Declarative business artifact centric modeling of decision and knowledge intensive business processes

Roman Vaculín¹, Richard Hull¹, Terry Heath¹, Craig Cochran², Anil Nigam¹, Piyawadee Sukaviriya¹

¹ IBM T.J. Watson Research Center, Hawthorne, NY 10532 USA, ² IBM Integrated Supply Chain, Atlanta GA 30328, USA
Email: {vaculin, hull, theath, clcochran, anigam, noi}@us.ibm.com

Abstract—In this paper we address the problem of modeling collaborative decision and knowledge intensive business processes (sometimes referred to as Decision Intensive Processes, or DIP processes). DIP processes assist users in performing decision intensive tasks, and provide users with a guidance relevant to process execution context. DIP processes are by nature collaborative, data-driven, need to support various kinds of flexibility at design and run time, and need to integrate with external services and information sources. Such a combination presents significant challenges for contemporary business processes technologies. We present a solution based on a business artifacts paradigm (a.k.a. business entities with lifecycles) using a Guard-Stage-Milestone (GSM) model for declarative lifecycles specification. We introduce a CoreControl-MicroProcess design pattern, which allows a natural blending of a business functional process structure (usual for most business processes), with a decision & knowledge driven structure providing domain specific decision guidance to users. The proposed design pattern along with the declarative GSM BA approach provide suitable design primitives for DIP process, as demonstrated on a real problem from the supply chain solutions enablement domain.

Keywords: decision intensive business processes, business artifacts, business entities with lifecycles, Guard-Stage-Milestone model, GSM, process design patterns

I. INTRODUCTION

The existing business processes technologies are suitable for problems that can be relatively easily represented in the form of well defined stable repeatable flow structures consisting of activities or tasks. As pointed out by various authors (e.g., [18], [6], [14], [5], [11]) many classes of problems exhibit characteristics which present significant challenges to the contemporary business process formalisms. One important such class are the Decision and Knowledge Intensive Processes (also called Decision Intensive Processes, or DIPs) [3] which are defined as business processes whose conduct and execution are heavily dependent on users (“knowledge workers”) performing various interconnected knowledge intensive decision making tasks. Decision intensive processes are genuinely knowledge, information and data centric and require substantial flexibility at design and run time [15].

In decision intensive processes a lot of knowledge and expertise is implicit, resides in experts heads or it has a form of (informal) best practices or organizational guidelines [8]. In our work we address the problem of modeling a class of decision intensive processes where at least some fragments of knowledge and/or expertise can be (at least) partially formalized and represented as an explicit machine processable knowledge (knowledge base). Example problems range over many knowledge & decision intensive domains including campaign management, insurance case investigation, background check, clearance determination, SW&HW appliances configuration, or technical sales support.

We propose a solution to modeling DIP processes using a business artifacts (BA) paradigm (a.k.a. business artifacts) [12], [2], [4]. We employ a declarative variant for lifecycle representation using the Guard-Stage-Milestone (GSM) model [9]. The GSM BA approach integrates the data and the process aspects in one unifying framework, supports parallelism within a single BA instance and hierarchical organization of lifecycle activities. To model DIP processes in the GSM BA framework, we introduce a new process design pattern, called CoreControl-MicroProcess (CC-MP) pattern. The pattern is based on an interplay of a (1) Core Control business artifact, which provides a top-level process control harness and placeholder for information sharing, and (2) Micro Process Template and Micro Process business artifacts, which together provide design and run time primitives for flexible decision driven process structures. One central notion of the pattern is an explicit representation of domain knowledge elements (that form the decision-driven structure) as BA instances, and dependencies among them in the form of declarative conditions.

Modeling DIP processes in the GSM BA framework has several advantages. The data-centric nature of GSM BA model is a good fit for a substantially data-driven nature of DIP processes. Next, the hierarchical structuring of GSM lifecycles corresponds to how the domain experts think about their activities, improves process understanding, and supports structured visibility, tracing and visualization of decision tasks. In combination with declarative conditions of the GSM model the hierarchical lifecycles allow an easy coordination of large number of interconnected decision artifacts, tasks and process instances, and it also allows for large degree of flexibility. Finally, comprehensive support for parallelism in the GSM model allows support of rich collaborative models. Additionally, the CC-MP pattern allows an easy integration of the process control with the domain knowledge elements,
which are represented as BA instances in the same model as the DIP process, and thus are directly accessible in all parts of the DIP process model. Also, this design choice allows easy maintenance and update of the decision-driven process structure by editing the BA instances in the knowledge base, without having to modify the BA types definitions.

To validate our approach, we present a real application, called the Solution Builder (SB), from the area of supply chain solutions design enablement, and we illustrate how the CoreControl–MicroProcess design pattern is employed to model the business process of this application.

