Exploiting Rules and Processes for Increasing Flexibility in Service Composition

Brahmananda Sapkota
Department of Electrical Engineering
Mathematics and Computer Science
University of Twente
The Netherlands

Marten van Sinderen
Department of Electrical Engineering
Mathematics and Computer Science
University of Twente
The Netherlands

***Abstract*—Recent trends in the use of service oriented architecture for designing, developing, managing, and using distributed applications have resulted in an increasing number of independently developed and physically distributed services. These services can be discovered, selected and composed to develop new applications and to meet emerging user requirements. Service composition is generally defined on the basis of business processes in which the underlying composition logic is guided by specifying control and data flows through Web service interfaces. User demands as well as the services themselves may change over time, which leads to replacing or adjusting the composition logic of previously defined processes. Coping with change is still one of the fundamental problems in current process based composition approaches. In this paper, we exploit declarative and imperative design styles to achieve better flexibility in service composition.**

I. Introduction

Modern enterprises document their business processes using modeling tools in order to understand and guide manage the execution of their business activities efficiently. Such efforts may include cross-organizational business processes, as enterprises often need to collaborate to provide solutions to their customers which are not achievable by any of the enterprises alone. Software systems form an integral part of most enterprises to support and realize these business processes. One of the major challenges faced by enterprises is to cope with changes and to maintain the alignment between business processes and software support, while external developments force them to introduce new business strategies, formulate new business goals, reconsider requirements, and adopt new technologies.

Enterprise Architecture (EA) has evolved as an established discipline to handle the complexity of business processes in relation to their software support, by providing concepts and architectural frameworks [1], [2] that help to bridge between concerns at business level and IT level. While EA also aids to address change, the support for this is mainly at an organizational or management level [3], not from a technical perspective in terms of modeling and software architecture styles.

Because of the changes that occur at different level, as mentioned above, a technical solution that can flexibly support these changes are required. Some of the these changes can be addressed easily by designing the system in a declarative style while other changes can be addressed by designing the system in an imperative style. We exploit declarative and imperative design styles to achieve better flexibility in service composition. We follow service oriented architecture principles to establish interconnection between declarative and imperative design styles when needed.

Service oriented architecture allows to encapsulate reusable application functionality as loosely coupled composable software services. It is being considered as a favourable paradigm for architectural design and implementation of collaborative distributed systems. In the scope of this paper, we assume that service composition techniques for software systems could be applied in the context of business services.

In service composition, services provided by various organisations can be connected to create composite services through a process called service composition. One of the important aspects of service composition is that it supports reusability and hence significantly reduces development costs and increases operational agility [4]. Service composition is generally defined on the basis of business processes in which the underlying composition logic is guided by specifying control and data flows. The flow of control is normally guided by the use of business rules to define constraints and influence the way a service is composed.

While defining a composition logic, rules are used for addressing frequently changing requirements and processes for addressing fairly static requirements. The identification and separation of processes and rules is done by the application designer based on the problem specification, generally stated as a set of requirements, provided by the domain expert. Such separation is primarily aimed at identifying frequently changing and fairly static requirements to reduce management and implementation costs.

Current approaches specify the overall composition-logic in business processes and business rules (e.g., decision rules) [5], [6], [7]. A common practice is to use rules to specify frequently changing requirements and to constrain control flows embedded in the business processes. Embedding rules into a process makes it difficult to adapt changing requirements and keeping the rules consistent without changing the composition logic [8], [9], [10]. Adaptation of changes may requires updates in rules or redesign of processes which may lead to remodification of composition logic. Such remodification or updates increases the management as well as the implementation cost in proportion to the complexity of the system. Moreover, interleaving processes and rules also makes their reuse difficult.

One of the daunting challenges of service composition is to deal with situations where changes in user preferences and service behaviours are either context dependent or may not be known at design time [11], [12]. These changes can occur more frequently than expected at design time, making it more difficult to adapt them to the composition logic. Designers, however, are interested in methods and technologies that allow them to reuse existing design parts into a new design. Such a reusable design should be flexible enough such that any future changes can be integrated easily into the existing design. This indicates that the reusability and flexibility issues of a design are proportional to the level of support available for requirements evolution. Hence it is imperative to find an approach that supports requirements evolution and allows to deliver corresponding design solutions.

To provide support for adaptability, reusability and increased flexibility, we aim at exploiting business processes and rules in a loosely-coupled fashion. We provide a design solution that allow independent changes of rules and composition processes when needed. Allowing changes, updates or replacement of rules at execution time, greatly simplifies tailorability or personalised service composition. We also present a guiding principles that help designers to choose between process-based and rule-based approaches while modelling a service.

We present a simple reminder-application scenario from the home-care domain for motivating the need of and for illustrating the usability of the proposed approach. The home-care domain was chosen because the requirements and their preferences are unique per user [13]. These requirements could also change making previously composed service insufficient to meet their requirements. User requirements in this domain are dependent on their medical conditions and preferences. Both the medical conditions and preferences of the users, however, would change due to ageing and related social activities and status. This calls for a flexible solution to service composition such that maximum reuse of existing composite service can be made while delivering composite services to meet user requirements.

