To be published in: *International Journal of Business Process Integration and Management (IJBPIM)*, VOL 3, 2008

Controlling business process instance flexibility via rules of planning

Ilia Bider, Alexey Striy

IbisSoft, Box 19567, SE-10432 Stockholm, Sweden

[Ilia,alexey]@ibissoft.se

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. Achieving flexibility when workflow management systems are used for support of business processes is widely discussed in the literature. The ways of achieving flexibility for other types of business process support is a much less studied topic. The paper discusses how a given level of flexibility can be imposed by a BPS system built based on the state-flow view on business processes. The flexibility is achieved 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. Additional flexibility is achieved by defining rules on the level of a single process instance. The discussion is illustrated with the help of a simplified example already implemented in a BPS system called ProBis.

Keywords: business process, flexibility, planning, policy, business rule, deontic logic

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, employee 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 (Bider, 2005a), namely: (1) input/output flow, (2) workflow, (3) agent related view, and (4) state-flow. The choice of view depends on the properties of the business processes in question and the task

that a BPS system should serve (Bider 2005a). For example, input/output flows are suitable for business processes that deal with physical objects. The workflow view is suitable when it is important to ensure that operations/activities are completed in a certain order. The agent related view is suitable when there is strict distribution of responsibilities between the agents. The state-flow view is suitable for loosely structured administrative processes, the ones that do not have strict order of activities or distribution of responsibilities between the agents and which consume and produce information rather than physical objects.

As follows from the above deliberation, the choice for a view on which a BPS system is being built is based on a number of factors that are not always directly connected to flexibility. Therefore there is a need to have an approach for dealing with flexibility for each view.

The issue of flexibility for Workflow Management Systems (WfMS) has been widely discussed in the literature in the last 10-15 years, see for example (Reichert & Dadam, 1998, Heinl et al., 1999). The ways of achieving flexibility for other type of business process support, is a much less researched topic. In this paper, we discuss how a desired level of flexibility can be introduced in a system built based upon the state-flow view proposed in (Khomyakov & Bider, 2000, Bider 2002). As the underlying conceptual models for different views differ, 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 without a substantial modification (if at all). Based on the same reasoning, we do not expect that methods suggested for other views on business processes will be easily applicable to the state-flow view. Thus we do not make any attempts to use them while developing our approach.

An approach to flexibility for a state-flow based system substantially differs from the approaches applied for WfMS. In WfMS, flexibility is introduced by moving from the rigid predefined control flow to allowing deviations. This, for example, can be done via introducing alternative patterns of behavior, by permitting the process to deviate on particular paths, etc. In a state-flow system, flexibility is achieved while starting from the other end. We begin with full chaos, and then introduce some means to restrict it. Thus, instead of achieving a given level of flexibility, we can speak about achieving a given level of rigidity (Bider 2005b). This interpretation of flexibility is in line with (Regev et al., 2007), where flexibility is defined through invariants, i.e. through what should not be changed rather than what can be changed.

In a state-flow system, control over process instances is realized via rules of planning. As was suggested in (Khomyakov& Bider 2000), the natural way of introducing flexibility/rigidity in these circumstances is by introducing several kinds of rules. In the work reported in this paper we further develop the suggestions from (Khomyakov& Bider 2000), and prototype them in a working system. We ensure various levels of flexibility through differentiating four types of rules, which is one more than initially suggested in (Khomyakov& Bider, 2000) as it lacks the fourth type of rules:

1. Obligations (must)
2. Recommendations (should)
3. Prohibitions (must not)
4. Negative recommendations (should not)

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.

The rule system mentioned above is defined on the type level, which means that all instances of a given business process type should abide these rules. In addition to the type-level rules of planning, we introduced ad-hoc rules of planning that are defined on the instance level. While type-level rules can be defined only by an expert responsible for a given process type, ad-hoc rules can be introduced by any BPS system user. An ad-hoc rule is defined for a given instance, and it is valid only for this particular instance. An ad-hoc rule represents a kind of generalized alert of the type: “if a particular situation occurs in the frame of the given instance, plan a particular activity in the frame of this instance”. An alert can plan any activity that can be planned manually, or by a type-level rule. The user introducing the alert can specified that this activity will be planned for him, or for some other process participant.

Ad-hoc rules do not increase the flexibility of the process instance for which they are defined. On the contrary, they introduce an additional level of “rigidity” by adding extra control flow not defined by the type-level rules. However, as we stated above, controlling the rigidity is another way of controlling flexibility; thus we can consider ad-hoc rules as a means of achieving process instance flexibility.

When a new theoretical approach to modeling business reality is developed, two issues need to be solved:

1. Whether the approach itself is consistent and can be formalized

2. Whether it can be applied to business practice

Consistency and formality without a possibility to be applied and thus verified can make an approach useless. Ad-hoc applicability without formality and consistency will severely limit the scope of application of the approach. Though the two issues above are interrelated, they can in some degree be pursued in parallel. We need not wait to full formalization before trying to apply an approach to practice. If we cannot find the way to apply it, all our efforts spent on formalization will be in vain, we will only get one more formalism that nobody can use.

The paper presents our findings connected to the second issue. The work on formalization is not finished yet. However, we felt that our understanding was sufficient to try to find ways of applying the approach to practice. We successfully implemented rules of planning in a working system to ensure ourselves of practical applicability of the approach. This work justifies our further efforts on formal theory.

