Apto: A MDD-based Generic Framework for Context-aware Deeply Adaptive Service-based Processes

Zakwan Jaroucheh, Xiaodong Liu, Sally Smith
School of Computing,
Edinburgh Napier University, UK
{z.jaroucheh, x.liu, s.smith}@napier.ac.uk

***Abstract*—Context-awareness and adaptability are important and desirable properties of service-based processes designed to provide personalized services. Most of the existing approaches focus on the adaptation at the process instance level [1] which involves extending the standard Business Process Execution Language (BPEL) and its engine or creating their own process languages (e.g. [2]). However, the approach proposed here aims to apply an adaptation to processes modeled or developed without any adaptation possibility in mind and independently of specific usage contexts. In addition, most of the existing approaches tackle the adaptation on the process instance or definition levels by explicitly specifying some form of variation points. This, however, leads to a contradiction between how the architect logically views and interprets differences in the process family and the actual modeling constructs through which the logical differences must be expressed. We introduce the notion of an evolution fragment and evolution primitive to capture the variability in a more logical and independent way. Finally, the proposed approach intends to support the viewpoint of context-aware adaptation as a crosscutting concern with respect to the core "business logic" of the process. In this way, the design of the process core can be decoupled from the design of the adaptation logic. To this end, we leverage ideas from the domain of model-driven development (MDD) and generative programming.**

Keywords-Context-awareness; MDD; adaptive service-based processes; BPEL.

I. INTRODUCTION

Context-awareness refers to the capability of an application or a service being aware of its physical environment or situation (e.g. context) and to respond proactively and intelligently based on this awareness [11]. We define the context-aware process adaptation as *the action that modifies the process in a way that causes process behavior to evolve according to the evolution of business and users' requirements, and the context considered relevant to that process.*

Many different solutions have been proposed by researchers to the problem of context-aware adaptation during process development and provision. However, three main issues could be identified in the existing approaches. Firstly, in many cases the context management and adaptation logic are handled at the code level by enriching the core logic of the service with code fragments responsible for context manipulation or adaptation rules. Significant examples of such approaches are Context Oriented Programming [9], context-aware aspects [10], and VxBPEL [8] which incorporates the variation points and variants inline in the process definition itself (i.e. BPEL code). However, as the service engineering process passes through the stages of analysis and design prior to the actual code development, the context and adaptation should be considered also in these stages.

Secondly, although the structure and behavior of the user centric process can be adapted to contextual information, the overall goal of the process core logic is indifferent to context change. Under this perspective, the adaptation to different contexts can be considered as an almost orthogonal task with respect to the core process logic. The separation of concerns is a promising approach in the design of such context-aware adaptive processes (CAAPs) where the core logic is designed and implemented separately from the context handling and adaptation logics.

Thirdly, process modeling must be flexible enough to deal with constant changes – both at the business level (e.g. evolving business rules) and the technical level (e.g. contextual information and platform upgrades). The flexibility could be provided or addressed by incorporating variabilities into a system [8]. Most of the approaches tackle process adaptation on the process instance or definition level by explicitly specifying some form of variation points. To date, a variety of different adaptation approaches have been proposed for capturing variabilities (e.g. [12]). Common to all these approaches is that they capture the process variant as a monolithic structure containing variation points to differentiate between process family members. By making appropriate choices to resolve the variation points, either at design time or at runtime, a single process variant could be constructed. The problem is that, for example, each task in the process is modeled as a variation point in and of itself, each governed by its own clause to determine inclusion or exclusion. This is in contradiction with how the developer or architect logically views the process variant i.e. in terms of the features that determine the difference between process variants in each usage context. Moreover, managing and understanding the process variants becomes more difficult when the number of variabilities and their relationships increase.

Motivated by these problems and directives in mind, we propose an MDD-based framework called *Apto* (the Latin word for adapt) that introduces the evolution fragment and

evolution primitive constructs to capture the variability in a more logical and independent form. In addition, it aims to tackle context-aware adaptation without interfering with the core functionality of the process. The proposed approach contributes to a solution to automatically generating a customized process based on the context. Another feature is that *Apto* supplies a set of automated tools for generating and deploying executable process definitions e.g. WS-BPEL (OASIS, 2007) which in turn significantly reduces the development cost.

The rest of the paper is structured as follows: Section II describes the proposed conceptual model for context-aware adaptation. Section III and IV describe the configuration and deployment of CAAP. In section V we present the proof-of-concept prototype; and in section VI we illustrate the Apto approach by giving a simple example of an airline booking process. The related work and concluding remarks end the paper.

II. A CONCEPTUAL MODEL FOR CONTEXT-AWARE ADAPTATION

Apto adopts MDD methodology whose primary objectives are: portability, interoperability and reusability. Therefore, software systems abstraction can be specified in platform independent models (PIMs), which are then (semi)automatically transformed into platform specific models (PSMs) using some transformation tool and possibly with some additional information that guides the transformation process.

The traditional process life cycle, as depicted in Fig. 1, consists of three phases, namely the design and modeling of the process, the selection or configuration of a particular process variant, and the deployment of this variant in the runtime environment. As the process may evolve over time there should be a feedback loop during which a process is continuously optimized. In the following subsections we explain the proposed approach in the light of these phases.