The rest of the paper is structured as follows. An application scenario is presented in Section II, showing the situation where the proposed mechanisms will be useful. In Section IV-B, guidelines for deciding whether to use processes or rules to model different activities and services are presented. Design solution to exploit business process and business rule is presented in Section V. Section VI takes this further, illustrating how the proposed approach can be utilised in service composition. Some of the relevant existing works are analysed in Section VII, describing their strengths and weaknesses in comparison to the work presented in this paper. Finally, Section IX concludes this paper, outlining possible future work and extensions.

II. Application Scenario

A flexible service composition approach is required in situations where possible future changes in requirements are not known *a priori*. We motivate this problem through the following example scenario from the home-care application domain.

John and Linda are a married couple. Despite having various medical conditions that come with advanced age, they still live independently in their private home. They are prescribed to take different medicines at regular times. The times at which John and Linda have to take medicine are different. They both suffer from amnesia, and consequently they have difficulty remembering when and what medicines to take. To help them remember the right time to take right amount of medicines, they need a a "medicine-reminder-and-dispenser" service, which should provide personalised reminders and control the release of right amount of right medicines.

The "medicine-reminder-and-dispenser" (MRD) service can be realised as a composite service, i.e., through the orchestration of more basic services such as a "reminder" service and a "dispenser" service. The reminder service sends a reminder message to a subscribed user at the predefined times (specified at subscription). The dispenser service enables the release of medicines and also monitors whether the medicines have actually been taken. If the medicine has not been taken within a certain time interval (Δt), then the same reminder has to be repeated. If the medicine has been taken, MRD service should stop sending the same reminder.

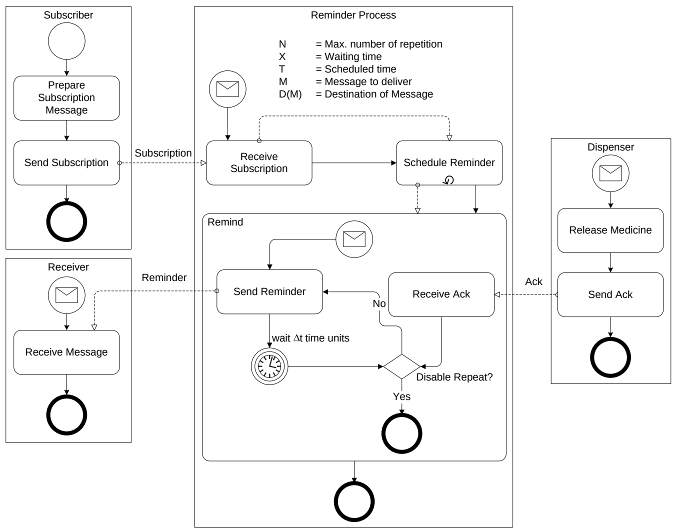


Fig. 1. Reminder Process in BPMN

The implementation details of the MRD service, having the above mentioned behaviour, can be described using Business Process Modeling Notation (BPMN) [14] as shown in Figure 1. The application as specified, sends the reminder message every Δt time units unless an acknowledgement, indicating that the medicine is taken, is received from the dispenser.

A commonly used alternative approach is the use of ECA (Event-Condition-Action) rules. In this approach, a system behaviour is defined as a set of rules that specify which action to take when a certain event occurs. The behaviour of the MRD service can be specified in ECA (roughly) as shown in Listing 1.

```
RULE <MRDRule> [(medicineTime, patientID, timeOut)]

WHEN (currentTime == medicineTime)
   IF (acknowledgement == false)
   THEN sendReminder(patientID)
        enableDispenser(patientID)
        MRDRepeatRule(patientID, timeOut)

RULE <MRDRepeatRule> [(patientID, timeOut)]

WHEN (timer == timeOut)
   IF (acknowledgement == false)
   THEN sendReminder(patientID)
```

Listing 1. ECA rule for MRD

There may be situations in which only the reminder and dispenser services many not be sufficient to fulfill the user requirements. John, for example, can simply ignore the reminder and not take his medicines. Such situations may need to be considered as a hazard situation and an alarm may need to be raised to seek external help (e.g., from a volunteer or a professional). If such a change in situation occurs, the existing solution shown in Figure 1 has to be changed to support new requirements.

The functionality offered by the MRD service, therefore, has to be extended with an "alarm" functionality provided by an "alarm" service. If the acknowledgement has not been received even after sending the reminder N number of times, we assume that the user has ignored the reminder. With this assumption, the implementation details of the MRD service with the new functionality is shown in Figure 2, where the new parts are highlighted with light gray colour.

