rules are applied in the following manner. If a person is added to the participants list, a new *Meeting* activity assigned to him/her is automatically added to the plan. If this activity is later manually deleted, it will appear once more. If a person is deleted from the participants list, his/her *Meeting* activity is also deleted (negative recommendation). If a *Meeting* activity is manually added to the plan and assigned to a person not on the participants list, the activity will stay in the list. In addition, date and time parameters are always corrected to the actual date and time from the process state. Multiple *Meeting* assignments to the same person are reduced to one.

Scenario 3 – Permission to skip, but no guests. Core participants are recommended to attend, and only them are allowed to attend. This scenario can be described by a combination of recommendations, obligations and prohibitions as follows:

- *Recommendation*. If a person belongs to the core participants, it is recommended to have an activity *Meeting* assigned to him/her in the plan.
- *Obligation*. Date and time of a *Meeting* activity must be the same as prescribed by the process state.
- *Prohibition*: A *Meeting* activity is allowed to be in the plan only if it is assigned to a core participant.
- *Prohibition*: Only one *Meeting* activity per person is allowed in the plan.

The rules are applied in the following manner. If a person is added to the participants list, a new *Meeting* activity assigned to him is automatically added to the plan (recommendation). If this activity is later manually deleted, it won't appear once more. If a person is deleted from the participants list, his/her *Meeting* activity is also deleted. If a *Meeting* activity is manually added to the plan and assigned to a person not on the participants list, the activity will be deleted. In addition, date and time parameters are always corrected to the actual date and time from the process state. Multiple *Meeting* assignments to the same person are reduced to one.

Scenario 4 – Full flexibility. Core participants are recommended to attend, and guests are allowed. This scenario can be described by a combination of recommendations, negative recommendations, obligations and prohibitions as follows:

- *Recommendation*. If a person belongs to the core participants, it is recommended to have an activity *Meeting* assigned to him/her in the plan.
- *Obligation*. Date and time of a *Meeting* activity must be the same as prescribed by the process state.
- *Negative recommendation*: A *Meeting* activity is not recommended to be in the plan if it is assigned to a person who is not a core participant.
- *Prohibition*: Only one *Meeting* activity per person is allowed in the plan.

The rules are applied in the following manner. If a person is added to the participants list, a new *Meeting* activity assigned to him is automatically added to the plan (recommendation). If this activity is later manually deleted, it won't appear once more. If a person is deleted from the participants list, his/her *Meeting* activity is also deleted (negative recommendation). If a *Meeting* activity is manually added to the plan and assigned to a person not on the participants list, the activity will stay in the list. In addition, date and time parameters are always corrected to the actual date and

time from the process state. Multiple *Meeting* assignments to the same person are reduced to one.

As we can see from the above scenarios, flexibility can be achieved by substituting an obligation to recommendation, or prohibition to negative recommendation. The difference between *obligation* and *recommendation*, and *prohibition* and *negative recommendation* shows itself only when planning is done manually, i.e., not through changes in the process state (core participants list in our example). An obligation does not allow manual removing of activities, while a recommendation does allow it. In the same way, a prohibition will remove manually added activities, while a negative recommendation will allow them to stay.

4 Implementation in ProBis

The approach to handling flexibility through rules of planning has been implemented in a BPS system called ProBis [6]. The planning system consists of a set of independent rules. Each rule is manually coded, but its inclusion in the system is done via an invocation table stored in the database. Rules can be added/deleted activated/deactivated via a special system administrator screen.

In ProBis, a process state is represented by a tree structure, the tree relevant to the example from the previous section being shown in Fig. 3. In this structure, the process itself is represented as a root node, whereas child nodes represent various elements included in the definition of the process state, like meeting participants, planned activities etc. Each node, root, as well as child, has a set of attributes assigned to it.