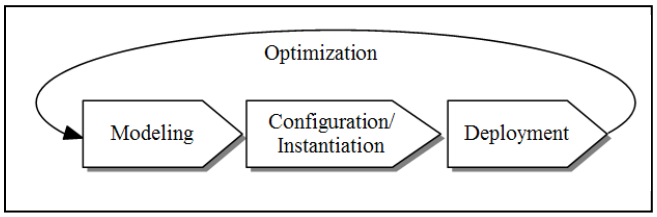

Figure 1. Process life cycle

The proposed conceptual model is structured in four main sections that address, respectively, the modeling of the service-based process, context, evolution, and linkage between evolution and context models (see Fig. 2).

A. Basic Process Model

In *Apto* we denote the original process as a basic process. This can be either an existing process model or a newly created one. The basic process could be defined for the most frequently executed variant of a process family, but this is not a requirement. We use a UML process definition model. For illustration purposes, Fig. 2 depicts some of the main meta-classes representing the key elements of BPEL process model, and their relationships.

B. Context Model

As in previous work [14] the main construct for representing context knowledge is the *ContextPrimitive* which represents the base context constructs (primitives): entity classes, entity attributes and entities associations.

- Entity class: represents a group of entities (e.g. users, places, devices, etc) sharing some properties.
- Attribute class: represents an entity's attributes e.g. preference, position, temperature, etc.

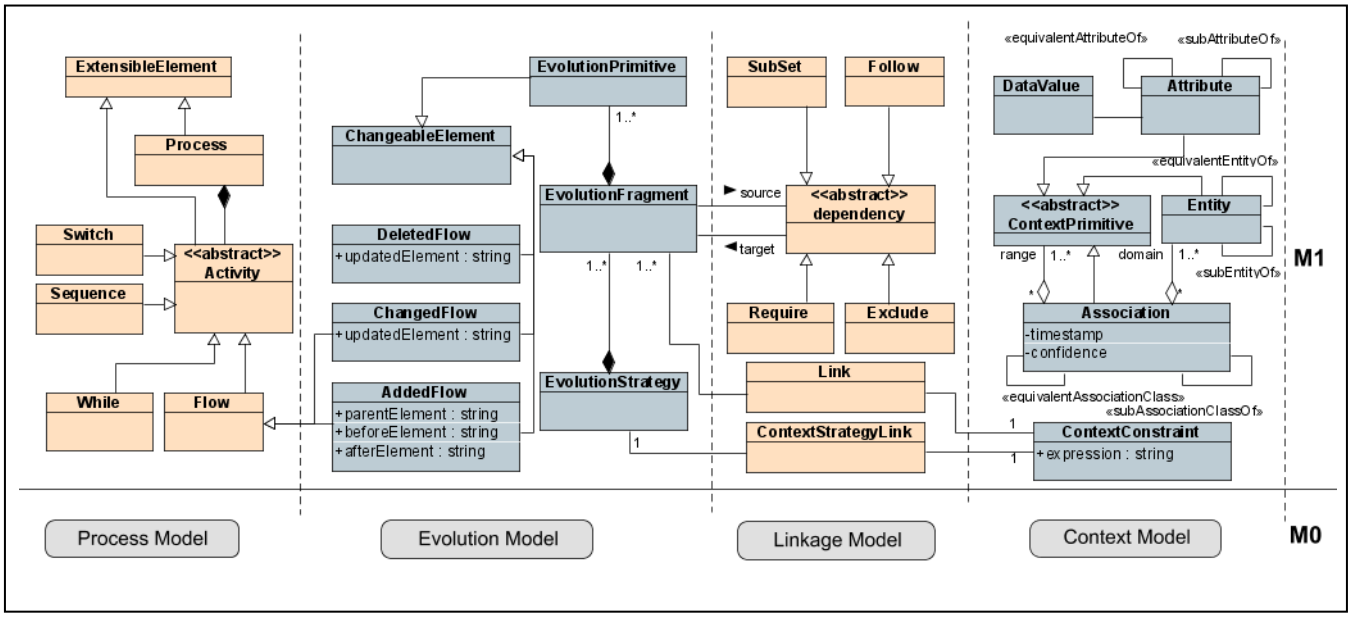

Figure 2. The conceptual model for context-aware adaptation

• Association class: represents a relationship between one entity and either another entity or an attribute.

Further optional modeling constructs are additional facts about the entities and attributes. These are: specialization and equivalence relationships that may be specified between two entity classes, two attribute classes, or two association classes. In addition, we introduce the *context-dependent constraint* concept which allows us to specify conditions that must hold to introduce some kind of context-aware adaptation by specifying the evolution fragments that should be applied to the process as described in the next sections.

The OCL language -an implementation of the Object Constraint Language (OCL) OMG standard for Eclipse Modeling Framework based models- is leveraged to express the constraint expression. We distinguish between two types of expressions:

- *Plain Expressions* which are OCL-based expressions that can be directly evaluated e.g. the expression of the `RainyWeather` context constraint (cf. Section VI).
- *Parameterized Expressions* that *contain* one or more variables whose values must be determined before evaluating them e.g. `ClientIsBrandConscious` constraint (cf. Section VI).