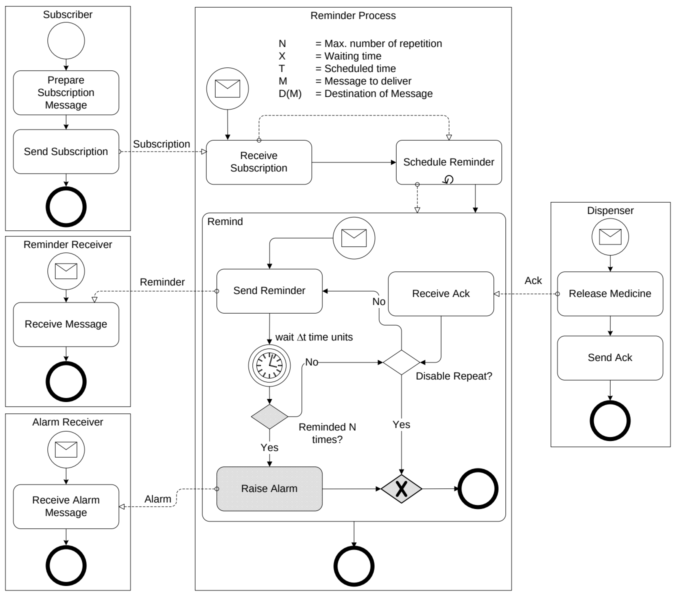


Fig. 2. Reminder Process in BPMN with Alarm

By using rules, this additional functionality can be included by adding a set of rules. The "alarm" functionality can be included by including the ECA rule described in Listing 2 in the existing rule set. In this example, we only showed the set of rules required to enable the required new functionality because the new rules, unlike in process based approaches, can simply be inserted in the rule repository.

```
RULE <MRDAlarmRule> [(receiverID, patientID, reminderCount)]

WHEN (reminderCount >= N)
   IF (acknowledgement == false)
   THEN sendAlarm(receiverID, patientID)
```

Listing 2. ECA rule for MRD service with alarm

A. Motivation

In the previous section, we presented an application scenario and identified a minimal set of services required to build a MRD service. We further specified a composition logic, in BPMN and ECA rules, illustrating how these services could be used to build the required MRD service. With the help of these examples, we motivate the work presented in this paper. The aim is to find what problems arise if the functionalities provided by an existing composite service has to be extended to adapt new (future) requirements. We illustrated this situation by introducing the requirement of "alarm" functionality from the existing MRD service.

In process based approach, extension of functionalities provided by the MRD service with a new functionality (i.e, alarm functionality), the existing processes, shown in Figures 1, required its modification resulting in a new processes as shown in 2. If the requirements are changing quite frequently, process based approaches will be an inappropriate choice because changing process specification is usually time consuming. This is especially the case since a process requires a corresponding change in the implementation which are quite expensive in terms of maintenance and management costs. Moreover, as the system becomes complex, identifying the right place where new changes has to be adapted becomes a challenging task.

While the extension of functionality required new rules, shown in Listing 2 was required in rule based approach, modification of existing rules, shown in Listing 1 was not necessary. In rule based approaches, new rules can be added incrementally as the requirements changes which makes rule based approaches modular. It can be argued that rule generation is a laborious task, however, it is easier to change them. If a change has to be reflected in a process, it is required to understand the whole process to decide which part of the process has to be changed. Therefore, a rule based approach is more flexible than a process based approach. However, as the number of rules increase managing these rules becomes an issue. In addition, unlike in process based approaches, it is difficult to guide or control the flow of activities in rule based approaches because sequence or data flows cannot be specified explicitly. Thus it is necessary to find a mechanism that can exploit complementary features of both process based and rule based approaches to increase flexibility and maximise reusability in a composition approach.

III. Background

Generally accepted approaches to model or to specify service composition are based on processes and rules. Process based approaches focus on defining control and data flows whereas rule based approaches focus on defining statements that constrain or control functional behaviour. In this section, we briefly introduce business processes and business rules which are the underlying approaches and techniques used in the work presented in this paper.

A. Process Based Approaches

Process based approaches specify a system and its behaviour in terms of control and data flows. These approaches focus on defining what sequence of activities/services is required to satisfy a particular user requirement. Different formalisms such as UML activity notation and business process modeling notation (BPMN) are used in process based approached to specify such a sequence.

In process based approaches, control flows are used is to specify the sequence of execution of services. Transfer of information between services is specified using data flows. Since the sequence of activities is the main focus in process based approaches, control and data flows are specified explicitly. Tasks that have to be performed are represented as activities, if something happens then it is represented as an event, split or merge of control flow is represented as gateways in process based approaches. We use BPMN notations to denote these concepts.

B. Rule Based Approaches

Rule based approaches specify a system and its behaviour in terms of constraints, conditions and actions which are usually represented as a set of rules [15]. Business rules can be seen differently from different perspectives. In business world, it is used to refer to directives which are intended to guide or influence business behaviour. In technical world, it is used to refer to reusable business logic which can be expressed declaratively. However, business rules are primarily intended for business world [16] and hence their ownership and management and validation responsibilities also rests with the business world.

