To be published in BPMDS 2012 proceedings, LNBIP series by Springer

Preparing for the era of cloud computing: Towards a framework for selecting business process support services

Ilia Bider[1,2], Erik Perjons[1]

[1] DSV, Stockholm University, Stockholm, Forum 100, SE-16440 Kista, Sweden
[2]IbisSoft AB, Stockholm, Box 19567, SE-10432 Stockholm, Sweden

{ilia,perjons}@dsv.su.se

Abstract. The shift to the cloud computing creates new opportunities for the IT usage in business. New standard and customizable services that do not require high initial investment allow business people to choose services to support their business activities without involving technicians. Business process solutions providers are already moving their products to the cloud offering them as services. The question arises of how a business person, e.g. a department manager, can decide on which service suits best his/her needs. The paper investigates this issue in respect to the services that provide fully customizable operational support to business processes. The paper suggests a practical framework for defining requirements based on characteristics of the process to be supported by the service. The framework determines the needs of such capabilities as process flow support, shared spaces, team collaboration, etc., based on the high-level analysis of a process in question. The framework is aimed at serving as a basis for designing a practical methodology for selecting business process support services.

Keywords: cloud computing, services, business process, workflow

1 Motivation

The current shift to the cloud computing is the fourth revolution in the application of computer technology to business that the authors have observed: counting the shift to mini-computers as the first revolution, shift to the personal computers as the second, and shift to the Web as the third one. Each technological revolution extended the usage of IT technology to the new areas, and gave much broader assortments of systems to business people. Each revolution also resulted in higher involvement of business people in systems development due to appearing easy to use tools, like dBase, Access, Dreamweaver, and user-centered methodologies, e.g. agile development.

The shift to the cloud computing is to continue the trend above. Inexpensive cloud computing services ready for deployment at the moment notice will allow business people to choose services they consider appropriate for them. The burden to consult

the IT department whether there is an appropriate IT-infrastructure to support a system they need will be lifted from their minds. The same applies to financial concerns; it will be possible to test a new service without high initial investments.

Business process solutions providers are already moving their products to the cloud offering them as services [1]. There already exist a number of such services, ranging from those that provide totally domain independent solutions [2] to the ones that are focused on supporting processes in a specialized domains like Salesforce [3] (CRM), or ProjectPlace [4] (Project management solutions). With growing popularity of cloud-computing, it is only expected that the number of different process support services will grow. The question arises of how a business person, e.g. a department manager, can decide on which service suits best his/her needs. The paper investigates this issue in respect to the services that provide fully customizable operational support to business processes.

Services that provide customizable process support are based on different principles; some of them are totally workflow based, others are based on case-management [5], including adaptive case management [6]. The suitability of a particular type of services depends on the process in question. Some business processes can be streamlined and optimized making workflow services to be the right match for operational support. For others, a social software service, like wiki, can be an appropriate choice. Therefore, choosing the right type of business process support (BPS) service for a particular process requires understanding this process nature. For example, employing workflow requires splitting the process in a number of predefined operations that can be ordered in a systematic way, while employing a wiki-type service does not require even to identify operations.

The goal of this paper is to suggest a framework for high-level analysis of a business process that *allows to determine requirements on a service without building a detailed process model (as details are better to investigate in terms and concepts of an already chosen service).*

The goal is achieved by splitting the process into relatively large chunks of work, called steps, and investigating relationships between them, such as input/output dependencies, possibility of parallel execution, intersecting teams, and some others. Presence or absence of particular relationships is then used for identifying requirements on a service aimed at providing support for the process.

The ideas presented in this paper have been derived from analysis of our own experience of building BPS systems (including cloud BPS services [7]) and introducing them in operational practice. Our experience in the latter shows that not all service features that theoretically could be imagined as useful, are really useful in practice, and more nuanced analysis of the needs when choosing a BPS service is required than just desiring to have everything that is technically possible. More on our experience, see, for example, [8,9].

As far as literature is concerned, we have not found a practically-oriented framework for choosing computerized operational support for business processes. However, the literature does identify the limitations on the scope of applicability of particular methods; see for example definition of workflowability in [10].

