Complex Systems Informatics and Modeling Quarterly
CSIMQ, Article 78, Issue 13, December 2017/January 2018, Pages 73–113
Published online by RTU Press, https://csimq-journals.rtu.lv
https://doi.org/10.7250/csimq.2017-13.05
ISSN: 2255-9922 online

Towards a Business Process Modeling Technique for Agile Development of Case Management Systems

Ilia Bider[*] and Erik Perjons

DSV, Stockholm University, Postbox 7003, 164 07 Kista, Sweden

ilia@dsv.su.se, perjons@dsv.su.se

Abstract. A modern organization needs to adapt its behavior to changes in the business environment by changing its Business Processes (BP) and corresponding Business Process Support (BPS) systems. One way of achieving such adaptability is via separation of the system code from the process description/model by applying the concept of executable process models. Furthermore, to ease introduction of changes, such process model should separate different perspectives, for example, control-flow, human resources, and data perspectives, from each other. In addition, for developing a completely new process, it should be possible to start with a reduced process model to get a BPS system quickly running, and then continue to develop it in an agile manner. This article consists of two parts, the first sets requirements on modeling techniques that could be used in the tools that supports agile development of BPs and BPS systems. The second part suggests a business process modeling technique that allows to start modeling with the data/information perspective which would be appropriate for processes supported by Case or Adaptive Case Management (CM/ACM) systems. In a model produced by this technique, called data-centric business process model, a process instance/case is defined as sequence of states in a specially designed instance database, while the process model is defined as a set of rules that set restrictions on allowed states and transitions between them. The article details the background for the project of developing the data-centric process modeling technique, presents the outline of the structure of the model, and gives formal definitions for a substantial part of the model.
Keywords: Business process, process modeling, data-centric, workflow, agile development, adaptive case management, ACM.

1 Introduction

The concept of business process (BP) covers a complex phenomenon that can be considered from several perspectives, e.g.: control flow (also called tasks/activities perspective or workflow) perspective, data/information perspective, resources perspective, including human resources, to

[*] Corresponding author

Reference: I. Bider and E. Perjons, “Towards a Business Process Modeling Technique for Agile Development of Case Management Systems,” *Complex Systems Informatics and Modeling Quarterly*, CSIMQ, no. 13, pp. 73–113, 2017. Available: https://doi.org/10.7250/csimq.2017-13.05

Additional information. Author's ORCID iD: I. Bider – orcid.org/0000-0002-3490-6092 and E. Perjons – orcid.org/0000-0001-9044-5836. Article PII S225599221700078X. Article received: 2017 October 7. Accepted: 2017 December 20. Available online: 2017 December 29.

name the three most important perspectives. A full-featured Business Process Support (BPS) system[*] has tp to handle most, if not all, of these perspectives to provide maximum help to business process participants when they run process instances/cases. There are two approaches to building a BPS, traditional and agile [1]. In the traditional approach a detailed business process model is built before starting the development of BPS, and such model is supposed to cover the majority of BP perspectives. In the agile approach, a minimum possible BPS system is created and set into operation before adding details. In this case, a business process model on which a minimum BPS system is built does not require to cover all perspectives of the business process that it aims at supporting. For instance, a system built using only control flow perspective can be useful for scheduling tasks, while adding the human resources perspective may enhance the support to forwarding information on the tasks to be completed by appropriate human participants of the process instance.

Which perspective(s) should be covered in the minimum BPS depends on the nature of the business process to be supported. If the process is workflowable [2], the control flow perspective can be appropriate for implementation in the minimal BPS. However, for non-workflowable business processes, perspectives other than control flow can be more important for using in the minimum BPS system. The most known classes of such processes are the ones that are supported by Case Management (CM) systems, sometimes called case handling systems [3], and Adaptive Case Management (ACM) systems aimed at supporting so-called knowledge workers in processes that are inherently dynamic and unpredictable [4]. ACM systems can be considered as a special kind of CM systems, see Section 2.1 for more detailed discussion.

In CM, and especially ACM systems, the sequence of tasks/activities (i.e. the control flow), quite often, is impossible to predict beforehand. The most important functionality of CM/ACM systems is providing means for storing structured information and making it available for the process participants who need it. Based on this information, the participants can themselves decide on which tasks/activities to complete and in which order, at least, when a minimum BPS system is employed. Such functionality, for instance, can be implemented via a built-in structured shared space, as in [5]. Accordingly, for CM and ACM systems, the data/information perspective could be more appropriate to be included in the minimum system when agile process development is used.

Though ideas and systems based on ACM are expanding in practice, research in this area only recently started to investigate them, for instance, through the AdaptiveCM workshop [6]. Due to CM/ACM only currently got much attention, so far, there is no widely accepted business process modeling technique that would be suitable for executable business process models employed in CM and ACM systems. OMG has suggested a modeling technique for CM [7], but the focus of this technique is still on tasks, while the data/information plays a less central role. This article is an attempt to suggest a formal foundation for creating a process modeling technique in which the data/information perspective is in the focus, and which could be suitable for employment in CM/ACM systems developed in the agile manner.

When coping with the task of devising a modeling technique for CM/ACM systems, we follow a path similar to the one through which the workflow modeling technique went. We start with devising a definition of a process model with the minimum set of concepts that, nevertheless, can be useful for building CM/ACM systems. Then, we gradually extend this definition to include more concepts thus facilitating the extended functionality of BPS systems. We base our design on the hypothesis that the data/information perspective is the best candidate for the starting perspective when building executable models for CM/ACM systems in the agile manner. The main operations for CM/ACM systems are considered to be storing and retrieving pieces of

[*] Term *BPS system* refers to a software system aimed at supporting people to run/execute business process instances/cases. The level of support can differ, some systems just facilitate storing and retrieving relevant information, others– issue reminders and automate some tasks/operations. The terms stresses that the system is focused on supporting a process as the whole, rather than individual functions.

data/information related to the given process instance, or case in the terminology of CM/ACM. Therefore, a process instance/case can be represented as a trace of changes in the data/information related to this instance/case, while a process model can be define as a set of rules identifying allowed traces[*]. The process model based on this idea substantially differs from the workflow one. It is not based on the notion of task/activity as a basic notion, as there are no specific tasks/activities defined for a given process, except a few general operations related to managing data.

The basis for the modeling technique we suggest is threefold:

1. Reflection made on the so-called form-based case management typical for the public sector practice (at least in Sweden). The management of such processes is not done by specifying activities, but by producing a package of templates/forms mandatory to be used during the handling of cases (see Section 4 for a detailed discussion). This practice was introduced long before the computers were installed in each public office, and it is still applicable when computer based BPS systems are being introduced in practice.
2. Our own experience of developing CM/ACM systems and tools to support the development of CM/ACM systems based on the idea of form-based case management.
3. Database theory and practice.

Based on the three sources above, we suggest a process modeling technique that we call data-centric. In this technique, we differentiate two types of process models: a basic data-centric process model and an extended one. In the basic data-centric business process model, all data related to the process are treated uniformly, independently of what these data represent: sub-goals that need to be reached; actors that are engaged in the process; tasks/activities that are planned or completed; or communication between various agents during the process instance life-time. The proposed technique considers that each process instance of the given process type has an instance database that is being filled with data when the instance develops in time. The databases of instances that belong to the same process type have the same scheme. Formally, the process instance is defined as a sequence of database states ending with some final state. A process model is defined as a set of rules that differentiate valid (allowed) sequences of database states from the invalid ones. These rules can also be defined as a procedure that generates all possible valid sequences, similar to the generative grammars that generate all possible language sentences.

In the extended data-centric model, the notion of task/activity appears and helps to introduce rules that further limit the set of allowed traces. The extended data-centric model will semantically interpret some data items, for instance, as a goal, task/activity, agent, plan, etc., in order to enrich the functionality of a BPS system. While enriching the data-centric model with finer grade semantics is in our plans, the focus of the current article is on designing a “pure” data-centric model and show that it may be useful for BPS systems development.

The specific goal of this article is to suggest a formal definition of the data-centric process model that could be useful for developing CM/ACM systems in the agile manner. As with many other formal definitions, it is too mathematical to be used directly when modeling business processes in practice. A more visual notation has to be developed, which is outside the scope of this article. The formal definition suggested in this article is aimed at guiding the development of possible visual notations, as well as developing internal data structure and model interpreters of the future BPS system of the CM/ACM type.

The rest of the article is structured according to the following plan. In Section 2, we explain in more detail the needs for a new business process modeling technique. Firstly, we give a short overview of the state of the art in CM/ACM field (Section 2.1), and explain the main ideas of

[*] Our way of defining data-centric process modeling substantially differs from other suggestion for non-workflow process modeling, e.g. as artifact-based process modeling [22], data-driven process modeling [23], declarative process modeling [20], or state-oriented process modeling [24]. These works will be considered in Section 3.

agile business process development (Section 2.2). Then, in Section 2.3, we present the reasoning behind the design of our data-centric business process modeling technique. In Section 3, we give an overview of the literature related to non-workflow business process modeling techniques. In Section 4, we present the background on which our modeling technique has been built. This section is extended in Appendix A, which presents a description of a tool that motivated current work. Section 5 presents an outline of our basic data-centric business process model. The substantial part of this outline is formalized in Appendix B. Section 6 discusses how the basic data-centric model can be extended to include the new concepts, like activity or task, via assigning semantics to the data elements. Section 7 analyzes a data-centric modeling technique implemented in an existing tool. More exactly, the analysis concerns which parts of the data-centric model are implemented in the tool, and which are missing or implemented in a way that requires improvement. Section 8 discusses methodological issues, summarizes practical and scientific contribution of the article, and draws plans for the future.

This article uses an illustrative example from our experience report presented at BPMDS 2013 conference [8]. The report concerned a project of building a data-centric process model of a course preparation process, where the goal of modeling was discovering requirements on a BPS system to support this process. Most of the report [8] is available in Appendix A of this article; the parts of the report that concern the project completion and lessons learned are left outside of this article. We refer the readers interested in the details of the project to the original BPMDS 2013 paper.

2 Motivation for Developing a New Business Process Modeling Technique

2.1 Case and Adaptive Case Management

Many researchers have emphasized that workflow based BPS systems have problems in managing change and supporting unpredictable processes [3], [4], [9]. As a result, case management (CM) systems have been introduced as a solution to be used in the environment where workflow based BPS systems are too restrictive [3], [10]. In [3], the differences between workflow based BPS and CM systems are summarized as the difference in focus and the difference in driving force.

The difference in focus means that actors using workflow based systems focus on individual tasks to be carried out in a case (or process instance), while actors using a CM system focus on the whole case and not only on the individual tasks. An actor of a workflow based system executes work items in his/her in-tray without considering the context of the case, which may be harmful for the case as the whole. An actor of a CM system views data/information related to the whole case when carrying out his/her work tasks and thus can take more informed decisions. The difference in driving force means that while workflow based systems are process driven with the focus on executing the control-flow (i.e. completing tasks/activities in a predefined order), CM systems are both process and data driven. It is the case related data/information and the knowledge workers experience that determine which tasks to carry out, and in which order, when a CM system is in use. Therefore, when using such a system, the process execution is decided predominantly at run-time, and not at design-time, which is the standard for traditional workflow based systems.

The next generation of CM systems is called Adaptive Case Management (ACM) systems. There is no commonly accepted definition of an ACM system, and thereby, the difference between CM and ACM is blurred. One interpretation is that ACM systems support situations that are inherently dynamic and unpredictable [4]. An example of such a situation could be a physician receiving a patient in an emergency room. While unsure of the real causes to the health issue, the physician still needs to carry out some treatments. Therefore, the results of decided treatments have to be assessed swiftly and continuously in order to better understand the root causes. As a result of such continuous assessments, new plans of treatments may need to be decided frequently.

From the practical perspective, an ACM system is defined as a BPS system that supports so-called knowledge workers in knowledge-intensive processes [4]. The system provides the knowledge worker with the required data/information at the right time, and helps the knowledge worker to handle frequent exceptions that occur in ACM processes by supporting continuous re-planning. The knowledge worker should also be able to make changes in the case without possessing any programming and modeling skills.

Based on the analysis of the ACM cases in practically oriented books, like [4], it is reasonable to identify processes that are suitable to be supported by ACM systems based on two characteristics of the process[*]: (a) the level of uncertainty typical for the instances/cases of this process, and (b) the level of volatility of the surrounding environment.

The level of uncertainty concerns the lack of data/information that exists for making decisions in the beginning and during a typical instance/case of the process. Having high-level of uncertainty means that the actions taken in the case are decided based on some hypotheses, which need to be confirmed based on the analysis of the results from these actions. From the system theory perspective, driving a case with uncertainty constitutes a system with a feedback loop. Actions are planned and corrected based on the assessment of the results produced. With a high-level of uncertainty, one expects a possibility to start over again with a loss of much or all of the results achieved in the previous actions, except lowering the level of uncertainty in the case.

Volatility of environment concerns unexpectedness of changes that are outside the control of the process participants. Volatility may be mixed with high-level uncertainty as defined above, but may also occur in the situations where uncertainty is low. In the latter situations, the actions are taken based on the current situation at hand. Sudden changes in the environment may result in the needs to abandon the current course of actions, and thereby lose the results already achieved.

A typical ACM process has a high level of uncertainty, or volatility, or both. This makes re-planning an integral part of the process, which is not the case for the workflow process or the CM process where there exists a fixed plan from the very beginning. As it is hard to provide a predefined plan for the processes typical for ACM, when developing an ACM system, the focus is moved to providing data/information and collaboration support. This can be done by introducing templates as a middle ground between blank slates and a completely specified process [11], or using shared spaces for facilitating collaboration as in [5]. More detailed discussion on using shared spaces in BPS is available in [12].

A typical way of using a shared space in ACM is described in [5]. The shared space usage is based on a blackboard metaphor, and it aims to facilitate cooperation between different subsystems, called lines of work by the authors. Each line of work has its own goals, experts, responsibilities and work scope. The authors of [5] suggest a shared space divided into several sub-spaces, one for each line of work. The shared space also provides common properties for the whole case, i.e. for all lines of work, such as goals and states of the whole case and general case information. The benefits of the shared space are that all case data/information is visually available to everybody during the case life-time, and, thereby, experts in the different lines of work can be inspired by each other and continuously adapt their work to the work of others, thus being more proactive and ensuring successful conclusion of the case at hand.

One of the main challenges with ACM reported in a recent review [13] is the lack of proper theory and model of ACM, also emphasized in [14]. This prevents creating a common understanding of the concept of ACM, and hinders comparisons between different ACM solutions. OMG recently introduced a meta-model [7] for CM/ACM business process modeling. Though this model is based on the fact that a CM/ACM process includes continuous planning

[*] This is the authors' own definition

and re-planning, it does not set in focus the data/information perspective. Instead, it continues the workflow tradition of having the focus on the task/activity flow. The difference between the suggestions from OMG and the workflow tradition is that task ordering in a CM/ACM process is defined at runtime, not at design-time. The meta-model suggested by OMG is quite new, and its usefulness has not been proven in practice so far. In our view, missing the data/information part will limit the usage of OMG suggestions.

2.2 Agile Business Process Development

The difference between the traditional and agile business process development based on the analysis of knowledge transformation in these two kinds of development projects is explained in [1], see also Figure 1. The analysis itself has been done based on the Nonaka's SECI model from [15], where SECI stays for Socialization-Externalization-Combination-Internalization. The SECI model describes how knowledge is transformed between tacit and explicit forms in the process of knowledge generation in organizations. For the sake of analysis of business process development, [1] adds to the two forms of knowledge used in SECI (i.e. tacit and explicit) an additional form called embedded knowledge. Embedded knowledge represents the knowledge built-in in a BPS system. According to [1], the main difference between the traditional and agile business process development are as follows:

- In agile process development, depicted on the right hand side of Figure 1, phases of *Process modeling*, *Support system design* and *Support system manufacturing* are merged into one phase *Support system manufacturing*.
- The nature of the first phase in the agile development is changed compared to the first phase in the traditional development (see the top right corners of the models in Figure 1). In the traditional business process development, the cycle starts with *Externalization*, i.e. converting the knowledge of stakeholders (process participants) from tacit to explicit form – a detailed process model. In the agile business process development, the cycle starts with socialization, i.e. transferring tacit knowledge on the desired process from the stakeholders (process participants) to the design team.
- In addition, in the agile business process development, one big cycle is substituted by many smaller and shorter ones. The BPS system is built iteratively starting with the basic functionality that does not limit flexibility of process participants to experiment with the new process. During the usage of the basic system, better understanding of the needs is acquired, which is converted in adding more details to the system in the next cycle.