Categorically, business rules can be either be seen as derivation, integrity, reaction and production rules [17]. Derivation rules specify rules for deriving new information based on existing information. Integrity rules specify constraints which guide behaviour of the system. Reaction rules specify which action to take while a certain event occurs under a certain condition. Production rules specify what action to generate when a certain condition holds. We use reaction rules and express them in WS-ECA language.

IV. Guiding Principles

A system is designed based on the requirements expressed by the system's stakeholders. These requirements are captured, analysed and expressed in terms of rules or processes by the designer. Depending on the nature of the requirements, a design may be rule-based at one level of design and process-based at another level of the design. Moreover, both rules and processes may be used side by side at the same level because some requirements may be naturally expressed by rules whereas the processes could be a suitable choice to express other requirements. It is useful to know when to use processes and when to use rules to make a design flexible enough to support requirements evolution.

When aiming at system's flexibility, it is difficult to indicate what has to or can more conveniently be expressed in rules and what in processes because of the presence of several uncontrollable factors such as the future (evolving) needs and preferences of the stakeholders. Nonetheless, it is possible to draw a set of guidelines which can help in the identification of the parts of the requirements which can be expressed in rules and in processes such that an acceptable level of flexibility can be achieved. Before outlining these guidelines, following assumptions are made with respect to the involved activities.

A. Assumptions

Given a set of requirements, a system designer identifies a number of services (i.e., parts of the system) required to fulfill these requirements. These services have to be executed in a coordinated way. The coordination between services have to be defined such that the overall requirements can be fulfilled. In doing so, we assume that the coordination between services is defined in composition logic. The composition logic defines how and under what conditions the services have to coordinate such that the given set of requirements can be satisfied.

We also assume that the services represent a task in the scope of this paper. The execution of a service, therefore, may produce some data or information. These events and information are utilised while defining a composition logic.

B. Selection Guidelines

Selection of an appropriate design tool for modeling such a system depends on several characteristics associated with the required behavioural and information aspects of the required system. In the following we discuss about these characteristics and provide selection guidelines. These guidelines are provided, building upon works presented in [18], to help designers in deciding whether parts of requirements can be expressed naturally in rules or in processes.

1) Dependency: It is required to preserve the dependency between services, if the execution of a service depends on the execution of another services while satisfying the requirements. If such requirement exists, it is necessary to execute them in the order of their dependency.

Rule-based approaches do not provide mechanisms to specify the order of execution of rules which makes it difficult to preserve dependency between services. Process-based approaches are suitable to model such dependency because they provide explicit construct do specify the sequence of activities. Dependency between services can be specified with the help of control flow construct provided by process-based approaches.

2) Alternative Solution: For a given requirement, if a service that satisfies the requirement cannot be executed, an alternative services may need to be executed. If such a situation exists, we need a mechanism which allows to specify these alternative situations. This mechanism should also be able to indicate which alternative solution can be used for a given requirement and under what circumstances. The circumstances that prevents from the execution of a service may result from internal restriction of the service or from the requirement itself.

The service execution could be asynchronous or synchronous. Since process based approaches allow to model asynchronous execution, alternative service execution is recommended to be specified in processes. Using gateway constructs from process based approaches such situations can be specified. The selection criteria between different services have to be evaluated synchronously to make the selection decision. Therefore, it is recommended to specify selection criteria in rule as rule execution is synchronous.

3) Change frequency: It is possible that the requirements may change with time. The degree at which the requirements change can be high or low. It is necessary that the modelled system is capable of handling such changes. To support this characteristics, a mechanism is required which supports modification of the requirements at ease.

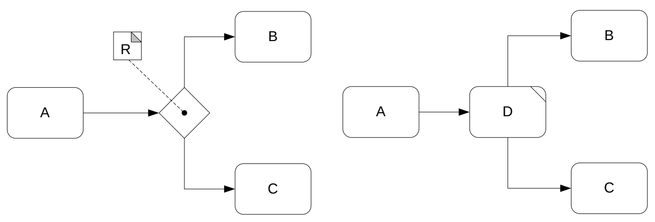
The change in requirements may require the change in behaviour of the system which may result in change in underlying implementation. If the degree of change is high, cost and time involved in changing underlying implementation is also high. In order to minimise such changes, it is recommended to specify such requirements in rule based approach. Rule based approaches allow to change or modify requirements easily at lower cost. If the degree of change is low, these changes can be planned and can be specified in process based approach.

V. Process and Rule Integration

Usually the extension of functionalities offered by services causes the introduction of new gateways in a process-based approach. In Figure 2, addition of an alarm functionality required two more gateways, highlighted in gray colour, than in the original process. The first one was to decide when to raise an alarm and the other one was to decide when to stop the whole process. In a rule-based approach, however, extension of functionalities can be done by simply adding or removing rules.

As shown by example snippets in Section II, rule-based approaches are more flexible than process-based approaches. In rule-based approaches, flexibility is achieved because the rules are executed directly by the rule engine and hence they can be added or removed when needed. Whereas in process-based approaches, any addition or removal of activities requires changes in the existing process and hence the underlying implementation.