C. Evolution Model

The adaptation in a process usually involves adding, dropping and replacing tasks in the process. In this respect, and in order to achieve deep change ability, we propose to add for each class `X` in the BPEL metamodel three classes: `AddedX`, `DeletedX`, and `ChangedX` describing the difference between the basic process model and the respective variant model (See Fig. 3). Other change types can be mapped to variations and combinations of these ones. For instance, moving an activity is achieved by dropping the activity and inserting it at a later position of the process.

The evolution metamodel (Fig. 2) consists of an `EvolutionStrategy` class that contains one or more `EvolutionFragments`. The `EvolutionFragment` in turn consolidates related Evolution Primitives (a set of elements of type `ChangeableElement`) into a single conceptual variation. Our approach promotes evolution fragments (EFs) to be first-class entities consisting of closely-related additions, deletions and changes performed on the basic process model.

The evolution metamodel could be automatically generated from the BPEL model. One possible approach is to use the ATL transformation language [15] as in the script of Fig. 4. Fig. 2 shows only one example of the three generated classes from the Flow class (`AddedFlow`, `DeletedFlow` and `ChangedFlow`).

D. Linkage Model

Because in the MDD world everything should be a model, the mapping between the context constraints and the EFs will be represented by the linkage model. This mapping will be used as information for driving the model transformation. Moreover, the linkage model is used to represent the dependencies between the EFs which we prefer to keep it separate from the evolution model itself. Dependencies are used to describe relations between EFs in order to constrain their use. Each dependency has at least one source EF and exactly one target EF. The relations supported in *Apto* are as follows: dependency (*Require*), compatibility (*Exclude*), execution order constraint (*Follow*), and hierarchy (*SubSet*). *Require* arises when elements introduced by one EF depends on elements introduced by another. The *Exclude* relationship dictates which EFs are incompatible with each another, based on conceptual design knowledge of the

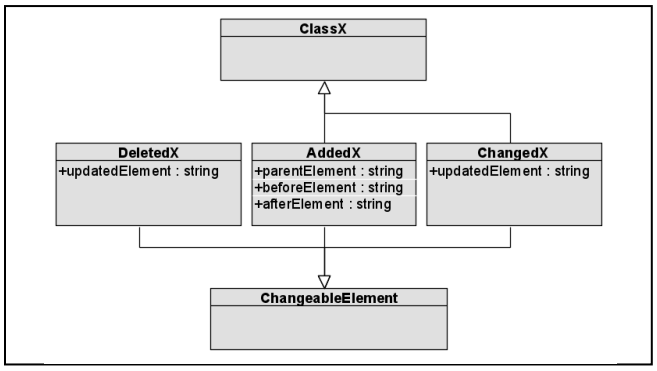

Figure 3. Generating evolution metamodel

```
create OUT : EvolutionMM from IN1 : BPELMM, IN2 :
MinimalEvolutionMM;
helper def: changeableElement: MinimalEvolutionMM!EClass =
MinimalEvolutionMM!EClass.allInstances()->select(i | i.name =
'ChangeableElement');

rule copyMinimalEvolutionMM {
   from s : MinimalEvolutionMM!EClass
   to t: EvolutionMM!EClass (
         name <- s.name,
         interface <- s.interface,
         eSuperTypes <- s.eSuperTypes,
         eStructuralFeatures <- Sequence
{s.eStructuralFeatures}
                  ...
   )
}
rule generateEvolutionMMElements {
   from s : BPELMM!EClass (s.name <> 'Process' and not
s.abstract)
   to t: EvolutionMM!EClass (
         name <- s.name,
         interface <- s.interface,
         eSuperTypes <- s.eSuperTypes,
         eStructuralFeatures <- Sequence
{s.eStructuralFeatures}
                  ...
    ),
      added_element: EvolutionMM!EClass (
         name <- 'Added' + s.name,
         eSuperTypes <- Sequence {t,
thisModule.changeableElement}
    ),
      changed_element: EvolutionMM!EClass (
         name <- 'Changed' + s.name,
         eSuperTypes <- Sequence {t,
thisModule.changeableElement}
    ),
      deleted_element: EvolutionMM!EClass (
         name <- 'Deleted' + s.name,
         eSuperTypes <- thisModule.changeableElement
    )
}
```