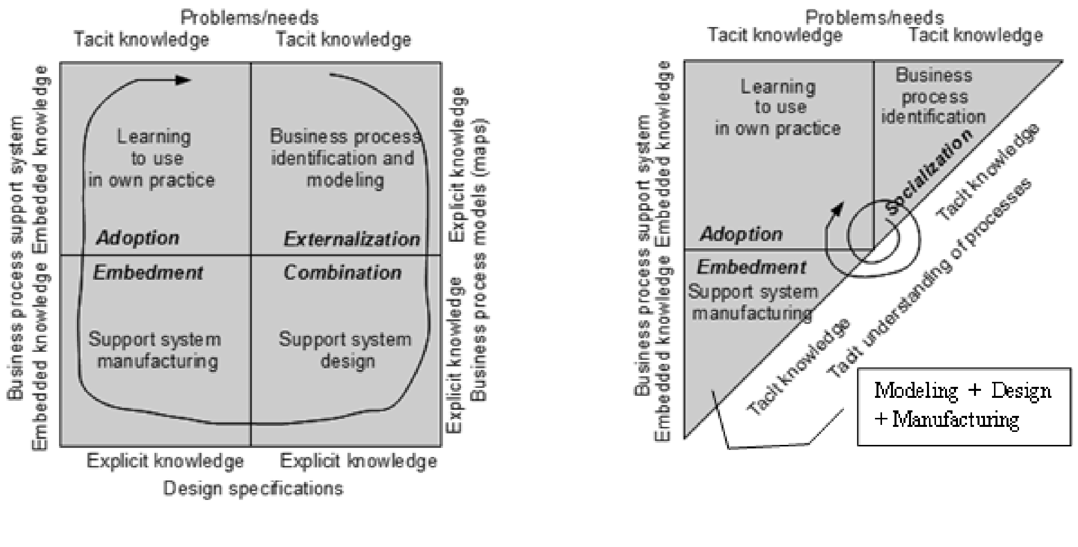

Problems/needs
Tacit knowledge
Tacit knowledge
Learning to use in own practice
Adoption
Business process identification and modeling
Externalization
Embedment
Support system manufacturing
Combination
Support system design
Business process support system
Embedded knowledge
Embedded knowledge
Explicit knowledge
Business process models (maps)
Explicit knowledge
Explicit knowledge
Design specifications
Problems/needs
Tacit knowledge
Tacit knowledge
Learning to use in own practice
Business process identification
Socialization
Adoption
Embedment
Support system manufacturing
Tacit knowledge
Tacit understanding of processes
Modeling + Design + Manufacturing

Figure 1. Models of knowledge transformation for the traditional business process development on the left and the agile business process development on the right. Adopted from [1].

Agile business process development is most suitable when a completely new business process is to be introduced in the organization in question, or an existing process is to be radically changed due to changes in the organization’s environment. In both cases, there might not be enough information to design the process in details, and a try-and-error approach needs to be taken. In the agile approach, both the business process and a BPS system to support it are developed in parallel.

2.3 Why a New Business Process Modeling Technique is Needed?

In this section we give a line of reasoning that guided us when developing our data-centric modeling technique. This line of reasoning is illustrated in Figure 2 and explained in detail below.

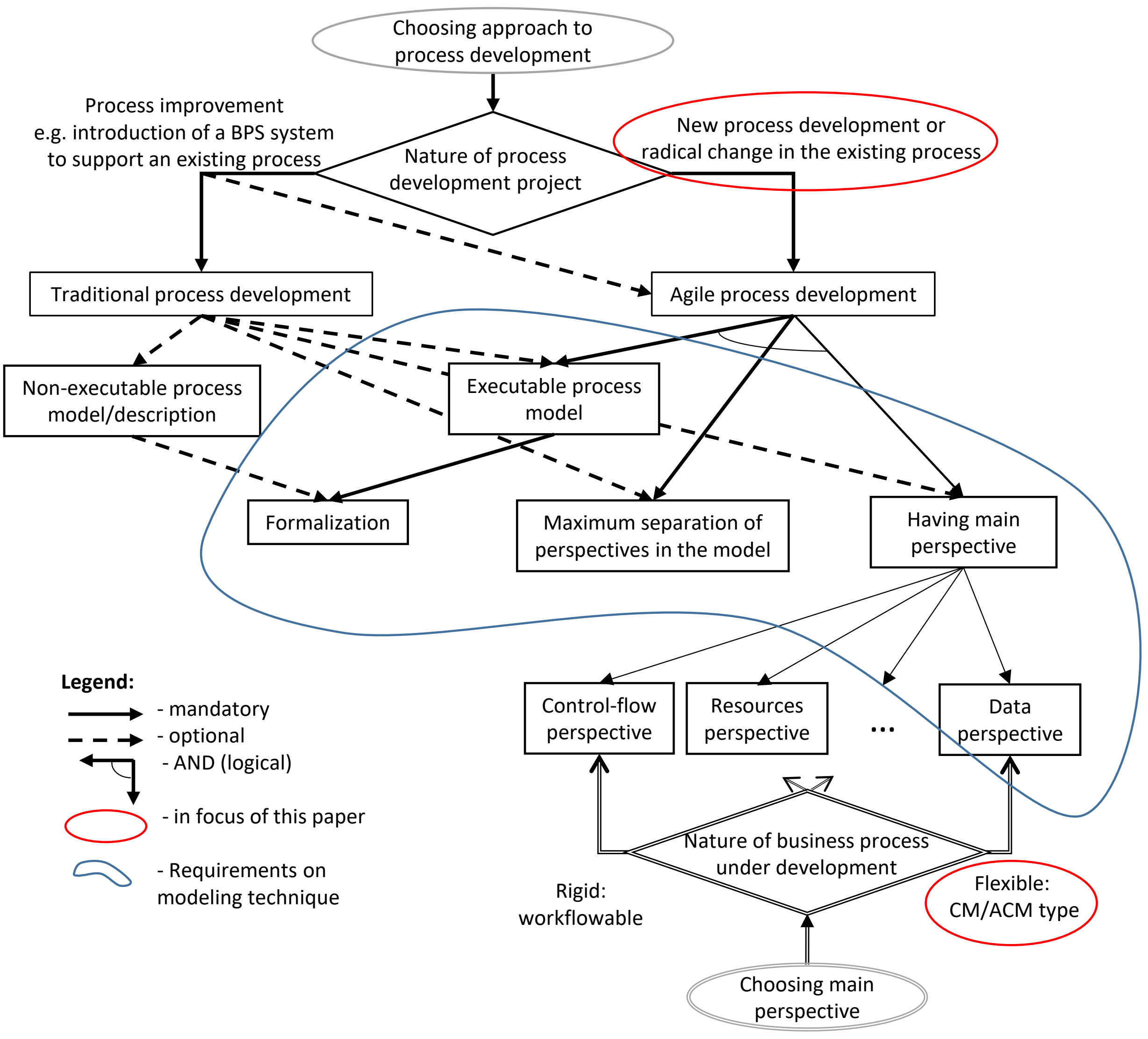
As was discussed in Section 2.2, agile business process development presumes having a high-level tool for building BPS system[*]. One, and most probably the only, way to create a tool that satisfies the two requirements from Section 2.2 is by using the principle of separation between the program code and the process description, or process model. This allows making changes in the process description/model without changing the system code. The separation between the process description/model and the program code is referred to as executable process models [16]. Implementing the idea of executable business process models in practice means having a tool that can interpret a business process model at runtime[†]. The tool has to be generic, i.e. capable of interpreting a wide range of business process models, to be practically useful. While executable process models can be successfully used in the traditional business process development, this is not mandatory. Therefore using executable models for the traditional process development is identified as optional in Figure 2, while their usage in the agile business process development is marked as mandatory.

To be used as an executable, a process model has to possess certain properties, the most important of which is formality, as the model is interpreted by a machine. The property of formality is essential here, in difference from process models used for other purposes, e.g. for better understanding of business, or for process improvement. In the latter cases, the model is interpreted by people, who can use their knowledge and experience to fill the gaps missing in the model, or understand “fuzzy” parts of the model. That is why the requirement on formalization is identified as mandatory for executable business process model, and optional for non-executable business process model in Figure 2.

As has already been discuss in Section 1, a business process has a number of different perspectives. To ensure quick iterations when using agile methodology, an additional requirement has to be added on the process model. Namely, different business process perspectives are separated from each other, which makes it easy to make changes in the model, and thus also in the BPS system that executes it. For instance, changing who is completing which activity should not affect which activities are included in the business process, and changing the order of activities does not need to change which data are used in the process, etc. Separation of perspectives is advantageous even when the traditional business process development is in place. It will make it easier to introduce changes in the process and BPS system when the changes in business environment occur. However, as with the previous requirement, for the traditional development method this is a “good to have” requirement, while with the agile method this is a mandatory requirement, as depicted in Figure 2.

[*] Such tools are often included in so-called BPM-suites.

[†] Besides having an interpreter, such a tool often include an environment for building (drawing) a business process model to be interpreted.



Choosing approach to process development
Process improvement e.g. introduction of a BPS system to support an existing process
Nature of process development project
New process development or radical change in the existing process
Traditional process development
Agile process development
Non-executable process model/description
Executable process model
Formalization
Maximum separation of perspectives in the model
Having main perspective
Legend:
- mandatory
- optional
- AND (logical)
- in focus of this paper
- Requirements on modeling technique
Control-flow perspective
Resources perspective
...
Data perspective
Nature of business process under development
Rigid: workflowable
Flexible: CM/ACM type
Choosing main perspective

Figure 2. A rational behind the modeling technique suggested in this article

From the requirements 1 in Section 2.2 (creating the first version of BPS in the shortest time), we can derive the needs to be able to start modeling with the perspective(s) most important for the given process type and given context without bothering to depict other perspectives. This need may require using different business process modeling techniques for different processes, unless a universal technique could be found that covers all perspectives to the required extent and, additionally, allows to start modeling with any of these perspectives. To the best of our knowledge, such modeling technique does not exit. Note that this requirement is most important for the agile development, which is reflected in Figure 2. The traditional development aims at building a full-featured BPS system in one go; therefore all perspectives have to to be presented in the model in one way or another. As a consequence, "Having main perspective" is marked as optional for the traditional business process development in Figure 2.

Which perspective to start with depends on the nature of the business process to be supported, see the bottom part of Figure 2. The most spread process modeling technique of today supports workflow thinking, which more or less forces to start modeling with the control-flow perspective that helps to ensure the right flow of tasks/activities in the process. A number of proprietary and open source tools and services use executable process models that have control-flow as the main perspective [17], while implementing other perspectives is often done relatively outside the process model. For instance, the data/information flow is often handled with the help of the notion of forms.

For the workflow modeling, there exist a number of formal modeling techniques and notations. These range from simple flowcharts to complex workflow diagrammatic languages such as BPMN, and they are supported by a number of modeling tools. What is more, there exists means that can handle formal semantics of workflow models, e.g. Petri-nets, or Colored Petri-nets [18], [19].

As has been discussed in Section 1 and Section 2.1, for CM and ACM systems, the data/information perspective is more appropriate as a starting point when designing executable models, see the bottom of Figure 2 in the neighborhood of the red circle. The absence of a formalizable modeling technique having the data/information view as the main perspective has motivated us to start developing a business process modeling technique that satisfies the requirements included in the blue circle on Figure 2, while following the plan presented in Section 1.

3 Existing Non-Workflow Business Process Modeling Techniques

As has been discussed in Section 2, the control flow perspective is not suitable as a starting perspective for development of CM/ACM systems based on the concept of executable business process models. The needs to have other types of business process modeling techniques besides workflow based ones are well understood in practice and research, and several attempts to switch focus from the flow of tasks/activities to data/information used in these tasks/activities have been made. In this section, we will review and analyze typical examples of such attempts, choosing the ones that have included relatively formal definitions of the process model. More specifically, we consider the following examples of non-traditional process modeling techniques: (a) declarative process modeling that includes the notion of state [20], (b) artifact-centered process modeling [21], [22], (c) data-driven process modeling [23] and (d) state-oriented process modeling [24].

Declarative process modeling [20] is based on the notions of state, activity and constraint. The state is defined with the help of a finite set of variables (i.e. the state is composite). Activities are considered as legitimate transitions from one set of states to another, and are used for moving from an initial state to one of the final states. Execution of activities is controlled by constraints that enables or fire them. Constraints are defined as predicates over the values of state variables. Besides the state, activity and constraint, the model includes the notion of external event that can affect certain state variables. It is suggested that the formalism could be useful for analysis of correctness of process models. At the time of writing, we do not have knowledge on such model being used for practical purposes, e.g. for practical process modeling or building BPS systems.

Artifact-centered process modeling [21] is based on the notion of artifact. [21] defines artifacts as “business-relevant objects that are created, evolved, and (typically) archived as they pass through a business”. Examples of artifacts are an air courier package, a deal, a supplier invoice and an asset. Each artifact is described in form of a life-cycle model represented as a kind of state machine. The life-cycle model describes the possible states that the artifact may go through while being alive. Related to each artifact and, thereby, to the life-cycle model, is also an data/information model containing all business relevant data managed by the artifact, including data/information obtained at the beginning of the lifecycle and added during the lifecycle. The benefit of the technique is that it enables modularity and componentization of business processes.

[22], which extends [21], introduces a formalization for the artifact life-cycle model. The technique is called Guard-State-Milestone (GSM). GSM supports parallelism and hierarchy within a single artifact instance. The data model related to each life-cycle model consists of data and status attributes. This model is fully formalized and has the fix-point semantics. It uses a rich set of notions that includes the notions of event, task, milestone and Event-Condition-Action rules. In the works on artifact-centered modeling available at the time of writing, we could not

find any examples of practical application of GSM modeling techniques in process modeling projects or for building BPS systems.

Data-driven process modeling [23] is based on the notion on data and states. In the technique, for each object type in the data model, an object life cycle model is designed. The latter is represented as a kind of state machine, called life cycle coordination structure. The benefit of the technique is that it supports coordination of many sub-processes of processes that deal with complex products, such as cars. For instance, the data model could consist of a system (e.g. a navigation system in a car) and a number of sub-systems (e.g. components of the navigation system). For each system and sub-system, an object life cycle model is designed that describes the states of the individual life cycle model and its communication with other life cycle models. The modeling technique suggested in [25] was used for analysis of IT systems that were employed for supporting a process of release management at a car manufacture.

State-oriented process modeling [24] is based on the state-oriented view on business processes. The main concept here is a state of the process instance that can be defined as a position in some state space. A state space is considered to be multidimensional, where each dimension represents some important parameter (and its possible values) of the business process. Each point in the state space represents a possible result of the execution of a process instance. If we add the time axis to the state space, then a trajectory (curve) in the space-time will represent a possible execution of a process instance in time. A process type is defined as a subset of allowed trajectories in space-time. Each process instance of the given type has a goal that can be defined as a set of conditions that have to be fulfilled before a process instance can be considered as finished (i.e. the end of the process instance trajectory in the space state). A state that satisfies these conditions is called a final state of the process. The process instance is driven forward through activities executed either automatically or with a human assistance. 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 planned and executed in the frame of the process should be aimed at diminishing the distance between the current position in the state space and the nearest final state. All activities currently planned for a process instance make up the process plan. The plan together with the current position in the state space constitutes a so-called generalized state of the process, the plan being an “active” part of it (i.e. an engine). With regards to the generalized state, the notion of a valid state is 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/model can be defined as a set of valid generalized states. A business process instance is considered as belonging to this process type if, for any given moment of time, its generalized state belongs to this set of valid generalized states.

The state-oriented modeling was used in practice for both building models of business processes, see, for instance [26], and for building business process support systems based on these models [27].

Summarizing the examples of non-workflow modeling approaches presented above, we can conclude that:

- All of them include definition of data-structures in one way or another, e.g. as a multidimensional state, or artifact.
- All of them also include notions of events, states, tasks/activities in one way or another, e.g. as labels marking the transitions between the states.
- Also, it seems that none of them has been tested for starting the design with the data/information perspective and getting a process model that could be effectively used in a BPS system without developing the full featured model that includes other perspectives, at least, to some extent. Moreover, from the descriptions available, it is clear whether they were not designed for this purpose in mind. Therefore, it is not clear that these process modeling

techniques could be tweaked to adjust them to our goal, and whether such tweaking makes sense.

Based on the deliberations above, we consider it feasible to start defining the new modeling technique from scratch based on our experience instead of trying to modify one of the existing techniques. The fact that none of them were tested for the aim we have in mind makes our decision at least as feasible as starting with any of them. This, however, does not exclude the indirect influence of the works summarized above on our line of thought, especially the influence of the state-oriented perspective on business process modeling. Our approach to data-centered business process modeling has been developed by reconstruction and critical analysis of our experience, especially the experience of building and using a service/tool called *iPB* aimed at agile development of CM/ACM systems.

4 Background for the Development of Data-Centric Modeling Technique

As was mentioned in the introduction, our design is based on the analysis of three sources (1) form-based case management in the public sector, (2) our own experience of developing form-based CM systems and tools that support such development, and (3) database theory and practice. As the first two sources are not well known, we present a short description of them in this section. As far as the database theory and practice is concerned, we presume that the reader knows this domain at the extent needed to follow the material of this article.

4.1 Form-Based Case Handling as an Example of Data-Centric Process Modeling

Our experience with CM systems started with the analysis of case handling in Swedish municipalities that included general case handling in the Municipality of Motala [26] and case handling in the social welfare office of municipality of Jönköping, e.g. handling applications for child adoption, and later handling cases of suspected child abuse (see Section 4.2). Case handling that we observed in municipalities was template/form driven, rather than task/activity driven. There were templates, often of the type of structured forms, for filling an application, investigation, and decision making.

Template/form driven case handling cannot easily be translated into the task/activity driven one, as the same template/form can be used for several different tasks/activities, e.g., for both application and decision making. Tasks/activities can be represented in such case handling in different ways. For instance, a task/activity of decision making can be represented in the form by three fields: (a) decision, (b) name of decision maker and (c) date when the decision has been taken. A set of tasks/activities can also be represented as a checklist on the form that requires putting a cross beside each task/activity on the list before the form can be considered as properly filled.

In the template-based case handling, there often are no strict rules as when starting to work with a certain template. For instance, the main result of an adoption case is a report that follows a certain template. The report is based on a series of meetings with the applicant(s), but there is no regulation when to start writing the report: directly after the first meeting or after the whole series has been completed. The choice is left to individual case managers. The same applies to the order in which the report is compiled. Though the report concerns 10 different issues placed in a certain order, obtaining and reporting information on them is not to mandatory be done in this order; the investigator can add information that concern any issue as soon as it becomes available.