The paper is written according the following plan. In Section 2, we introduce main concepts needed for the high-level analysis. In section 3, we present our framework and show how it helps to understand the complexity of a business process that we

want to support. In Section 4, we list the capabilities that one can expect from BPS services. In section 5, we suggest guidelines for specifying which capabilities to choose based on the nature of the process in question. Section 6 discusses the results achieved and plans for the future.

2 Basic concepts

2.1 Business processes and process support services

There are many definitions of what a business process is, each of them highlighting different aspects of this phenomenon. Actually, term *business process* encompasses two concepts (which often confuses outsiders), *business process type* and *business process instance* (or case). We give both concepts the following pragmatic definitions sufficient for the issues discussed in the paper:

- *Business process type* (BPT) is a plan/template for handling business situations of a certain type
- *Business process instance/case* (BPI) is a situation (being) handled according to the plan/template

The plan/template can include information on any or a combination or all of the following:

- A situation that warrants application of the plan, i.e. triggers a new instance creation
- A goal to reach
- Sub-goals and an order in which they could/should be achieved (goal decomposition)
- Operations/actions/activities that should be completed for achieving goals/sub-goals and the order in which they should be completed (operational decomposition)
- Rules of responsibility/participation (both for sub-goals and operations)
- Rules of collaboration/communication between participants pursuing common goals/sub-goals (communication/collaboration channels)

For example, consider a situation of developing a customized software system for a particular customer. A general plan for handling this situation can be presented as a simplified flowchart in Fig. 1. To this flowchart, any number of details can be added, e.g. the first step in Fig. 1 should be carried out by requirements engineers, the second step should produce use-case diagrams, or the third step requires using Java as a programming language. The more details are added, the more rigid the process will be. For example, setting the requirement that all programming should be done in Java will force the developers using this language even in cases where it does not fit, e.g. for development of operating systems.

The plan/template can reside in any or a combination of all of the following:

- In the heads of members of staff who participate in the process instances of the given type (tacit knowledge). This knowledge guides the process participants what can/should be done or/and what is prohibited, without much thinking about it.

– As written documents, including process maps and other kind of process description (explicit knowledge) on the paper or inside a computer, e.g., in the form of web-based hypertext. These documents contain explicit instructions of what can/should be done or/and what is prohibited to do.
– In software systems/services used to support running process instances (built-in knowledge). The usage of such systems/services forces to do some actions in a certain way and/or in a certain order, or/and prohibit to do it in other ways.

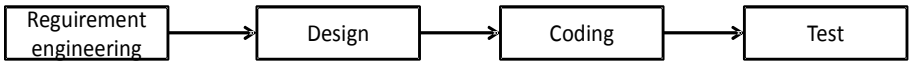

Fig.1. A plan/template for handling a situation when there is a need to develop a customized software system

In other words, the knowledge on processes can range from being completely tacit (e.g., resides in the heads of the process participants), to being totally explicit (e.g., depicted in detailed process maps).

We define *Business Process Support* (BPS) Service as a cloud service that helps the participants of a business process instance to follow the plan/template defined by the business process type. It can, for example, automate certain operations or support coordination/collaboration between the workers who participate in the same process instance. Note that using a BPS service for operational support does not imply that the whole process needs to be defined in the term of this service, and the service needs to supports all operations included in the process.

2.2 Process steps

As was pointed out in the introduction, the property of the process that we want to introduce is based on splitting the process in chunks of work called steps. To do this, we need to introduce some concepts related to the idea of process step. We start with identifying concepts related to the instances and then proceed to abstracting them to the concepts that belongs to the process type.

