Enabling Flexibility of Business Processes by Compliance Rules

A Case Study from the Insurance Industry

Thanh Tran Thi Kim[1], Erhard Weiss[1], Christoph Ruhsam[1]

Christoph Czepa[2], Huy Tran[2], Uwe Zdun[2]

[1] ISIS Papyrus Europe AG, Maria Enzersdorf, Austria
{thanh.tran,erhard.weiss,christoph.ruhsam}@isis-papyrus.com
[2] Research Group Software Architecture, University of Vienna, Austria
{christoph.czepa,huy.tran,uwe.zdun}@univie.ac.at

Abstract. The Swiss insurance company *Die Mobiliar* creates insurance documents with a wizard application utilizing the Papyrus Communication and Process Platform. Based on predefined processes, wizards guide business users through document generation processes. Although wizards can be amended by business administrators to respond to changing requirements, a high degree of process automation restricts adaption to the rapidly changing insurance market due to rigidness and bureaucratic efforts. In our approach, the concept of compliance rules in combination with process redesign has been applied to enable flexibility for insurance processes. The original processes are split into predefined reusable sub-processes and a set of individual ad hoc tasks which can be added by the business users at runtime as they assess the current insurance client situation. Compliance rules guarantee overall process execution compliance whilst enabling the needed flexibility. The number of process templates can be reduced considerably to a few predefined core processes in combination with a set of ad hoc tasks. The flexibility achieved by this compliance-rule approach enables the adaptability of insurance processes by business users, from which the whole insurance industry can benefit.

Keywords: Compliance Rules, Wizard, Insurance processes, Adaptive Case Management, Papyrus Communication and Process Platform, Consistency checking, Knowledge intensive processes, Process Collaboration

1 Introduction

The Swiss insurance company *Die Mobiliar*[1] is the oldest private insurance organization in Switzerland. As a multiline insurer, offering a full range of insurance and pension products and services, *Die Mobiliar* needs to handle a huge amount of docu-

[1] https://www.mobi.ch/

adfa, p. 1, 2011.
© Springer-Verlag Berlin Heidelberg 2011

ments, exchanged with approximately 1.7 million customers [1]. An insurance document issued by *Die Mobiliar* is not only a piece of paper; it serves as a business card, representing the company to their customers. Moreover, *Die Mobiliar* considers well-designed and rich content documents as an opportunity to communicate and build a strong relationship with customers.

1.1 The Business Context of *Die Mobiliar*

Insurance case work can follow established procedures only to a certain degree as it usually demands for knowledge workers knowing their business very well in order to decide the best solutions for their clients. This is what Adaptive Case Management (ACM) is designed for: customer-oriented work driven by goals allowing the knowledge workers to select from a context-sensitive set of ad hoc tasks rather than to follow strictly predefined process flows. To produce quality documents in such a knowledge intensive environment, business users of *Die Mobiliar*, i.e. clerks, acting as knowledge workers, are guided by a wizard application to compose insurance documents from predefined building blocks. Typically several actions are composed by the wizard, interacting between clerks and IT systems to retrieve data for the generation of documents. Usually such wizard steps and processes are hardcoded in most of the wizard systems. Thus, the processes cannot be changed quickly to adapt to the requirements of a new insurance product and related document types.

1.2 Application Solution Overview

In order to overcome this problem, *Die Mobiliar* uses the Papyrus Communication and Process Platform[2] as basis for their customized "Mobiliar Korrespondenz System" (MKS, Mobiliar Correspondence System), with full functionality for online interactive business document production [2]. MKS assists several thousand business users handling various types of documents in different insurance cases with wizard processes. Hence, the documents and associated processes must be well-prepared and suitable for such a huge and complex working environment. To ensure a high quality of the generated documents, many aspects of the processes must be strictly controlled. This could be achieved by rigid, hardcoded process models. In MKS, however, instead of being hardcoded by IT staff, the processes are directly created by business administrator by a model editor built in the Papyrus platform. Clerks simply follow the steps defined in the processes that are not editable at runtime. Thus, a wizard template must be defined at design time by business administrators responsively to new document requirements.