Swedish public sector is not the only domain where designing forms are the most important part of designing a business process. This, for instance, this also applies to designing a process of reviewing journal or conference papers. The two most important forms to be designed here are a submission form that should include the information on the paper, e.g. title, authors, category, and one or several reviewing forms for different categories of papers. The reviewing work is controlled by the latter forms that prescribe the results desired to be achieved and not the tasks that should be completed, e.g., *download* the paper, *read* it, *think* of it critically, *write* a review, etc., and their order. The tasks that are used for describing the reviewing process, e.g. *submit*, or *review*, formally, are equivalent to “fill out a particular form and save its content”. The order in which these tasks are completed can be described as rules for filling out the forms. For instance, the reviewing form cannot be filled out before the corresponding submission form, including the text of the submission has been fully filled. Another set of rules could set restrictions on the human resources as follows: none of the authors on the submisions form could appear as a reviewer on the corresponding reviewing form*.

Summarizing the above, for some processes, a package of templates or forms to be filled can serve as a better model of the business process than a flowchart of tasks/activities to be performed. Such a package will say much more about the process than the flowchart. Adding the rules that restrict the ways the forms can be filled can serve as an alternative way of defining the order of tasks (different from the flowcharts).

4.2 The *iPB* Project – Developing a Tool for Supporting Loosely Structured Processes

iPB [28] is a tool from *Ibissoft AB* for developing BPS systems that support loosely structured business processes. The *iPB* project was initiated in connection to the Swedish National Board of Health and Welfare effort to standardize handling cases of suspected child abuse in Swedish municipalities. The process got the name *BBiC*, which is the Swedish abbreviation for “Childs’ Needs in the Center” (“Barnens Behov i Centrum” in Swedish). The process concerned investigation, decision making, and following up decisions in cases of suspected child abuses, or families that could not take care of their children. The standardization in *BBiC* was done not by producing a workflow diagram of the process, but by producing a package of templates/forms mandatory to be used during handling cases. Each municipality that licensed the process was free to choose their own way of implementing the forms. They could use the forms as RTF, or Word templates or incorporate them in their case-handling systems themselves or with the help of their ordinary system vendors or integrators.

Having been invited to participate in the project as a system vendor, *IbisSoft* decided not to start directly with developing a system to support the *BBiC* process, but to develop a tool that would allow to relatively quickly “configure” such a system by defining a process model. The goal was to be able to use the tool for developing BPS system for other form-driven processes later. One of the main requirements on the tool was that a person with knowledge in the domain, and with the minimum of technical knowledge, could use it for configuring support for a new business process. The development started in 2007 and it resulted in a tool called *iPB* [28], [29]. The theoretical and practical basis for *iPB* development was the state-oriented view on business processes [24] and prior experience of building business process support systems based on this view [27].

In the *iPB* project, the *BBiC* process has served and continues to serve as a pilot for the tool development. The *iPB*-based system that supports this process went through several major revisions after the first introduction, where the complexity of the process model had been gradually increased. Currently, the *iPB*-based system that supports *BBiC* has around 270 active participants, including 80 % of heavy users who work with the system on a daily basis, and 20 % of occasional users who work with it rarely and complete only few tasks.

The first version of *iPB* included means for form building and user interface design for navigating between the forms. In consequent versions, new functionality was added that allowed introducing of business rules that set restrictions on the usage of forms. The result was a tool based on the concept of executable business process model in which management of data has

* More detailed description of this process and forms used in it can be found in [40].

been placed in the focus. *iPB* is described in more details in Appendix A alongside with an example of a process that can be supported by an *iPB*-based BPS system.

5 An Outline of a Basic Data-Centric Business Process Model

In this section, we give an outline of our basic data-centered centric business process model. For this end we will employ an analogy with relational databases, assuming that the reader is familiar with this domain. This outline can be formalized, which is shown in Appendix B, where formal definitions for a substantial fragment of the model are presented.

5.1 Data Perspective

Let us assume that each process instance/case has its own relational database which state represents the state of this process instance. The database consists of *tables*, each table can be considered as related to some *form* of the type discussed in the previous sections. Each *column* in the table can be considered as a placeholder for a value of the corresponding field. Naturally, the columns are *typed*, and a type can refer either to a *simple object*, e.g., integer or string, or *complex object*, e.g. a company or person, that can be represented in the form as an area with several fields.

A row in the table represents the content of a form filled out by the user working with the corresponding BPS system. Some tables allow *only one row*, which means that the corresponding forms can be filled only once. Other tables allow *multiple rows*, which means that corresponding forms can be filled multiple times. Permitting/not permitting multiple rows represents a special property of a table in our database.

Each row in each table has its own unique identifier. This identifier is not important for the tables that accept only one row, but is useful for the tables that accept multiple rows. The unique identifier corresponds to the concept of surrogate keys in relational databases and the auto-numbering data type that exists in some DBMS.

Now, we can define a notion of *business process state st* as a set of rows stored in the instance database. A sequence of the states, called a *trajectory tr*, represents the development of the instance in time. Having the notions of *state* and *trajectory*, we can define rules that govern such development by restricting the set of allowed states, and allowed transitions from one state to another. To create such rules, however, we need to extend the notion of state to include additional components. Besides a set of rows in the database tables, the following components are included in the *extended state*:

1. A set of locked rows that cannot be updated until unlocked. A locked row represents a form that is no longer available for updating.
2. A set of locked tables. A locked table means that no rows can be added to or deleted from the table. In terms of forms, locking a table means that the user is not allowed to work with the forms related to the locked table. This can happen when the table is empty, which means it is not yet possible to work with the form, or when the table has some rows, which means that the work with the forms related to the locked table has been finished. Note that if the table is locked, all its records (if any) are also locked.
3. A set of row dependencies in the form of pairs $<r_1,r_2>$, where r_1, r_2 are rows in two different tables. Such a pair represents a dependency of row r_1 on row r_2. Dependencies are important for tables that allow multiple rows. If forms filled at one stage need to be complemented with other forms later, then there is a need to establish which of the latter forms correspond to which of the former ones. In terms of relational databases, a dependency is just an additional table (or set of tables) with two columns. It represents relations (or associations in terms of UML) between objects stored in other tables.

A sequence of extended states is called an *extended trajectory etj*. Having a notion of extended state and extended trajectory as a representation of a development of the process instance/case in time, we can define several types of rules that govern such development. The rules can be of general character, i.e. valid for all business process types; pr they can be specific for a particular process type.

To the general rules, belong, for instance, the following:

- If a row in a specific table is locked in a given state *st*, and it exists in the next state st_1, than all its fields in state st_1 have the same values as in state *st*.
- If a table *t* is locked in a given state *st*, then all rows in this table are also locked in state *st.*
- If a table *t* is locked in a given state *st*, then its content cannot be changed in the next state $st_{1.}$
- In the last state of the extended trajectory all tables are locked.

The specific rules can be introduced by assigning certain properties to tables, columns or relations between the tables. Examples of such rules are as follows:

1. *Mandatory columns*. We introduce two types of mandatory columns in a table – one for saving, and one for locking. Each row in the table should have values in the columns mandatory for save; if some of them are missing, the record could not be saved. In the same way, each locked row in the table should additionally have values in the columns mandatory for locking; if they are missing, the record cannot be locked. The latter will even prevent locking the table, as in a locked table all records should be locked.
2. *Multiple rows* is a Boolean property of a table (which has already been discussed earlier) that allows/forbids inserting more than one row in the table. If a table does not allow multiple records, then in any state *st* there can be at the maximum one row in this table. This property can be enriched by having a pair <*min*,*max*> of integer numbers as a value that defines the minimum and maximum number of records in the table. The *min* number can be used for prohibiting locking the table that has less than required minimum number of records. The *max* number will prohibit adding more records than defined by this number.
3. *Unlocking rules* – are conditions that should be fulfilled when an empty table can be unlocked for adding records. At the start of the process, all tables are considered to be empty and locked. If there are no unlocking rules for the given table, an empty table can be unlocked at any moment. An unlocking rule may require that before unlocking the table some conditions are satisfied. Such condition refers to other tables that have to be non-empty and/or locked in order for the condition to be satisfied. The unlocking rules constitute a mechanism of establishing order in which records can be added to the tables. This concept can also be extended to automatic unlocking, when the table is automatically unlocked in case the condition is satisfied. Note that the demand on some other table to be locked will also imposed for all mandatory lock columns in this table to have some values to ensure integrity of the database state.
4. *Conditions on the final states* – are conditions that should be satisfied for the extended trajectory to be considered as properly finished. One general condition that all tables are locked in the last state has already been mentioned. This condition ensures that all mandatory columns have values in all rows of all tables. The specific conditions set requirements which tables should have records and which ones could or should remain empty.
5. *Table dependencies* – is a set of rules related to the dependencies between the rows from different tables. For instance to such rules belongs the requirement that any row r_i in table t_1 should have a corresponding row r_j in table *t2* such that there is a dependency < r_i , r_j>. Deleting row r_j in table t_2 automatically leads to deleting row r_i in table t_1.

In Appendix B, we present formalization of the outline discussed above using slightly different terminology. We use term *variable* instead of *column*, *group* instead of *table*, and *record* instead of *row*. We will use this terminology in the rest of the article. The main concepts that are presented in this section and Appendix B, and their analogs in the world of relational databases are summarized in Table 1, the last column of which refers to the sections of Appendix B that discusses the corresponding concepts.

Table 1. Main concepts of the formal data-centric business process model

Concept of data-centric business process model	**Similar database concept**	**Section in App. B**
Scheme	Database Scheme (Schema)	B.1
Group	Table	B.1
Variable	Column/Attribute	B.1
Variable name	Column name	B.1
Set of variable values	Domain of a column/attribute	B.1
Record	Row	B.1
Record identifier	Primary key, surrogate key, auto-number, etc.	B.1
Trajectory	The concept does not exist in the classical relational database, but implicitly exists in temporal databases	B.1
Mandatory variables	NOT NULL columns	B.3.3
Non-repeatable groups	The concept does not exist in the classical relational database, but could be implemented with the help of triggers	B.3.3
Record and Group locking	The concept does not exist in the classical relational database, but rows and table locking is used in technical implementation to ensure integrity in the multi-user environment.	B.3.2
Rules of unlocking	The concept does not exist in the classical relational database	B.3.5
Condition on final states	The concept does not exist in the classical relational database	B.3.6
Records and groups dependencies	Relationships via join, definitions of foreign keys plus definitions of alternative keys or unique indexes to ensure one-to-one or one one-to-many relationships between the records.	B.3.2 and B.3.7

5.2 Human Resources Perspective

Human resources perspective has two dimensions. The first dimension is to define which participants have rights and/or obligations to participate in the process instances and in what capacity. The second dimension concerns who are assigned to do what in the frame of a particular process instance/case. To reflect these two dimensions in the frame of our data-centric business process model, we follow the database practice and introduce the concept of roles and users to represent these dimensions accordingly. Roles are defined on several levels:

1. *Global*, were the role correspond to the position of the user in the given organization, e.g., manager.
2. *Process instance*, were the role defines what a given user does in the frame of the whole process instance/case, e.g. *owner, secretary*.
3. *Group* (or *table*) in the given process instance, where the role defines the responsibilities of the user in relation to this group, e.g., *author* (the person who adds data), or controller (the person who is responsible for checking the data in this group before the group is locked.
4. *Record* (or *row*) in the given group of the given process instance, where the roles are similar to the ones of the group but the actions are limited to a particular record in the group.

A user is assigned a role on the Global level as soon as he/she can potentially become a participant of the process. Assigning the role on the lower levels is done for a particular process instance and particular groups and records in this instance database.

Two types of information are connected to each role:

1. Which roles at the lower levels a user with a particular role on the upper level can have. For instance, a user with the *manger* role on the global level (level 1) can be assigned the *owner* role on the *process instance* level (level 2).
2. Which access rights to the components of the instance database a given user has if he/she has been assigned a certain role. Access rights are defined on three levels (1) *process instance database*, (2) *group* (table) and (3) *record* (row). Detailed description of possible access rights is presented in Table 2.

A user can have several roles in relation to a particular database component through different levels of engagement in the given process instance. For instance, a user can be an *owner* of the given process instance database, an *administrator* of a particular group in this database, and have some role on the global level. Each role can bring certain access rights to a particular database component. In this case, the actual access rights to the database component are defined as a union of access rights from all different roles for this user in relation to the database component.

Table 2. Access rights

Scope	Access right	Comment
Process instance database	Start	Create a new and empty instance database
	Finish	Finish the given process instance by locking all groups in the database
	Assign users	Assign a user a certain role in the frame of the given process instance on the Instance, Group or Record level
	View	Be aware of the given instance (database) existence, e.g. see it in a list of all process instances
	Revive	Revive a finished process instance to make it possible to continue updating the database
Group (table)	Add	Add new records to the group
	Delete	Delete records from the group
	UnlockEmpty	Unlock the empty group (table) thus signaling a start for adding records to this group
	Lock	Lock the group
	Assign users	Assign a user a certain role in the frame of the given group on the Group or Record level
	View	Be aware of the given group existence in the database
	UnlockNonEmpty	Unlock an already locked group that have records allowing modification of data in this group
Record (row)	Read	Read record
	Update	Update record
	Lock	Lock
	Assign users	Assign a user a certain role in the frame of the given group on the Record level
	View	Be aware of the given record existence in the group
	Unlock	Unlock a locked record allowing changes in it

5.3 Data Presentation Perspective

Besides tables, relational databases introduce the concept of views where data from several tables are combined to create a new virtual table. This helps in achieving more flexible access control, and independence of data usage from the structure of storage. A concept of views can be helpful for data-centric process modeling for the same reasons as for the databases. Views can be used instead of groups (tables) when defining the state and trajectory (extended state and trajectory) for a given business process instance. They can be subjected to unlocking rules instead of tables. They can also be subjected to the same rules of access control as tables. The latter allows creating completely different views on the same process for different categories of users. It also allows hiding some information from some users, while giving full access to others.

The following ways of creating views can, for instance, be employed for the basic data-centric business process model:

1. Group projection on a subset of variables that belong to the group, which corresponds to choosing only some columns from a table. This is the simplest way of creating a view that can be used for hiding some variables from some users.
2. Combination of group projections for two or more non-repeated groups that creates an aggregated view. This type of view can be used for providing easy access to the information already available to the users who need to add extra information. Suppose we have two groups g_1 *and* g_2 that according to the unlocking rules are to be filled with records in a sequential order, first g_1 and then g_2. Suppose also that the user that deals with g_2 needs some information from g_1. Instead of the user going back and forth between the records in these two groups, we can provide him/her with a view that includes all variables from g_2 and relevant variables from g_1 making the latter read-only in the view.
3. Combination of group projections for two or more repeated groups for which one-to-one record dependencies are established. This is a variation of 2 above, but the view created will be with repeated records, part of the variables coming from one record, other parts from the other(s). This type of views can be used for the same purpose as in point 2 above.
4. Master-detailed view, a combination of projections of one or more non-repeatable groups and one or more repeatable groups with or without dependencies between them. This type of views helps to group data with which a user completing a certain task has to deal. Some of the projections can be read-only (filled via other views), others are filled when working with this particular view. Another way of using such views is creating a kind of reports related to the progress of the whole process instance.

5.4 Pragmatic Mechanisms Outside the Process Model

Besides a good theoretical basis, the field of (relational) databases gives examples of pragmatic mechanisms implemented in modern DBMS that are widely used for developing database-centric software systems. Similar mechanisms can also be employed in connection to our data-centric business process model. A typical example of such mechanisms is triggers that allow adding actions to any attempt to change the database state. Such actions can be of internal nature, i.e. result in additional changes in the database or preventing changes in the database, or completely external, i.e. invoking an external function or service. Though triggers may not become part of the theoretical process model, they can be useful when building a BPS system based on the execution of the data-centric business process model. For instance a trigger can be set on any action of assigning a user some role on the instance, group or record level to inform the user, e.g. via email or SMS, about his/her new assignment. Another trigger can be set for any change of any record or group informing the users having some roles on this record or group level about changes related to their current assignments.

6 Extending the Basic Data-Centric Process Model by Adding New Semantic Concepts

One of the distinctive features of the basic data-centric process model from the previous section is that no semantics is assigned to the variables (columns) in the model, except that the values of the given variable should belong to some predefined domain. A variable can represent pure data, or an action to be or already completed in the frame of the given business process instance. While the basic data-centric model can be extended in different ways, its extension without identifying the semantics of data has limitations. For instance, the condition of unlocking introduced in the basic model is expressed in terms of groups having no records and locked. In principle, the unlocking rules can be extended so that any kind of logical expressions that include predicates defined over the content of the instance database can be used as conditions for unlocking. In this case, we can introduce data-dependent unlocking rules. However, while such extension greatly increases the expressive power of the modeling language, it will also makes it difficult for business people to define and understand such rules. To avoid the latter, the rules have to take into consideration the semantics of data included in various groups and views. The idea is discussed in more details below with the reference to the example presented in Appendix A.

Analyzing the *iPB* model presented in Appendix A, we can see that data can represent different things, e.g.:

- Artifacts, such as *Lecture presentation* in Figures A3 and A4 .
- Artifacts characteristics, such as *Title* in Figures A3 and A4.
- Resources assigned to an action, e.g. *Teacher 1*, *Teacher 2* in Figures A3 and A4.
- Time for action, e.g. *Start* and *Finish* in Figures A3 and A4.

Taking into consideration the meaning of data and connection between groups and records we can create rules that would facilitate some automation. For instance, group *Lecture/Lessons Teacher Feedback* (see Figures A2 and A5 in Appendix A) can be automatically unlocked when the real time becomes equal to the start time of the first planned lecture. Moreover, at the time of the lecture start, a new form for recording feedback can be added to the Feedback group, In addition, *Teacher 1* and *Teacher 2* can become *responsible* (role) for completing the feedback form of Figure A9. More detailed discussion on this matter see in [14].