The goal of the paper is twofold. Firstly, we want to present main ideas of how flexibility can be controlled based on the state-oriented perspective. Secondly, we want to show that these ideas can be applied to practice. Consequently, in this paper, we do not present any formal definition of rules of planning. We concentrate on explaining the ideas using a working 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. Actually, the example was chosen due to its simplicity, as it allows explaining the main ideas in a paper of limited size.

The paper is written in a style that could be easily understood by practitioners. The number of references was intentionally kept to the minimum to make the paper easier

to read for the intended audience. Screenshots were used instead of abstract descriptions to give better understanding that the approach discussed can be used in practice.

The paper is written according to the following plan. In section 2, we briefly review the main ideas of the state-flow view on business processes. In section 3, we suggest a classification of the type-level rules of planning that allows flexible control over the process instances. In section 4, we demonstrate how the type-level rules work on a simplified example. In section 5, we shortly review current implementation of type-level rules of planning in a BPS system. Section 6 reviews ad-hoc rules and their current implementation in a BPS system. Section 7 is devoted to the related research. Section 8 contains concluding remarks and plans for the future.

2 State-flow view on business processes

The main concept of the state-flow view on business processes is the process's *state* (Khomyakov & Bider, 2000). The process's state is aimed to show how much has been done to achieve the operational goal of a process instance, and how much is still to be done.

A *state* of a business process is defined by a construct that reflects the relevant part of the business world at a given moment of time. The internal structure of the state construct depends on the business process type to which the current instance belongs. An example, of such structure for a business process related to a customer order is represented in Fig. 1, which is a screenshot from a state-oriented business process support system.

The state structure in Fig.1 includes: (a) attributes (variables), such as *To pay*, *Paid, Ordered*, etc., and (b) references to various human and non-human participants of the process, like *Customer*, *Products*, etc. The names of the attributes above correspond to the labels in the screenshot from Fig. 1; the positions of some of these attributes are additionally pinpointed by arrows. Note that the state structure may include repeating groups, like *Products* in Fig. 1, which means that some attributes, like *Ordered* and *Delivered* will have multiple values, one for each item of the group.

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 (end of the process instance trajectory in the state space). A state that satisfies these conditions is called a *final state* of the process. The set of final states for the process in Fig. 1 can be defined as follows: (a) for each ordered product *Ordered* = *Delivered*; (b) *To pay* = *Total* + *Freight* + *Tax*; (c) *Invoiced* = *To pay*; (d) *Paid* = *Invoiced*. These conditions define a surface in the state space of this process type.

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.

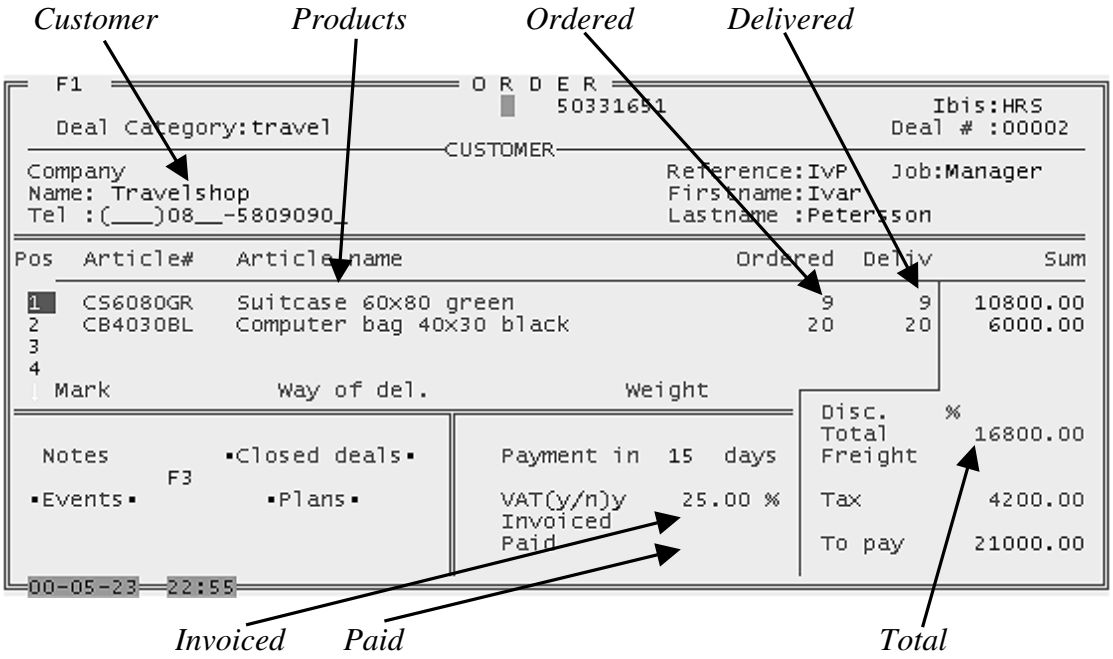

Figure1. State representation of order processing.