Figure 4. Evolution metamodel generation script

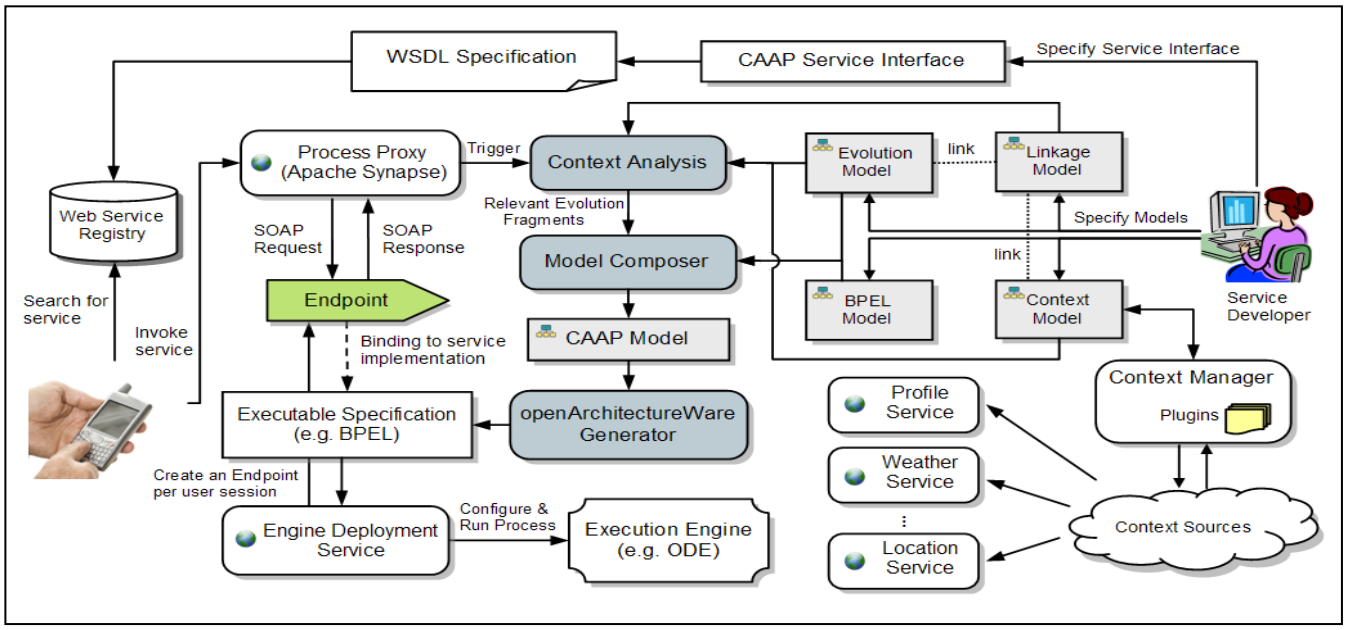

Figure 5. *Apto* Architecture

architect. *SubSet* denotes composition relationship which means that when choosing the child EF the parent EF must be applied first. As one EF might insert an activity whose attributes are changed by a second one, the execution order of these EFs becomes crucial. Therefore, the *Follow* relationship enables the order in which EFs are applied to the basic process to be specified.

III. Process Instantiation/Configuration

The selection of a process variant in a particular context should be done automatically. Therefore the process context in which this selection takes place has to be considered. To this end, the basic process model, the defined EFs, the context and the linkage models are used to configure the models of the different variants. A single process variant is created by applying a number of EFs and their related evolution primitives to the basic process.

Step 1. Select EFs: the EFs that are relevant to configuring a particular variant are selected based on the current values of the context model; i.e., an EF will be selected if all context constraints associated with it –via the linkage model– evaluate to "true".

Step 2. Check EFs relations: EFs relations are considered to ensure process consistency. The selected EFs have to be extended if dependent EFs are missing. Also, it could happen that some of these EFs are mutually exclusive; in this case the process variant cannot be generated. In addition, the EFs are sorted by the order in which they should be applied to the basic process.

Step 3. Apply the EFs: After defining and evaluating the relevant set of EFs, the related evolution primitives are applied to the model of the basic process.

Step 4. Check for consistency: Although the EFs are validated, applying these EFs in combination with each other may result in a deadlock or data inconsistency in the resultant process variant. Therefore, a consistency check is necessary and it is considered for our future work.

We can distinguish here between "instance level changes" that should be made on a user request basis and the "permanent changes" that are due to changes of the regulation or the business rules. In the latter case, *Apto* is flexible enough to accommodate this type of evolution by assigning it to a context constraint always evaluated to true. One of the advantages of this approach is that the evolution in the process definition can be easily documented.

Further, the evolution fragment concept is used to specify the process adaptation during runtime namely the adaptation strategy. But, what about the evolution of the adaptation

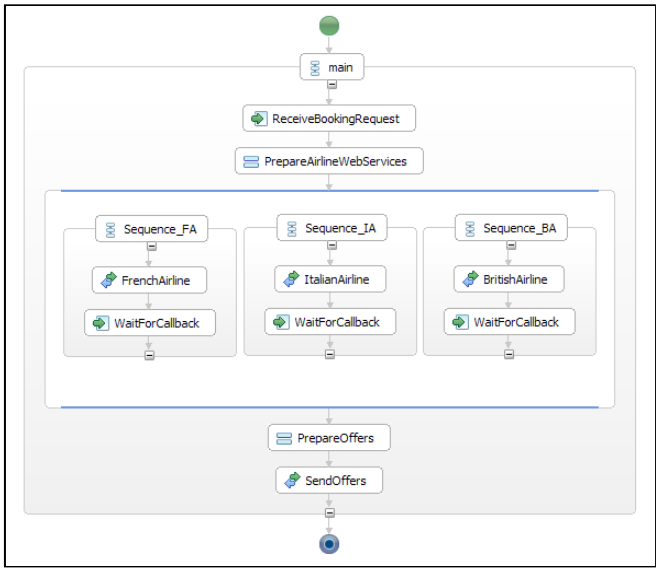

Figure 6. Ticket booking process

strategy? This is the role of the evolution strategy concept. An example of strategy evolution is that the business owner may choose to apply a different adaptation strategy during the Christmas days which require them to eliminate, add or change some activities and later to return to the basic strategy. To this end, the evolution strategy could also be linked to a specific context constraint.