The example above demonstrates that adding semantics to data elements can make rules of data-dependent automatic unlocking and user assignment easy to understand from the business point of view. This example is typical for ACM processes where planning is followed up by execution of the plan, see the discussion in Section 2.1. As the semantic relationships between the groups representing planning and executing plan are quite general for the ACM processes, it is worth to include this relationship in the model in an explicit form, like linking variable in one group to the condition (time) of unlocking of another group.

Another important issue of business process modeling is process decomposition that allows invoking subprocesses common for several processes. Process-subprocess relationships can formally be represented via introduction of variables that take values of process instances. These variables require special treatment, as it should be allowed to include the values of the subprocess variables in the views that belong to the main process to reflect the changes taking place in the subprocess. From the data-base point of view this is analogous to having views over several databases.

7 Implementation of the Data-Centric Model in *iPB*

The fact that the design of our formal data-centric model was inspired by analysis of *iPB* does not mean that the model just formalizes what has already been built-in in *iPB*. *iPB* has been built in a pragmatic manner in several iterations that extended the initial design in several directions, while testing the tool in several practical projects. The formal data-centric model has been built based on the database theory that helped to generalize and formalize the features implemented in *iPB*. The goal of formalization has been to give exact semantics to practical modeling techniques implemented in the tools like *iPB*. This model can be used to guide further development of the existing tools, like *iPB*, and/or development of completely new tools based on this model. In other words, the formal model is to be used by the tool developers to analyze existing tools in order to find the areas of improvement, and/or create new tools.

As *iPB* is a practical tool created through experimentation, it does not implement the data-centric business process model exactly as defined in Section 5 and Appendix B. In particular:

- *iPB* “mixes” internal structure of the database with data representation to the end-user in the concepts of process maps and forms.
- *iPB* uses business terms, such as *map, step, form, widget, open, close, business rule,* etc., while formal model uses more abstract notions, such as *variable, group, record, scheme, state, locked, unlocking rules,* etc.

Still, *iPB* implements many of the concepts introduced in Section 5 and Appendix B, though some time this is done indirectly or in a mixed features manner, In Table 3, we present the correspondence between the concepts of the formal model and *iPB* to highlight similarities and differences between them.

Table 3. Correspondence between *Formal and iPB* modeling concepts

Formal concept	Section	*iPB* concept	Comments
Variable (column)	5.1 B.1	*Form Field*	*Form Field* defines both the variable, i.e. the name and a set of values, and its visualization in the web form. See Figure A3.
Variable Assignment	B.1	The value that the end-user inputs in a *Form Field*	See Figure A4.
Group (table)	5.1 B.1	*Form*	*Form* defines both a set of variables and visualization of them. In addition, *Form* allows including references to the fields in another form. The latter means that *Form* is also used for defining a view that combines records from more than one group. See Figure A3.
Record (row)	5.1 B.1	*Form filled and saved by the user*	See Figure A4.
Process scheme	B.1	*Process map*	
Locked record (locked row)	5.1 B.2	*Filled form* saved by the user and marked as *write protected*	See the *Write protect* button in the tool menu of Figure A3 (with lock icon to the left of it).
Locked group (locked table)	5.1 B.2	*Step* in the *process map* that is marked as *gray* (cannot be started yet)*, or blue* (finished)	The gray color corresponds to the empty locked groups. The blue color corresponds to the non-empty locked groups. See Figure A5.
Record dependencies (row dependencies)	5.1 B.2	*Filled and saved forms* that belong to *synchronized* steps	Synchronization means that two steps have the same number of filled forms represented as tabs in Figure A9. Dependencies are shown in two ways: (a) through having the same tab labels, and (b) through reference fields showing the values from the ordinary fields that correspond to the parent form. For instance, reference fields *Title* and *Type* in Figure A9 shows the values from the corresponding field in the *Seminar* form.
Extended scheme	B.3.2	*Process map* + business rules of all types, see below	
Mandatory variables (mandatory columns)	5.1 B.3.3	Field property *Required* with values: *Not required, On save, On close* (Locking)	For instance, field *Title* in Figure A3 and A4 has property *Required* set to *Required on save*, while fields *Type*, *Mandatory*, *Start* have property *Required* set to *On close.*

Formal concept	Section	*iPB* concept	Comments
Non-repeatable groups (multiple rows)	5.1 B.3.4	Step property: *One form only* of *Yes/No* (Boolean) type	*One form only* corresponds to non-repeatable groups. For instance, step *Course general description* in Figure A2 has *One form only* set to *Yes.*
Rules of unlocking	5.1 B.3.5	*Step start rules*	See the matrix in Fig A3. The matrix limits the unlocking expressions to the ones that include operators *OR* only or *AND* only.
Conditions on final states	5.1 B.3.6	*Not yet implemented*	
Group dependencies (record depedndencies)	5.1 B.3.7	*Step synchronization rules*	See the matrix in Fig A4. The matrix imposes strict rules introduced at the end of Section B.2, and at the end of Section B.3.7.
Human resources	5.2	*Partly implemented*	Roles are assigned on *Global* (via profile manager) *Instance*, and *Group* level. No explicit assignment on record level is implemented. Access rights are defined only on the *Global* level (via profile manager); no access rights are connected to roles assigned on the *Instance* and *Group* level.
Views	5.3	The concept of *form* implements both groups and views	The views functionality is limited in a way that all fields that refer to other forms are defined as read-only.
Pragmatic mechanisms	5.4	Notification to participants on instance and group level	Any assignment of a role to a particular user can result in this user getting an email message. The same happens when there is the change in the data related to the step in which the user has some role.
Assigning semantics to variables	6	The first step made: the relationship between the process instances of process-subprocess type has been implemented	The only feature implemented from the extended model is process variables (fields in *iPB*) that refer to other process instances.

8 Discussion and Plans for the Future

8.1 Notes on the Research Paradigm Used in the Project

The research reported in this article has been completed in the frame of Design Science (DS) paradigm [30], [31], [32]. Althought normal practice is to discuss methodological aspects in the beginning of the article, we have chosen to put it at the end in order not to distract the readers unfamiliar with DS to follow the ideas and arguments[*]. DS is related to finding new solutions for problems known or unknown [33]. To count as a DS solution, it should be of a generic nature, i.e. applicable not only to one unique situation, but to a class of similar situations, cf. Principle 1 of [34]. DS research can be considered as an activity aimed at generating and testing hypotheses for future adoption by practice [35].

DS, as a way of generating and testing hypotheses for generic solutions, requires researchers to act in two different worlds: (a) the real world of specific situations, problems and solutions in local practices, and (b) the abstract world of generic situations, problems and solutions [35]. DS does not impose any particular order of movement in the two worlds. A researcher can start with a specific problem in a specific situation, find a solution for it (situation to-be that solved the problem), and then generalize all three parts of his/her test case: situation, problem, and solution. Classification of the ways of working in this manner is presented in [33]. The researcher can also

[*] On the problems that can arise when wrapping the text in methodology, see [39].

start from the other end – with finding a generic solution for a known generic problem and then try to find and implement a test case for its demonstration. A kind of "shuttling" between the two spaces is suggested in [34] in the Formalization of Learning stage of Action Design Research (ADR), which aims at transforming situated learning results into classes of field problems and generically applicable solutions.

This research was initiated by a practical problem of incremental (agile) development of BPS systems of the CM/ACM kind. This problem has been formulated on the general level as a having a business process modeling technique that complies with the following requirements (see Section 2.3):

1. Different business process perspectives are separated from each other in the model, which makes it easy to make changes in the process model, and thus in the BPS system that executes it.
2. Process modeling can start with the perspective most important to support in a minimal BPS for a given process. For CM/ACM processes, the data perspective is the one that can help in building a minimal BPS.

The general solution as a data-centric business process model was sought through analysis of a local practice – the *iPB* tool in our case. This analysis followed by designing an outline of the data-centric business process model based on the database theory and practice as presented in Section 5 and Section 6. A substantial fragment of this model has been formalized in Appendix B. Other parts can be also formalized, but this task is beyond the scope of this article.

DS also sets a requirement of a generic solution being practically useful, so that it can be used for solving specific problems. This issue is discussed in the next section.

8.2 Practical Usefulness

As was discussed in Section 4.2, *iPB* development was started as a reaction on a concrete practical task of developing CM systems based on the idea of executable business process models. As the commonly accepted workflow modeling techniques were not suitable for our task, we devised a practical modeling technique implemented in *iPB*. The development had several iterations in which new features were successively added to the process modeling technique and *iPB*. For instance, such features as the step start rules (see Figure A3 in Appendix A) and step synchronization rules (see Figure A7 in Appendix A) appeared quite late in the development, when the *iPB*-based *BBiC* system was already running at the municipality of Jönköping.

The data-centric process model presented in this article is a reaction on the needs of having a scientific base for development of *iPB* to avoid ad-hoc solutions that might be reversed at the later stages of development. The model suggested in this article gives formal semantics of *iPB*-based modeling and can be used to analyze the features already implemented in *iPB*, and guide further development. Even in its incomplete form as in Section 5, Section 6 and Appendix B, the formal model can be used for this end, as it is more general than the model built-in in *iPB*. For instance, unlocking rules in the formal model are of a more general nature that can be expressed in the matrix of Figure A7 in Appendix A, which may lead to extending the ways the unlocking rules are defined in *iPB*. Another example is the rules on the final states that are currently absent from *iPB*.

Analysis of roles and access control presented in Section 5.2 gives better understanding of what is missing in *iPB* and how to make access control more elaborated. Also, the idea of views helps to understand the conceptual mixture crawled in the *iPB* design, and how to separate different concepts to make this design cleaner. It also gives directions to new extensions, where different users can have different views on the process. This can be especially useful for inter-organizational processes where there is a need to hide details pertained to each individual organization revealing only information that can be useful for others.

Note that *iPB* is not the only tool that takes form-based perspective on business processes. Other tools, like IBM Forms Experience Builder [36], also try to combine forms with process management, though built on the workflow perspective. We believe that our data-centric model, especially extended with the concepts discussed in Sections 6, could be of help for analysis and further development of the existing form based tools, as well as building new tools from scratch. Moreover, even traditional workflow based vendors as Bizagi started to look for database related business process theories, as has been expressed in their keynote at BPM 2015 conference [37].

8.3 Research Contribution

The main goal of this article was to suggest a formalizable data-centric business process model that would be suitable for agile development of CM/ACM systems. This has been done in Section 5 and Appendix B where the main concepts of such model have been outlined and partly formalized in a step-wise manner based on the following two ideas:

1. A business process instance is viewed as a sequence of this instance database states called a trajectory.
2. A business process model is viewed as a database scheme that includes rules of correctness of allowed trajectories. Such a scheme, called extended scheme, includes rules that ensure integrity of the database state and sets restrictions on transitions from one state to another.

In addition to the basic data-centric process model of Section 5 and Appendix B, the article discusses its possible extensions via introduction of semantics of data that could enhance the expressive power of the model (see Section 6).

The approach to business process modeling suggested in this article radically differs from the traditional approaches. A traditional approach, normally, starts with the notion of task/activity and tasks/activities flow, and only later the concept of data/information flow might be added. Our approach starts from the opposite end, i.e. with defining data structures to be used when running the process instances, and data/information flow (through the rules of unlocking). Only after that, the notion of task/activity might be introduced through assigning semantic interpretation to some data elements, e.g. considering them as tasks/activities planned or executed. Our approach also differs from other non-traditional approaches to business process modeling, which introduce both the notion of data and task/activity at the same time, see Section 3. To the best of our knowledge, an approach similar to ours has not been discussed in the contemporary research literature.

Our approach has its roots in practice. It was built based on the analysis of the existing ways of case management in the Swedish public sector, and practical experience of development of BPS systems of the CM/ACM type based on the idea of separation of code from the description/model of the business process to be supported. More specifically, our formal model was built based on the analysis of *iPB* and *BBiC* projects presented in Sections 4.2 and 4.3.

We envision that the formal data-centric process model would be useful for analysis of the existing CM/ACM systems and tools for building such systems. This can be done by mapping the concepts built-in in the given CM/ACM system/tool to the ones of the formal model in order to better understand the system/tool properties and have a scientific base for planning further development of this system/tool. This procedure has been applied to *iPB* in Section 7 and 8.2, which constitutes the first proof of the formal model usefulness. We believe that such mapping could be of use for planning further development of other existing ACM systems/tools, and for the development of new BPS systems and tools that support such development.

In our view, the following features of our data-centric business process model make it potentially useful for BPS system development aimed at supporting processes of CM/ACM type:

1. It has the focus on data structures and data flow which is of importance in this kind of systems, especially for the ACM systems aimed at supporting knowledge workers in

knowledge intensive processes. In this kind of support, providing different views on the data/information obtained or created in the process is of utter importance.

2. The model has a theoretical and experience-based ground from the database theory and practice, which is a mature discipline with a long history of practical usage. This gives the possibility to continue the development of the model by “borrowing” the concepts from the database domain, as we have demonstrated in Section 5.2 and Section 5.3. The existing experience of building Database Management Systems (DBMS) could also be helpful for building tools that supports CM/ACM development (see Section 5.4).

The model is built using the idea of separating different concerns from each other as much as possible. The extended scheme that defines the process model consists of seven elements, which can be changed relatively independently from each other. For instance, new variables can be added to a group without any need to change the unlocking rules. This separation of concerns was systematically followed in *iPB*, where form design is separated from the step start rules, and from step synchronization rules. What is more, even inside one component of the model, e.g. unlocking rules, rules are separated from each other; adding or deleting a rule does not necessarily affects other rules[*]. The latter is also respected in *iBP*, changing a cell in the step start rules matrix in Figure A6 (Appendix A) on its own does not warrant changes in other cells.

The separation of concerns is important for achieving three practical goals when using our formal model for building CM/ACM systems/tools:

1. It is possible to start developing a BPS system and especially a tool that supports development of CM/ACM systems with implementing only some concepts of the formal model, while leaving the rest of them to be implemented in the next iterations of the development. Having such a possibility is especially important if the agile method of software development has been chosen. As our experience from *iPB* and *BBiC* projects shows, it is possible to develop a useful system/tool without implementing all concepts of the formal model. For instance, as has already been mentioned, the *step start rules* that implements the *unlocking rules* of the model were introduced in *iPB* at a relatively late stage.
2. The separation of concerns makes it easier to develop new processes in an agile manner when the process development is conducted concurrently with the BPS system development [1]. Using a tool with separation of concerns allows to start with very loosely defined process, e.g. as a set of forms to fill when running process instances. Then, based on the gathered experience, some restrictions on the order in which the forms are to be filled are introduced and/or some fields are made mandatory for save or locking. This can be done and redone without any or with only minor restructuring of the process that is already in operation.
3. Even without employing the agile development, separation of concerns makes it easier to maintain the system when changes in the business environment require changing the process, and thus – the BPS system that supports the process.

8.4 Plans for the Future – Challenges to Overcome

The formal definitions presented in Appendix B formalize the substantial fragment (that corresponds to the ideas from Section 5.1) of the basic data-centric business process model outlined in Section 5. Other components of the basic model, i.e. access control and views presented in Section 5.2 and 5.3, still need formalization. Also, our suggestions for the extended data-centric process model discussed in Section 6 need, firstly, further analysis and, secondly,

[*] This can be seen in contrast to the workflow models were moving an activity from one place to another may require quite many changes. The problem caused several researchers to become engaged in the task of reordering activities simpler.

formalization. Both formalizing the rest of the basic model and working on the extended model are included in our plans for further research.

However, extending and formalizing the model, on its own, does not guarantee the model being adopted by practice. As discussed in Section 5, the data-centric business process model is too formal to be of use for business people, and even for majority of software developers. Therefore, the main challenge for promoting the data-centric process modeling is in creating a more suitable notation for depicting process models.

Fully understanding the main challenge above, we include the development of a practically oriented notation for data-centric process models in our plans. Such notation should be graphical and expressed in business related terms. In developing this notation, we hope to get help from the experience obtained in the *iPB* project in which graphical means were used for representing various formal concepts. Though the *iPB*-based notation is not ideal from the research point of view, it proved to be useful for non-academics and non-technical people. The executable model for the *BBiC* project discussed in Section 4.2 was built by one domain specialist who continues to develop the *BBiC* system having only the second-tier support from *IbisSoft*. The person is a professional in the domain of the social welfare who acquired some technical knowledge to become a go-between between technical and business people. Besides building and supporting the executable *BBiC* model, he used *iPB* for creating IT support for about a dozen of other processes in the office of social welfare of Jönköping municipality.

Acknowledgement

This work could not have been completed without having an experience from the practically oriented projects related to *iPB* development and usage. The authors are in debt to all people that participated in these projects, especially to Tomas Andersson, Alexey Striy, Rogier Svensson and Zakria Riaz Dar.

References

[1] I. Bider and A. Jalali, “Agile business process development: why, how and when – applying Nonaka’s theory of knowledge transformation to business process development,” Information Systems and e-Business Management, vol. 14, no. 4, pp. 693-731, 2016. Available: https://doi.org/10.1007/s10257-014-0256-1

[2] L. Baresi, F. Casati, S. Castano, M. Fugini, P. Grefen, I. Mirbel, B. Pernici, and G. Pozzi, “Workflow design methodology,” in Database Support for Workflow Management: The WIDE Project, Grefen, Pernici, and Sanchez (eds.), Kluwer, pp. 47–94, 1999. Available: https://doi.org/10.1007/978-1-4615-5171-3_4

[3] W.M.P van der Aalst, M. Mathias Weske, and D. Grünbauer, “Case handling: a new paradigm for business process support,” *Data & Knowledge Engineering*, vol. 53, pp. 129–162, 2005. Available: https://doi.org/10.1016/j.datak.2004.07.003

[4] K.D. Swenson (ed.) *Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done*. Tampa, Florida, USA: Meghan-Kiffer Press, 2010.