The rest of this paper is structured as follows. Section 2 briefly introduces DIP processes and the GSM BA model. Sections 3 and 4 introduce the Solution Builder problem and the BA types modeling it. In Section 5 we present the CoreControl–MicroProcess design pattern and we discuss details by employing the SB example. Section 6 gives a discussion of the approach and it briefly reviews the implementation. Finally, Sections 7 and 8 review related work and conclude the paper.

II. PRELIMINARIES

A. Decision Intensive Processes

Collaborative decision intensive business processes (DIPs) is a class of business process which provide a guidance and support to one or several users in performing information and decision intensive tasks that need to be solved in a collaborative fashion with the help of various information and knowledge resources. Figure 1 shows a schema of DIP business processes.

As opposed to traditional business processes approaches, which more or less completely prescribe the process execution flow, in DIP processes, similar to case management [18], a user has much more control over the possible flow choices. **Role (and goal) of users**, when interacting with DIP processes, is to perform (knowledge intensive) decision making tasks by making decisions and providing required information, in order to generate information artifacts, such as plans, list of requirements, or recommendations relevant to solving a particular problem. **Role of DIP processes** is to provide users with a guidance in the form of contextual information, (only) relevant choices, options, decision advices and recommendations appropriate in a given context and execution state of the process. At the same time, as pointed out by Gronau and Weber [8], the knowledge and decision processes need to be integrated well with the traditional business processes. Therefore, the overall **structure and control DIPs** is defined by a “blend” of the following two structures:

1. **Business functional perspective** represented typically as a traditional structured flow that typically provides the framing of the overall process. The formal structure may originate in usual procedural flows and patterns of a particular problem, or in organizational or problem specific guidelines.

2. **Decision-driven process structure** is a less formal structure formed and governed by problem domain specific knowledge, for example in the form of best practices or problem related experiences. Such knowledge, represented for example in some form of a knowledge base, serves as a major driver of the DIP processes since it represents the specific contextual information needed for providing the user with the guidance at proper stages of the process. The process flow, control and structure can be derived from the knowledge, dependencies and relationships applicable in a particular context.

The overall flow of DIP processes is controlled by a possibly complex interplay of structures 1 and 2. The mode of operation we advocate as natural (see Section V) is to perceive the functional structure 1 as a mechanism for providing a top-level organization to the decision-driven structure 2.

DIP processes must allow for a smooth integration with external (e.g., SOA) environments, in particular with external services and information sources which provide necessary information inputs for decision making. Similarly, DIP processes are collaborative by nature and therefore need to provide multi-user, multi-roles support, with appropriate notions of visibility [10], traceability and accountability.

Finally, DIP processes require various kinds of flexibility at design time (changes of structure, knowledge base, etc.) and at run time [15] (flexibility in execution flow).

Table I summarizes our discussion as application and corresponding technical requirements of DIP processes.

B. Business Artifacts

The business artifact centric approach [12], [2], [4] considers data as an integral part of business processes, and it defines the business processes and its operations in terms of interacting key business artifacts (BAs). Each BA type is characterized by an information model and a lifecycle model. The information model records all business-relevant information about a BA instance as it moves through the business. The lifecycle specifies all possible evolutions of a BA instance over time. As an example BA, consider a bank CheckingAccount, which records all information about an account since it has been opened by a customer till it gets eventually closed and archived, with the lifecycle capturing all relevant states of the account, possible transitions, operations, etc. In this paper we employ the Guard-Stage-Milestones (GSM) model [9] for
Table I: Requirements for DIP processes

<table>
<thead>
<tr>
<th>Application requirements</th>
<th>Technical requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1. Support informed decisions</td>
<td>R1. Knowledge, dependencies representation</td>
</tr>
<tr>
<td>- Status, context &amp; risks visibility</td>
<td>- Focus on data, data-driven</td>
</tr>
<tr>
<td>- Information and experience driven</td>
<td>- Unified storage, unified visibility</td>
</tr>
<tr>
<td>- Incorporate guidelines &amp; domain expertise</td>
<td>- Suitable data and process granularity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R2. Structured guidance &amp; control</th>
<th>R2. Process primitives (language) &amp; patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Decisions/flow: show right options, choices</td>
<td>- Allow providing guidance</td>
</tr>
<tr>
<td>- Information recommendations: concerns, resources, information sources, ramifications-</td>
<td>- Allow flexibility within (minimal) constraints</td>
</tr>
<tr>
<td>- Capture business &amp; functional structures-</td>
<td>- Flow driven by dynamic (data) dependencies</td>
</tr>
<tr>
<td>- Structured, modular, hierarchical processes-</td>
<td>- Structured, modular, hierarchical processes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R3. Allow &amp; support collaboration</th>
<th>R3. Multi user / role process primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Multi user / role collaborative processes</td>
<td>- Access control</td>
</tr>
<tr>
<td>- Controlled, structured sharing &amp; visibility</td>
<td>- Controlled process and data views (visibility)</td>
</tr>
<tr>
<td>- Delegation of work fragments</td>
<td>- Adequate process structuring</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R4. Traceability &amp; accountability &amp; visibility</th>
<th>R4. Unifying data / process model &amp; patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Summary of attributes and decisions made</td>
<td>- Unifying (holistic) data model</td>
</tr>
<tr>
<td>- Collect all information (answers / decisions)</td>
<td>- Unified storage (at least virtual)</td>
</tr>
<tr>
<td>- Provide explanation for why decision made</td>
<td>- Process pattern for traceability &amp; account.</td>
</tr>
<tr>
<td>- Structured visibility, tracing and control</td>
<td>- Language constructs supporting structuring</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R5. Integrate external information sources</th>
<th>R5. Enabling business application integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Access data storages, service, applications</td>
<td>- Data-sources integration &amp; svc SOA integ.</td>
</tr>
<tr>
<td>- Make available to apps &amp; services</td>
<td>- Service access points (RESTful, WSDL)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Change KB offline / on the fly</td>
<td>- Process driven by data</td>
</tr>
<tr>
<td>- Change decisions on the fly + propagation of changes and ramifications</td>
<td>- Complex patterns of rework and invalidation of relevant dynamic process fragments</td>
</tr>
<tr>
<td>- Ability to update, change the schema</td>
<td>- Model-driven + flexible process schema</td>
</tr>
</tbody>
</table>