All activities currently planned for a process instance make up its *operational plan* or to-do list, see Fig. 2 for an example. The plan lists activities the execution of which diminishes the *distance* between the current state of the process instance and the *projected* final state. The meaning of the term distance depends on the business process in question. Here, we use this term informally. For example, activities to plan for the process in Fig.1 can be defined in the following manner:

- If for some product *Ordered* > *Delivered*, *shipment* should be performed, or
- If *To pay* > *Invoiced*, an *invoice* should be sent, etc

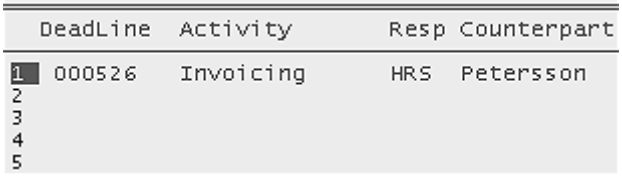

Figure 2. Process's plan that complements the state from Fig. 1

The plan together with the "passive" state (attributes and references) constitutes a *generalized state* of the process, the plan being an "active" part (engine) 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 a 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 state towards the goal. A business process type can be defined as a set of valid generalized states. A business process instance is considered as belonging to this type if for any given moment of time its generalized state belongs to this set. This definition can be converted into an operational procedure called *rules of planning*. The rules specify what activities

could/should be added to/deleted from 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; as a result, some actions may be added to the plan and some be removed from it.

3 Automated generation/modification of plan with type-level rules

This section is devoted to classifying rules of planning and discussing how rules from different categories can be used for achieving flexible control of business process instances. The basic ideas for our classification were taken from the works in three areas: deontic logic (R.Hilpinen, 1971), policy-based thinking (ISO/IEC 1995, Lupu & Sloman, 1997) and theoretical findings in the domain of Business Rules (BR) (Wan-Kadir & Loucopoulos, 2004). Our classification does not follow exactly any of the theories listed above, as it is adjusted to our specific goal of controlling business processes via rules of planning. We differentiate 4 categories of rules of planning:

1. *Obligation:* for a given class of the process states, some activities **must** be present in the process's plan. In case of absence, they are added. For example, suppose in an order processing process, all goods have been delivered, but not all money has been invoiced. Then the *invoice* activity must be included in the plan.
2. *Recommendation*: for a given class of the process states, some activities **are, normally, present** in the process's plan. In case of absence, they are suggested for inclusion.
3. *Prohibition*: for a given class of the process states, some activities **cannot** be present in the process's plan. In case of presence, they are removed. For example, if all goods have been shipped, no shipping activity can be present in the plan.
4. *Negative Recommendation*: for a given class of the process states, some activities **are, normally, not present** in the process's plan. In case of presence, they are suggested for removal.

Only prohibitions and recommendations facilitate rigid control of business processes. A BPS user can manually plan anything that is not required but not prohibited. It may give the user too much flexibility, which may be more than a normal user is willing to cope with (Bider, 2005b). Recommendations and negative recommendations are aimed at "limiting" this flexibility. This limitation is not in the sense that the user will be forced to do something, as he/she is free not to follow recommendations. The limitation is in the sense that the user will need to break recommendations explicitly by changing an automatically generated plan. As not following recommendations requires additional actions, there, normally, will be some business reason for completing these actions.

The wording in the names of categories above implies that the rules can be used for communication between the system and its users, for example, by *recommending* some actions, or *not recommending*, i.e. warning against some others. While the rules can certainly be used in this way, we do not discuss this issue in the paper. We focus on *automated generation/modification* of the plan, and *automated correction* of the manual planning done by the user. While discussing automated generation/modification of the plan we differentiate 3 situations:

1. The process plan is empty, for example at the start of the given process instance, or after the last of the previously planned activities has been executed.
2. The process plan is not empty, but the current state of the process instance has been changed.
3. The user has manually changed the plan.

Note that situations 1 and 3, and 2 and 3 can be mixed. For example, the plan can become empty because the user manually deleted all activities from it, or when in a situation 2, the user manually corrected the plan directly in connection to registering changes in the process state.

In each situation listed above, applicability of a particular rule of planning depends on to which category this rule belongs. The following scheme is applied:

1. The process plan is empty. Obligations and recommendations are treated equally to suggest a plan.
2. The process plan is not empty, but the current state has been changed, i.e. the user registered changes that just happened in the real world, or has executed a planned activity that changed the state. Then, as in the first situation, obligations and recommendations are treated equally to suggest new activities to be added. In addition, prohibitions and negative recommendations are treated equally to remove activities that no longer correspond to the new state of the given process instance.
3. The user manually changes the plan. Only obligations and prohibitions are used to correct the plan. Obligation will restore mandatory activities, while prohibitions will remove activities that are not allowed in the given state. Neither recommendations, nor negative recommendations are used in this case. The latter means that recommended activities can disappear from the plan, and non-recommended activities can appear in it.
4. When a mixed situation has occurred, rules are used dependent on whether the given activity was added/deleted manually, or was in the plan before the last change of the state, see the next section for examples.

As we see from the discussion above, when the system does planning on its own it treats obligations and recommendations equally. The same is true for prohibition and negative recommendations. However, when the system corrects manual planning performed by the user, it interprets these categories in a different manner.

4 Examples of type-level rules of planning

The main ideas of our approach are demonstrated and explained in an example of a process of organizing a meeting. The state of such process can be represented as a screen capture in Fig. 3. Each meeting has a number of so-called core participants (see "*Core participants*" in Fig. 3), *Meeting date and time* and *Place*. Fig. 4 represents the list of activities (called tasks in the figure) currently planned for the meeting process from Fig. 3. 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. 4 represents the "normal" correspondence between the state and the plan. Below we consider several scenarios that ensure that such correspondence is

imposed strictly or with some deviations. The scenarios are ordered in a way to exemplify increasing flexibility.