IV. Deployment and Execution

After the configuration and instantiation phase, the resultant variant process model has to be translated into an executable artifact e.g. specified by WS-BPEL. As the context as well as the user and business requirements is in constant change, we retain the evolution and context models in the runtime as well. This gives the ability to switch between variants during runtime.

V. *Apto* Architecture

As a proof-of-concept we implemented an Eclipse-based prototype for the process variant generation. The Eclipse Modeling Framework (EMF) was used to model the aforementioned models. Having specified these models, the *Apto* framework is able to deliver CAAP on a basis of user request as follows (See Fig. 5). The user request for the process service is intercepted by the Process Proxy service which in turn triggers the Context Analysis module. The Context Analysis module evaluates all context constraints of the context model. Using the constraints elements evaluated to "true" and the linkage model we are able to determine the relevant EFs and the order in which they should be applied to the basic process model. We consider that the context model is managed by the Context Manager but due to space limitation we omit further details here.

These relevant EFs are used by the Model Composer module which supports context-aware process configuration; i.e., it allows for the configuration of a process variant by applying only those EFs relevant in the process context. The result is the CAAP Model. This model is automatically transformed, using a set of transformation rules, to generate the executable specification of the target platform. At this time, the proxy service creates a new virtual end point which will be bound to the resulting deployed process. Then it invokes the service deployment of the corresponding execution engine (ODE [18] in our prototype) to deploy the generated process. The client request is then transferred to the new end point; and the client will be provided with a personalized process that takes into account her context and preferences.

For the proxy service, we employed the Apache Synapse [19] which is designed to be a simple, lightweight and high performance Enterprise Service Bus (ESB). One of the key features of Synapse is that it is easily extended via a custom Java class (mediator); therefore the Synapse engine is configured with a simple XML format to use our proxy service as the mediator. This mediator is responsible for coordinating and running all the above-mentioned framework modules. The Context Analysis and Model Composer modules are implemented via a Java application. The engine used to run the process is ODE [18] which is an

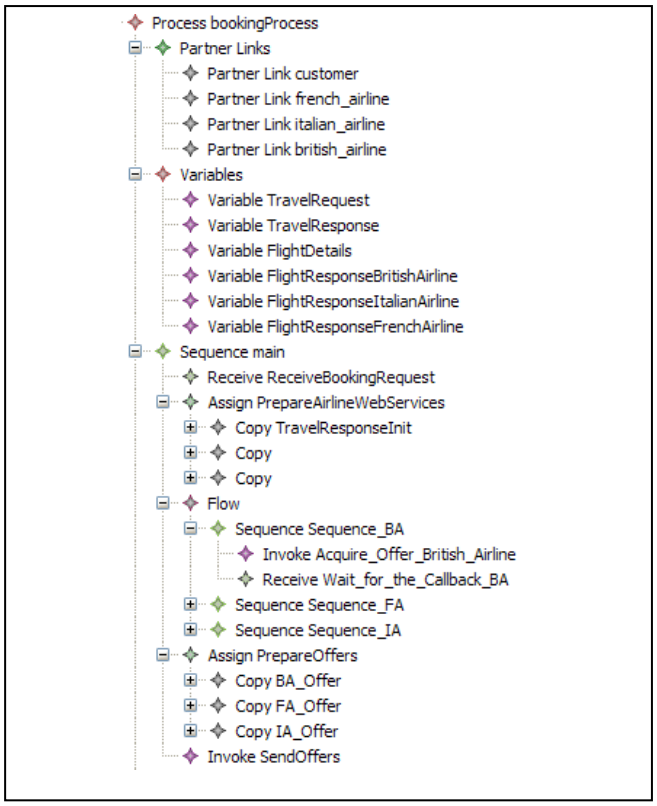

Figure 7. Ticket booking process model

engine for executing processes described using the WS-BPEL 2.0 standard. One possible deployment option that is used in the prototype is to deploy ODE as a simple service in Axis 2 (the Apache Web Services/ SOAP/WSDL engine) which is invoked using plain SOAP/HTTP and deployed in the Tomcat application server [20].

In *Apto*, we use the model-to-code transformation that takes as input the CAAP model and generates code in an executable language (e.g. BPEL). In the literature there are numerous code generation techniques such as templates+filtering, template+metamodel, inline generation, code weaving, etc. [13]. In our prototype, we used the template+metamodel technique – which is realized in the openArchitectureWare framework (oAW) [16] to implement the model transformations. But any of above-mentioned techniques can be utilized in our framework with reasonable modifications.