- A milestone is associated with one or more expressions of the form “on triggering event if condition” (both optional). A milestone gets achieved when one of its expressions becomes true (i.e., the triggering event gets detected and the condition is true). A guard is represented as an expression of the same form. The conditions of expressions of milestones and guards range over the information model of the corresponding BA instance, and possibly of related BA instances. The conditions are expressed using an extension of the OMG Object Constraint Language (OCL) [1]. The triggering events for milestones and guards might originate from the external world (e.g., from a human actor or an incoming service call), from other BA instance, or from internal processing of the lifecycle, e.g., as a result of milestones changing values.

The logic of stages operations is based on a flavour of an Even-Condition-Action semantics and it works as follows (for complete semantics see [9]). Each stage can be either active (opened) or inactive (closed). Stages get activated by means of their guards. A stage gets opened only if one of its guards gets triggered and only if its parent stage (if there is any) is active. When an atomic stages opens, its task is initiated automatically. When a composite stage opens, its immediate sub-stages become eligible for opening. The stage gets closed when one of its milestones is achieved or when its parent stage gets closed (in both cases the purpose of the stage was achieved). Notice that multiple stages of a single BA instance may run in parallel.

### III. SOLUTION BUILDER PROBLEM

To give a concrete example of an DIP business process we introduce a Solution Builder (SB) application which exemplifies many of the above described features. The SB application was developed to allow supply chain teams to simplify, streamline, and standardize idea to market deployment of cross-brand solutions. The process of enabling a new solution is an elaborate effort (taking up to several weeks or months) involving several team members who work together to understand the solution structure, requirements and constraints. The work of participating users largely depends on their expertise, and due to lack of standardization and appropriate tooling most of the process is seemingly ad-hoc, supported mostly by use of spreadsheets and text documents. The outcome of the SB process is a plan for solution enablement, called a Prescribed Course of Actions (PCA), which in essence is a structured plan consisting of solution specific parametrized actions that need to be performed to enable the solution. To create a PCA, many parameters and questions need to be considered, and depending on the solution nature different sets of decisions need to be taken into account.

Analysis of the SB domain revealed that a substantial part of the solutions enablement expertise can be formalized as a generic knowledge base. The individual knowledge elements are modeled as Questions, with each question instance capturing information such as the question text, type, possible answers, considerations, concerns, ramifications, resulting actions, time constraints, and relevant information resources. Additionally, each question is associated with dependencies specifying the context when the question is applicable.
When the work on a new solution enablement starts, the SB application first identifies the initial set of questions from the knowledge base that are relevant for the solution and presents them to users who need to answer them. As the users answer the questions, the SB application iteratively identifies other appropriate questions (using their dependencies) relevant to a solution and to what has been answered so far. Eventually, the SB will collect all answers and information needed to complete the PCA for this new solution. Notice that for one particular solution enablement typically only a subset of all questions will be asked, that are relevant for the particular solution. An act of answering a question is modeled as a QuestionAnswer instance (QA for short). Answering a QA is a small process on its own, since it often requires performing research, the QA can be delegated, postponed, submitted, etc.