On the one hand, a rigid process configuration ensures the process execution is under control and satisfies the compliance requirements identified and adhered to during the process definition. Compliance requirements result from different sources related to laws, contracts with business partners, general standards, best practices and company-internal regulations. A usually underestimated source is coming from tacit busi-

[2] http://www.isis-papyrus.com/e15/pages/software/platform-concept.html

ness knowledge of the knowledge workers themselves [3]. Thus, process models implicitly express compliance regulations for the execution of tasks. This has the disadvantage that business users may not be able to respond to exceptions and unforeseen situations.

On the other hand, instead of mapping the whole process into predefined task sequences, the execution of individual tasks could also be controlled by compliance rules, defined by business administrators. These rules loosely define task sequences that have to be adhered to for compliance reasons but allow knowledge workers to decide ad hoc which tasks shall be executed based on the assessment of the current situation. This way, the case evolves over time to enable the necessary flexibility, instead of being predefined by business administrators, who will never be able to predict every knowledge-intensive scenario. Therefore, we aim to enhance the simplification of the wizard design and the flexibility of execution by a compliance rule and consistency checking system. This approach will guard business users at runtime when they select from ad hoc task templates, as well as business administrators during design time defining new or amending existing sub-processes. The consistency checking system continuously observes process actions to ensure that the execution produces results in compliance with company regulations. We propose to restructure the wizard process in order to enable such flexibility for business users during process execution.

This paper is structured in four sections. Section 1 introduces *Die Mobiliar* and the insurance industry business application MKS based on the Papyrus platform and the current process situation as the motivation of our new approach. Section 2 describes a typical document generation process in MKS. Section 3 discusses the consistency checking methods using compliance rules and how to apply them in MKS. Section 4 elaborates on the results achieved and the benefits gained when our approach is applied to MKS. Section 5 discusses the lesson learnt.

2 Typical MKS Document Generation Process

MKS is built on the Adaptive Case Management (ACM) and Correspondence frameworks of the Papyrus platform. While the correspondence solution handles the design and content of documents, the ACM solution manages the processes for the document generation. The process management in ACM is very flexible as processes can be predefined and executed automatically, while ad hoc changes at runtime are also supported [4].

At *Die Mobiliar*, MKS supports clerks to interactively generate documents based on wizard templates and to retrieve data from different sources. The wizard is defined by ACM cases to define processes composed from interactive user steps entered by the clerks as well as from service tasks like web service requests executed automatically by the system for data retrieval. A document template is composed from predefined text building blocks and data variables. Forms are an integral part of the wizard definition and request the clerks to enter the document variable values for a document template in a step-by-step approach. Fig. 1 shows a typical wizard input form and the

preview of the corresponding document at that stage of the document generation process. The data is directly exchanged with the document. For instance, the value "*13579*" for "*Antrag-Nr."* (application number) is inserted in the form on the left side and immediately displayed by the related building block in the document on the right hand side.

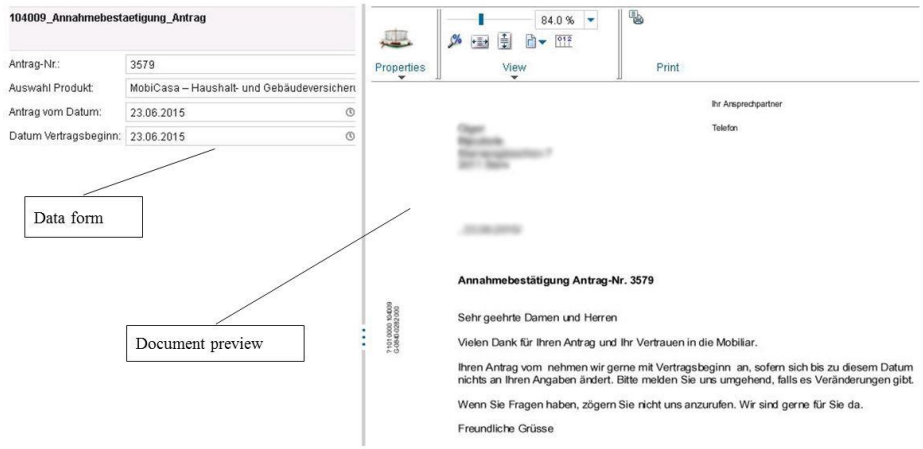

Fig. 1. Wizard with data form and document preview

The wizard processes executed by clerks at runtime are prepared and defined by business administrators at design time in form of templates stored in a template library as shown in Fig. 2. The processes are defined with an editor (1) that has full functionality to edit (2), visualize (3) and simulate (4) the wizard execution before being released into production. Transitions (5) connect the steps and each step has actions (6) defined which select the text building block to be added dynamically to the document.