VI. Case Study

To demonstrate the realization of these concepts, we introduce a simple but realistic case study, namely, a travel agency running an Airline Ticket Booking (ATB) process (see Fig. 6). The BPEL syntax is adopted to model the Booking process, and the graphical notations are borrowed from the Eclipse BPEL Designer environment [17]. We consider a generic service application that travelers can access through a wireless connection using their own portable devices. The application displays a GUI through

which travelers may use ATB services for ticket purchase. Fig. 6 depicts a part of the static structure of this process. The booking process is initiated when the process's customer issues an airline ticket offer request. The request is received by the "Receive Request" activity. The process then invokes three services to get three offers for different companies. The airline service only needs some necessary information such as the `OriginFrom`, `DestinationTo`, `DepartureDate` and `ReturnDate`. The process performs a preparation step that extracts this information from the user request. Finally, the booking offers are prepared and sent back to the customer during the last step, `SendOffers`. After that, the booking process successfully finishes.

The agency manager might want to customize this process by adding some context-aware enhancement to the process. For example, the ATB process could be enhanced by automatically filling in the `ClientType` parameter, using for this purpose information provided by an existing User Profile service. Being a brand conscious customer means that the customer is not interested in getting several offers from different companies. Therefore there is a need to change the process structure so that the activities that invoke, for example, the `FrenchAirline` and `ItalinaAirline` are deleted. Moreover, should the weather be rainy and depending on the time left before plane departure, a new pickup to the airport activity may be added after booking the tickets.

In the next paragraphs, we describe the simple process life cycle as follows. Firstly, the models of the ATB process, the context, the evolution strategy, and the linkage models are designed based on the proposed meta-models (See Fig. 2). Secondly, the relevant EFs are applied to the process model. And finally, the newly created CAAP model is used to generate executable code in BPEL that can be deployed into any BPEL engine. Fig. 7 depicts the graphical representation of the ATB process model.

Fig. 8 shows a simple example of the context model that contains three entities: two customers; Alice and Bob; and the Weather entity. The association elements assign the attributes to the entities so that Alice has an attribute `ClientType` whose value is `PriceConsious` whereas Bob's `ClientType` is `BrandConscious`. The context constraint named `RainyWeather` is an example of the constraints having *plain expressions*. Whereas, the constraint `ClientIsPriceConsious` uses a *parameterized expression*. It contains a variable named $`UserName`. The value of the parameter is extracted either from the user request information or from any other data source. In either case the above-mentioned proxy service is responsible for assigning the variables' values.

For the sake of simplicity, the linkage model example contains one link element that links between the context constraint named `ClientIsBrandConscious` and the EF named "ef1".

Being a brand conscious customer means that the sequence activities responsible for invoking the French and Italian airlines should be deleted. This means deleting the Copy activities that copy the company offers to the resulting

```
<ctxt:ContextModel xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:ctxt="http://napier.ac.uk/context">
  <associations name="weather_attributes"
entities="//@entities.2" attributes="//@attributes.0
//@attributes.1"/>
  <associations name="Alice_attributes"
entities="//@entities.0" attributes="//@attributes.2"/>
  <associations name="Bob_attributes"
entities="//@entities.1" attributes="//@attributes.3"/>
  <entities name="Alice"/>
  <entities name="Bob"/>
  <entities name="Weather"/>
  <attributes name="Temperature" value="20"/>
  <attributes name="RainLikelihood" value="90"/>
  <attributes name="ClientType" value="PriceConsious"/>
  <attributes name="ClientType" value="BrandConsious"/>

  <contextconstraints expression="associations->select(a |
a.entities->exists(e | e.name='Weather') and a.attributes-
>exists(a1 |a1.name = 'Temperature' and a1.value='5') and
a.attributes->exists(a1 |a1.name = 'RainLikelihood' and
a1.value='80'))" name="RainyWeather"/>

  <contextconstraints expression="associations->select(a |
a.entities->exists(e | e.name='$UserName') and a.attributes-
>exists(a1 |a1.name = 'ClientType' and
a1.value='PriceConsious'))" name="ClientIsPriceConscious"/>

  <contextconstraints expression="associations->select(a |
a.entities->exists(e | e.name='$UserName') and a.attributes-
>exists(a1 |a1.name = 'ClientType' and
a1.value='BrandConsious'))" name="ClientIsBrandConscious"/>
</ctxt:ContextModel>
```

Figure 8. The context model

```
<linkage:LinkageModel xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:linkage="http://napier.ac.uk/linkage">
  <links name="l1"
contextConstraintName="ClientIsBrandConscious"
changeFragmentName="ef1"/>
</linkage:LinkageModel>
```

Figure 9. The linkage model

```
<es:EvolutionStrategy xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:es="http://napier.ac.uk/es">

<evolutionFragments name="ef1">

 <children xsi:type="es:DeletedSequence"
updatedElement="Sequence_FA"/>
 <children xsi:type="es:DeletedSequence"
updatedElement="Sequence_IA"/>
 <children xsi:type="es:DeletedCopy"
updatedElement="FA_Offer"/>
 <children xsi:type="es:DeletedCopy"
updatedElement="IA_Offer"/>
 <children xsi:type="es:ChangedCopy"
updatedElement="TravelResponseInit">
     <to variable="..." part="offersData"/>
     <from literal="..."/>
 </children>

</evolutionFragments>

</es:EvolutionStrategy>
```