Figure 2 shows an overview of the SB process focusing on its basic structure. In Figure 2 the flow progresses from left to right. The top-level SB process consists of 5 top-level logical phases which capture the functional structure and control of the process, with Phase 2 being further divided into 3 sub-phases corresponding to selecting offerings in the Hardware, Software and Services categories. These three sub-phases of Phase 2 can be executed in parallel, and similarly Phase 3 and Phase 4 can be executed in parallel with different users being responsible for working on these phases, while the remaining phases are performed in a sequential manner. The top-level phases serve as an organizing and controlling structure for Questions and QA instances (in Figure 2, QA instances are represented as dots). Specifically, every question in the knowledge base is assigned to one of the top-level phases of the overall process. During the process execution when some phase starts, the SB only considers questions assigned to this phase as candidates to be asked at this phase. For example, in Figure 2 when Phase 1 started, three questions in the KB were identified that are both assigned to this stage and whose dependencies are satisfied, and accordingly the three QA instances \( qa_1, qa_2 \) and \( qa_3 \) were created which users need to answer. These three QA instances can be answered and modified by the users if and only if Phase 1 is active. Such organization allows users to focus only on QAs relevant to an active phase instead of being overwhelmed by all QAs. Further, the phases serve as a mechanism for specifying responsibilities and views of various user roles – e.g., the role Team Lead is responsible for controlling Phase 2 and Phase 3 and answering the QAs controlled by these 2 phases.

The arrows connecting QA instances represent dependencies among them. For example QA instance \( qa_4 \) depends on \( qa_1 \) meaning that \( qa_4 \) was asked by SB because the value answered to \( qa_1 \) made the dependence of \( qa_4 \)‘s question satisfied (which enforced \( qa_4 \) to be asked). Thus, \( qa_1 \) can be seen as a reason for why \( qa_4 \) was created and asked. The dependencies are used by the SB to “guide” the user only to those QA instances which are relevant given the answers provided so far in the process.

IV. Modelizing the SB Problem Using BAs

Figure 3 shows the three BA types, namely PCACreation, Question, and QuestionAnswer that model the SB application, showing their information models represented as lists of selected attributes with types and cardinalities, and lifecycles with the most important stages (rounded rectangles), guards (diamonds) and milestones (small circles). A diamond with the cross inside serves as an entry-point guard of the lifecycle of a newly created BA instance.

The PCACreation BA type (Figure 3a) represents the end-to-end logic of the PCA creation process from its inception to its deployment. The information model tracks information about the new prospective solution (such as its name, content, background, etc.), information about the plan for enabling the solution (represented as list of actions, possible flags, concerns, team members, etc.), and it also records information about the progress and the history of collecting all the above information (represented as references to QuestionAnswer instances that were used to collect all the above information).

The top level stages of the PCACreation lifecycle as shown in Figure 3a correspond to the phases and their logic as showed in Figure 2 and described in the previous section. The substages are not showed in the figure. The flow logic is represented declaratively by means of guards and milestones of stages, which correspond to specific application requirements. For example, Define Functional Properties stage of a PCACreation instance pca can open only when the user decides to do so (by sending the event OpenDefineFunctionalProperties) and when the milestone OfferingsSelected of Select Offerings stage is achieved. This is expressed as a guard of Define Functional Properties stage as follows:

\[
\text{on pca.OpenDefineFunctionalProperties.onEvent() if pca.OfferingsSelected}
\]

In Figure 3a dashed arrow (1) indicates that the guard of Define Functional Properties stage depends on the value of milestone OfferingsSelected. We use dashed arrows to indicate dependencies among guards and milestones of the same BA type, while dotted arrows are used to illustrate dependencies among different BA types.
The Question BA type (Figure 3c) represents all relevant information related to one particular question that the SB application might need to ask the user in order to collect information needed for constructing the PCA. Question instances represent the expert knowledge of the solutions enablement which drives the SB process and helps its users in asking the right questions at the right time and making the correct decisions. The information model includes the question text, type (e.g., multiple choice question, yes/no question, free form text response, person selection, etc.), list of possible answers, and many context and expert information related to considerations such as cost, timing, and other ramifications. Additionally, each question instance might have a logical condition (dependenceCondition attribute) which specifies the context when the question is applicable, and a reference to one of the PCACreation stages specifying the appropriate phase of the SB application when the Question need to be answered (controlStage attribute). The lifecycle is concerned with authoring and editing the Question instances.

The QuestionAnswer (QA for short) BA type (Figure 3b) models the overall process of answering a question by the user. Every QA instance is derived from / based on (basedOn attribute) some instance of a Question BA. The information model contains the actual answer provided or selected by the user, reference to the owning PCACreation instance (owningPCACreation attribute), the name of PCACreation stage where the QA is answered (controlStage attribute), and some other attributes whose values are copied from the associated Question instance. As mentioned earlier, answering a QA instance is a small process on its own. There are two top-level stages in the PCA lifecycle, the Waiting stage that represents the phase when QA cannot be answered, and the Answering stage that contains all substages and activities that can happen when the QA is eligible for answering. For example, depending on the value of the questionType attribute one of the inside substages (e.g., Answer Free Form Text Question or Answer Multiple Choice Question, etc.) will present the user with the right information and allow the user to enter the right type of the response. The lifecycle in Figure 3b further illustrates some other features supported by the QA BA type, such as the ability to postpone the answering (Postponed milestone) or allowing rework of the QA before it gets committed (Commit Answer stage).