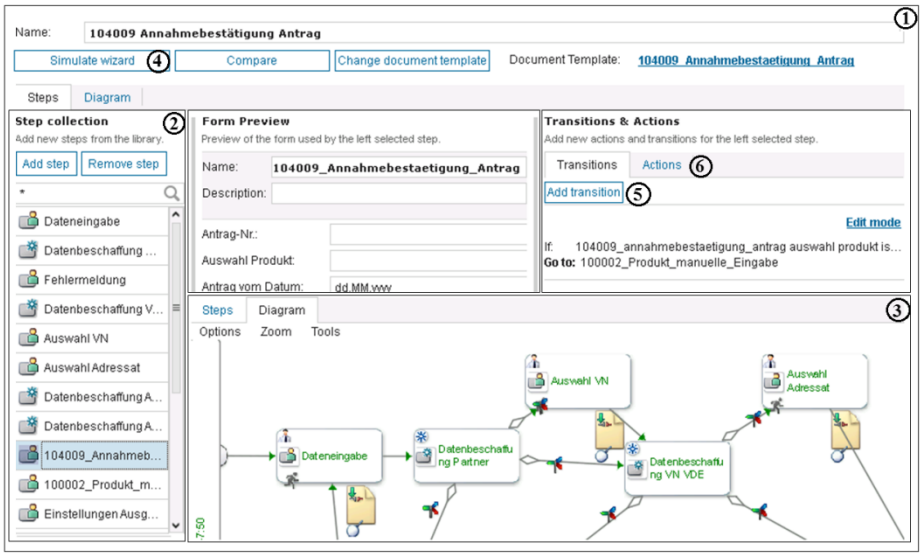

Fig. 2. Wizard template composition editor

The capability of MKS to edit wizard templates any time when new business requirements demand for it, supports *Die Mobiliar* to define new document and wizard templates within the boundaries imposed by the predefined processes. In order to support process flexibility even at runtime, we address this challenge by applying consistency checking methods in combination with compliance rules as discussed in the next section.

3 A Consistency Checking Method for Enhancing Flexible Execution of Wizard Processes

This section introduces a generically applicable consistency checking method used in our approach to enable the flexible execution of wizards for insurance document creation. The following subsections describe the overview of the approach, the modified process design of the wizards, and how to guarantee the process compliance as defined by a set of compliance rules.

3.1 Key Innovations

The original operating principle of MKS is shown in Fig. 3a. A wizard process template is defined at design time and instantiated for execution at runtime. Clerks must strictly follow the steps predefined in the wizard to create a document. The system is not intended to allow clerks to adapt the process at runtime, thus they cannot react to unforeseen situations of an insurance case.