[5] H. Sem, S. Carlsen, and G. Coll, “How can the blackboard metaphor enrich collaborative ACM systems?” in *EDOCW – Proceedings of the 17th IEEE International EDOC 2014 Conference Workshops, Ulm, Germany, 1-2 September 2014*, ULM, Germany, pp. 139 – 142, 2014. Available: https://doi.org/10.1109/EDOCW.2014.29

[6] AdaptiveCM. (2014) AdaptiveCM – 3rd International Workshop on Adaptive Case Management and other non-workflow approaches to BPM, 2014. [Online]. Available: http://acm2014.blogs.dsv.su.se.

[7] OMG, “Case Management Model and Notation,” Object Management Group, 2014.

[8] I. Bider, E. Perjons, and Z. RiazDar, “Using Data-Centric Business Process Modeling for Discovering Requirements for Business Process Support Systems: Experience Report,” in *BPMDS 2013 and EMMSAD 2013, LNBIP 147*, Valencia, Spain, pp. 63–77, 2013. Available: https://doi.org/10.1007/978-3-642-38484-4_6

[9] M Reichert, S. Rinderle, and P. Dadam, “Adept workflow management system,” in *Business Process Management*, pp. 370–379, 2003. Available: https://doi.org/10.1007/3-540-44895-0_25

[10] H. A. Reijers, J. Rigter, and W. M. P. van der Aalst, “The case handling case,” *International Journal of Cooperative Information Systems*, vol. 12, no. 3, pp. 365–391, 2003. Available: https://doi.org/10.1142/S0218843003000784

[11] D Khoyi, “Templates, not programs, Mastering the unpredictable: How Adaptive Case Management Will Revolutionize The Way That Knowledge Workers Get Things Done, 2010.,” in Mastering the unpredictable: How Adaptive Case Management Will Revolutionize The Way That Knowledge Workers Get Things Done, K.D. Swenson (ed.), 2010.

[12] I. Bider, P. Johannesson, and R. Schmidt, “Experiences of Using Different Communication Styles in Business Process Support Systems with the Shared Spaces Architecture,” in Proceedings of CAiSE 2011, LNCS, Springer, vol. 6741, pp. 299–313, 2011. Available: https://doi.org/10.1007/978-3-642-21640-4_23

[13] M. Hauder, S. Pigat, and F. Matthes, “Research Challenges in Adaptive Case Mangement: A Literature Review,” in EDOCW – Proceedings of the 17th IEEE International EDOC 2014 Conference Workshops, Ulm, Germany, 1–2 September 2014, Ulm. Germany, pp. 98–107, 2014. Available: https://doi.org/10.1109/edocw.2014.24

[14] I. Bider, A. Jalali, and J. Ohlsson, “Adaptive Case Management as a Process of Construction of and Movement in a State Space,” in *On the Move to Meaningful Internet Systems: OTM 2013 Workshops, LNCS 8186*, Graz, Austria, pp. 155–165, 2013. Available: https://doi.org/10.1007/978-3-642-41033-8_22

[15] I. Nonaka, “A dynamic theory of organizational knowledge creation,” *Organ. Sci.*, vol. 5, no. 1, pp. 14–37, 1994. Available: https://doi.org/10.1287/orsc.5.1.14

[16] P. Kueng and P. Kawalek, “Goal-based business process models: creation and evaluation,” *BPMJ*, vol. 3, no. 1, pp. 17–38, 1997. Available: https://doi.org/10.1108/14637159710161567

[17] Bonitasoft, 2014, Bonita BPM. [Online]. http://www.bonitasoft.com/

[18] W.M.P. Aalst van der, “Three Good Reasons for Using a Petri-Net-Based Workflow Management System,” in *Information and Process Integration in Enterprises*, T. Wakayama, et al. (ed), Springer, pp. 161–182, 1998. Available: https://doi.org/10.1007/978-1-4615-5499-8_10

[19] R.M. Dijkmana, M Dumasb, and Ouyang C., “Semantics and analysis of business process models in BPMN,” *Semantics and analysis of business process models in BPMN*, vol. 50, no. 12, pp. 1281–1294, 2008. Available: https://doi.org/10.1016/j.infsof.2008.02.006

[20] P. Soffer and Y. Tomer, “A State-Based Context-Aware Declarative Process Model,” in *BPMDS 2011 and EMMSAD 2011, LNBIP 81*, London, UK, pp. 148–162, 2011. Available: https://doi.org/10.1007/978-3-642-21759-3_11

[21] D. Cohn and R. Hull, “Business artifacts: A data-centric approach to modeling business operations and processes,” *IEEE Data Eng. Bull.*, vol. 32, no. 3, pp. 3–9, 2009.

[22] R. Hull, E. Damaggio, and R. De Masellis, “Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events,” in *DEBS’11*, NewYork, USA, pp. 51–62, 2011. Available: https://doi.org/10.1145/2002259.2002270

[23] D. Müller, M. Reichert, and J. Herbst, “Data-Driven Modeling and Coordination of Large Process Structures,” in *OTM2007, Part I, LNCS 4803*, pp. 131–149, 2007. Available: https://doi.org/10.1007/978-3-540-76848-7_10

[24] M. Khomyakov and I. Bider, “Achieving Workflow Flexibility through Taming the Chaos,” in *OOIS 2000 – 6th international conference on object oriented information systems*, pp. 85–92, 2000. Available: https://doi.org/10.1007/978-1-4471-0299-1_7

[25] D. Müller, J. Herbst, M. Hammori, and M. Reichert, “IT Support for Release Management Processes in the Automotive Industry,” in *BPM 2006, LNCS 4102*, pp. 368–377, 2006. Available: https://doi.org/10.1007/11841760_26

[26] T. Andersson, A. Andersson-Ceder, and I. Bider, “State flow as a way of analyzing business processes--case studies,” *Logistics Information Management*, vol. 15, no. 1, pp. 34–45, 2002. Available: https://doi.org/10.1108/09576050210412657

[27] T. Andersson, I. Bider, and R. Svensson, “Aligning people to business processes experience report,” *Software Process: Improvement and Practice*, vol. 10, no. 4, pp. 403–413, 2005. Available: https://doi.org/10.1002/spip.243

[28] IbisSoft. “iPB Reference Manual,” 2009 [Online]. http://docs.ibissoft.se/node/3

[29] I, Bider, P, Johannesson, and E, Perjons, “In Search of the Holy Grail: Integrating social software with BPM. Experience report,” in *Enterprise, Business-Process and Information Systems* Modeling, *LNBIP,* vol. 50, pp. 1–13, 2010. Available: https://doi.org/10.1007/978-3-642-13051-9_1

[30] A. R. Hevner, S.T. March, and J. Park, “Design Science in Information Systems Research,” *MIS Quarterly*, vol. 18, no. 1, pp. 75–105, 2004. Available: https://doi.org/10.2307/25148625

[31] K. Peffers, T. Tuunanen, M.A. Rothenberger, and S. Chatterjee, “A Design Science Research Methodology for Information Systems Research,” *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–78, 2007. Available: https://doi.org/10.2753/MIS0742-1222240302

[32] R.L. Baskerville, J. Pries-Heje, and J. Venable, "Soft Design Science Methodology," in *DERIST 2009*, pp. 1–11, 2009. Available: https://doi.org/10.1145/1555619.1555631

[33] J. Anderson, B. Donnellan, and A.R. Hevner, "Exploring the Relationship between Design Science Research and Innovation: A Case Study of Innovation at Chevron," in *Communications in Computer and Information Science, 286*, pp. 116–131, 2012. Available: https://doi.org/10.1007/978-3-642-33681-2_10

[34] M.K. Sein, O. Henfridsson, S. Purao, M. Rossi, and R. Lindgren, "Action Design Research," *MIS Quarterly*, vol. 35, no. 1, pp. 37–56, 2011. Available: https://doi.org/10.2307/23043488

[35] I. Bider, P. Johannesson, and E. Perjons, "Design science research as movement between individual and generic situation-problem-solution spaces," in *Organizational Systems. An Interdisciplinary Discourse*, Springer, pp. 35–61, 2013. Available: https://doi.org/10.1007/978-3-642-33371-2_3

[36] IBM, "IBM Forms Experience Builder," 2014. [Online]. http://www-03.ibm.com/software/products/en/ibmformexpebuil.

[37] G.I. Gomez, "Adaptability, Architecture & CX: The Bizagi Way," 2015 [Online]. http://bpm2015.q-e.at/keynote-gomez/.

[38] J. Biggs and C. Tang, "Teaching for Quality Learning at University," 4th ed.: Open University Press, 2011.

[39] R. Baskerville and J. Pries-Heje, "Discovering the significance of scientific design practice: new science wrapped in old science?" in *ADWI-2013: The 2nd international SIG Prag workshop on IT Artefact Design & Workpractice Improvement*, Tilburg, Netherlands, 2013 [Online]. http://www.vits.org/adwi2013/RBaskerville_JPries-Heje-ADWI2013.pdf.

[40] I. Bider, "Towards Process Improvement for Case Management. An Outline Based on Viable System Model and an Example of Organizing Scientific Events," in *BPM 2015 Workshops, LNBIP 256*, Innsbruck, Austria, pp. 96–107, 2016. Available: https://doi.org/10.1007/978-3-319-42887-1_9

Appendix A. An Example of an *iPB*-Based Process Model

In this appendix, we present an example of a process and its *iPB*-based executable model to which we refer in Section 6 and Section 7 of the article. We decided to abstain from using a real case from *BBiC* project, because of it being very specific for the social offices of municipalities in Sweden, and rather two complicated for using in illustrations. Instead we have chosen an example that is easy to understand for the reader. Though no real system has been built for the process used in the example, an executable model for it has been built and tested [8]. The presentation of the example consists of informal description of the process, and its model created with the help of *iPB*. Alongside with the presentation of the process model, we also explain the ideas of business process modeling built-in in *iPB*.

A.1 A Process Example – Course Preparation Process

A business process that we use for illustration purposes is the process of preparing and giving a course round at a university. The course round can be the first round of a completely new course, or a round for a regularly given course. This business process has been chosen based on the following two reasons:

1. It is a process in which the authors are personally involved.
2. The process as it is run in our department would be difficult to support with a workflow based BPS system.

Below, we present an informal overview of the chosen process. The description below is intentionally made to look like a sequential description of phases, which will not be followed when building a data-centric process model. Namely, we identify five major phases in the course process:

1. *Planning course* includes a number of meetings with involved teachers to decide which teaching and learning activities to carry out during the course as well as their sequence. The phase also includes selecting a course book, and deciding on and producing the course material. Moreover, the exam has to be designed, at least a draft of it, to govern the teachers in their teaching and learning activities, following the principles of constructive alignment [38]. Finally, an evaluation form has to be designed, which will be filled out by the students when the course has ended.
2. *Schedule course* consists of composing a schedule with dates, times and locations for the lectures, lessons and seminar as well as a date, time and location for the written exam and other teaching and learning activities. The phase includes a number of interactions between the teacher responsible for the course and the person responsible for scheduling courses in the department.
3. *Publishing course material* consists in printing course material. The printing is done by the person responsible for printing.
4. *Learning and teaching* includes a number of teaching and learning activities, such as lectures, lessons (tutorial) and seminars, managing assignments and carrying out exams. It also includes receiving feedback from students and involved teachers for each teaching and learning activity to be used for adjusting the course immediately, if needed, or before the next round.
5. *Evaluation* includes the students evaluating the course after the end of the course. The phase also includes analysis of the evaluation carried out by the teacher responsible for the course.

Though the process is split in a number of phases, the latter are not always executed in a sequence, but can run iteratively or in parallel, which makes the process non-workflowable according to tradicional workflow. The sequential execution, though it might be considered as preferable, works well only in limited cases, e.g., for a repetition of an already established course when it is given to the students with a known level of preparedness. In Figure A1, we present a version of relatively sequential execution of activities in the course process in the format of UML activity diagrams so it can be compared with a data-centric model presented in the next section[*].

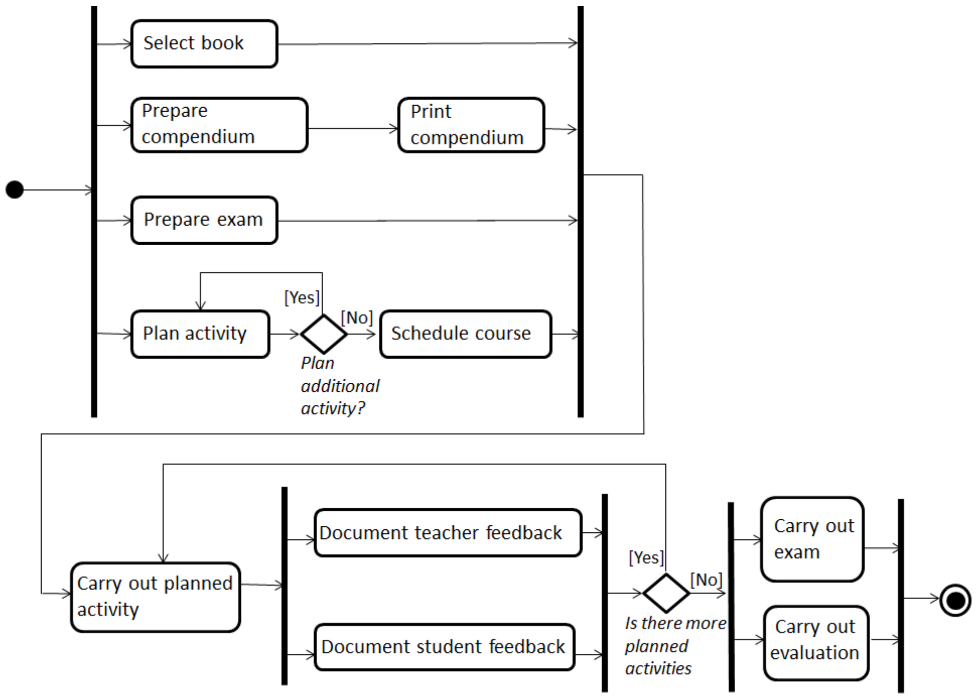


Figure A1. A possible workflow model for the course process

The workflow-based way of preparing and giving a course was predominant in the past. However, according to the latest research on university teaching [38], the landscape of university teaching has changed in the last few decades. The main characteristic of the new landscape is the diversity of students, which requires adjusting the course to their level and background. This is especially true when giving a completely new course or an old course to a different or unknown audience. In the latter cases, there might be a need for changing the teaching/learning activities, material used in them, teachers, or the schedule when the course has already been started. In the "worst" case, the preparation before the course can be limited to devising an introductory lecture or/and a test to identify the level of preparedness of the students. Everything else is designed on the fly during the course.

The worst case described above corresponds to the high-level of uncertainty discussed in Section 2.1 of the article, therefore the course preparation process could be considered as a good candidate for being supported by an ACM system. Note that in the authors' own department, based on the practice of which we designed an *iPB*-model presented in Section A.2 and Section A.3[†], many courses are prepared and run in non-sequential manner.

A.2 Business Process Modeling Built in *iPB*

As was noted in Section 4.2 of the article, *iPB* was designed as a tool for developing business process support systems/services for loosely structured business processes. *iPB* consists of two components – Design studio, and Runtime environment. *Design studio* is used for building a

[*] Note that the diagram is not complete, as some activities, e.g. "Prepare evaluation form" are left out.

[†] Details on how and for what purpose this model has been built can be found in I. Bider, E. Perjons, and Z. RiazDar, "Using Data-Centric Business Process Modeling for Discovering Requirements for Business Process Support Systems: Experience Report," in *BPMDS 2013 and EMMSAD 2013, LNBIP 147*, Valencia, Spain, 2013, pp. 63–77. Available: https://doi.org/10.1007/978-3-642-38484-4_6

process model, while *Runtime environment* interprets (executes) this model while helping process participants to run their process instances/cases.

Process modeling in *iPB* is based on four *main* abstract concepts: *Process map*, *Process step*, *Step form*, *Form field* (additional concepts are described later). The basic relationships between these *main* 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 two dimensional matrix underneath the upper row called *flow-dependent* steps, see Figure A2[*].
- Each *process step* in a *process map* has a *step form* attached to it.
- A *step form* consists of a collection of named *form fields* arranged in a two-dimensional matrix called *form layout*, see Figure A3. 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, such as uploaded file, journal, person, organization. In addition, field collections that belong to different step forms can intersect. The latter is done by defining fields in one form, and then referring to them in other forms.

A process map with underlying step forms defines a kind of a database scheme for the given process type. This scheme is then used by the runtime system for "creating" an “instance database” for each new instance/case of the process defined by this process map. The runtime system also interprets step forms as web forms for inputting, viewing and changing information in the instance database, see Figure A4. The process map is used for creating an *instance map*, see Figure A5, that serves as a mechanism for user navigation through the web forms. To open a web-form, the user just clicks on the step in the *instance map*. As we can see from Figure A4, some steps are allowed to have multiple forms at runtime (see the tabs for each lecture in Figure A4). In this case, besides filling a form, the user can also add and fill new exemplars of the form, or delete existing exemplars. Besides filling a form and saving it, the user can mark the form as write-protected, or close the step. In the latter case, the step will be marked as finished in the instance map, see steps with the blue background in Figure A5.