V. CoreControl–MicroProcess DESIGN PATTERN

In this section introduce a re-usable design pattern, which we call a CoreControl–MicroProcess (CC-MP) design pattern. This pattern in combination with the GSM BA approach provides suitable design primitives for DIP processes. We first introduce the structure of the pattern, illustrated on the SB application, and then we describe informally the interaction semantics (in paragraphs §1–§5). We illustrate the interactions by several examples of guards and milestones and their expressions from the SB application which show the power of the declarative GSM model to relatively easily specify intricate relationships between many BA instances in a succinct, easy to understand and easy to modify form.

Figure 3: Process model of the Solution Builder application represented as Business Artifact types with declarative GSM lifecycles. For each BA type its information and lifecycle models are showed.
A. Structure of CC-MP pattern

Figure 4a presents the BA types of the CC-MP design pattern as a UML class diagram with BA types showed as classes decorated with a small life-cycle icon (lifecycle are omitted). The CC-MP design pattern, is a generalization of the roles and the structure of the SB BA types we described in the previous section. The CC-MP design pattern consists of three BA types and several other elements as follows.

(i) Core Control (CC) BA type (generalizes the role of the PCACreation BA) models the top-level business functional structure of the DIP process. It defines a top-level control of the DIP process. The information model serves for tracing DIP process progress, results, and for sharing information with the decision-driven process structure layer.

(ii) Micro Process (MP) BA type (generalizes the role of the QA BA) defines a content and a behavior of a (“micro”) artifact whose (possibly many interdependent) instances constitute the decision-driven process structure (possibly in an interplay with many interdependent instances of some other Micro Process BA type).

(iii) Micro Process Template (MP-Template) BA type (generalizes the role of the Question BA) defines a content and a behavior of an artifact whose instances represent and store reusable decision/knowledge elements, such as Question instances, which are used as templates for creating/instantiating instances of the associated MP BA.

Remark 1. Although the MP and MP-Template BA types are tightly related to each other, we clearly separate these two types. Also, we do not use the inheritance to capture the relationship between MP-Template and MP. The reason is that MP-Template and MP have different purpose, different lifecycles, and typically, their relationship is “one-to-many”.

(iv) basedOn association represents the relationship between an MP instance and the MP-Template instance which created it and whose data served as a template for creating the MP instance under consideration. (see §2)

(v) ownedBy association represents the ownership relation between the CC instance and MP instances. The CC instance serves as an organizing and coordinating structure for the MP instances that it owns and controls. (see §1)

(vi) controls association captures the basic control relationship between the CC instance and MP-Template / MP instances. Typically the control and organization of MP instances is related to some relevant part of the CC’s lifecycle, e.g., to a stage, an event, etc. Therefore the controls association links the MP type to the Life_Cycle_Component class which represents the CC’s lifecycle parts. (see §1)

(vii) dependenceCondition of an MP-Template instance is used to specify under what conditions this instance shall be used to spawn new instances of an associated MP BA type. The condition is an OCL expression that might refer to any attribute of any BA instance of the types constituting the CC-MP design pattern. Often the dependenceCondition refers to other MP and MP-Template instances and their values, which effectively constitutes the notion of “dependencies” between MP-Template instances (and also the notion of a dependence graph). (see §3, §4)

(viii) predecessors association specifies the structure of how MP instances are dependent on each other. Each MP instance can have zero or more predecessors. The predecessors relation defines a partial order of the MP instances. (see §4)

(ix) createMP method of MP-Template is a factory method [7] used for creation of new MP instances. (see §2)

(x) findAndCreateMPs method of CC identifies appropriate / matching MP-Template instances which are used to create corresponding new MP instances. (see §4)

Example 1. Figure 4b illustrates application of the CC-MP
pattern to the BA types of the SB process model, presented as a UML class diagram. Compared to Figure 3 which focused more on the lifecycle aspects, Figure 4b stresses the structural relationships. In the SB application, the Core Control BA is realized by the PCACreation type, the Micro Process is realized by the QA BA, the MP-Template is realized by the Question BA, the ownedBy association is realized by the owningPCACreation association, the predecessors association is realized by the motivatingQAs association, and the controls association is realized by the controlStage association. controlStage links the QA instances to particular stages of the owning PCACreation instance.

Remark 2. While in the SB application, one CC BA type, as well as one MP and MP-Template type is used for modeling the problem, in other applications more than one type of each might be used to represent the business process faithfully.2

B. Informal semantics of CC-MP pattern interactions

The instances of BA types of the CC-MP design pattern operate in a close relationship. Although, different specific patterns of control will apply in different applications, we summarize here the shared principles of the CC-MP pattern.