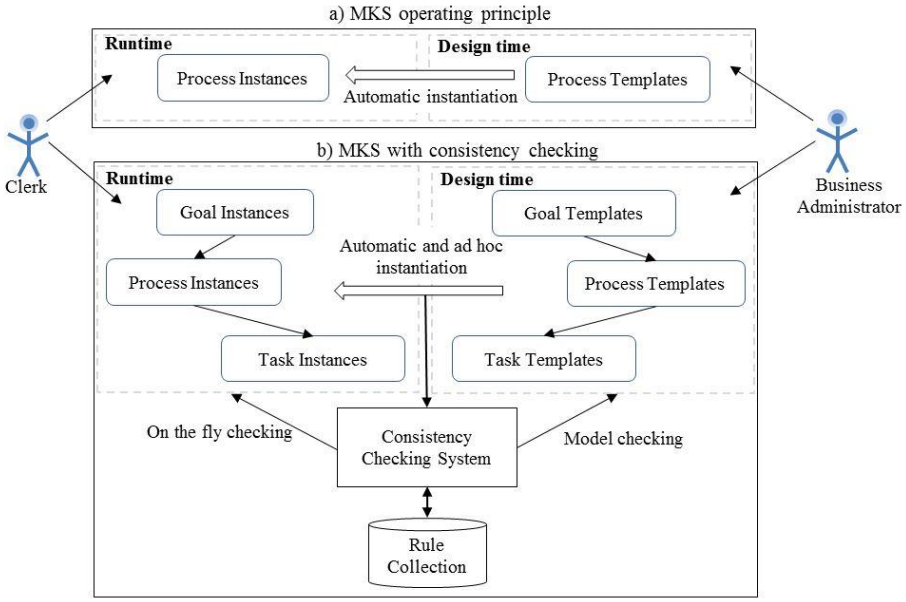

Fig. 3. (**a**) MKS operating principle and (**b**) MKS with consistency checking

The overview of MKS extended by the consistency checking system is provided in Fig. 3b. At design time, a wizard ACM case template is assembled from goals, sub-processes and individual tasks. Each sub-process is attached to a goal and combines the necessary tasks in a given sequence. A goal is reached when all its tasks are completed and the relevant data was achieved. Consequently, a document is finished when all goals of a wizard are reached.

The enhanced structure of the ACM wizard case allows clerks to execute ad hoc actions at runtime to achieve flexibility. Goal and task templates can be instantiated manually by the clerks into the predefined wizard case for adding new actions during the execution at runtime. An important component are compliance rules defined by business administrators to enable process compliance. Compliance rules are expressed by a constraint specification language, built upon temporal logic patterns [5, 6] allowing to define rules like "A.finished leads to B.started" or "A occurs only one time" or "C precedes D". The rules are verified during ad hoc actions executed at runtime by on-the-fly checking performed by the consistency checking system using Complex Event Processing (CEP) [7]. Additionally, the compliance rules are also verified by model checking [8] during design time when business administrators work on sub-process templates. Model checking also allows to verify the structural consistency of the predefined sub-processes. Thus, business administrators are guarded in the same way as clerks against violating compliance requirements. Due to the compliance rule definitions, the sequences of tasks are guaranteed in a similar way as with completely predefined wizard processes. The major difference is that tasks can now be added by the business users as needed for each specific situation. This way process compliance is guaranteed by the definition of compliance rules observed at runtime as well as

during design time. Clerks and administrators have to stick to the boundaries defined by the rule system and thus, the compliance of the overall case execution is guaranteed.

With the redesigned structure, the goal, process, and task templates can be reused in various wizard cases. Moreover, the ad hoc tasks instantiated from task templates can be added at runtime by clerks to adapt to an unforeseen situation requiring a new document. We use this to reduce the predefined processes in the library. In particular, a wizard case is predefined with goals and related sub-processes which reoccur repeatedly in several wizards. The variable tasks between the predefined processes can be added by clerks at runtime. In other words, instead of designing a full-blown wizard case at design time, the case is partly defined at design time and completed with ad hoc tasks at runtime.

3.2 Design of Compliance Rules Enabled Wizard Cases

The results of the analysis and design of the MKS processes are displayed in Fig. 4. Tasks originally contained in MKS processes are anonymized with capital letters. A typical wizard process can be divided into three parts: beginning, middle and ending. The beginning part is for retrieving information of the insured person and the insurance product selector. The ending part is the definition of the document delivery channel whereas the middle part is dealing with steps necessary for the specific insurance case. A significant amount of processes have the same beginning part *(A, B, C)* and an ending part *(X, Y, Z)*. The middle part of each process is quite distinct from the others, such as *(K, F, D)* or *(K, L, G, O)*, although they might have common tasks, for example K and O.