As discussed in Section IV, some tasks can be expressed more naturally in rules whereas it is more appropriate to describe some other activities in processes. The composition approaches are, therefore, influenced by such factors. Considering the assumptions presented in Section IV-A and the selection guidelines given in Section IV-B, a flexible approach for service composition is proposed. In this approach, first we separate between the part of the requirements which can be specified in rule and and the parts which can be specified in processes following the guidelines presented in IV-B. Second, we identify the place of rule in the process and attach them there as shown in Figure 3(a). In doing so, we focus on rules that play role in the composition logic and not those internal to the service. Finally, we externalise the rules as shown in Figure 3(b) by introducing *decision point* activity to support flexibility and reusability.



(a) Process with internal rules (b) Process with externalised rules

Fig. 3. Design Stages

A decision point activity is like an activity in the BPMN sense. The difference is that, instead of representing some 'work' is being done, it represents some 'service' is being invoked. To distinguish a decision point activity from other activities, we denote them as a rounded rectangle with a line slant line on top right corner. We replace gateways in processes by decision point activities to externalise rules used to capture alternate activities and branching conditions. Because a decision point activity replaces a gateway of process based approaches, it allows multiple incoming and outgoing control flows.

A. Decision Service

Since we externalise the rules using a decision point activity, a mechanism is needed to establish a communication between this activity and the externalised rules. To support these communication, we use *decision service* to encapsulate decision expressions that determine the decision. A decision service can be invoked as web services [19] from the decision point activity. It is worth noting that the decision service is not used only for handling decision logic changes but also for managing the changes that occur in the system.

Decision expressions, encapsulated in a decision service, may refer to other decision services which again have decision expression or it may directly retrieve rules from a rule repository R. If rules $r_1, r_2, ..., r_n$ are used, for example, to make a certain decision while performing an activity a, these rules are grouped in a decision expression d. This results in a hierarchical design structure as shown in Figure 4. This hierarchical structure allows to separate rules from processes supporting reuse of rules across an enterprise as opposed to sharing only at the local process.It can also be argued that the changes in decisions would influence or even change the structure of the processes. We capture these structural changes, if any, through the use of sub-processes.

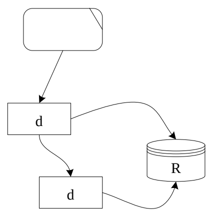


Fig. 4. Reference hierarchy

By introducing decision service, we allow separation of process, rules and decision expression. Since we aim at supporting flexibility in terms of adapting changing requirements, we externally store rules and decision expressions. Decision expressions are introduced to allow reuse of rules across processes. Since decision expressions evaluates to certain decision, decision expressions can also be reuse across processes. Moreover, this separation allows to update processes, decisions and rules independently from each other.

As we discussed, decision service is invoked from the decision point activity when a certain decision has to be made. To make a decision, decision service may need certain information to evaluate the decision expression. While evaluating the decision expression, received information may need to be passed further to the rules, e.g, to substitute variables in condition expressions.

The outcome of the decision should indicate which next service has to be executed. This means that the outcome of the decision service has to be associated to one (or more) outgoing links of the decision point activity. For this purpose, we annotate incoming and outgoing links of a decision point activity with the names of their source and destination activities respectively. We then define the interface of the decision service such that takes as input a set of tuple $<(l_i, \alpha)>$, and provides a set of values (l_o) as output, where l_i, α and l_o represent incoming link, data coming from incoming link, and outgoing link. In cases where there is no data associated with incoming link, the decision service will take a set of (l_i) as its input.

VI. Service Composition

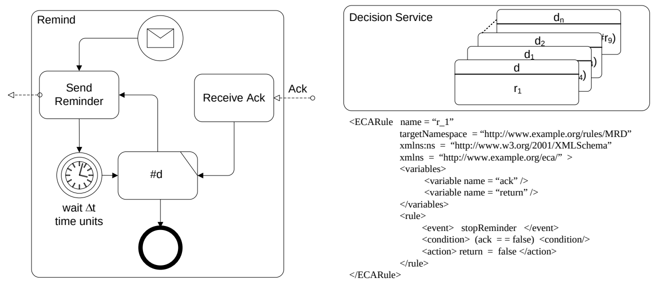
One of the main features of service oriented architecture is its support for functional adaptation (including extension) of an application. The adaptation is supported by means of composition, i.e., services providing different functionalities can be composed to create/provide a new service. A service composition is a mechanism for creating a customised service through the aggregation of available services.

Service composition is not only concerned with the aggregation of individual services to achieve new functionality, but also to promote reuse of existing services and reduce the overall development cost. Individual services that form a part of composite service are invoked in a certain order and following certain logic. If the functionalities of either the constituent services or the composite service changes at a later point in time these changes have to be adapted. To support the requirements related to the adaptation of service capabilities, we develop a composition logic following the techniques discusses and developed in Section V. In the following, we illustrate the usability of the proposed hierarchical approach for composing a MRD service as required by the application scenario described in Section II. In the following we focus on the "remind" subprocess from Figure 1 as this is the only part affected, in the given scenario, due the change in requirements. Therefore, we do not consider other activities and assume that the "remind" process receives reminder message at scheduled time (defined at subscription).