§1. Control and coordination of MPs: MP instances are always owned by some CC instance (ownedBy relation) and controlled by some of its lifecycle components (controls relation). The controls association specifies which part(s) of the CC lifecycle are controlling the MP instances. While the controls association defines a basic control relationship between MP instances and the CC instance, it is the role of the respective lifecycles of the MP and CC types to define the specific interactions. For example, coordination might be addressing creation of MP instances ($\S2$,$\S3$), synchronization of the CC and MP lifecycles, or of the information flows ($\S5$).

Example 2 (PCACreation controlling QAs). In the SB application, the lifecycle evolution of many QA instances depends on changes of the PCACreation lifecycle. For example, switching between the Answering and Waiting stages of a QA instance is indirectly controlled by the lifecycle of the owningPCACreation instance. Specifically, a QA instance qa can be answered (be in Answering stage) only when its owningPCACreation has its stage open which is a controlStage of qa. Arrows (2), (2') and (3) in Figure 3 show fragments of this indirect control mechanism. Consider for example the milestone PCAControlStageOpened with one of its expressions having the following form:

$$\text{on qa.owningPCACreation.}$$
$$\text{DefineFunctionalProperties.onOpen()}$$
$$\text{if qa.controlStage='DefineFunctionalProperties'}$$

Assume for an instance qa the Waiting stage is open and the Answering stage is not. Expression 2 becomes true only when the stage DefineFunctionalProperties of the owning-PCACreation instance opens (the event part of expression 2)

and when qa is controlled by this specific stage (the if part of expression 2). As soon as PCAControlStageOpened milestone is achieved, Waiting stage gets closed and subsequently the Answering gets opened, since its guard depends on this milestone (as indicated by the arrow (2')). This mechanism makes sure that all QA instances controlled by the DefineFunctionalProperties stage switch from Waiting to Answering, while other instances remain unaffected.

Similarly, the following expression of the milestone PCAControlStageClosed makes sure that the QA instances controlled by the DefineFunctionalProperties stage switch from Answering to Waiting when the milestone FunctionalPropertiesDefined is achieved:

$$\text{on qa.owningPCACreation.}$$
$$\text{FunctionalPropertiesDefined.onAchieve()}$$
$$\text{if qa.controlStage='DefineFunctionalProperties'}$$

Example 3 (PCACreation depending on many QAs). Define Functional Properties stage of PCACreation can close only when the user decides to do so (indicated by sending an appropriate event), and if every QA instance that is controlled by this stage is answered, i.e., the milestone Answered is true (illustrated by arrow (4) in Figure 3). The expression of FunctionalPropertiesDefined of milestone capturing this constraint has the following form:

$$\text{on pca.CloseDefineFunctionalProperties.}$$
$$\text{onEvent()}$$
$$\text{if pca.qasList->forall( qa |}$$
$$\text{qa.controlStage='DefineFunctionalProperties'}$$
$$\text{and qa.Answered)$$

In this expression we use the OCL operator $\text{forall}$ applied to a $\text{qasList}$ collection which gets satisfied if all items in the collection satisfy the condition specified in the $\text{forall}$ body.

§2. Creation of MP instances: A particular MP BA type is usually associated with a corresponding MP-Template BA type which specifies how MP instances shall be created. Specifically, MP-Template instances work similarly as a Factory design pattern [7] known in Object Oriented languages. A given MP-Template instance is used for creating new MP instances, with data of the MP-Template instance serving as a specification of how the MP instance(s) should be created. Creation of new MP instances is realized by the $\text{createMP}$ method of MP-Template instances. The $\text{basedOn}$ association is used to indicate the relation between the MP instance and the MP-Template instance that created it.

Typically, some attributes of the MP-Template are the so called “template attributes”, i.e., attributes whose values are set by a user in MP-Template instances with an intention of propagating the same values into MP instances created by a particular MP-Template instance. For example, a $\text{controls}$ relationship is a template attribute (in Figure 4a the $\text{controls}$ attribute of MP is marked as derived indicating this fact).

Example 4 (Creation of QA instances). In the SB application,
data of a particular Question instance is used while creating a corresponding QA instance. Some part of the data is simply copied from the Question instance to the QA instance (e.g., the controlStage attribute, questionText, etc.), and some data (e.g., questionType) is used to decide about the particular type of the QA instance that should be created.

§3. Coordination of MP instances creation: As part of the control & ownership relationships the CC BA is responsible for coordinating creation of new MP instances. The pattern we envision and that we implemented in the SB application is similar to the Builder design pattern [7]. In this pattern, at certain point(s) in the lifecycle of the CC instance cc the appropriate MP-Template instances are identified (e.g., found in the database) and used to create corresponding new MP instances. Two default mechanisms for identification of MP-Template instances are provided, namely the dependenceCondition attribute and the controls attribute of the MP-Template. The findAndCreateMPs method of the CC instance can use these attributes, for example to match/retrieve only those MP-Template instances with their dependenceCondition satisfied and with a value of their controls attribute matching the actual lifecycle point of cc.