- Each process instance has a *goal* to reach, for example sell to a customer one or more particular products from the company's assortment for a given price.
- An instance goal, usually, can be decomposed into a number of *sub-goals* that could be pursued sequentially or in parallel.
- Pursuing a sub-goal produces *results* that are used when pursuing other sub-goals.
- Often, a sub-goal cannot be reached at once; thus there is a need for recording the *progress* achieved when pursuing this sub-goal.
- Reaching sub-goal require resources that can be divided into two categories: *passive*, like energy or money, and *active* or *agents* that perform actions, like people, or robots.
- To reach a sub-goal an agent or several agents need to perform one or more *operations/actions/activities*

As we consider process instances that belong to the same type to be similar to each other, we assume that their instance goals, results, sub-goals, are also similar. This allows us to define meta-concepts for all concepts listed above, i.e. meta-goal, meta-result, meta-sub-goals, etc. A meta-concept means having a pattern with place-holders (variables) that can be used for any instance. For example a meta-goal for the sales process can be roughly defined as a following sentence:

To sell customer X product Y for the minimum price Z having budget B for the effort

When it is clear from the context that we discuss process types rather than instances, the prefix meta- is omitted.

In the rest of the paper, we consider only "somewhat structured" processes. The minimal requirement on the process structure is that a (meta) goal of the process type is decomposable in several (meta) sub-goals, and pursuing of each sub-goal could be entrusted to different process participants or groups of participants.

Now, we are ready to introduce a new concept of *process step* as a *sub-goal with associated to it components* – results, participants, operations. The concept is applied to both instance level and type level. On the instance level, a step represents a particular sub-goal, result achieved so far, people engaged in achieving the sub-goal, and operations – the ones already completed and those that are planned. On the type level, the step represents a sub-goal template, roles of participants to be engaged, template for formatting the result. Graphically, process steps are represented as boxes (rectangles), as it is done in the systems development process in Fig. 1. This process will be used in the rest of the paper for illustrating the ideas being developed.

3 The framework

To formulate requirements on a BPS service we need to understand the complexity of a business process to be supported from different points of views. Here, we are looking at the complexity of the process itself, not the complexity of operations included in the process. A process that includes complex operations completed in a strict order without any needs for participants of the process to communicate with each other is considered to be a simple process. A process that includes relatively simple operations completed iteratively is considered to be a complex process, especially if its participants need to collaborate when completing these operations.

Our analysis of process complexity is based on investigation of relationships between the steps identified in the process. Relationships are represented with the help of a set of square matrices where both columns and rows correspond to the process steps. Intersection between a row and a column in a matrix shows a relationship between two steps. The type of content in the cells depends on the relationship in question.

3.1 Input-output

The *input-output* matrix shows dependencies of one step on the results achieved in another. A cell (*a*,*b*) in the matrix, where *a* refer to a column and *b* – to a row,

specifies what result (i.e. output) from step *a* (if any) is used as input to step *b*. In addition to the name of result, a cell can be marked with asterisk (*) which means that the result is required for step *b* to be started the first time. An example of input-output matrix for the process in Fig. 1 is presented in Table 1. In Fig. 2, the input-output dependencies are presented in a graphical form.

Table 1. Example of input-output relationships

Output / Input	**Requirements**	**Design**	**Coding**	**Test**
Requirements				
Design	*Requirements specifications			Test results
Coding		*Design specifications		Test results
Test	*Test specifications		*Code	

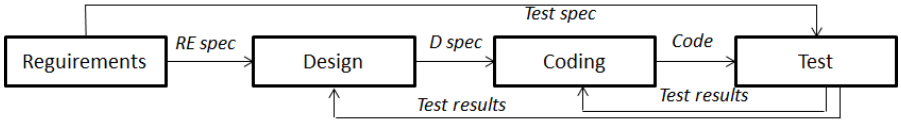

Fig 2. Graphical representation of Table 1

Presence of a symmetric pair of non-empty cells (*coding*, *test*) and (*test*, *coding*) in Table 1, points to a loop in steps execution, i.e., return from *test* to *coding* for bug fixing (Fig.2). To make all loops explicit, we can take the "transitive closure" of the *input-output* matrix creating a derived matrix called the *transitive input-output* matrix, see Table 2. In it, cell (*a*,*b*) is marked with cross *x* if (*a*,*b*) is nonempty in the *input-output* matrix, or there is a sequence of steps $c_1,\ldots,c_n$ such that cells $(a,c_1),(c_1,c_2),\ldots,(c_{n-1},c_n),(c_n,b)$ are non-empty in the *input-output* matrix. In Table 2, there are two pairs of symmetric non-empty cells (*coding*, *test*), (*test*, *coding*) and (*design*, *test*), (*test*, *design*). The second pair points to the loop of going from *design* to *test* via *coding* and returning to *design* in case the requirements are not satisfied.