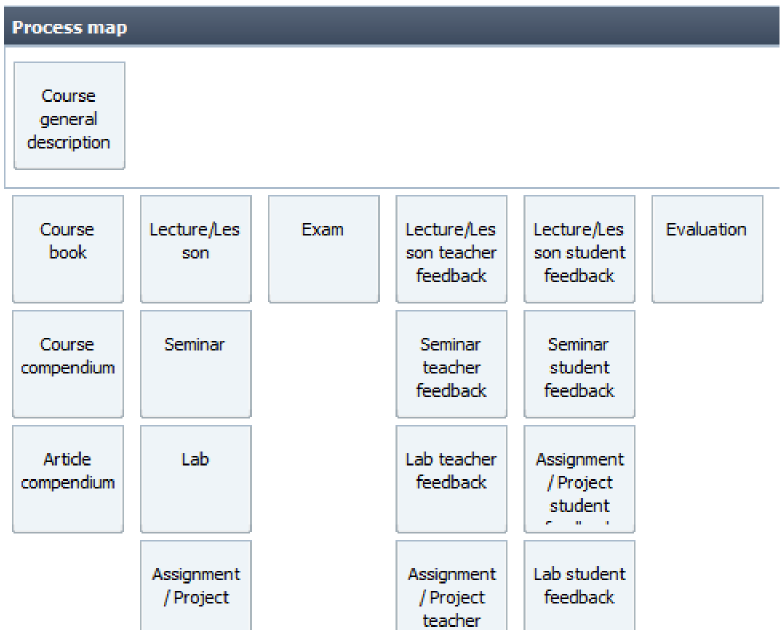
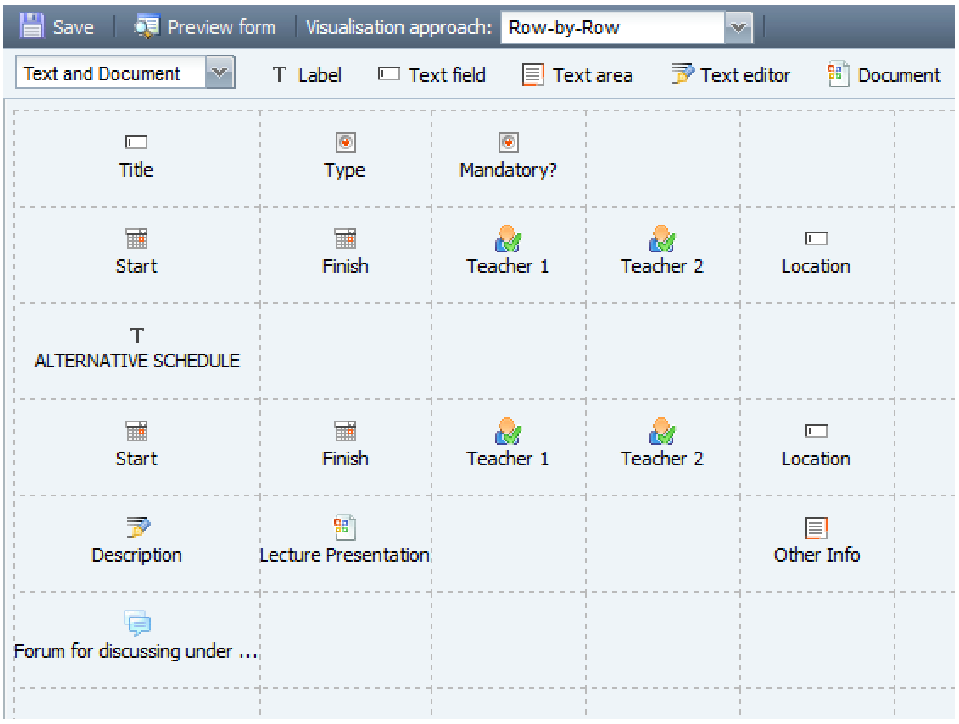


Figure A2. A process map

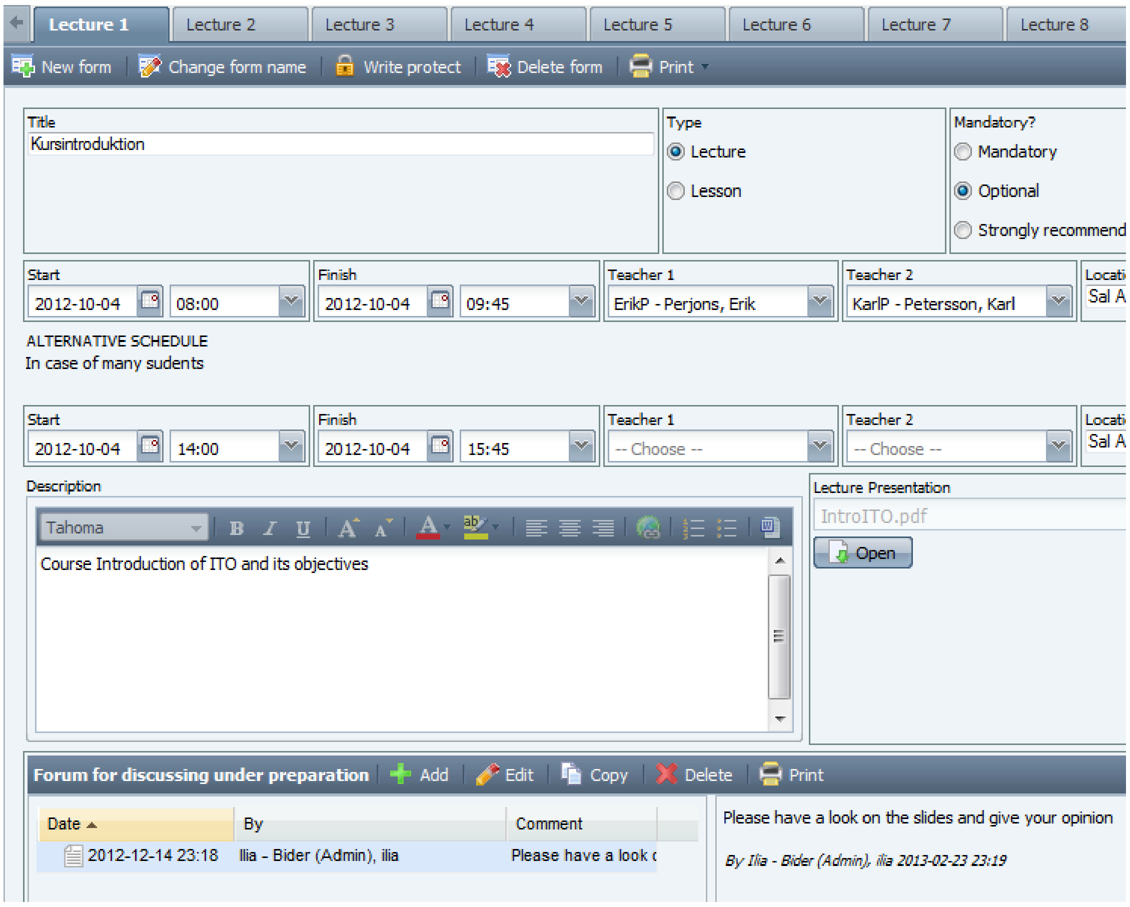
[*] All figures that illustrate the main concepts of *iPB* are taken from the course process model that will be explained in details in the next section.


Save | Preview form | Visualisation approach: Row-by-Row

Text and Document | Label | Text field | Text area | Text editor | Document

Title	Type	Mandatory?		
Start	Finish	Teacher 1	Teacher 2	Location
ALTERNATIVE SCHEDULE				
Start	Finish	Teacher 1	Teacher 2	Location
Description	Lecture Presentation			Other Info
Forum for discussing under …				

Figure A3. Step form for step Lectures/Lesson preparation from Figure A2


Lecture 1 | Lecture 2 | Lecture 3 | Lecture 4 | Lecture 5 | Lecture 6 | Lecture 7 | Lecture 8

New form | Change form name | Write protect | Delete form | Print

Title: Kursintroduktion

Type: [✓ Lecture] Lesson

Mandatory?: Mandatory [✓ Optional] Strongly recommend

Start	Finish	Teacher 1	Teacher 2	Locati
2012-10-04 08:00	2012-10-04 09:45	ErikP - Perjons, Erik	KarlP - Petersson, Karl	Sal A

ALTERNATIVE SCHEDULE
In case of many sudents

Start	Finish	Teacher 1	Teacher 2	Locati
2012-10-04 14:00	2012-10-04 15:45	-- Choose --	-- Choose --	Sal A

Description: Course Introduction of ITO and its objectives

Lecture Presentation: IntroITO.pdf | Open

Forum for discussing under preparation | Add | Edit | Copy | Delete | Print

Date	By	Comment
2012-12-14 23:18	Ilia - Bider (Admin), ilia	Please have a look c

Please have a look on the slides and give your opinion

By Ilia - Bider (Admin), ilia 2013-02-23 23:19

Figure A4. Step form from Figure A3 as a web-form

Besides the *main* concepts described above, there are some auxiliary concepts implemented in *iPB* to provide additional capabilities, e.g.:

- *Step properties*, such as permission to have more than one form (as in Figures A4 and A5).
- *Mandatory fields* that establish when data in a step form can be saved or made write-protected. Such rules are expressed via defining some fields on the form as mandatory for save, or close.
- *Step start* rules that control whether the user can open a particular step form based on the states of completion of other forms. Steps that cannot yet be clicked on are colored grey, see Figure A5. Such rules are specified with the help of a square matrix where both rows and columns correspond to the steps defined in the process, see Figure A6. In this matrix, the content of a cell can, for instance, determine that the row step can be started only after the column step has been completed (blue color in Figure A5), or started (green color in Figure A5).
- *Step synchronization rules* for steps with multiple forms. When step *A* is synchronized with step *B*, a form in step *A* cannot exist without being connected to a corresponding form in step *B*. For instance, step *Lecture/Lesson teacher feedback* in Figures A2 and A5 is synchronized with step *Lecture/Lesson*, thus each feedback form is connected to the corresponding lecture/lesson form. These rules are defined by a square matrix where a checked box means that the row step is synchronized with the column step, see Figure A7.
- *User profiles* that specify categories of users and their access rights to view and change information in each step form.
- Visual properties assigned to the fields that define their look and feel at runtime. This part of *iPB* is not related to process modeling, but to system development.
- *Notifications* send to the users when they become participants of the process or a particular process step; or when data has been changed in a form of a step for which they are registered as participants.

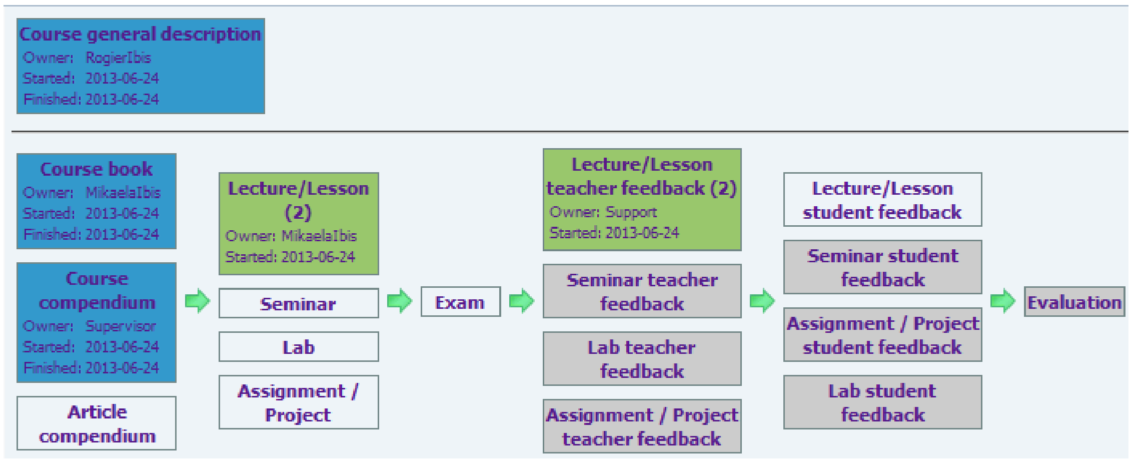
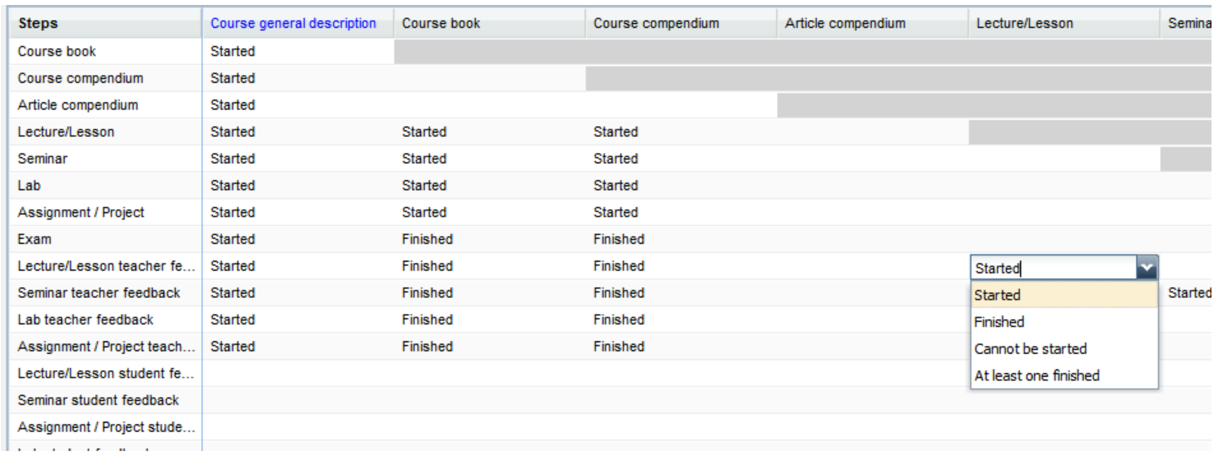


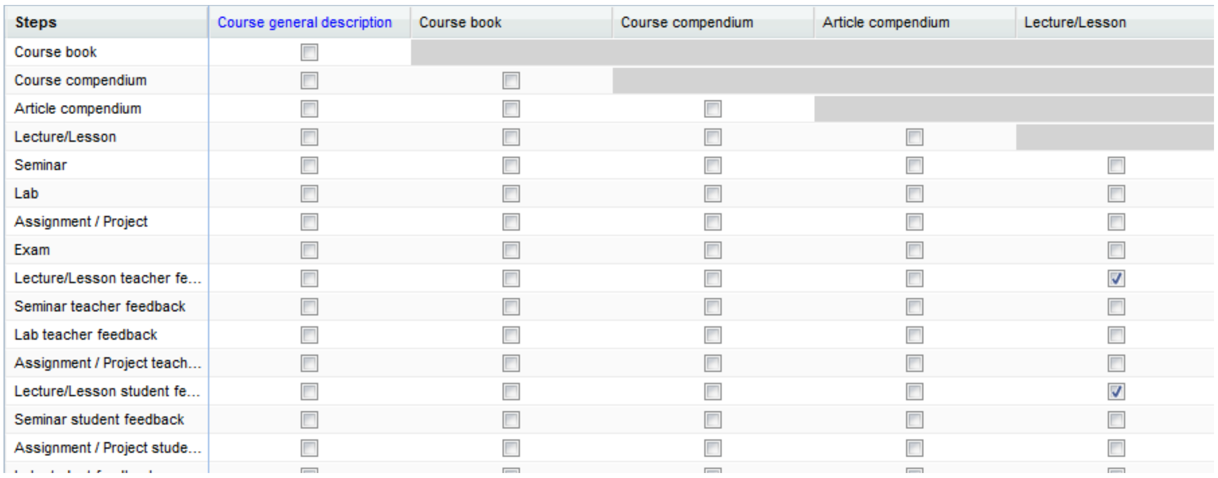
Figure A5. Instance process map from Figure A2 as a user navigation mechanism



Steps	Course general description	Course book	Course compendium	Article compendium	Lecture/Lesson	Semina
Course book	Started					
Course compendium	Started					
Article compendium	Started					
Lecture/Lesson	Started	Started	Started			
Seminar	Started	Started	Started			
Lab	Started	Started	Started			
Assignment / Project	Started	Started	Started			
Exam	Started	Finished	Finished			
Lecture/Lesson teacher fe...	Started	Finished	Finished		Started	
Seminar teacher feedback	Started	Finished	Finished			Started
Lab teacher feedback	Started	Finished	Finished			
Assignment / Project teach...	Started	Finished	Finished			
Lecture/Lesson student fe...						
Seminar student feedback						
Assignment / Project stude...						