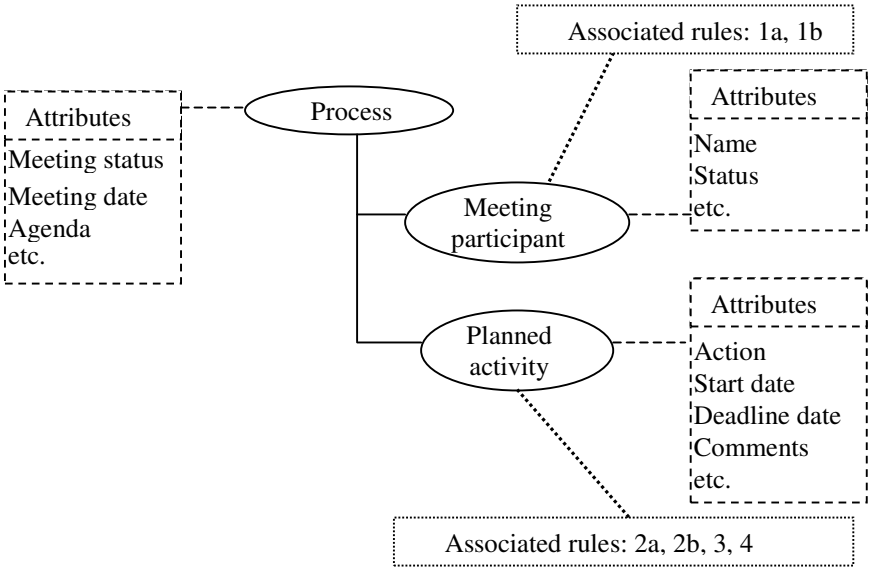

Figure 3. Process tree structure

Each individual rule in the invocation table is associated with a node in the process tree. A rule may be supplied with a condition on attribute values of the node with which it is associated, or the upper nodes. A condition can take into account current values of attributes as well as information about changes. If a condition is supplied, the rule is applied only if the condition yields true for a given process instance.

Application of rules of planning is governed by a so-called session principle. A session is started when a user presses the *Edit* button (see Fig. 1), or chooses an activity in the plan for execution. The session ends when the user presses the *Save* button; then all changes made on the screen are introduced in the database. Rules are applied after the *Save* button has been pressed, but before the changes are introduced in the process state stored in the database. Thus, the rules of planning can be considered as a kind of ECA rules, where ECA stands for Event-Condition-Action. In our case, pressing the *Save* button serves as an event trigger.

The process tree is traversed in the top down left to right manner starting from the root. Rules associated with each node are applied in an order defined in the invocation table: a rule can be executed either before traversing the sub-tree attached to the given node, or after traversing.

A rule associate with a node other than *planned activity* can only add new activities to the list, while a rule associated with a *planned activity* node can delete an activity being traversed or change the values of its attributes. Rules of the first kind implement obligations and recommendations, while rules of the second kind implement prohibitions and negative recommendations. Rules of the second kind are applied last.

Rules that cover the examples from the previous section are defined in Table 1. Assignment of these rules to the nodes of the process tree is shown on Fig 3.

Table 1. Rules for examples in Section 3

#	Condition	Action
1	A person is a core participant and there is no activity *Meeting* assigned to him/her in the plan.	**Add** a new activity *Meeting* to the plan. **Assign** it to the person in question. **Make** *Start* and *Finish* of the activity (see Fig. 2) equal to *Meeting date and time*, and *Meeting duration till* from the current process state (see Fig. 1).
a)	**Obligation:** No additional conditions.	
b)	**Recommendation: and** this person appeared on the participant list in the current session, i.e. he/she was not on the list before the session was started.	
2	An activity *Meeting* is assigned to a person who is not on the participant list.	**Delete** the activity.
a)	**Prohibition:** No additional conditions.	
	Negative recommendation: and this person was on the participants list before the current session was started, i.e. he/she disappeared from the list in the current session .	

#	**Condition**	**Action**
3	A *Meeting* activity assigned to a person *X* is in the plan (see Fig. 2), **and** there exist other activities of type *Meeting* that are also assigned to *X*.	**Delete** the activity.
4	A meeting activity has *Start* not equal to *Meeting date and time* **or** *Finish* is not equal to *Meeting duration till* (see Fig. 1 and 2).	**Make** *Start* and *Finish* of the activity equal to *Meeting date and time*, and *Meeting duration till* from the current process state.