Table 2. The transitive input-output matrix derived from Table 1

	Requirements	**Design**	**Coding**	**Test**
Requirements				
Design	x			x
Coding	x	x		x
Test	x	x	x	

3.2 Parallel execution

The *parallel execution* matrix shows whether two steps are allowed to be executed in parallel. If ongoing activity inside step *a* do not totally forbid carrying out activity in step *b*, then both cells (*a*,*b*) and (*b*,*a*) are marked with *x* (the matrix is symmetrical). If none of the steps can run in parallel, the parallel execution matrix will be empty.

Suppose our systems development process template provides for hard project deadlines and allows some degree of parallelism. For example, *requirements* is allowed to partially run in parallel with both *design*, and *coding*, meaning that test specifications from the requirements team are continued to be prepared while the design and coding are already in progress. Such case is depicted in Table 3.

Table 3. Example of parallel execution matrix

	Requirements	**Design**	**Coding**	**Test**
Requirements		x	x	
Design	x			
Coding	x			
Test				

In a process template that provides for very tight deadlines, all steps can be allowed to run in parallel. One starts designing as soon as basic requirements are gathered, and coding when some "implementable" part of the design has been completed. Such course might require a lot of re-doing, but it could be the only one possible if there is no possibility to extend the project length. This approach may succeed provided that the systems development team is experienced and accustomed to work in such a fashion.

3.3 Parallel dependencies

By combining the *input-output* matrix with *parallel execution* matrix we can get a new view on complexity of a business process. Table 4 is produced by merging Tables 1 and 3 according to a simple rule: cell (*a*,*b*) get crossed in the new table only if the cell is not empty in both input-output matrix and parallel execution matrix. We will refer to the merged matrix as to *parallel dependencies* matrix. The cross in a cell (*a*,*b*) in this matrix means that steps *a* and *b* can run in parallel at the same time as *b* is dependent on the result from *a*. In Table 4, there is only one cell that is crossed – (*requirements, design*), which means that steps *design* and *requirements* can run in parallel while *design* depends on results from *requirements* (see deliberations in Section 3.2).

A cross in cell (*a*,*b*) of the *parallel dependencies* matrix requires special attention as it warrants tight coordination between these steps, otherwise the work done in step *b* may need to be totally re-done after substantial changes in the result from step *a*. Even more tight cooperation is required when both cells (*a*,*b*) and (*b*,*a*) are crossed.

Table 4. Example of parallel dependencies matrix

	Requirements	**Design**	**Coding**	**Test**
Requirements				
Design	x			
Coding				
Test				

Sometimes a cross in the *parallel dependencies* matrix appears due to the steps we have chosen are too big. In this case, we can try to remove parallel dependencies by decomposing (splitting) the original steps into smaller ones. For example, we can split *requirements* into two steps: *specifying requirements* (SR) and *specifying requirements tests* (SRT). Then, the input-output matrix may get the form of Table 5, and the parallel execution matrix will get the form of table 6. As the result the *parallel dependencies matrix* becomes empty.

Table 5. The new input-output relationships matrix

	SR	**SRT**	**Design**	**Coding**	**Test**
SR					
SRT	*Requirements specifications				
Design	*Requirements specifications				Test results
Coding			*Design specifications		Test results
Test		*Test specifications		*Code	

Table 6. The new parallel execution matrix

	SR	**SRT**	**Design**	**Coding**	**Test**
SR					
SRT			x	x	
Design		x			
Coding		x			
Test					

Note that it is not always possible to get an empty *parallel dependencies* matrix through decomposition (see deliberation in Section 3.2). Decomposition may not remove all parallel dependencies, or it can introduce new dependencies instead of the old ones. This, for example, happens if we allow steps *SR* and *SRT* run in parallel.