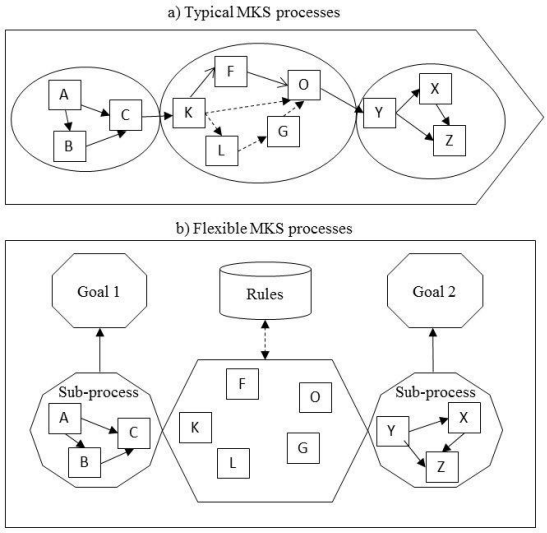

Fig. 4. (**a**) Typical MKS processes and (**b**) Flexible MKS processes

In our approach, the typical wizard processes are transformed into flexible processes using a goal-driven structure. The beginning and ending parts are modeled as predefined sub-processes, which can be reused in various cases. The middle part is split into several single ad hoc tasks. A case holding the restructured processes is driven by the two goals: *Goal 1* is linked to the beginning sub-process and *Goal 2* to the ending sub-process. The tasks of the middle part are either related to *Goal 1* or *Goal 2* or could be assigned by the clerks to newly defined goals. If the tasks of the middle part have to follow a certain execution sequence, they are controlled by the compliance rules associated with the affected tasks. In other words, the rules express the middle parts without rigidly predefining the processes. Based on these rules, during the execution of the middle part, the consistency checking system supports clerks adapting to a specific situation by suggesting the tasks that temporarily violate a certain rule. If certain tasks are optional and have to follow no specific sequence, they can be just added by the business users when needed as no rules would be violated. The same rules apply when business administrators define process templates and thus, enforce compliance also during design time.

3.3 Constraint Definitions by Compliance Rules

In order to express the middle part of a wizard process, we support two kinds of rules: state-based rules and data-based rules.

State-based rules. State-based rules define the sequences of tasks based on the states of tasks**.** These states are *"started"*, *"finished"* and *"running"*. For example, the sequence from Task *K* to Task *F* can be described by a rule: "*Task F can be started only after Task K was finished"*. This rule is expressed in the constraint language as

```
Constraint No1 for MobiliarCase{
  K.finished leads to F.started
  }
```

In the above example the states are *finished* and *started* of tasks *K* and *F*, respectively. The temporal pattern of type precedence is defined by the keyword *leads to*.

To define the temporal patterns in more detail, we use the following temporal expressions:

- Existence: K.finished *occurs*
- Absence: K.finished *never occurs*
- Response: K.finished *leads to* F.started. It means, if K.finished happens, F.started must follow eventually.
- Precedence: K.finished *precedes* F.started. It means, if F.started shall happen, K must happen first.

Data-based rules. Data-based rules allow business users to define task dependencies related to data conditions. State-based and data-based rules can be combined to ex-

press a compliance requirement. E.g. Task *F* must be started only when Task *K* is finished and the value of a certain data attribute, such as the birth year of a customer, is less or greater than 1981.

```
Constraint No2 for MobiliarCase{
   (K.finished and CustomerBirthyear >= 1981) leads to F.started
   }
```

Access to data might not always be available when business users have to deal with unpredictable situations as the underlying data models might not yet support the needed data. In order to support flexibility in such situations without the need for explicit data definitions by IT, the checking of conditions can be done manually by business users using voting tasks which will be guarded by compliance rules. Let us assume a voting task called *"Inquire additional customer interests"* which must be preceded before the final pricing can be finished:

```
Constraint No3 for MobiliarCase{
   InquireInterest.approved leads to Pricing.finished
   }
```

The voting task *InquireInterest* is used to quickly adapt to a new situation. The business administrator can create the task template without the support of database experts or IT people and specify with checklists which items have to be verified with the customer. Alternatively the business user defines the checklist at runtime to even more adapt to the current situation. Unstructured data is quite popular in real-life systems since data definitions cannot be amended short term in IT systems following bureaucratic change management cycles. For example, in a car insurance case, the result of the investigation on whether the car was damaged intentionally or by an accident can return the value by a voting task decided by the clerk.