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

1. *Obligation*: If a person belongs to the core participants, there should be an activity "Meeting" assigned to him/her in the plan.
2. *Obligation*: Date and time of a *Meeting* activity must be the same as prescribed by the process state.
3. *Prohibition*: A *Meeting* activity is allowed to be in the plan only if it is assigned to a core participant.
4. *Prohibition*: Only one *Meeting* activity per person is allowed in the plan. Note that this prohibition concerns only the plan of the given *Meeting* process. There can be several *Meeting* processes running at the same time, which means that any person can have several meeting activities in his calendar, each belonging to a different *Meeting* process.

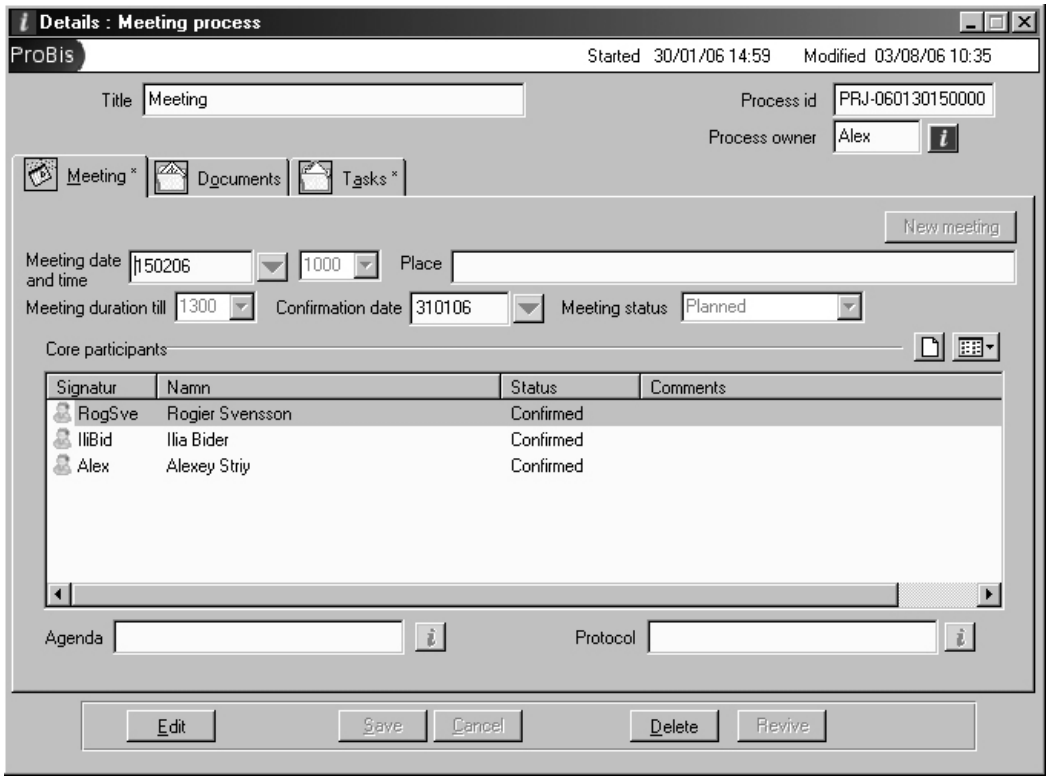

Figure 3. State of the meeting process

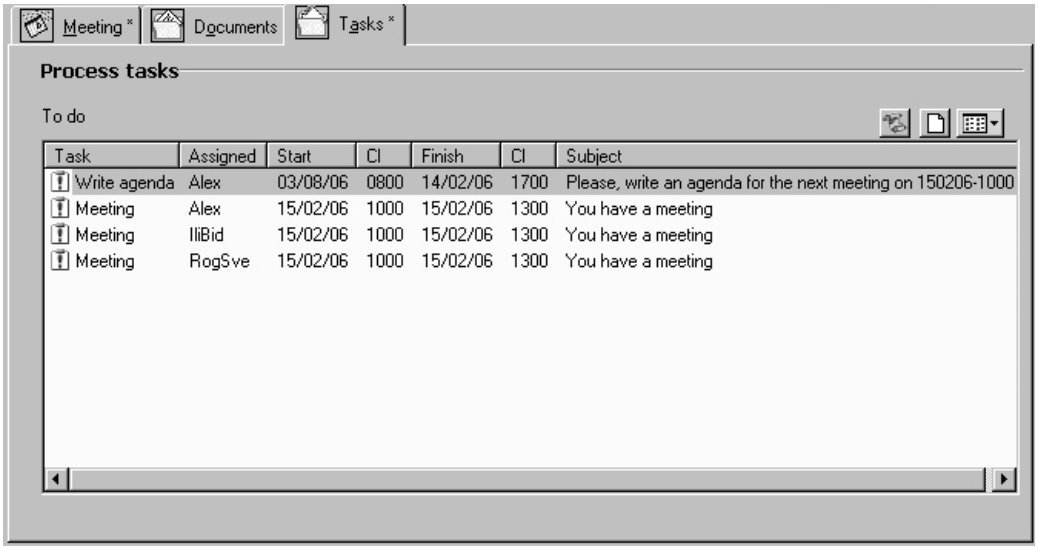

Figure 4. Activities planned in the frame of the process instance

The rules are applied in the following manner:

a) If a person is added to the participants list, a new *Meeting* activity assigned to him/her is automatically added to the plan.

b) If this activity is later manually removed, it will automatically be added to the plan again.

c) If a person is removed from the participants list, his/her *Meeting* activity is also removed.

d) 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 automatically be removed.

e) In addition, date and time parameters are always corrected to the actual date and time from the process state.

f) Multiple *Meeting* assignments to the same person in the same *Meeting* process are reduced to one.