Based on the scenario presented in Section II we identified that the specific requirements that a MRD service has to fulfill are: 1) to remind users to take medicine at right time, 2) to control the release of medicines, and 3) to repeat reminder every Δt time units unless an acknowledgement is not received.

We associate the task of controlling the release of medicine to enabling the dispenser box, such that the user can press the "release" button on the box to get get the medicine. We assume that a dispenser box is fitted with such button. If the release button is pressed, the dispenser box will send an acknowledgement to MRD. This task is represented as an activity "Receive Ack" and sending of the reminder is represented as an activity "Send Reminder". By sending a reminder, dispenser box is also enabled, i.e., a reminder informs the user that this is the time to take a medicine and enables the dispenser box.

Following the guidelines presented in IV, we observed that the repetition of activity "Send Reminder" depends on activity "Receive Ack" because "Send Reminder" activity can be repeated only if the acknowledgement is not received. Therefore, a decision is required to determine whether to stop or repeat the "Send Reminder" activity. This situation is shown in Figure 5(a), where the decision point activity refers to a decision identified as #d.



```
<ECARule   name = "r_1"
           targetNamespace  = "http://www.example.org/rules/MRD"
           xmlns:ns  =  "http://www.w3.org/2001/XMLSchema"
           xmlns  =  "http://www.example.org/eca/"  >
           <variables>
                <variable name = "ack" />
                <variable name = "return" />
           </variables>
           <rule>
                <event>   stopReminder   </event>
                <condition>  (ack  = = false)  <condition/>
                <action> return  =  false </action>
           </rule>
</ECARule>
```

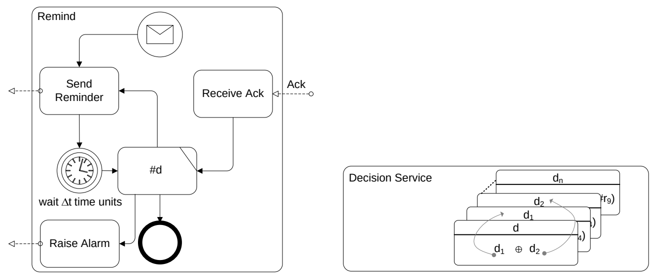
(a) Use of decision point activity (b) Decision service and rule

Fig. 5. Decision Point Activity, Decision Service and Rule

This decision evaluates a decision expression which is specified in terms of other decision expression or rules. A decision expression is specified in the form of a logical formula and encapsulate them in a service. For the purpose of deciding whether to repeat the reminder to stop, we define the decision expression d as $d = r_1$, where r_1 is a rule and is specified using WS-ECA syntax and semantics. In a decision expression we allow to use logical operators OR, AND, and XOR. The upper part in Figure 5(b) shows how decision expression are defined and the lower part specified how rules are encoded in WS-ECA.

The additional requirement of the changed situation in the application scenario presented in Section II is to extend the functionality of DRM as specified in 5 with the alarm functionality. The condition to raise the alarm is that the user does not take medicine even after sending the reminder N number of times. As in the case of previous situation (i.e., situation without the need of alarm functionality), we represent the task of raising alarm with "Raise Alarm" activity.

Following the guidelines presented in IV, we again observed that requirements for the repetition of activity "Send Reminder" is unchanged. However, the activity "Raise Alarm" depends on the activity "Receive Ack" because "Raise Alarm" activity can only be performed if the acknowledgement is not received after sending N reminders. In addition, we have additional choices: either repeat "Send Alarm" activity for N number of times, perform "Raise Alarm" activity or stop the process. Therefore, a decision is required to determine which of these activities to perform and when. This new situation is shown in Figure 6(a), where the decision point activity refers to a decision identified as #d.



(a) Use of decision point activity (b) Decision service and rule

Fig. 6. Decision Point Activity, Decision Service and Rule

In the new situation, we need to decide when to repeat "Send Reminder" activity and when to perform "Raise Alarm" activity. We also need to decide on when to stop the whole process. The process should stop either after the user takes the medicine of the "Raise Alarm" activity has been performed. Let $d_1 = r_1$ is the logical expression for making a decision on when to repeat the "Send Reminder" activity and $d_2 = r_1 \bigwedge r_2$ is the logical expression for making a decision on when to perform "Raise Alarm" activity, then the decision expression d referred from the decision point activity can be defined as $d = d_1 \bigoplus d_2$. Figure 6(b) shows how decision expressions can be used from a decision expression. The examples of rules r_1 and r_2 are not shown because they can be encoded similarly as shown in Figure 5.

The example solutions presented in Figure 6 shows that the proposed approach is flexibly supports adaptation of changing requirements. It also illustrates how rules can be reused across the process. In comparison to the purely process based or purely rule based approach, the proposed solution is more capable of separating the concerns and minimising the dependency between process designer and rule experts. This allows for adaptation of changing requirements at minimal cost with respect to changes in composition specification as well as their implementation. Instead of changing the composition specification each time the requirements changes, it is possible to change only the part of the specification affected by the changes.