Figure 10. The evolution strategy model

offer list. Therefore, two elements of type `DeletedSequence` are added. In addition, the variable `TravelResponse` which is initialized to have a three-element list should now be initialized to contain just one element. Therefore, an element of type `ChangedCopy` is

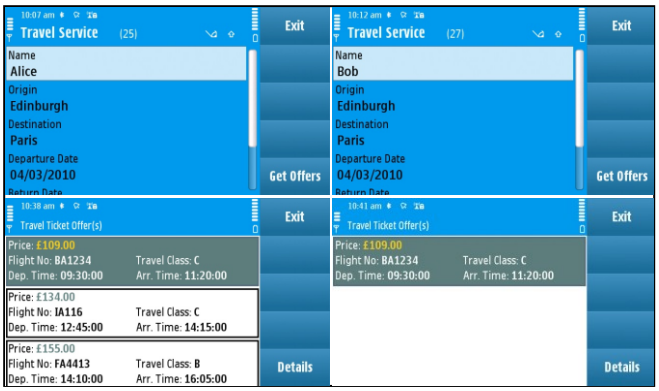

Figure 11. The result of invoking the CAAP

added which will replace the Copy activity named `TravelResponseInit`. Each of these deletions and changes is considered an evolution primitive element; and all these elements should be regrouped into one evolution fragment named "ef1" which should be applied when the customer is brand conscious (Fig. 10). Obviously the variables and partner links related to the irrelevant airline invocation should also be removed; to make the example as compact as possible we omit these evolution primitives from the evolution model in Fig. 10. Finally, Fig. 11 shows the result of running the prototype and the context-aware processes delivered to Alice and Bob respectively.

VII. DISCUSSION

Apto may provide several benefits as follows: firstly, besides the advantage of splitting a single process design task into hopefully simpler subtasks, this approach may provide the possibility of "plugging" more easily within the same basic process different adaptation logic tailored for different contexts. It can result in adaptive context-aware processes developed independently of specific usage contexts, where context information and adaptation rules are efficiently handled outside the main process logic.

Secondly, the software reusability principle is respected thanks to the reusable evolution fragments and the inheritance relations between them.

Thirdly, in the context of the Software as a Service (SaaS) [12], a new delivery model for software, the service provider provides the same application or process for several different customers. However, each individual customer has different requirements for the same application logic. In order to allow each customer to customize the process application to their specific needs the application needs to provide a set of architectural variants (i.e. evolution fragments) that can be determined by customers by specifying their preferences. Further, *Apto* could be easily extended to perform the adaptation on any software system as long as it has been modeled. That is because the generation approach of the evolution metamodel is rather generic.

On the other hand, in order to achieve the possibility of making deep changes we intend in our future work to extend the *Apto* idea to regroup different process views' models. Indeed, as the number of services or processes involved in a process grows, the complexity of developing and maintaining these processes also increases. One of the successful approaches to managing this complexity is to represent the process by different architectural views [5]. Examples of these views are collaboration view, information view, orchestration view etc. The idea is to give the developer the possibility of applying the necessary evolution fragments in each view and then the automated tool verifies the integrity of the changes and generates the adapted process variant artifacts accordingly.

VIII. RELATED WORK

Our work can be viewed from different perspectives e.g. context-aware process adaptation, managing process variants, dynamic process configuration, or customization of SaaS process-based applications perspectives.

AO4BPEL [4], is an aspect-oriented extension to BPEL. In AO4BPEL, the business logic is treated as the main concern in workflows, while crosscutting concerns, such as data validation and security, are specified using workflow aspects in a modular way. By their nature, evolution fragments are close to aspects in their compositional capability. However, unlike *Apto*, there is a need to modify the BPEL engine to support aspects before and after executing each activity.

In the context of SaaS, [12] presents an approach that allows the generation of customization processes out of variability descriptors. Variability descriptors can be used to mark variability by defining variability points in the process layer and related artifacts of a SaaS application. The *Apto* approach is different in the way it presents the variation points and variants. It regroups the different variants into more abstract and meaningful constructs to ease the adjustments of the basic process.

Another interesting work that is similar to our work is the Provop approach [6], which provides a flexible solution for managing process variants following an operational approach to configure the process variant out of a basic process. This is achieved by applying a set of well-defined change operations to it. However, *Apto* deviates from Provop in that it uses the MDD approach and defines the evolution fragments as evolution model elements not as change operations.

Choi et al. [7] propose an adaptation approach in a pervasive environment to support the modification of workflow at runtime. Each service is modeled as a sub workflow which can be inserted into the main workflow. If the context conditions are satisfied, that service will be executed. Like *Apto*, the adaptation takes place at the workflow definition level and is reflected in the running instance. However, their approach may not be sufficient to derive workflow variant; that is because this may involve rolling back executed tasks or adding new activities. They consider only the activities that will be executed but not the activities that have already been executed.