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 made more sophisticated, so that a user is not allowed to delete a mandatory activity (task), however, we will not discuss this issue in more details in this paper.

Scenario 2 – Guests allowed. Core participants must attend but guests are allowed. This scenario can be obtained from Scenario 1 by changing the status of Rule 3 from *Prohibition* to *Negative recommendation*:

3. *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.

The rules application would differ from scenario 1 only in situation (d):

d) 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. As was discussed in Section 3, the system will be "silent" in this case, assuming that the user who has chosen to manually plan a *Meeting* activity (without adding a person to the participant list) knows what he is doing.

Scenario 3 – Permission to skip, but no guests. Core participants are recommended to attend, and only they are allowed to attend. This scenario can be obtained from Scenario 1 by changing the status of Rule 1 from *Obligation* to *Recommendation*:

1. *Recommendation*. If a person belongs to the core participants, it is recommended to have an activity *Meeting* assigned to him/her in the plan.

The rules application would differ from scenario 1 only in situation (b):

b) If this activity is later manually deleted, it will not appear again after the rules have been applied.

Scenario 4 – Both guests allowed and permission to skip. Core participants are recommended to attend, and guests are allowed. This scenario can be obtained from Scenario 1 by changing the status of Rule 1 from *Obligation* to *Recommendation*, and the status of Rule 3 from *Prohibition* to *Negative recommendation*:

1. *Recommendation*. If a person belongs to the core participants, it is recommended to have an activity *Meeting* assigned to him/her in the plan.

3. *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.

The rules application would differ from scenario 1 only in situations (b) and (d):

b) If this activity is later manually removed, it will not appear again after the rules have been applied.

d) 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.

As we can see from the above scenarios, flexibility can be achieved by substituting an obligation to recommendation, or prohibition to negative recommendation.

5 Implementation of type-level rules in ProBis

The approach of controlling flexibility via type-level rules of planning has been implemented in a BPS system called *ProBis* (Andersson et al., 2005). 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 and activated/deactivated via a special system administrator screen.

In *ProBis*, a process state is represented as a tree structure; the tree relevant to the example from the previous section is shown in Fig. 5. 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.

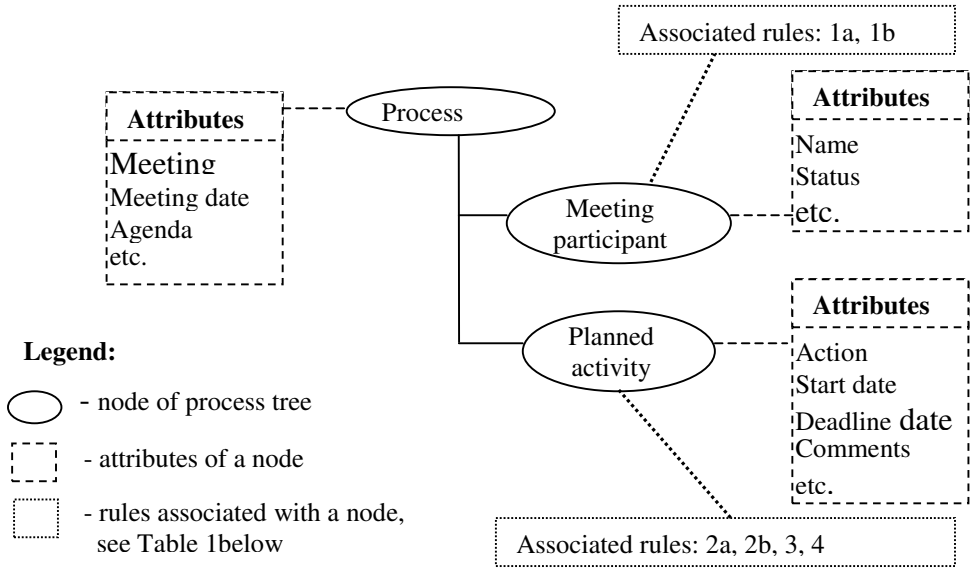

Figure 5. Process tree structure

Each individual rule in the invocation table is associated with a node in the process tree. A rule may be supplied with a pre-condition on attribute values of the node with which it is associated, or the upper nodes. A pre-condition can take into account current values of attributes as well as information about changes. If a pre-condition is supplied, the rule is applied only if the pre-condition yields true for a given process instance. The rule itself is implemented as as procedure written in a programming language for which *ProBis* has an API. This procedure may include additional conditions that should be met before any changes in the plan can be introduced by this rule. If the conditions are not met, the rule invocation does not affect the plan.

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. 3), 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. Our 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 associated 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 associated with planned activities are always 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 5.

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. 4) equal to *Meeting date and time*, and *Meeting duration till* from the current process state (see Fig. 3).
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.	
b)	**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.	
3	A *Meeting* activity assigned to a person *X* is in the plan (see Fig. 4), **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* not equal to *Meeting duration till* (see Fig. 3 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. 5 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 respect 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.

6. Implementation of ad-hoc rules

We differentiate two types of ad-hoc rules of planning:

1. Rule with positive condition: plan a given activity as soon as the generalized state of the given process instance changes in a specified way.
2. Rule with negative condition: plan a given activity if the generalized state of the given process instance does not change in a specified way before the given date and time.