Example 5 (Coordination of QA instances creation). In Section III we mentioned that in the SB application new QA instances get created as the process (i.e., the PCACreation instance) moves between its top-level stages. For example, when the Define Profile stage is entered for the first time, the Question instances are identified that have an attribute controlStage set to “Define Profile” and whose dependenceCondition is true. The matched Question instances are used for creating corresponding new QA instances.

§4. Dependencies & predecessors: DependenceCondition and predecessors relations are very closely related to each other. In fact, the actual values of the predecessors relation of the MP instances are derived from (implied by) the dependenceCondition of the MP-Template instances as follows. Consider MP instances p₁ and p₂, where p₂ was created by some MP-Template instance t. p₁ and p₂ are in the predecessors relationship (p₁ is the predecessor of p₂) if and only if p₂ was created by t because t’s dependenceCondition was true and p₁ contributed to this condition to be true.

Example 6 (Predecessors of QA instances). A dependenceCondition of a Question instance in the SB application typically refers to answer values provided by users to QA instances already answered. For example, assume that question instance q₁ has two possible answers, say yes or no. Assume that for some other question, say q₄, it is required that q₄ should be asked by the SB only if the user provided the yes answer to the question q₁. The dependenceCondition of q₄ in OCL can be expressed as follows (assuming pca variable refers to the current instance of the PCACreation):

\[
pca.qasList->exists(qa | qa.basedOn=q₁ and qa.actualAnswer='yes')
\]

Assume a QA instance qa₁ is basedOn Question q₁ and the user answered qa₁ with the value yes. After qa₁ is answered, the findAndCreateMPs method is invoked to find those Question instances whose dependenceCondition became newly satisfied. Since the dependenceCondition of qa₄ is satisfied, qa₄ will be found and its createMP method is used to create a new QA instance basedOn qa₄, say qa₄. Obviously, qa₁ can be seen as an immediate reason for why qa₄ was created and asked, and therefore the value of the motivatingQAs of qa₄ will contain qa₁ as indicated by the arrow in Figure 2.

§5. Storage & information flow coordination: In the CC-MP pattern different information can be stored in instances of the three BA types. CC instance(s) store(s) information related the overall DIP process and its results, MP-Template instances store information which is reusable across several runs of the DIP process, and MP instances store information related to a particular micro process instance in the context of one run of the DIP process. Two types of information flows are prominent in the CC-MP pattern. First, an information flow from MP-Template instance(s) to MP instances is happening during construction of the MP instances. Specifically, data of MP-Template instances typically impact (e.g., get often copied to) the newly created MP instances. Second, some data of the MP instances impact (e.g., get often copied to) the CC instance. For example, the values collected by CC instances might contribute to constituting the results of the overall CC execution (output generation).

Example 7. While the information model of the PCACreation is fixed, its specific content is dependent on the particular configuration of the Questions database and on the specific instantiation of the QAs during one PCACreation run, since the answers of key QA instances act as fillers of the PCACreation structure. For example, in Figure 2 it can be seen that the answers of qa₁, qa₂ and of some other QA instances are used as part of the resulting PCACreation structure. In the information model of the Question and QA BA it is specified which part of their information model contributes to which part of the PCACreation. For example, the Question instance collecting a “solution name” contains a specification of an assignment which is mapping the answer of a corresponding QA instance to the solutionName attribute of the PCACreation.

VI. DISCUSSION & EVALUATION

We evaluate our approach qualitatively by reviewing it with respect to the requirements (R1)–(R6) as summarized in Table I and by discussing the actual implementation. We implemented the SB application (following the design in this paper) in the Barcelona GSM design & runtime environment which supports a rapid model-driven development of GSM
Our approach supports making of informed decisions (R1) by representing expert knowledge as modular knowledge elements as MP-Template (Question) instances. The explicit declarative dependenceConditions allow flexible guidance of process flow (R1,2). Notice, that the knowledge elements are defined as business artifact instances in the same framework as the process itself, i.e., not in some external system or in an external database or rule-base. This allows integration of the process control and data flows with the expert knowledge (R2) since the explicitly represented data are directly accessible in all dependenceConditions, and in all guards and milestones of BA instances. Also, such design allows easy maintenance and update of the knowledge driven process layer by editing the MP-Template instance knowledge base, without having to modify the BA types definitions. For example, the questions knowledge base was created solely by end users and it is updated the same way (R6). Process schema modifications are supported by the model driven nature of both the information and the lifecycle models of the BA types, as implemented in the Barcelona environment.