3.4 Weak dependencies

Cell (*a,b*) in the *weak dependencies matrix* shows whether step *b* may require something more than the formalized result from step *a*, e.g. a historical trace of how the result has been achieved. For example, it is not unusual for the designers to need more information than exists in the formal requirements. They might need to understand the rationale behind one or more requirements, or need some other background information. Cell (*a,b*) in this matrix specifies what kind of information from step *a* might be needed to complete step *b*. An example of such matrix for our systems development process is given in Table 7.

The concept of *weak dependencies* reflects the needs for informal communication in the frame of a process instance. It is not always possible to include everything that might be needed for the next step in the formal results, as different instances might

require completely different information from the previous steps. It is better to start looking for this information on the demand basis, i.e. when there is a need for it.

Table 7. Example of weak dependencies

	Requirements	**Design**	**Coding**	**Test**
Requirements				
Design	Rational behind requirements Communication with the customer			
Coding		Clarification of diagrams		
Test				

3.5 Teams and their relationships

The *teams* matrix shows the presence of collaborative teams and their relationships. The presence of teams is shown in the diagonal of the *team*s matrix: cell (*a*,*a*) is marked with the light gray color if the team for step *a* consists of more than one person. The non-diagonal elements show whether the teams participating in different steps intersect. If the teams for steps *a* and *b* intersect but not coincide, we mark both cells (*a*,*b*) and (*b*,*a*) with the light gray color. If the teams coincide, we mark these cells with the dark gray color.

An example of *teams* matrix for our systems development process is shown in Table 8. Here, we assume that each step does have a team; *requirements* and *design* teams intersect but not coincide; *coding* and *test* teams coincide.

Table 8. Example of teams matrix

	Requirements	**Design**	**Coding**	**Test**
Requirements				
Design				
Coding				
Test				

The diagonal of the *teams* matrix identifies steps that may require support for intra-step collaboration, which is discussed in Section 5. The non-diagonal part of the matrix is used for analyzing the needs for support of inter-step coordination/ collaboration, which is discussed in Section 3.6 and 5.

3.6 Inter-step collaboration

By merging the *weak dependencies* matrix (Section 3.4) with the *teams* matrix (Section 3.5), we get a view on the needs for inter-step collaboration. The result of the merger of matrices in Tables 7 and 8 is presented in Table 9. In it, one non-empty cell (*requirements, design*) has the light gray background, the other one (*design, coding*) has the white background. In the first case, *requirements* and *design* teams intersect, thus additional information from one step to another can be carried out tacitly via

intersecting members. In the second case, the *design* and *coding* teams do not intersect, thus there is a need to make information (other than formal design documentation) that might be needed from *design* for *coding* available on demand. Considering that the design team can be dissolved before coding starts, or be not easily available, this issue needs to get special attention.

Table 9. The weak relationships matrix merged with teams matrix

	Requirements	**Design**	**Coding**	**Test**
Requirements				
Design	Rational behind requirements Communication with the customer			
Coding		Clarification of diagrams		
Test				

4 BPS services capabilities

In this section we discuss capabilities one can expect for a BPS service to provide. The term capability here is understood as ability to provide support for certain aspect of running business process instances. The capabilities could be provided separately or tied up in a clump where one cannot be used without the others. The list of basic capabilities that we believe could be expected from BPS services is presented below.

The list has been compiled mostly based on our experience of supporting business processes, and analysis of BPS tools from other vendors. Many of the capabilities listed below are also mentioned in various research works. However, due to the lack of space, we cannot produce a detailed analysis of the literature on the topic in this paper. We do not insist that the list is comprehensive; in this paper it is only important that it includes capabilities the needs for which could be derived from the content of the matrices from Section 4.

1. *Information logistics support* (ILS) is aimed at providing process participants with all information they need to complete their work without being overwhelmed by the details that are not relevant. ILS is particular important for steps where the inputs-outputs constitute information objects, like documents, program code, test protocols, etc. ILS can be provided in two different ways:
   - By actually sending the results to the next step team, e.g. via email. We refer to such kind of logistics as to *conveyor belt logistics* [9].
   - By providing a shared space where the results are stored and made available for the participants of the "next step". We refer to such kind of logistics as to *construction site logistics* [9].

   The ILS capability can also provide version control for the information objects that are produced more than once. Version control is easier to achieve by using the construction site logistics than the conveyor belt one.