Each scenario from Section 3 is governed by its own set of rules, namely:

- Scenario 1 — 1a, 2a, 3, 4
- Scenario 2 — 1a, 2b, 3, 4
- Scenario 3 — 1b, 2a, 3, 4
- Scenario 4 — 1b, 2b, 3, 4

The process tree is traversed in the following manner, see Fig. 3 for illustration. First, the rules attached to the root are invoked, we do not have any in our examples. Then, the rules attached to node *Meeting Participant* (1a or 1b) are invoked for each core participant. And lastly, the rules attached to node *Planned activity* (2a or 2b, 3 and 4) are invoked for each activity on the to-do list.

Note that the conditions for rules 1a (obligation) and 1b (recommendation) are almost identical. The only difference is that in rule 1b, there is an additional condition on the previous state of the process. The same is true in respects to rules 2a, 2b.

As was mentioned in the beginning of this section, currently, each rule is coded manually in a low level programming language. For this end, we use the *ProBis* development environment that permits programming in C and JPL. The latter is a proprietary interpretative language included in the Panther development platform from Prolifics Inc.

5 Conclusion

We started with the task of finding a way of controlling process instance flexibility via a business process support (BPS) system. We narrowed down our task to consider only BPS systems built upon the state-oriented view on business processes. Flexibility control in this case should be incorporated in the rules of planning that are the primary mechanism of process control. Through a set of scenarios, we showed that flexibility of business process instances could be controlled through changing the status of rules of planning. Changing from an obligation to recommendation, or from a prohibition to negative recommendation yields more flexibility, while reversing from a recommendation to obligation/prohibition yields less flexibility.

As we mentioned in the introduction, the literature search gave us no results as far as formalization of rules of planning is concerned, especially for recommendations. Thus, we need to work out our own approach to this task, which is one of our research

directions at present. Though we have not discussed our general approach in this paper, some ideas of it can be derived from the practical implementation discussed in Section 4. In this section, we showed that as far as planning rules are concerned the difference between a recommendation and obligation, and between prohibition and negative recommendation can be defined as an extra condition added to the recommendation, or negative recommendation. This condition adds a temporal aspect to the rule, as it checks whether the main condition was true before the last modification or not. As far as we know, such way of formalizing recommendations is new, and it has not been described in the literature before.

As we succeeded in implementing rules of planning in a real BPS system, suggested approach deserves to be considered as a practically feasible alternative for imposing a predefined level of flexibility into operational practice. Rules of planning discussed in the paper will be included in the production version of ProBis in the nearest future.

Acknowledgements: Writing of this paper was supported by the Swedish Agency for Innovation Systems (Vinnova) under the grant for a project on "Integration Business Process Support with Knowledge Management".

References

1. Bider, I. "Choosing Approach to Business Process Modeling. Practical Perspective". *Journal of Conceptual Modeling*, Issue 34, January 2005. http://www.inconcept.com/jcm/January2005/IBider.html
2. Khomyakov, M., and Bider, I., "Achieving Workflow Flexibility through Taming the Chaos." In OOIS 2000 - 6th international conference on object oriented information systems, pp. 85-92, Springer, 2000.
3. Bider, I., *State-oriented business process modeling: principles, theory and practice*. PhD thesis, KTH (Royal Institute of Technology), Stockholm, 2002.
4. Regev G., Soffer P., Schmidt R. *Taxonomy of Flexibility in Business Processes.* http://lamswww.epfl.ch/conference/bpmds06/taxbpflex
5. Wan-Kadir W.M.N., Loucopoulos P. "Relating evolving business rules to software design". *Journal of Systems Architecture* 50 (2004), pp. 367–382.
6. Andersson T., Bider I., Svensson R., "Alignig people to business processes. Experience report." *Software Process: Improvement and Practice (SPIP)*, V10(4), 2005. pp.403-413.