4 Results Achieved

To demonstrate the results achieved in our approach, we experimented with the MKS wizard case for *"Annahmebestaetigung Antrag"* (*"Acknowledgement of Application"*). In this insurance case, a clerk creates a document confirming the application submitted by an insurance customer. First, some identification numbers, such as the insurance ID, customer ID or case ID, are entered to the system by the clerk. The customer's data is retrieved by the predefined process through web service tasks from different sources based on the entered IDs. After that, the clerk selects the matched insurance holder and address. Specific information for this insurance case is inserted. A document confirming the acceptance of the insurance application is generated and sent to the customer, based on the output channel, determined by the clerk at the end of the process.

The redesigned wizard process of the *Acknowledgement of Application* case is divided into three parts as shown in Fig. 4. The first and last parts are linked to goals owning predefined processes as no flexibility is needed and which are shared between

several wizards. The flexible part is contained in the middle where the sequence of the tasks is determined by the compliance rules introduced by our approach.

As mentioned, the MKS is built based on the ISIS Papyrus ACM framework. Processes and tasks are reusable components of the ACM system, to be shared with other goals and cases. The sequence of the tasks in the sub-processes is modeled with transitions and gateways following BPMN standards. The tasks of the middle part are ad hoc tasks selected by the clerks at runtime. Each of these tasks can be added to the case, when the clerk sees the need, based on content or the context of the case. Their order of execution is not predetermined, but controlled by rules.

Before the redesign an ACM wizard case was completely driven by a predefined process containing all the steps of the wizard. On the contrary, in our enhanced approach the redesigned ACM wizard case is driven by goals. The wizard process of the *Acknowledgement of Application* case is divided into three parts. The first and last parts are linked with goals holding predefined sub-processes. The sequence of the tasks of the middle part is determined by the rules introduced in our approach. Thus, the redesigned *Acknowledgement of Application* ACM case contains goals, processes and tasks as follows: *First Core Goal* holds the beginning part of the process that is configured as a sub-process for retrieving insurance customer data from the database. The *Last Core Goal* holds the ending part of the wizard process for choosing the delivery channel of the document to the customer, such as by post or by email. The tasks of the middle part are not predefined in the wizard template, but will be added by the business user at runtime among others that may be needed to satisfy the specific customer situation. The ad hoc task templates are prepared as "*100002 Produkt manuelle Eingabe"* (*"manual input product"*) and "*104009 Annahmebestaetigung Antrag"* (*"acknowledgement of application"*).

The process model of the middle part is controlled by three compliance rules. The first rule *R0* expresses the need that Task *acknowledgement of application* must be present at least one time. *R1* defines the dependency of Task *acknowledgement of application* from Task *manual input product* when the *selection product* is *manual input*. The second rule expresses the dependency of Task *acknowledgement of application* from Task *selection output channel* when the *selection product* is not *manual input.*

```
Constraint R0 for MobiliarCase{
   acknowledgement_of_application.started occurs at least 1x
   }
Constraint R1 for MobiliarCase{
   (acknowledgement_of_application.finished and selection_product
   equal to "manual_input") leads to manual_input_product.started
   }
Constraint R2 for MobiliarCase{
   (acknowledgment_of_application.finished and selection_product not
   equal to "manual_input") leads to selection_output_channel.started
   }
```

The two data-based rules *R1* and *R2* can be alternatively expressed by using a voting task to check whether the selection product is manual input. The voting task is named *check_selection_product_manual_input.*

```
Constraint R3 for MobiliarCase{
   (acknowledgment_of_application.finished leads to
   check_selection_product_manual_input.started
   }
Constraint R4 for MobiliarCase{
   check_selection_product_manual_input.approved leads to manu-
   al_input_product.started
   }
Constraint R5 for MobiliarCase{
   check_selection_product_manual_input.denied leads to selec-
   tion_output_channel.started
   }
```

Compliance rules are composed by the Papyrus rule editor in natural language as shown in Fig. 5. To facilitate the composition, the editor provides elements named by business terms as used by the language of the business administrators and allows to intuitively define the rule with auto-completion features as the users type.