VII. Related Works

Use of process and rules interchangeably has already taken the attention of both research and academia. Integration of business processes and rules has been considered in literature [5], [7], [8], [12], [20], [21], [22], [23] in an attempt to introduce flexibility in business process model.

A hybrid approach for web service composition is presented in [5]. In this approach, business processes are used for describing core part of the composition logic and the rules are used for defining policy-sensitive aspects of the composition. These rules and processes are kept separately to reduce complexity and to increase adaptability. However, the integration of these processes and rules for use at runtime is poorly specified.

Combination of SOA, BPEL and ontology is considered in [12] to improve maintainability and achieve flexibility of knowledge intensive business processes. The variable tasks are described more abstractly and stored in task pool during design time whereas their flow is defined during run time. Ontologies and business rules are used for defining these abstract parts to support selection of execution of tasks during rune time. The link between business process and rules, however, is poorly defined. In our approach, we provide mechanisms to define such links clearly at required granularity.

The work presented in [7], propose an approach to customise a business process to a particular case of usage. In their work, authors utilise business rules and workflow patterns to model variable parts of process flow to support dynamic pattern composition. They focus on isolating the parts of the process that are likely to change from the rest of the process. Similar to the solution proposed in [12], a set of workflow patterns is identified and implemented in business rules. These rules are kept separately from the process but, unlike our approach, concrete solution to incorporate those isolated process parts is not defined properly.

An ECA-based framework to facilitate coordination between active devices interacting with each other in an ubiquitous environment is discussed in [20]. An XML-based language is defined to specify reactive bahaviours of web services as well as service interactions. The rules describing a particular service are embedded into the device providing such service. These rules are triggered by certain events and the corresponding service is invoked. This approach considers rule-based approach only which is difficult to manage if the number of devices or the service increases.

Possibility for combining business rules with business processes is also investigated in [10], [24]. In their work, authors proposed rBPMN language by integration of BPMN and R2ML at meta level. In particular, rule gateways are introduced to replace normal gateways from BPMN. Through the rule gateways business rules can be specified explicitly in the business process. The level of flexibility, i.e., the reuse of rules across processes, due to such integration is, however, poorly specified.

A representational analysis of business process and business rules is presented in [6] comparing their representation capabilities. The paper concludes that neither processes-based nor rule-based languages are complete in their own but their combination is able to provide a better modelling power.

In [22], a "find and bind" mechanism for supporting dynamic binding of Web services to Web service flow instance at run-time. This mechanism facilitates a policy-based selection of Web services at run time to repair process instances. It extends the existing process technologies and thus requires upgrading of existing tools.

The work presented in [8] utilise rules for managing business collaboration models which are developed following the model driven approaches. In their approach, process specifications were generated using a composition engine which takes process elements and rules as its inputs. The facts regarding the process elements and their flows are described in rules. The proposed approach, however, is not described in detail.

VIII. Conclusions and Future Work

One of the appealing features of service composition is its support for flexibility and reusability of existing services while creating value-added services to satisfy user requirements. Users requirements may change more frequently than anticipated as seen, for example, in healthcare domain. Current approaches either do not provide support for adapting such changes or the offered solutions are too restrictive in terms of cost and time required for responding these change in requirements.

In this paper, we proposed an approach for increasing flexibility and decreasing implementation cost in service composition. We provide guidelines to identify which part of the requirements can be expressed in process and which parts can be expressed in rules. By identifying these parts, we aim at minimising negative implications at different level of abstractions and their implementations. Following a design based approach, we introduced a decision point activity. We externalise rules from processes and refer them from the decision point activity. Since the rules are externalised, a mechanism is needed to establish a communication between this activity and the externalised rules. To support these communication, we use *decision service* to encapsulate decision expressions that determine the decision. A decision expression can refer to other decision expression or access rules from the rule repository. Through this mechanism we supported reuse of rules across processes and offered higher level of flexibility in support of requirements evolution and their adaptation.

The usability of the proposed approach is illustrated with MRD application scenario, but we believe this approach can be used in other application domains such as health information systems and supply chain management. In the proposed approach we considered only the rules involved in a composition logic and not the internal rules. It would be interesting to see whether these rules has any implications in supporting flexibility. We further aim at developing complex application scenarios from other domain to investigate applicability of the proposed approach in other domains. We are also implementing the prototype to support execution of the cases shown in this paper. We are further refining this work to experiment with more complex examples where both decision logic and structural changes can occur simultaneously. Users' requirements and preferences can be more complex than those show in this paper. The proposed approach should be able to respond to these requirements and preferences in a flexible manner. We aim at achieving this by separating user's preferences from functional requirements. Development of preference model is considered as part of our future work. We are also going to work on modifiability analysis and observe the effect of the proposed work as the requirements evolve.