Conceptually, the ad-hoc rules are very simple; all that is needed is a condition on the change and an activity (task) to plan. The main challenge here is not in complexity of the idea, but in how to make it understandable for an average system user, i.e. a person who is not accustomed to thinking in terms of logical formulas. To meet the challenge, the conditions should be made simple, and a good enough user-interface should be provided so that the user can understand how to create an ad-hoc rule. As usual in system development practice, good enough interface can only be obtained through an iterative process of making suggestions and discussing them with the end-users until the users accept the final suggestion.

To simplify the conditions we decided to permit only one element of a process tree to be included in the condition definition. More exactly an ad-hoc rule can be attached to:

1. an attribute of a process itself, e.g., *Owner*, *Title*, *Meeting Status* (see Fig. 3)
2. a group of items in a process tree, e.g. *Core participants* (see Fig. 3), or *Planned activities* (*To do* list on Fig. 4)
3. an item in a group, e.g., a particular participant, or a particular task
4. an attribute that belongs to a particular item of a group, e.g., *Status* of a particular core participant.

If the element to which an ad-hoc rule is being attached is an attribute, the condition of the rule can be expressed in terms of changes in the value of the attribute:

- has been changed
- has become NULL
- has become not NULL

- has become equal or not equal to a certain value

If the element to which an ad-hoc rule is being attached is a group, the condition is expressed in terms of the content of the group:

- a new item has been added
- some item has been removed
- some item has been changed

If an ad-hoc rule is being attached to a particular member of a group, the condition is expressed in terms of what can happen to this member:

- member has been removed
- member has been changed

For a member of the *Planned activities* group (*To do* list on Fig. 4) it is possible to specifying an additional condition “has been executed”.

Not all elements of the process tree are suitable for defining ad-hoc rules. The permission to attach ad-hoc rules is controlled by a special table that can be accessed only by a system administrator.

The user attaches an ad-hoc rule in two steps. First, he/she chooses an element of the process tree to attach a new rule, and then he/she defines a condition and what should be planned when it happens/does not happens. The choice of an element to which to attach a rule is made through a screen which reflects the tree structure of the given process instance, see Fig 6. A checkbox before an element shows that there already exists an ad-hoc rule attached to this element.

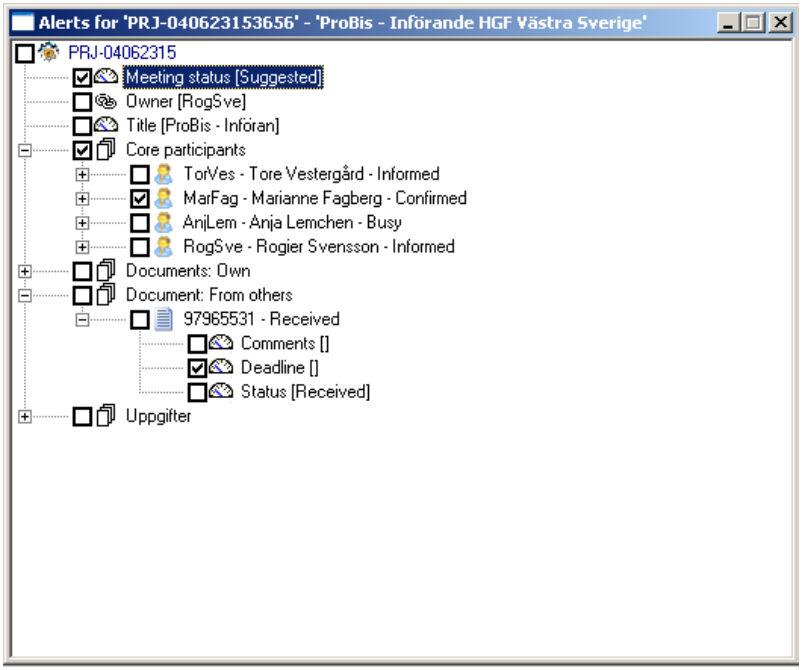

Figure 6. Tree representation of a process instance for ad-hoc rules

To add a new ad-hoc rule or to see and change an existing rule, the user double clicks on the element he is interested in and gets a rule specification screens shown in Fig. 7, 8. The upper part of this screen is formed differently dependent on what kind of a process element the rule is being attached to. In Fig. 7, the rule is being attached to the

group *Core participants*. In Fig. 8, the rule is being attached to attribute *Meeting status*. The user chooses the condition he wishes and then defines the activity to plan when this condition is satisfied (Fig. 7), or not satisfied by a certain date and time (Fig. 8). By default, *Attention* activity is planned to the person specifying the rule. The rule can be defined as a one-time rule, or as a permanent rule, i.e. it will work each time the specified condition happens. The permanent rule is defined by checking the box *Do it each time the situation occurs*. Only a rule with positive condition can be defined as permanent (Fig. 7). If the negative condition is chosen, the checkbox *Do it each time the situation occurs* disappears (Fig. 8).