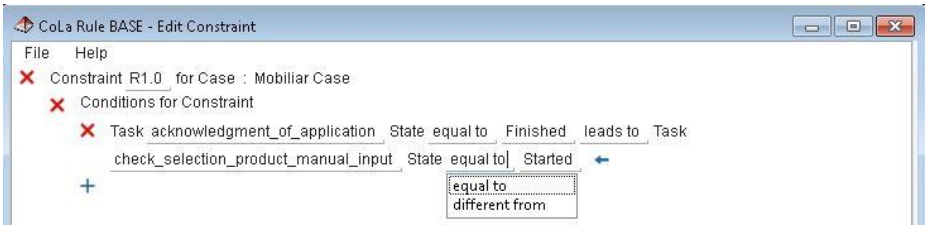

Fig. 5. Compliance Rule Editor

At runtime, when a clerk needs to create a confirmation for a customer who submitted an application, an instance of the *acknowledgement of application* case is instantiated as soon as the template was selected by the clerk.

The *First Core Goal* is processing offering a form to the clerk to enter the insurance ID, customer ID or case ID. When all predefined tasks of the first core goal are finished, the clerk adds Task *acknowledgement of application* as suggested by the consistency checking system, which is used to create a confirmation for an application, as shown in Fig. 6.

Although Task *acknowledgement of application* is suggested to the clerk because of Constraint *R0* being temporarily violated, the clerk can do other tasks as well. However, as soon as an ad hoc task related to the compliance rule constraints is added, it will be controlled by the consistency checking system. The constraint defining the occurrence of tasks, like Constraint *R0*, is used to ensure the task initiating the variable middle parts, like the task *acknowledgement of application*, is not missing.

Therefore, to complete a case successfully, the clerk eventually needs to execute that task.

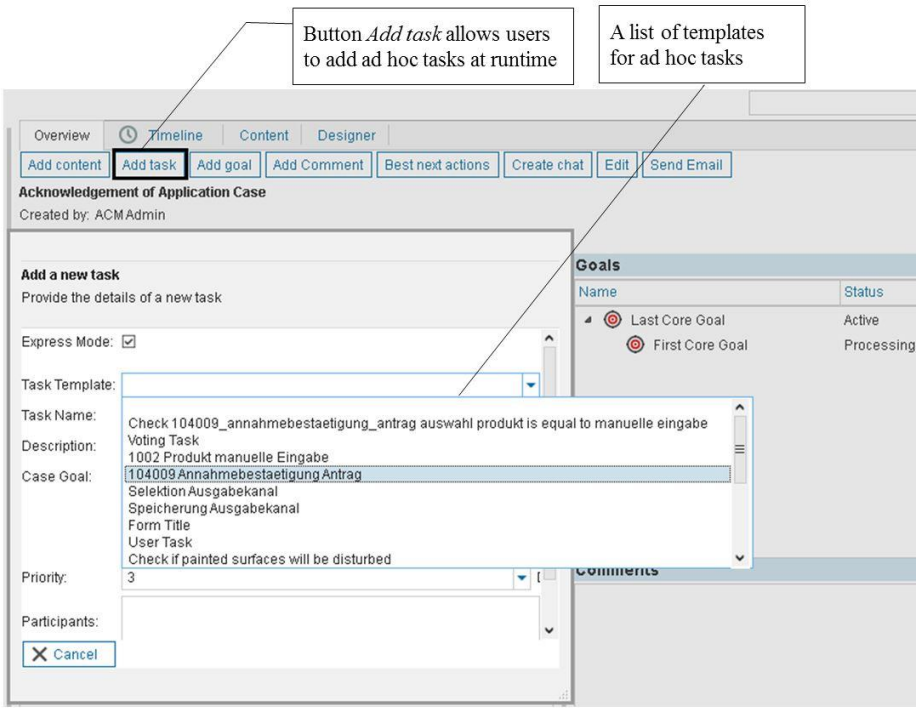

Fig. 6. Add an ad hoc task at runtime

When Task *acknowledgement of application* is finished, the constraints *R1* and *R2* are investigated by the consistency checking system. Since Task *acknowledgement of application* is finished and if the *selection product* is chosen as *manual input*, Task *manual input product* is suggested to the clerk by the consistency checking system. If *selection product* is not chosen as *manual input*, Task *selection output channel* is suggested to the clerk. This way, the execution of the ad hoc tasks is controlled by the consistency checking system. When there are no more suggestions, the clerk can continue the steps defined in the *Last Core Goal*. When the goal is reached, a confirmation document for the application is generated and the case is closed.