IX. Acknowledgements

This material is based upon works jointly supported by the IOP GenCom U-Care project (http://ucare.ewi.utwente.nl) funded by the Dutch Ministry of Economic Affairs under contract IGC0816 and by the DySCoTec project funded by the Center for Telematics and Information Technology (CTIT) of the University of Twente.

References

[1] R. V. McCarthy, "Toward a Unified Enterprise Architecture Framework: an Analythical Evaluation," *Issues in Information Systems*, vol. 7, no. 2, pp. 14–17, 2006.

[2] H. Jonkers, M. M. Lankhorst, H. W. L. ter Doest, F. Arbab, H. Bosma, and R. J. Wieringa, "Enterprise Architecture: Management tool and Blueprint for the Organisation," *Information Systems Frontiers*, vol. 8, no. 2, pp. 63–66, 2006.

[3] M. Lankhorst et al., *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer, 2009.

[4] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, "The next step in Web services," *Communications of the ACM*, vol. 46, no. 10, pp. 29–34, 2003.

[5] A. Charfi and M. Mezini, "Hybrid Web Service Composition: Business Processes Meet Business Rules," in *Proc. of the 2nd International Conference on Service Oriented Computing*, 2004, pp. 30–38.

[6] M. zur Muehlen, M. Indulska, and G. Kamp, "Business Process and Business Rule Modeling: A Representational Analysis," in *Proc. of Workshop on Vocabularies, Ontologies, and Rules for the Enterprise*, 2007, pp. 1–8.

[7] T. van Eijndhoven, M.-E. Iacob, and M. L. Ponisio, "Achieving business process flexibility with business rules," in *Proc. of the 12th International IEEE Enterprise Distributed Object Computing Conference*, 2008, pp. 95–104.

[8] B. Orriens and J. Yang, "A Rule Driven Approach for Developing Adaptive Service Oriented Business Collaboration," in *Proc. of IEEE International Conference on Services Computing*, 2006, pp. 182–189.

[9] T. Graml, R. Bracht, and M. Spies, "Patterns of Business Rules to Enable Agile Business Processes," *Enterprise Information Systems*, vol. 2, no. 4, pp. 385–402, 2008.

[10] M. Milanović, D. Gašević, and G. Wagner, "Combining Rules and Activities for Modeling Service-Based Business Processes," in *Proc. of the 2008 12th Enterprise Distributed Object Computing Conference Workshops*, 2008, pp. 11–22.

[11] M. Cilia and A. P. Buchmann, "An Active Functionality Service for E-Business Applications," *ACM SIGMOD Record*, vol. 31, no. 1, pp. 24–30, 2002.

[12] D. Feldkamp and N. Singh, "Making BPEL flexible," Association for the Advancement of Artificial Intelligence (www.aaai.org), Technical Report SS-08-01, 2008.

[13] F. Wang and K. J. Turner, "Towards Personalised Home Care Systems," in *Proc. of the 1st International Conference on PErvasive Technologies Related to Assistive Environments*, 2008, pp. 1–7.

[14] S. A. White, "Introduction to BPMN," Object Management Group (OMG), Tech. Rep., 2004.

[15] D. Hay and K. A. Healy, "Defining Business Rules ~ What Are They Really?" The Business Rules Group, Final Report Revision 1.3, 2000.

[16] B. R. Group, "The Business Rules Manifesto," 2003, available online at: http://www.businessrulesgroup.org/brmanifesto.htm (Last seen on Feb, 2010).

[17] G. Wagner, C. V. Damasio, and G. Antoniou, "Towards a general web rule language," *International Journal of Web Engineering and Technology*, vol. 2, no. 2/3, pp. 181–206, 2005.

[18] M. zur Muehlen, M. Indulska, and K. Kittel, "Towards Integrated Modeling of Business Processes and Business Rules," in *Proc. of the 19th Australasian Conference on Information Systems*, 2008, pp. 690–697.

[19] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services Concepts, Architectures and Applications*. Springer, 2004.

[20] J.-Y. Jung, J. Park, S.-K. Han, and K. Lee, "An ECA-based framework for decentralized coordination of ubiquitous web services," *Information and Software Technology*, vol. 49, no. 11-12, pp. 1141–1161, 2007.

[21] S. Demeyer, T. D. Meijler, O. Nierstrasz, and P. Steyaert, "Design Guidelines for 'Tailorable' Frameworks," *Communications of the ACM*, vol. 40, no. 10, pp. 60–64, 1997.

[22] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. Buchmann, "Extending BPEL for Run Time Adaptability," in *Proc. of the 2005 Nineth IEEE International EDOC Enterprise Computing Conference*, 2005, pp. 15–26.

[23] F. Corradini, A. Plzonetti, and O. Riganelli, "Bsuiness Rules in e-Government Applications," *Electronic Journal of e-Government*, vol. 7, no. 1, pp. 45–54, 2000.

[24] M. Milanović and D. Gašević, "Towards a Language for Rule-enhanced Business Process Modeling," in *Proc. of the 2009 IEEE Enterprise Distributed Object Computing Conference*, 2009, pp. 64–73.