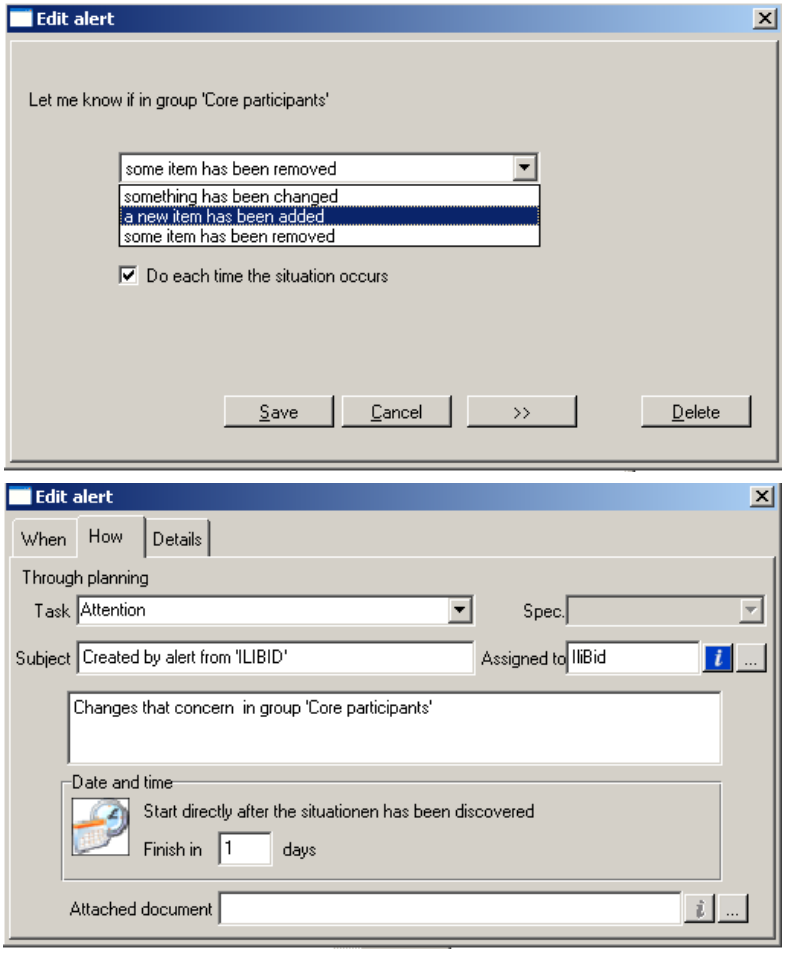

Figure 7. Ad-hoc rule specification screen

The ad-hoc rules are applied in two situations. When a process state has been changed, all ad-hoc rules specified for this process instance are checked. This happens immediately after application of the type-level rules. If the change in the process state satisfies the condition, the action is undertaken. If the rule has a positive condition (e.g., as on Fig. 7) then an activity (task) specified in the rule is added to the process plan. After that the rule is deleted, unless it was defined as permanent. If the rule has a negative condition (e.g., as on Fig. 8), then this rule is deleted without anything being added to the plan.

To invoke the overdue rules with negative conditions, a special server has been created. It periodically checks the *before* date and time of the ad-hoc rules with negative conditions. If a rule is overdue, the specified activity is added to the process plan and this rule is deleted.

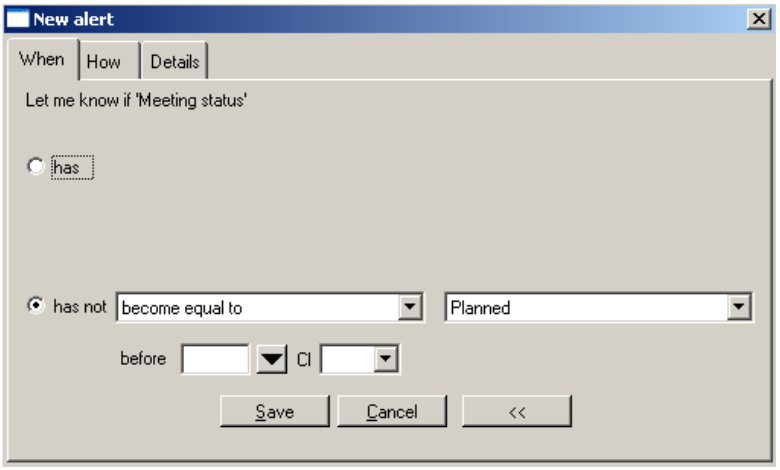

Figure 8. Another example of the rule specification screen

7. Related research

As far as our goal of controlling business process instance flexibility is concerned, we share it with many other researchers working in this area. However, as we already pointed out in the introduction, the ways of achieving such control depend on the principles on which a BPS system is built. Most of the work in this direction is being done in the frame of the workflow view on business processes, see for example, (Reichert & Dadam, 1998, Heinl et al., 1999). As we investigate BPS systems based on a completely different view, i.e. state-flow, our methods of achieving flexible control over business process instances differ from those that are proposed for the systems based on the workflow view. Comparing approaches based on completely different principles is a complicated task, and it was outside the scope of the research reported in this paper.

As far as the idea of using business rules for defining business processes is concerned, this idea is not new, see for example, (Knolmayer et al. 2000). Even in this area, the research is conducted based on the workflow view on business processes, which means that results of this research are not easily applicable to the state-flow view.

Our approach to controlling flexibility of business processes is related to a number of areas outside the field Business Process Management. The basic concepts of obligation and prohibition come from deontic logic (R.Hilpinen, 1971). Its application to our rules of planning was inspired by policy-based thinking (Lupu & Sloman, 1997). The main difference of our approach from (Lupu & Sloman, 1997) is that:

- The deontic concepts are integrated in the framework of the state-flow view on business processes.
- We do not use the concept of permission, everything that is not prohibited is permitted. Instead, we use recommendations and negative recommendations to show various degrees of permissibility.