2. *Intra-step collaboration support* is aimed at providing a team working on the same step with means to store/retrieve intermediate results and communicate with each other synchronously and/or asynchronously.
3. *Inter-step collaboration support* is aimed at providing the teams, or individuals working on different steps with means to access intermediate results obtained in each others' steps and communicate between the teams synchronously and/or asynchronously.

   Note that intra-step and inter-step collaboration may require different means of support. In the first case, the communication can be between people of the same profession who reside in the same department. In the second case, it can be between people of different professions who reside in different departments.

   Note also that term collaboration in this paper is used in a special way. It does not cover cases of accepting inputs from the previous steps, or forwarding outputs to the next steps. The latter two cases are considered as belonging to the ILS issues.
4. *Process flow restrictions enforcement* ensures that the rules establish for the process flow are strictly followed. Examples of such rules:
   - Ensure that the steps that cannot run in parallel run in turns.
   - Ensure that if a step needs a result (output) from some previous step it waits until the latter step is finished.
5. *Process flow support* ensures smoothness of the process flow; it sees to that the steps that can be activated (i.e., inputs are ready) are activated at once. This for example, can be done by informing process participants that they should start working on their step through sending them required inputs (when the conveyor belt ILS is employed), or notifying that the inputs have been placed in the shared space designated for them (when the construction site ILS is employed).
6. *Participation restrictions enforcement* ensures that "right" people are participating in various process steps. The rules can be established because of external legislation (e.g., Sarbanes-Oxley) or decided on internally. The rules concern who can participate in what steps, which information is available to each kind of participants, whether the step teams can intersect, etc.
7. *Resource assignment support*. This capability means automatic or semi-automatic formation of step teams based on qualifications and availability of process participants.
8. *Support for domain-specific operations*. This capability includes tools to complete operations inside the step, like compiling or testing a program. These tools can be general, like an office package (MS, OpenOffice, etc.), or specialized like compilers for specific languages.

5 Guidelines for choosing capabilities

The easiest way to choose a BPS service is when the process is workflowable in a high degree [10]. With the help of our framework workflowability can be defined as:

– The *parallel execution* matrix is empty - dependent steps are executed in turns
– The *teams* matrix is empty – one and unique person per step
– The *weak dependencies* matrix is empty (only formalized inputs are relevant for steps execution).

In this case, any service that provides a workflow solution, e.g., [1], would be suitable.

In a case where workflowability is not present and cannot be obtained by decomposition of steps (see Section 3.3), each capability in the list of Section 4 needs to be considered separately against the properties of the business process in question. Fig. 3 shows which matrices from Section 3 can be used for determining the needs for which capabilities. In addition to relationships between capabilities and matrices, the bottom part of Fig.3 shows which matrices are basic – white background, and which are derived – gray background and arrows into them. Note that in this paper, only capabilities 1-6 of the list from Section 4 are covered.

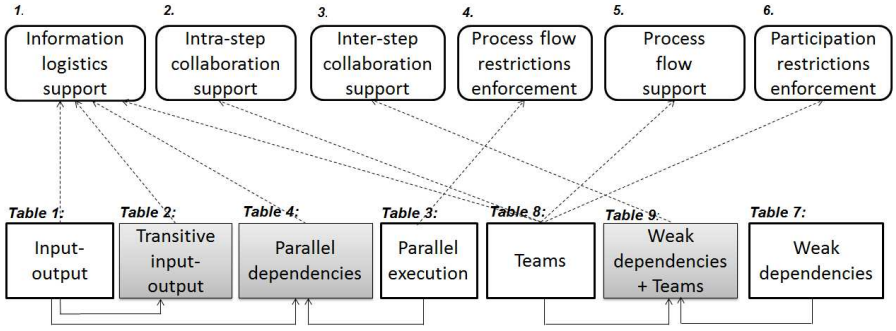

Fig 3. Guidelines for identifying capabilities.