Started
Finished
Cannot be started
At least one finished

Figure A6. Step start rules



Steps	Course general description	Course book	Course compendium	Article compendium	Lecture/Lesson
Course book	☐				
Course compendium	☐	☐			
Article compendium	☐	☐	☐		
Lecture/Lesson	☐	☐	☐	☐	
Seminar	☐	☐	☐	☐	☐
Lab	☐	☐	☐	☐	☐
Assignment / Project	☐	☐	☐	☐	☐
Exam	☐	☐	☐	☐	☐
Lecture/Lesson teacher fe...	☐	☐	☐	☐	☑
Seminar teacher feedback	☐	☐	☐	☐	☐
Lab teacher feedback	☐	☐	☐	☐	☐
Assignment / Project teach...	☐	☐	☐	☐	☐
Lecture/Lesson student fe...	☐	☐	☐	☐	☑
Seminar student feedback	☐	☐	☐	☐	☐
Assignment / Project stude...	☐	☐	☐	☐	☐

Figure A7. Synchronization rules

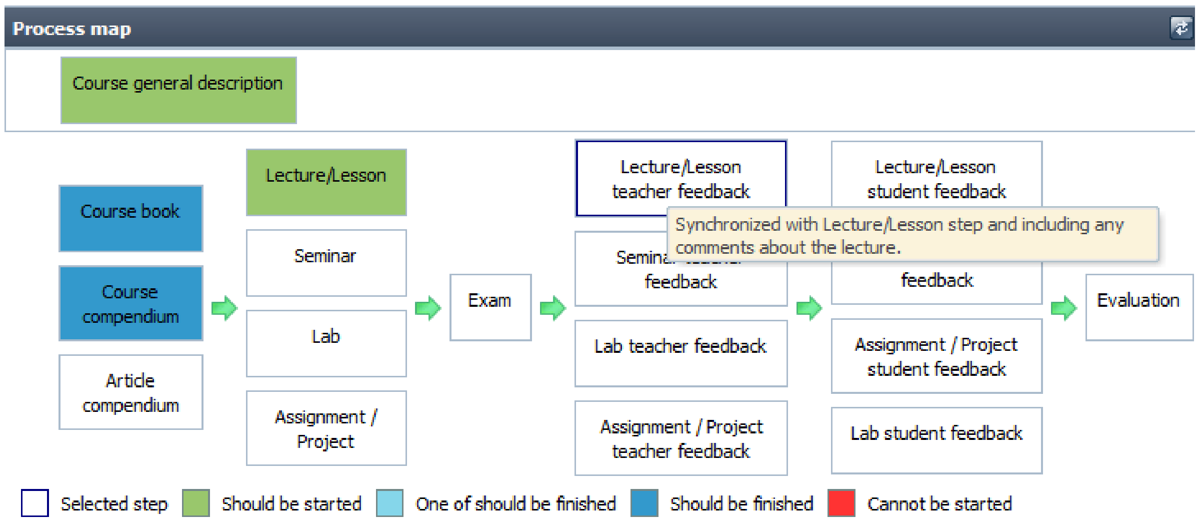
Though *iPB* is built on the few basic concepts, it is powerful enough for representing various aspects/perspectives of business processes. Below, we give some examples of how different aspects of processes are represented when building a process model in *iPB*:

- *Structure of data/information* created and utilized in the process is defined through creating step forms.
- *Data/Information flow* in the process is defined by including references to the form fields from the previous step forms into the form where this information is used.
- *Participant collaboration* in the frame of process instances is defined by inserting complex fields of the type *Journal* in the step form; see the field "Forum for discussing …" at the bottom of Figures A3 and A4. Users working with the same form add records to the journal registering questions, answers, comments, etc. relevant to the particular step.
- *Tasks/activities* included in the process and *restrictions* on the order in which they can be completed are specified in the following manner. The concept of *step* is used for representing larger chunks of work– work-packages; if smaller tasks are needed they are represented in the step forms as checklists in the way accepted for case management/adaptive case management systems. The order of steps is represented by (a) process layout that gives a recommended sequence of steps (from left to right and from top to bottom), and (b) by business rules that restrict the possibility to "jump over" some steps. On the lower level, a possibility of making task checklists mandatory gives an opportunity to prevent finishing the step to which they belong until all tasks have been completed. The latter can prevent starting the next step if it is forbidden by business rules that require finishing the previous step first.

- *User roles* – categories of users engaged in the process and limitations on the data/information they can access (read, write, or modify). They are defined with the help of user profiles.

A.3 The *iPB*-Based Course Model

The *iPB* process map for the course process from Section A.1 is presented in Figures A2, A5 and A8. Figure A2 presents the map in the design studio in the form convenient for model developers. Figure A8 presents the map in the form convenient for discussion with business people. Figure A5 presents a process instance map in the form used for navigation in runtime. Figures A3 and A4 demonstrate a step form typical for this process, more exactly the form for step "Lecture/Lesson". Figure A3 presents the step form as viewed in the design studio, and Figure A4 – at runtime.



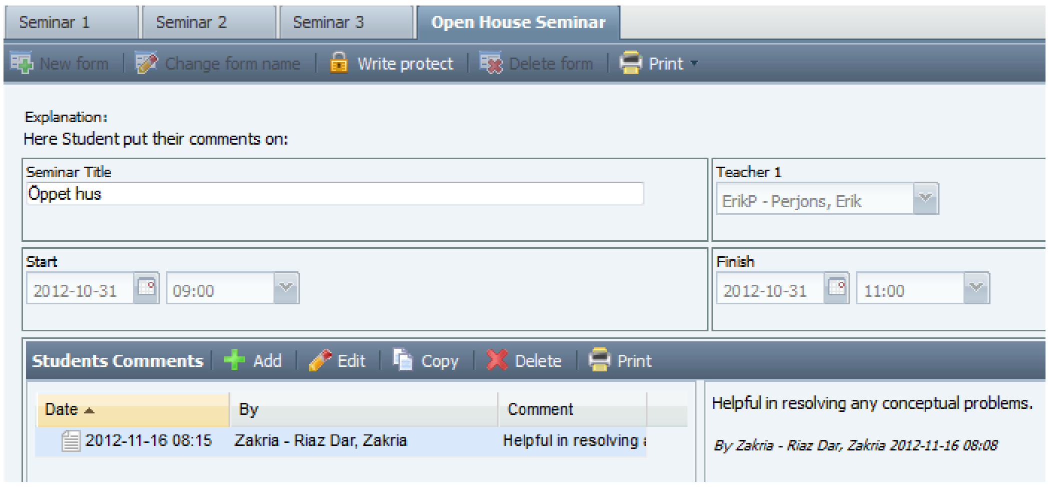
Synchronized with Lecture/Lesson step and including any comments about the lecture.

Figure A8. The process map of course preparation in the form suitable for discussion with stakeholders

As follows from Figure A8, the process map for the course process is clearly divided in two parts: (1) preparation of the course to the left and including *Exam*, and (2) giving the course. The first part is structured according to the artifacts that need to be prepared for the course, each component being represented as a step in the model (see Figure A8). This part includes (a) general components: *Course general description* (mandatory), *Course book* (optional), *Course compendium* (mandatory), *Article compendium* (optional), *Exam* (questions, and instructions for assessment), and (b) specific components that correspond to teaching and learning activities: *Lecture/Lesson, Seminar, Lab,* and *Assignment/Project*.

Each step has a step form attached to it that defines the data structure for this step. A typical form, the one for step *Lecture/Lesson*, is presented in Figure A3 (the design studio view), and Figure A4 (runtime view). It includes the descriptive fields, like: *Title*, *Type*, *Description*, *Lecture presentation*, and the administrative fields, like: *Teacher*, *Start*, *Finish, and Location*. In addition, it has a journal like widget named *Forum for discussing under preparation* were the teachers preparing this teaching/learning activity can leave comments for themselves and for each other. Note that the form in Figure A4 allows multiple forms at runtime (see tabs) – one for each teaching/learning activity. Multiple forms are also allowed for all other steps except *Course general description*, *Course compendium*, *Exam*, and *Evaluation*.

The second part of the model, all steps on the right of step *Exam*, deals exclusively with getting feedback for course activities from the teachers and students. More exactly, the steps that end with "teacher feedback" are aimed at gathering feedback from the teachers immediately after the corresponding teaching/learning activity. The steps that end with "student feedback" are aimed at gathering feedback from the students immediately after the corresponding teaching/learning activity. An example of a feedback form is presented in Figure A9. In it, journal field *Students Comments* is designated for recording feedback from students attending the course.



Explanation:
Here Student put their comments on:

Seminar Title: Öppet hus
Teacher 1: ErikP - Perjons, Erik
Start: 2012-10-31 09:00
Finish: 2012-10-31 11:00

Date	By	Comment
2012-11-16 08:15	Zakria - Riaz Dar, Zakria	Helpful in resolving a

Helpful in resolving any conceptual problems.

By Zakria - Riaz Dar, Zakria 2012-11-16 08:08

Figure A9. Step form for Student comments on Seminars

The feedback steps are synchronized with corresponding preparation steps. When step *A* is synchronized with step *B*, the former will automatically get the same number of forms as the latter. If there are references on the *A*'s form to the fields on the *B*'s form, then the references are also synchronized so that they show the contents from the corresponding forms. For instance, fields *Seminar title*, *Teacher 1*, *Start* and *Finish* in Figure A9 are references to the fields on step form *Seminar*, and show their content from the form *Open House Seminar*.

The order of steps in the process model given via the layout represents only the recommended order. We found that it is almost impossible to establish a strict order to which all teachers in our department would abide. Although it is highly recommended to have all material ready before the start of the course round, it happen that changes in some materials and in the schedule are done very late, sometimes after the course has already been started. This is especially true when the course is given for the first time. Step start rules where used in this model mainly in their “soft” form – some steps cannot be started unless some others have already been started. The effects of these rules at runtime can be seen in Figure A5 – in which steps with gray background cannot be started.

Appendix B. A Fragment of Formalization of the Basic Data-Centered Business Process Model

B.1 Basic Notions – Scheme, State and Trajectory

Below, we introduce a set of basic definitions to be used for defining a data-centric business process model. The objective of this section is to introduce the minimum number of notions that allows defining sequences of states among which the process model can differentiate the valid ones. The notions are illustrated on fragments of a model for the course preparation process from Appendix A.

A *variable* is a pair $v = <n, O>$ that consists of a *variable name n* and a set of *variable values O*, which is called the domain of this variable. Values can be simple, like *integer*, *string*, or complex, like *Person* that is a vector of simple values e.g. *last name*, *first name*, *year of birth*, etc. Denote a set of all possible variables in which the name of a variable is never repeated as V (we consider that variables in V have unique names).

A *variable assignment* is a pair $a = <n, o>$ where n is a name of a variable $<n, O> \in V$, and $o \in O$. Denote the name of the variable in assignment a as $name(a)$, and the value of assignment as $value(a)$.

A *group* is a nonempty finite set of variables $g = \{v_1, \ldots, v_m\}$ from V, $m \geq 1$. Denote the set of all possible groups as G (G is a powerset of set V).

To illustrate the formal definitions in this section we will use three groups from the realm of the course preparation process of Appendix A: *Book* (course book), *Lecture*, *StudentFeedback*. They can be defined as follows:

```
Book = {bookTitle, bookAuthor, yearOfPublication, bookPublisher}.
Lecture = {lectureTitle, lectureTeacher, lecturePresentationSlides, lectureRoom,
          lectureDate}.
StudentFeedback ={studentFeedbackName, studentFeedbackContent}.
```

Let Id be a countable set that we call a set of identifiers id. A *record r* for group $g = \{v_1, \ldots, v_m\}$, where $v_i = <n_i, O_i>$, is an identified nonempty set of assignments for variables from g in the form of a pair:

$r = <id, \{a_1, \ldots, a_k\}>$, $id \in Id$, $1 \leq k \leq m$, $a_i = <n_j, o_j>$, where $< n_j, O_j> \in g$ and $o_j \in O_j$, and for each pair $a_i\ a_j \in r$, $name(a_i) \neq name(a_j)$.

Denote the group for which r is a record as $group(r)$. Denote the identifier of record r as $ident(r)$. Denote also all possible records for group g as $R(g)$.

Examples of records[*]:

```
r₁ = <1, {bookTitle = "Database Design", bookAuthor = "Anna Alm", yearOfPublication =
    "2010", bookPublisher = "Springer"}>,
r₂ = <2, {lectureTitle="SQL", lectureTeacher="Ilia Bider",
    lecturePresentationSlides="sql.ppt", lectureRoom="501", lectureDate=
    "2013-11-05"}>.
```

A *process scheme* is a finite set of nonintersecting groups of variables $S = \{g_1, \ldots, g_m\}$, $S \subset G$, and for each pair $g_i, g_j \in S$, $gi \cap gj = \emptyset$.

An example of a process scheme is $S = \{Book, Lecture, StudentFeedback\}$.

A finite set st of uniquely identified records for groups that belong to the scheme S is called a *state*: $st = \{r \mid group(r) \in S, ident(r_1) \neq ident(r_2), r_1, r_2 \in st\}$. Denote the set of all possible states for scheme S as $St(S)$.

[*] In all examples we will use the set of all positive integers for identifiers.

An example of a state with one record:

```
st = {<1, {bookTitle = “Database Design”, bookAuthor = “Anna Alm”, yearOfPublication
    = “2010”, bookPublisher = “Springer”}>}.
```

This state shows that a book for a course has been chosen, but no lectures have been defined so far.

Let *T* be an ordered set of timestamps given with some precision, e.g. minutes, seconds or milliseconds – time axis. Then a sequence of pairs $tj = <<t_1,\varnothing>, <t_2, st_2>, \ldots, <t_m, st_m>>$ where $t_i \in T$, $st_i \in St(S)$, $i= 1, \ldots, m$, $t_1 < t_2 < \ldots < t_m$ and $st_i \neq st_{i+1}$ for $i= 1, \ldots, m-1$, is called a trajectory of a process with scheme *S*. Denote the set of all trajectories for scheme *S* as *Tj*(*S*).

An example of a trajectory with three states defined (the rest is marked as …):

```
tj = <<2013-10-05 20:15, ∅>, <2013-10-13 18:30, {<1, {bookTitle = “Database Design”,
      bookAuthor = “Anna Alm”, yearOfPublication = “2010”, bookPublisher =
      “Springer”}>}>,<2013-10-14 10:05, {<1, {bookTitle = “Database Design”,
      bookAuthor = “Anna Alm”, yearOfPublication = “2010”, bookPublisher =
      “Springer”}>, <2, {lectureTitle=”SQL”, lectureTeacher=”Ilia Bider”,
      lecturePresentationSlides=”sql.ppt”, lectureRoom=”501”, lectureDate=”2013-11-
      05”}>}>, …>
```

In this example, we start with an empty database, then add a book for the course, and then add a lecture description.

Note that two sequences that can be converted to one another with the help of a mapping μ: $Id \rightarrow Id$ are considered equivalent for all practical purposes.

B.2 Extended State and Extended Trajectory

In the previous section, we introduced a number of notions to define the idea of a process instance as a time-stamped sequence of database states called a trajectory. The definitions in Section A.1 of Appendix A have no mechanisms for differentiating between trajectories; all trajectories are equally valid from the database point of view. Though, we have introduced the notion of group to tie some variables to each-other, this on its own does not constitute any limitations. In this section, we introduce additional notions aimed at creating convenient means for differentiating trajectories, so that some of them can be considered as valid. The concepts introduced in this section extend the notions of state and trajectory so that rules can be added to the scheme for differentiating valid and invalid trajectories. The extension of the scheme itself is discussed in Section A.3 of Appendix A.

Let *S* be a scheme, and *st* is a state, $st \in St(S)$. Let *lr* be a subset of records from *st* ($lr \subseteq st$), and *lg* be a subset of groups from *S* ($lg \subseteq S$). Furthermore, let *d* be an asymmetric binary relationship defined on *st* that does not have loops, which is called *record dependencies*. Then a quadruple $est = <st, lr, lg, d>$ is called an extended state for scheme *S*. A record *r* from *st* that belongs to *lr* ($r \in lr$) is said to be locked. A group *g* from *S* that belongs to *lg* ($g \in lg$) is said to be locked. If $<r_1,r_2> \in d$ then r_1 is called a parent of record r_2 in respect to *d* while r_2 is called a child of r_1 in respect to *d*.

Denote all possible extended states for scheme *S* as *ESt*(*S*). A sequence $etj = <<t_1,\varnothing, \varnothing, lg_1,\varnothing>, <t_2, st_2, lr_2, lg_2, d_2>, \ldots, <t_m, st_m, st_m, S, d_m>>$ where $t_i \in T$, $<st_i, lr_i, lg_i, d_i> \in ESt(S)$, $i= 1, \ldots, m$, $t_1 < t_2 < \ldots < t_m,$ and $<st_i, lr_i, lg_i, d_i>, \neq <st_{i+1}, lr_{i+1}, lg_{i+1}, d_{i+1}>$, for $i= 1, \ldots, m-1$ is called an extended trajectory of a process with scheme *S*. Denote the set of all extended trajectories for scheme *S* as *ETj*(*S*).

For the extended trajectories, it is possible to introduce general rules of validness even before defining extensions to the scheme. Rules of validness related to locked records and groups are as follows. An extended trajectory $etj = <<t_1,\varnothing, \varnothing, lg_1,\varnothing>, <t_2, st_2, lr_2, lg_2, d_2>, \ldots, <t_m, st_m, st_m, S, d_m>> >$ is considered to be valid in scheme *S* only if:

- for any $<st_i, lr_i, lg_i, d_i>$, $i = 1,\ldots m-1$, and record *r* such that $r \in st_i$ and $r \in lr_i$: $r \in st_{i+1}$, i.e. a locked record cannot be modified until it is unlocked, and

- for any $<st_i, lr_i, lg_i, d_i>$, $i = 1,\ldots m\text{-}1$, and group g from lg_i ($g \in lg_i$): $st_i \cap R(g) \subseteq lr_i$ and $st_i \cap R(g) = st_{i+1} \cap R(g)$, i.e. all records in a locked group are locked, and no records of the type represented by the group can be added or deleted until the group is unlocked.

Locking records and groups allow controlling the order in which records are added to or changed in the database. Locking groups has two different meanings. If the group is empty and locked, it means that conditions to start adding records have not been yet fulfilled[*]. If the group is non-empty and locked, it means that adding information related to this group has been completed, and no new information is expected to be coming.

Note that in the last state of the trajectory, which is called a final state, all groups and records are locked, thus it runs as $<t_m, st_m, st_m, S, d_m>$. The latter indicates the fact that no more information can be added to the finished process instance.