Another research area related to our work is Business Rules (BR). The literature on BR and its application to software design is vast, see, for example: (Wan-Kadir & Loucopoulos, 2004) and its list of references. BR literature also categorizes rules according to the degree of obligation with which they should be followed. For example, (Wan-Kadir & Loucopoulos, 2004) differentiates two types of constraint rules: *Mandatory Constraint* and *Guideline*. Business events that violate the *Mandatory Constraint* should be rejected, whilst violation of *Guideline* rules should raise a warning to the user. From the BR view, our rules of planning represent a special kind of BRs aimed at providing help in running business processes.

One more area related to our work is Expert Systems (ES) (Silverman, 1992). Initially, the aim of ES was to help humans in finding solutions in cases where a large knowledge base was needed for finding a solution. But later, a new field of ES, called Critiquing Systems (CS) emerged (Silverman, 1992). A critiquing system analyses solutions chosen by humans, warning them when a solution might be wrong, and explaining why. From the ES perspective, our work consists of integrating both traditional ES (obligations and recommendations) and CS (prohibitions and negative recommendations) capabilities into a business process support system.

8 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-flow view on business processes. Flexibility control in this case should be incorporated in the rules of planning that are a primary mechanism of process control. Through a set of scenarios in an example, 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. We also showed how through ad-hoc rules an additional control could be established over a particular process instance. That enhances the control established by the type-level rule of planning, giving possibility to increase rigidity of a given process instance.

As was explained in the introduction, the research presented in this paper was mainly directed at proving that our approach to controlling flexibility could be implemented in practice. We were not concerned with defining details of a possible formalism, nor with formal methods of discovering inconsistency between different rules. We believe that the approach can be applied to practice without waiting solutions for these issues. Rules can be hard-coded, and inconsistencies can be detected through testing in the same way as it is still done in programming.

As we succeeded in implementing rules of planning in a real BPS system, the suggested approach deserves to be considered as a possible alternative for imposing a predefined level of flexibility in operational practice. However, a lot more is required to make our findings practically feasible. Work in this direction includes fine classification of rules of planning in order to propose a formal language for writing the rules instead of manual coding them in a programming language.

Though our approach to rule definition is specific to the state-flow view on business processes, some details of it can be of general interest for the reader. Suggested

approach, when fully developed, represents an operational interpretation of the main concept of deontic logic. Through our rules of planning, we can formally express the difference between obligations and recommendations, as well as prohibitions and negative recommendations. In this formalism, obligations, recommendations, prohibitions and negative recommendations are defined uniformly without introduction of any special (deontic or modal) operators. All types of rules are expressed through logical formulas on valid transitions from one generalized state to another.

Though in our work we concentrated on automatic generation/correction of the plan, we do not reject the needs of having explanatory mechanism that will clarify to the user why certain activities have been added/removed, or are not appropriate for the current state of the process. This can constitute a new direction in our future research.

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". The authors are also grateful to the anonymous reviewers whose comments helped to improve the text.

References

(Andersson et al, 2005) Andersson T., Bider I., Svensson R., "Aligning people to business processes. Experience report." *Software Process: Improvement and Practice (SPIP)*, V10(4), pp.403-413, 2005

(Bider, 2002) Bider, I., *State-oriented business process modeling: principles, theory and practice*. PhD thesis, KTH (Royal Institute of Technology), Stockholm, 2002.

(Bider, 2005a) 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

(Bider 2005b) Bider I. Masking flexibility behind rigidity. Notes on how much flexibility people are willing to cope with. In Proceedings of the CaiSE'05 workshops, Vol. 1, FEUP, Porto, Portugal, pp. 7-8, 2005.

(Heinl, etal., 1999) Heinl, P., Horn, S., Jablonski, S., Neeb, J., Stein, K. and Teschke, M., A Comprehensive Approach to Flexibility in Workflow Management Systems, Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration, 1999, San Francisco, California, USA, February 22-25, 1999, pp79-88, ACM 1999,

(Hilpinen, 1971) R.Hilpinen (ed) *Deontic Logic: Introductory and Systematic Readings*. Reidel Publishing Company, 1971

(ISO/IEC, 1995) ISO/IEC 10746-2. Open Distributed Processing – Reference Model – Part 2: Foundations. 1995.

(Knolmayer et al. 2000) Knolmayer, G., Endl, R. and Pfahrer, M. Modeling Processes and Workflows by Business Rules. *Business Process Management, LNCS* 1806, pp. 16–29, Springer 2000.

(Khomyakov & Bider, 2000) 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.

(Lupu & Sloman, 1997) Lupu E., and Sloman M. *A Policy Based Role Object Model*. In the Proceedings of EDOC'97, pp. 36-47. Gold Cost, Australia, October 1997.

(Reichert & Dadam, 1998) Reichert, M., Dadam, P.: ADEPTflex Supporting Dynamic Changes of Workflows Without loosing Control. Journal of Intelligent Information Systems 10(2): 93-129,1998.

(Regev et al., 2007) Regev G., Bider I., Wegmann A. Defining Business Process Flexibility with the help of Invariants. Software Process: Improvement and Practice (SPIP), Wiley, Vol. V12, No.1, pp. 65-79.

(Silverman,1992) Silverman, B. G.: Survey of Expert Critiquing Systems: Practical and Theoretical Frontiers, Communications of the ACM 35: 106–127, 1992.

(Wan-Kadir & Loucopoulos, 2004) Wan-Kadir W.M.N., Loucopoulos P. "Relating evolving business rules to software design". *Journal of Systems Architecture* 50 (2004), pp. 367–382.