The preliminary guidelines for choosing BPS service capabilities are as follows:

1. *Information logistics support.* The capability is desirable as long as the *input-output* matrix (Table 1) identifies results in form of information objects that need to be passed between the steps. It is less critical when the *teams* matrix (Table 8) identifies that the step teams intersect. In this case, responsibility of moving the results from one step to another could be assigned to the intersecting parts of the step teams and be completed outside the frame of a BPS service. If the teams coincide, there is even less need for information logistics support.

   In cases when the *transitive input-output* matrix (Table 2) shows that there are no iterative loops (no symmetric non-empty cells in it) and the *parallel dependencies* matrix (Table 4) is empty, the conveyor belt information logistics will work satisfactory. When the loops are present, there can be many versions of the same information objects. When these versions are just sent from one step to another, there is a risk that a wrong version will be used instead of the right one. Having a shared space (construction site ILS) where a new version totally substitutes the old one would be preferable in case of loops. Having version control for such shared spaces can give additional advantages in case one needs to understand the difference between the old and new version.

A non-empty *parallel dependencies* matrix (Table 4) requires even more attention to information logistics. This is especially so when the teams of two steps with parallel dependencies neither coincide nor intersect (Table 8). In this case, any new piece of information should also include explanation on whether it is a complement to what already has passed, or substitution of the old piece, or both. Having dedicated shared space with version control and explanatory comments would constitute appropriate information logistics support in this case.

2. *Intra-step collaboration support.* The capability is desirable when the *teams* matrix (Table 8) identifies steps that do have teams (cells marked by the light gray background in the diagonal of the matrix). The capability should allow to store and share the intermediate results. In addition, it may include messaging and on-line communication, like chat, voice or teleconferencing. The basic needs could be solved by a shared space structured according to the needs of the team, with or without version control capabilities. Such a space can include forums for discussions, and journals to record the internal or external events, e.g., communication with customer/supplies. A step shared space could be useful even when a step "team" consists of one member (the white background in the diagonal of the *teams* matrix). This is true when he/she cannot complete the whole step in one go and need to return to the step several times before it is completed.
3. *Inter-step collaboration support.* The needs for this capability can be identified by the merged *week dependencies* + *teams* matrix (Table 9). The capability is desirable when there are non-empty elements in the matrix. The needs for this kind of support is even greater if the teams for steps with weak dependencies do not intersect – non-empty cells in the matrix with white background. One way of arranging such collaboration is by having a communication channel between the teams. The (week) dependent step team can send a request for extra information, and get it back through the same channel. This will work if the members of the team which have the information are still available for questioning.

   Another way of arranging inter-step collaboration is possible if the shared spaces technique is employed for intra-step collaboration. If the step shared space is made accessible to the team of a dependent step, the members of the latter can themselves find the information they need. This will work provided that the shared spaces are structured in a way that makes it easy to navigate in them for the process participants who do not participate in the correspondent step.
4. *Process flow restrictions enforcement.* This capability is desirable in case of the *parallel execution* matrix (Table 3) is empty or sparse. If many steps can and should run in parallel one may have very little use of this capability.
5. *Process flow support.* This capability is very useful if the *teams* matrix (Table 8) shows that steps teams neither intersect nor coincide. However, it does not harm to have it even if they do intersect. In case when many steps can run in parallel and steps teams do not intersect, a more sophisticated coordination mechanism is required than just process flow support, see the ILS-related discussion above.
6. *Participation restrictions enforcement.* This capability might be needed if steps teams do not coincide, which can be easily figured out from the *teams* matrix (Table 8). The actual need for this capability depends on the reasons that are not

revealed by the steps relationships matrices. If the conveyor belt information logistics is employed, the restrictions are established by sending results only to the participants of the steps in which these results are to be used. If shared spaces are employed for information logistics support, the participation restrictions are realized by limiting access to some parts of shared spaces.

7 Discussion and plans for the future

In the Sections 3-5 above, we have demonstrated how a business process can be analyzed with the help of steps relationships matrices, and how the needs for various BPS capabilities can be derived from the content of these matrices. The ideas presented in this paper can be summarized in the process of choosing BPS services presented in Fig. 4. The ideal situation here would be if the only steps in the process that require human intervention where *identifying steps* and *filling basic matrices.* The rest would be done pretty formally, or even fully automated.