Muller et al. [3] propose "AgentWork", an interesting approach for workflow adaptation to customize the hospital

cancer treatment workflow to suit each patient's medical profile by adding and deleting tasks in the running workflow instance according to the predefined extended ECA rules [3]. The adaptation in this approach provides dynamic and automatic workflow adaptations and suggests and implements a predictive adaptation strategy. *Apto*, on the other hand, takes another approach so that adaptation can be applied to processes modeled and developed without an adaptation possibility in mind and independently of specific usage contexts.

VxBPEL [8] is an adaptation language that is able to capture variability in processes developed in the BPEL language. VxBPEL provides the possibility to capture variation points, variants and realization relations between these variation points. Defining this variability information allows capture of a family of processes within one process definition and switching between these family members at run-time. Unlike *Apto*, VxBPEL works on the code level and the variants are mixed with the process business logic which may add complexity to the process developer task. Further, unlike the generative approach of *Apto*, VxBPEL is specific to the BPEL language.

IX. CONCLUSION

Change is the only certainty in the software/service development world due to the evolution in business or user requirements. Therefore, there is a need to customize processes by generating a process variant that corresponds to the change in the business and user requirements. We have described the *Apto* model-driven approach for managing and generating process variants. One of the advantages of using MDD is that the context management and adaptation logic are included in models rather than directly implemented in code. Based on logically-viewed well-defined evolution fragments and evolution primitive constructs; on the ability to group evolution fragments in reusable components; and on the ability to regroup these components in a constrained way, necessary adjustments of the basic process can be correctly and easily realized when creating or configuring a process variant.

We have adopted the viewpoint that this kind of adaptation can often be considered as a crosscutting concern with respect to the core application logic. Hence, one of our main goals has been the decoupling of the design and implementation of the adaptation logic from the design and implementation of the main process logic. Finally, *Apto* allows for the dynamic configuration of process variants based on the given process context. Our future work involves tackling the correct combination of evolution fragments when creating a variant. Sophisticated techniques are needed to prevent errors (e.g., deadlocks) or other consistency problems.

REFERENCES

[1] S. Smanchat, S. Ling, and M. Indrawan, "A survey on context-aware workflow adaptations," Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia - MoMM '08, 2008, p. 414.

[2] M. Adams, A.H. Hofstede, D. Edmond, and W.M. Aalst, "Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows," On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, 2006, pp. 291-308.

[3] R. Muller, U. Greiner, and E. Rahm, "AW: a workflow system supporting rule-based workflow adaptation," Data & Knowledge Engineering, vol. 51, 2004, pp. 223-256.

[4] A. Charfi and M. Mezini, "AO4BPEL: An Aspect-oriented Extension to BPEL," World Wide Web, vol. 10, 2007.

[5] H. Tran, U. Zdun, and S. Dustdar, "View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA," Internaltional Working Conference on Business Process and Services Computing (BPSC'07), 2007, pp. 105-124.

[6] M. Reichert, S. Rechtenbach, A. Hallerbach, and T. Bauer, "Extending a Business Process Modeling Tool with Process Configuration Facilities: The Provop Demonstrator," BPM'09 Demonstration Track, Business Process Management Conference, Ulm, Germany, 2009.

[7] J. Choi, Y. Cho, K. Shin, and J. Choi, "A Context-Aware Workflow System for Dynamic Service Adaptation," Computational Science and Its Applications – ICCSA 2007, Kuala Lumpur, Malaysia: Springer Berlin / Heidelberg, 2007, pp. 335-345.

[8] M. Koning, C. Sun, M. Sinnema, and P. Avgeriou, "VxBPEL: Supporting variability for Web services in BPEL," Information and Software Technology, vol. 51, 2009, pp. 258-269.

[9] R. Keays and A. Rakotonirainy, "Context-oriented programming," Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access - MobiDe '03, 2003, p. 9.

[10] E. Tanter, K. Gybels, M. Denker, and A. Bergel, "Context-Aware Aspects," 5th International Symposium on Software Composition, Springer, 2006, pp. 227-242.

[11] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," Int. J. Ad Hoc and Ubiquitous Computing, vol. 2, 2007.

[12] R. Mietzner and F. Leymann, "Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors," 2008 IEEE International Conference on Services Computing, 2008, pp. 359-366.

[13] M. Volter and T. Stahl. Model-Driven Software Development: Technology, Engineering, Management. Wiley, 2006.

[14] Z. Jaroucheh, X. Liu, and S. Smith, "CANDEL: Product Line Based Dynamic Context Management for Pervasive Applications," International Conference on Complex, Intelligent and Software Intensive Systems (ARES/CISIS 2010), IEEE Computer Society Press, 2010.

[15] ATL Language http://www.eclipse.org/m2m/atl/

[16] http://www.openarchitectureware.org.

[17] Eclipse WS-BPEL Project. http://www.eclipse.org/bpel

[18] Apache ODE http://ode.apache.org/user-guide.html

[19] Apache Synapse (ESB), http://synapse.apache.org/

[20] Apache Tomcat, http://tomcat.apache.org/