In conclusion, ad hoc tasks added at runtime can be controlled by compliance rules. Some of them have to be executed in a certain order, which has to be defined by the compliance rules. Others are controlled by rules, so they could be freely added by clerks. For example, Task *add additional information* can be added by the clerk when additional information is needed in the document. Therefore, our approach enables clerks to add ad hoc tasks at runtime under the control of the compliance rule system based on the current context, which could not be foreseen by an initial wizard design.

5 Lessons Learnt

Wizard processes for generating insurance documents are predefined at design time and executed sequentially at runtime. The configuration of wizard processes can ensure the compliance of insurance documents when compliance regulations are known and being followed during the design process. However, rapidly changing insurance markets require new types of documents day by day and demand business users to react spontaneously to client requests.

Although predefined processes are appropriate for automation where sufficient process knowledge exists, they might hinder innovation and business agility that is critical in insurance markets. The clerks who directly face specific insurance situations should have flexibility to adapt the case at runtime. However, the operating principle of wizards does not support adapting to unpredictable circumstances. Instead, the clerks have to report change requests to business administrators and wait for the release of a new wizard.

Enabling the flexibility of wizard cases along with compliance rules is a challenge addressed by our approach. Existent rigid wizard processes of the Swiss insurance company *Die Mobiliar* could be easily amended to enable ad hoc changes at runtime. Supported by compliance rules, the ad hoc actions are performed in the necessary order, similar to the corresponding predefined process model. With the flexibility supported at runtime, clerks can respond immediately to the requirement of their clients. Thus, the business strategy can reflect holistically customer orientation focusing on quality services and customer experience.

Based on compliance rule definitions by business administrators, consistency checking algorithms can be applied at runtime to verify process compliance. When applied also during design time even the consistency of predefined processes including their structural correctness can be verified, which is a big benefit for the change management and release process, reducing test and error correction efforts.

The process redesign can enhance the simplification of the wizard processes. Moreover, this approach can decrease the number of process templates in the library and support flexibility while handling unpredictable insurance cases. Thus, the maintenance effort can be reduced while the performance can be improved by a smaller size of templates in the library.

Acknowledgement. This work was supported by the FFG project CACAO, no. 843461 and the Wiener Wissenschafts-, Forschungs- und Technologiefonds (WWTF), Grant No. ICT12-001.

References

1. Mobiliar: Die Mobiliar, Versicherungen und Vorsorge. https://www.mobi.ch/
2. ISIS, Papyrus: ISIS Papyrus solution catalog - Swiss Mobiliar. http://www.isis-papyrus.com/e14/pages/solutionscatalog/2/solutionscatalog_mobiliar.html
3. Governatori, G., Rotolo, A.: Norm Compliance in Business Process Modeling. Semantic Web Rules - International Symposium, RuleML 2010, Washington, DC, USA, October 21-23, 2010. Proceedings. pp. 194–209 (2010).
4. Tran, T.T.K., Pucher, M.J., Mendling, J., Ruhsam, C.: Setup and Maintenance Factors of ACM Systems. In: Demey, Y.T. and Panetto, H. (eds.) OTM Workshops. pp. 172–177. Springer (2013).
5. Aalst, W.M.P. van der, Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. Web Services and Formal Methods, Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006, Proceedings. pp. 1–23 (2006).
6. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in Property Specifications for Finite-state Verification. Proceedings of the 21st International Conference on Software Engineering. pp. 411–420. ACM, Los Angeles, California, USA (1999).
7. Esper Tech.: Complex Event Processing (CEP). http://www.espertech.com/products/esper.php.
8. Clarke, E.: The Birth of Model Checking. In: Grumberg, O. and Veith, H. (eds.) 25 Years of Model Checking. pp. 1–26. Springer Berlin Heidelberg (2008).