The hierarchical stages of the GSM lifecycle, and in particular of the Core Control BA lifecycle, provide suitable mechanism for domain experts for structuring their activities. Our practical experience confirms that thinking in terms of stages and milestones was very natural for domain experts in the context of the Solution Builder application. Hierarchical stages also improve understanding of the complex decision-driven structure and allow collaborative support by supporting structured visibility, control, tracing & visualization (R2, R3), and it enables easy and flexible control of many MP instances (R2). The later is conveniently achieved by declarative guards & milestones which enable flexible linking of the business functional process structure with the decision-driven process structure. For example, the conditions (2), (3) and (4) described in Example 2 indirectly control up to 170 QA instances as the user opens or closes one top-level stage of the PCACreation instance. A different global behaviour can be easily achieved by modifying some of these three conditions. Collaborative aspects of DIP processes (R3) are further supported by comprehensive support for parallelism in the GSM model, and by support for multiple roles and users in the Barcelona execution engine (client-server model, with advanced access control). CC-MP keeps track of delegations, but the particular delegation patterns appropriate for specific applications need to be defined as part of the application logic. The CC-MP pattern supports traceability of decisions and possible causes (R4) in the form of dependencies and predecessors relationships. Furthermore, the BA approach fundamentally supports the notions of traceability, accountability and visibility by providing a unifying model covering the end-to-end evolutions of key data artifacts. Finally, regarding the integration with external processes, services and information sources (R5), the GSM model supports tasks for invocation of services and the Barcelona runtime provides RESTfull service endpoints for accessing the host processes.

VII. RELATED WORK

Our work is based on a data-centric business artifacts paradigm [12], [2], [4]. Recently different data-centric approaches have been proposed. Inspired by artifact-centric models, Redding et. al. [14] propose a FlexConnect meta-model where processes are organized as interacting business objects. In FlexConnect lifecycles are defined as finite state machines. Flexibility is achieved by introducing a pattern consisting of coordination objects, job objects and referral objects. Workflow patterns were studied by Aalst et. al. [17], and there are some similarities of the CC-MP pattern and the multiple instances without a priori run-time knowledge workflow pattern [17]. Müller et. al. propose a COREPRO framework for the data-driven modeling of large process structures, where the data structure types, with object lifecycles (defined as state transition systems), and relationships are used to characterize process structures. Similarly to our approach, domain specific data model consisting of object and relation types is defined at design time, and such a data model can then be instantiated to create specific data structures which serve as a definition of the run time process.

While the FlexConnect and COREPRO approaches are somewhat similar to our work, there are significant differences. Most importantly, neither of the above mentioned approaches addresses the problem of integrating knowledge elements (MP-Template and their instances) and the related concept of explicit dependencies, which is critical for representing the context of when a particular knowledge/decision element is applicable. Additionally, the approaches have no support for declarative conditions, hierarchical organization of life-cycles and parallelism in single instance.

Bromberg [3] provides analysis of decision intensive processes and identifies some key requirements, however no
specific solution is presented. Related to DIPs are “knowledge-intensive processes” defined by Gronau and Weber [8]. The proposed Knowledge Modeler Description Language (KMDL) addresses the problem of modeling and analyzing knowledge-intensive business processes. Compared to our approach, KDML does not intend to capture and incorporate the actual structure of knowledge elements and use it for structuring process flow and providing guidance.

In general, our work is related to the area of flexible process management (e.g., in [5], [15], [13], [15]), and in particular, to case management [18], [6]. Case management specifically focuses on supporting the work of knowledge workers that is usually ad-hoc, centered around knowledge and relies heavily on data and their flows. Recent trend in case management is to incorporate declarative constructs for managing flow of control. Our approach using GSM BAs and the proposed CC-MP pattern can be viewed as a prospective case management design pattern enabling support for DIP-like processes. Compared to other declarative approaches to workflows specification, such as DECLARE [13] where the possible sequencing of activities is defined entirely by constraints expressed in a temporal logic, GSM is a “hybrid” approach that combines procedural aspects, e.g., hierarchical stages, with declarative rules-based paradigm for determining what activities and stages should be performed. As mentioned earlier, both these aspects were essential when modeling the SB problem and its domain knowledge.

VIII. CONCLUSIONS

This paper introduced a declarative data-centric solution for modeling decision intensive processes. We employed the Guard-Stage-Milestone BA model for representation of artifact types, and we proposed a CoreControl–MicroProcess design pattern for modeling DIP processes. We validated on a real application the suitability of the proposed solution. The CC-MP design pattern supports definition and a blend of a top-level business functional process structure (usual for most business processes), with a bottom-up decision (or knowledge) driven structure providing domain specific decision guidance to process users. To support desirable features of DIP processes, we took advantage of GSM constructs, in particular of the hierarchical stages, declarative guards and milestones and embedded parallelism.

Currently we are extending the approach by incorporating complex invalidation and rework patterns that will allow better support for iterative aspects of DIP processes. Another topic we are exploring are extensions of the GSM model for support of complex visibility features based on views and windows [10]. Finally, we consider improving the guidance features by re-using the historical solution fragments and by constraining the offered choices by what users have done historically in similar situations.

REFERENCES