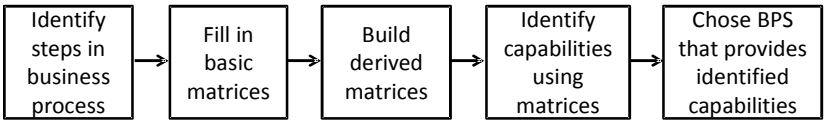

Fig. 4. A simplified process for choosing BPS

The first problem on the way to the ideal is the last step of the process – choosing a BPS services based on the list of capabilities. The following issues can be identified here:

– BPS vendors do not describe their services in terms of capabilities we listed. They, usually, do not provide information in such terms as information logistics, or process flow restrictions. The descriptions are in terms of a business domain at which the service is aimed, or/and in form of functional specification. To formalize the last step in Fig. 4, there is a need to introduce a standard on capabilities provided by BPS service vendors, or design a practical methodology of BPS service analysis that produces a list of capabilities. The latter approach seems to be more feasible. In addition, just having a list of capabilities provided by a BPS service may not be enough, as there can be dependencies between them, so that some capabilities cannot be provided without the others. These dependencies need to be revealed so that choosing a service that provides one useable capability with a set of unusable ones could be avoided.

– When choosing a BPS service, one also needs to take into account other requirements than capabilities from our list. Security provision and SLA level are typical examples of such requirements. Easiness to customize the service to a particular process is another example. Different vendors will be providing capabilities in different ways, e.g. conveyor belt vs. shared spaces, using different

modeling notations to specify details of the process in order to make customization. As the final selection of service can depend on additional requirements, they also need to be listed and understood.

The second problem concerns the following. Suppose we have done analysis according to the guidelines in Section 5, and established a list of capabilities to be requested from a BPS service. Should we seek a service that provides all these capabilities? To answer this question, one needs to take into account factors that characterized the environment in which the process instances are running. To such factors, for example, belong:

– *People engaged in the process*. For example, with a low staff turnover, and already established efficient ways of personal communication, one might choose not to impose a new collaboration mechanism, as it can create a resistance to using a new service. An opposite situation, i.e. with a significant staff turnover, warrants standardization of collaboration mechanisms.
– *Dynamism of environment*. For example, if a process definition is expected to be the same during considerable period of time, it can be advantageous to have a capability for strict process flow enforcement. If, however, the process definition is to be changed quite often, this capability might be useless, especially if the time for customizing the service to a new definition is comparable with the time to the next change.

Our future plans include investigation of the two problems identified above, as well as testing the main ideas in practice. Designing computerized support for the process in Fig. 4 is also on our research agenda. To accomplish this task we would, probably, need to introduce more derived matrixes than it was done in Section 4, and establish some structure in the capabilities list from Section 5. The next problem on the agenda is choosing a BPS service suitable for multiple business processes.

References

[1] *Apian cloud: BPM* http://www.appian.com/bpm-software/cloudbpm.jsp
[2] *ActionFlow:* http//www.actionflow.com
[3] *SalesForce:* http://www.salesforce.com
[4] *Projectplace:* http://www.projectplace.com
[5] Van der Aalst W. M. P. , Weske M., Grünbauer D. Case handling: a new paradigm for business process support," *Data & Knowledge Engineering*, vol. 53, no. 2, pp. 129-162, 2005.
[6] Swenson K.D. ed. Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done. Meghan-Kiffer Press, Tampa, Florida, USA, 2010.
[7] *iPB Reference Manual*. On-line documentation: http://docs.ibissoft.se/node/3.
[8] Andersson T., Bider I., Svensson R. Aligning people to business processes experience report. Software Process Improvement and Practice, vol. 10, no. 4, pp. 403-413, 2005.
[9] Bider I., Perjons E., Johannesson P. In Search of the Holy Grail: Integrating social software with BPM. Experience Report. *LNBIP*, No 50, Springer, pp. 1-13, 2010.
[10] Baresi, L. et al. 1999. Workflow design methodology. Grefen, Pernici, and Sanchez, eds. *Database Support for Workflow Management: The WIDE Project.* Kluwer pp. 47–94, 1999.