An extended version of the example trajectory from Section B.1 can look like:

```
etj = <<2013-10-05 20:15, ∅, ∅, {Lecture, StudentFeedback}, ∅ >,
      <2013-10-13 18:30, {<1, {bookTitle = "Database Design", bookAuthor = "Anna
      Alm", yearOfPublication = "2010", bookPublisher = "Springer"}>}, {1}, { Book,
      StudentFeedback}, ∅>,
      <2013-10-14 10:05, {<1, {bookTitle = "Database Design", bookAuthor = "Anna
      Alm", yearOfPublication = "2010", bookPublisher = "Springer"} , ∅ >, <2,
      {LectureTitle="SQL", LectureTeacher="Ilia Bider",
      LecturePresentationSlides="sql.ppt", LectureRoom="501", LectureDate="2013-11-
      05"}> }, {1,2}, {Book, StudentFeedback}, ∅ >, …>
```

In the first extended state of the trajectory above, groups *Lecture* and *StudentFeedback* are locked, which means that no records can be added to these groups. In the second state, the record that defines the book is added and locked, group *Lecture* becomes unlocked, but group *Book* becomes locked (group *StudentFeedback* continues to be locked). In the third element, both record for book and record for the first lecture are locked, but there is no change in the set of locked groups.

Relationship *di*, which is called *record dependencies*, is included in the extended state to show that a particular record in one group is connected to a particular record in another group. For instance, this type of connection is needed for the course preparation process to express relationships between the records that represent lectures, and feedback on them after the lectures. Without connecting a lecture record to a particular feedback record it would be impossible to know which feedback concerns which lecture.

Introduction of general rules of validness related to record dependencies can help in reducing the complexity of the model, and with it, the complexity of BPS systems that interpret them. For instance, the following rules could reduce the model complexity by introducing some restrictions on record dependecies:

- *One parent only*. For any $<st_i, lr_i, lg_i, d_i>$, $i = 1,\ldots m\text{-}1$, if $<r_1,r_2> \in d_i$ there is no $r_3 \neq r_1$, such that $<r_3,r_2> \in d_i$.
- Prohibition to change a parent. For any $<st_i, lr_i, lg_i, d_i>$, $i = 1,\ldots m\text{-}1$, if $<r_1,r_2> \in d_i$ and $r_2 \in st_{i+1}$, then $r_1 \in st_{i+1}$ and $<r_1,r_2> \in d_{i+1}$.
- *One child only*. For any $<st_i, lr_i, lg_i, d_i>$, $i = 1,\ldots m\text{-}1$, if $<r_1,r_2> \in d_i$ there is no $r_3 \neq r_2$, such that $<r_1,r_3> \in d_i$.
- Prohibition to abandon a child. For any $<st_i, lr_i, lg_i, d_i>$, $i = 1,\ldots m\text{-}1$, if $<r_1,r_2> \in d_i$ and r_1, $r_2 \in st_{i+1}$, then $<r_1,r_2> \in d_{i+1}$.

Note, however, that having the rules like above is not mandatory.

[*] They may never be fulfilled, leaving the group empty in the final state of the trajectory.

B.3 Extended Scheme

B.3.1 Motivation – Procedural Semantics

The definition of valid trajectories can be used for defining procedural semantics of a data-centric process model as a generative procedure that:

1. Starts from an empty database where some groups (are locked).
2. Adds records to non-locked groups, or change existing non-locked records in the database, and/or lock or unlock records in the database. This step can be repeated any number of times.
3. Finishing at any time when all groups are locked.

By extending the scheme with the rules that controls locking of records and establishing relations between the records, it is possible to control the order in which changes are introduced in the database. Note that here, we have no ambitions to extend the scheme with all possible means for defining valid trajectories. The intention is to show how it could be done based on analysis of our practice. Actually, only very basic rules are discussed here, however, they are powerful enough to be useful for practical purposes.

B.3.2 Definition of the Extended Scheme

Let S be a scheme then a 7-tuple $<S,Ms,Ml,N,U,fc,D>$ is called an extended scheme over S, if:

- *Ms is a* subset of variables included in the groups that belong to S, $Ms \subseteq \{v \mid v \in g,\ g \in S\}$. Variables from Ms are called *mandatory for save* variables.
- *Ml is a* subset of variables included in the groups that belong to S, $Ms \subseteq \{v \mid v \in g,\ g \in S\}$, such that $Ms \cap Ml = \emptyset$. Variables from Ms are called *mandatory for lock* variables.
- N is a subset of groups from S, $N \subseteq S$, called *non-repeatable* groups.
- U is a set of unlocking rules of the form $ur = <g,uc>$, where $g \in S$, and uc is a predicate of a form defined in Section B.3.5.
- fc is a logical expression defined in Section B.3.5 that sets conditions on the final state of the trajectory.
- D is an asymmetric binary relationship defined on S that does not have loops called group dependencies.

In the subsections below we explain each element of the extended scheme and show how it limits the set of valid extended trajectories.

B.3.3 Mandatory Variables Ms, Ml

This section explains the second and the third components of extended scheme $<S,\mathbf{Ms},\mathbf{Ml},N,U,fc,D>$.

Ms, Ml are subsets of variables that are included in the groups of S: $Ms, Ml \subseteq \{v \mid v \in g,\ g \in S\}$ and , $Ms \cap Ml = \emptyset$,. Variables in Ms are called mandatory for save variables, and variables in Ml are called mandatory for lock variables.

A record $r =<id, \{a_1, ..., a_m\}>$ for a group $g \in S$ is considered to be m-valid in respect to a set of variables M if for each variable v from g ($v \in g$) such that $v \in M$, record r contains an assignment for this variable, i.e. there is $a_i \in \{a_1, ..., a_m\}$ such that $name(a_i) = v$.

An extended trajectory $etj = <<t_1,\emptyset, \emptyset, lg_1,\emptyset>, <t_2, st_2, lr_2, lg_2, d_2>, \ldots, <t_m, st_m, lr_m, lg_m, d_m>>$ is considered to be valid in extended scheme $<S,Ms,Ml,N,U,fc,D>$ only if for any $st_{i,}$, $i = 1, \ldots, m$, and for any record $r \in st_{i,}$: r is m-valid in respect to Ms, and in addition if $r \in lr_i$: r is also m-valid in respect to Ml.

Set *Ms* sets restriction on records allowed in the database independently whether they are locked or not. In the term of procedural semantics from Section B.3.1, this is a set of rules that prohibit saving a record that does not have values for variables in *Ms*; the rule corresponds to not allowing null values in the standard relational DBMSs. Set *Ml* sets restrictions on which records can be locked. In procedural semantics, this is a prohibition to lock a record if certain variables are not defined; the rule does not have a prototype in the theory and practice of DBMS. Mandatory for lock fields are the rules that define the minimum information to be entered for the record to be considered good enough for the final state.

For our example scheme, we can define *Ms* and *Ml* as follows:

```
Ms = {bookTitle, bookAuthor, lectureTitle, lectureTeacher, studentFeedbackName,
      studentFeedbackContent} and
```

```
ML = {yearOfPublication, bookPublisher, lecturePresentationSlides, lectureRoom,
      lectureDate}.
```

Note that the definition of extended trajectory in Section B.2 requires all groups and records to be locked in the final state of the trajectory. This implies that in the final state of the trajectory, all mandatory variables both for save and for lock should have values in all records in the database.

B.3.4 Non-Repeatable Groups

This section explains the fourth component of extended scheme <*S*,*Ms*,*Ml*,***N***,*U*,*fc*,*D*>, which is a set of non-repeatable groups from *S*, $N \subseteq S$. A state $st \in St(S)$ is considered to be *n*-valid in respect to a set of groups *N* if for each group $g \in N$ there is no more than one record in state *st*.

An extended trajectory $etj = <<t_1,\varnothing, \varnothing, lg_1,\varnothing>, <t_2, st_2, lr_2, lg_2, d_2>, \ldots, <t_m, st_m, lr_m, lg_m, d_m>>$ is considered to be valid in extended scheme <*S*,*Ms*,*Ml*,*N*,*U*,*fc*,*D*> only if for all $i = 1,\ldots m$, $st_{i,}$ is *n*-valid in respect to *N*.

Continuing our simplified example, we can set *N* = {*Book*}, which means that only one book can be defined for a course.

Adding *N* to the scheme allows setting restrictions on the number of records allowed in each group. In this section, we demonstrated the simplest rule that limits the maximum number of records to one. This rule can be easily extended, for instance, by defining up to two numbers for each group: maximum number of records and minimum number of records. In procedural semantics, the maximum will prevent adding a record over the maximum allowed, while the minimum will prevent locking the group until the minimum number of records has been entered.

B.3.5 Rules for Unlocking Groups

This section explains the fifth component of extended scheme <*S*,*Ms*,*Ml*,*N*,***U***,*fc*,*D*>, which is a set of unlocking rules *U*. First, we introduce some additional definitions. Let *g* be a group from *S*, $g \in S$, denote as *Nempty*[*g*] a predicate over *St*(*S*) such that:

Nempty[*g*](*st*) = *true* if *st* contains at least one record *r* ($r \in st$) for group *g*, i.e. *group*(*r*) = *g*,
false otherwise

Denote as *Closed*[*g*] a predicate over the set of extended states *ESt*(*S*) such that:

Closed[*g*](<*st*, *lr*, *lg*, *d*>) = *true* if *Nempty*[*g*](*st*) = *true* and $g \in lg$,
false otherwise

Some examples: predicate *Nempty*[*Book*] evaluates to *true* for states in which a record for group *Book* exists, while predicate *Closed* [*Book*] evaluates to *true* for states in which a record for group *Book* exists and the group is locked.

Let $g \in S$ and *uc* is an expression that includes predicates *Nempty*[*g′*] and *Closed*[*g′*] connected by logical operators AND, OR and NOT such that $g' \neq g$ for any of the predicates that are included in *uc*. Then, pair *ur* = <*g*,*uc*> is called an unlocking rule for group *g*. Example *ur* = <*StudentFeedBack, Closed*[*Book*] AND *Nempty*[*Lecture*]>.

Denote as *uc*(*est*) a Boolean value of expression *uc* for extended state *est* = <*st, lr, lg, d*> ∈ *ESt*(*S*) obtained by (a) substituting each predicate *Nempty*[*g′*] or *Closed*[*g′*] that is included in *uc* for its value *Nempty*[*g′*](*st*) or *Closed*[*g′*](*est*), and (b) calculating the result based on the logical operators that connect the predicates. For instance, expression *Closed*[*Book*] AND *Nempty*[*Lecture*] is true for the states, in which a record for group *Book* exists, group *Book* is locked and there is at least one record for group *Lecture*.

The fifth component *U* is a set of unlocking rules <*g*,*uc*> for groups in *S* such that it contains at the maximum one rule per group, i.e. if $<g_1,uc_1>,<g_2,uc_2> \in U$ then $g_1 \neq g_2$.

An extended trajectory $etj = <<t_1,\varnothing, \varnothing, lg_1,\varnothing>, <t_2, st_2, lr_2, lg_2, d_2>, \ldots, <t_m, st_m, lr_m, lg_m, d_m>>$ is considered to be valid in extended scheme <*S*,*Ms*,*Ml*,*N*,*U*,*fc*,*D*> only if

1. $lg_1 = \{g \in S \mid$ there is $<g,uc> \in U\}$

and

2. for all $i = 2,\ldots m$, and $g \in S$ such that there is $<g,uc> \in U$ and $Nempty[g](st_i) = false$: $g \notin lg_i$ only if $uc\,(<st_i, lr_i, lg_i, di>) = true$.

In terms of procedural semantics, condition 1 above means that in the beginning of a valid trajectory all groups that have locking rules are locked. Such locked record can be unlocked only when the condition expression regulating its unlocking becomes true.

For our simplified example unlocking rules can be defined as

```
U = {<Lecture, Closed[Book] >,<StudentFeedBack, Closed[Book] AND Nempty[Lecture]>},
```

which means that *Lecture* can be unlocked only when *Book* is nonempty and locked, and *StudentFeedback* can only be unlocked when *Book* is nonempty and locked and there is at least one record for group *Lecture*. Note, that the example trajectory in Section B.2 satisfies these unlocking rules.

B.3.6 Condition on Final States

This section explains the sixth component of extended scheme <*S*,*Ms*,*Ml*,*N*,*U*,***fc***,*D*>, which is conditions on the final state of the trajectory. The conditions are defined with the help of a logical expression *fc* that includes predicates *Nempty*[*g*], $g \in S$, over *St*(*S*) connected by logical operators AND, OR and NOT. The value of *fc*(*st*) for $st \in S$ is obtained by (a) substituting each predicate *Nempty*[*g*] that is included in *fc* for its value *Nempty*[*g*](*st*), and (b) calculating the result based on the logical operators that connect the predicates.

An extended trajectory $etj = <<t_1,\varnothing, \varnothing, lg_1,\varnothing>, <t_2, st_2, lr_2, lg_2, d_2>, \ldots, <t_m, st_m, lr_m, lg_m, d_m>>$ is considered to be valid in extended scheme <*S*,*Ms*,*Ml*,*N*,*U*,*fc*,*D*> only if $fc(st_m) = true$.

Condition *fc* on the final state together with mandatory variables define the minimum information to be entered into the instance database for the process to be considered as finished. In procedural semantics, condition *fc* means that the generative procedure from Section B.3.1 cannot stop (step 3) until condition *fc* yields *true* for the current state of the database.

For our example scheme, we can define *fc*=*Nempty* (*Book*) AND *Nempty*(*Lecture*), which means that the process instance cannot be finished without defining a course book and at least one lecture.

B.3.7 Group Dependencies

This section explains the seventh component of extended scheme <*S*,*Ms*,*Ml*,*N*,*U*,*fc*,***D***>, which is group dependencies.

First, we introduce some additional definitions. Let *S* be a scheme, and *D* is an asymmetric binary relationship defined on *S* that does not have loops, i.e. its transitive closure D^+ does not include tuples of type $<g,g>$ for any $g \in S$ (i.e. $<g,g> \notin D^+$). The elements of *D* are called dependencies. When a pair of groups g_i,g_j from *S*, $g_i,g_j \in S$, is a dependency, i.e. $<g_i,g_j> \in D$, we say that g_j depends on g_i; g_j is a dependent of g_i.

Extended state *est*, $est \in ESt(S)$ and $est = <st, lr, lg, d>$ is considered to be *d*-valid in respect to *D* if :

(a) for each pair of records $r_1,r_2 \in st$, such as $< r_1,r_2> \in d$: $<\text{group}(r_1), \text{group}(r_2)> \in D$,

(b) for each *r* such that there exists $<g_i,g_j> \in D$, and $group(r) = g_j$ there exists at least one record $r^1 \in st$ such that $<r^1,r> \in d$ (existence of at least one parent if parents are possible).

An extended trajectory $etj = <<t_1,\emptyset, \emptyset, lg_1, \emptyset,>, <t_2, st_2, lr_2, lg_2, d_2>, \ldots, <t_m, st_m, st_m, S, d_m>>$ is considered to be valid in extended scheme $<S,Ms,Ml,N, fc, D>$ only if for all $i = 2,\ldots m$, $<t_i, st_i, lr_i, lg_i, d_i>$ is *d*-valid in respect to *D*. In other words, it is not allowed for two records to have a connection, if it is not prescribed by their groups having a relationship in the scheme. It is also not allowed for a record in a dependent group not to have at least one parent.

Let for our example scheme define *D* as *D* = {<*Lecture*, *SudentFeedback*>}, then extended trajectory

```
etj = <<2013-10-05 20:15, ∅, ∅, {Lecture, StudentFeedback}, ∅ >, <2013-10-13 18:30,
      {<1, {bookTitle = “Database Design”, bookAuthor = “Anna Alm”,
      yearOfPublication = “2010”, bookPublisher = “Springer”}>}, {1}, { Book,
      StudentFeedback}>,<2013-10-14 10:05, {<1, {bookTitle = “Database Design”,
      bookAuthor = “Anna Alm”, earOfPublication = “2010”, bookPublisher =
      “Springer”}>, <2, {lectureTitle=”SQL”, lectureTeacher=”Ilia Bider”,
      lecturePresentationSlides=”sql.ppt”, lectureRoom=”501”, lectureDate=”2013-11-
      05”}> }, {1,2}, {Book}, ∅ >,<2013-11-05 11:25, {<1, {bookTitle = “Database
      Design”, bookAuthor = “Anna Alm”, yearOfPublication = “2010”, bookPublisher =
      “Springer”}>, <2, {lectureTitle=”SQL”, lectureTeacher=”Ilia Bider”,
      lecturePresentationSlides=”sql.ppt”, lectureRoom=”501”, lectureDate=”2013-11-
      05”}>, <3, {studentFeedbackName=”Anna Svensson”, studentFeedbackContent=”More
      SQL example is needed”}> }, {1,2}, {Book}, {<2,3>}>, …>
```

to be valid for this scheme. Record dependency *d4* = {<2,3>} that appears in the last item of the trajectory connects records 2 (lecture) and 3 (student feedback) together.

Note that that the rules of d-validness presented above are very general, they only require that a record that can be a child of another record should be a child of some record. More strict rules could be applied when defining a *d*-validness. For instance, the following rules could be useful:

(a) a record may have only one child that belongs to a particular group,

(b) a record may have only one parent that belongs to a particular group,

(c) if a record that belongs to a particular group have a child that belongs to another group, all records in the parent group, have children in the child’s group, the groups thus becoming fully synchronized.

The rules as above could be universal for the whole model, or specific for particular dependencies and/or groups. Note that the rules discussed in this section are of the same type as more general restrictions on record dependencies described in Section B.2.

In terms of procedural semantics, a record that belongs to a dependent group can only be introduced together with its relation to its parent. Furthermore, it cannot continue to exist unless there is at least one parent. In addition, if established rules, (a)-(c) above introduce even more stringent restrictions on relationships between parent and child records.

Note that in relational databases, the concept of record dependencies is represented through the foreign keys and rules of integrity attached to them. In our formalization, there is no notion of keys, as records are uniquely identified, thus we use explicit ways of expressing the rules of record dependencies.