Dynamic Homecare Service Provisioning Architecture

Alireza Zarghami, Mohammad Zarifi Eslami, Brahmananda Sapkota, Marten van Sinderen
Department of Electrical Engineering, Mathematics and Computer Science, University of Twente
Enschede, The Netherlands
{a.zarghami,m.zarifi,b.sapkota,m.j.vansinderen}@utwente.nl

***Abstract*—The realization of homecare services is difficult because of dynamicity requirements and constraints that exist in this domain. These requirements call for a dynamic service provisioning, i.e., adaptivity and adaptability of the (composition of) homecare services in response to a) frequently occurring changes like change in the location or vital signs, or b) slowly developing changes like extent of impairments of a care-receiver. In this paper, we explain our understanding of a dynamic service provisioning platform, its requirements and constraints. As such, we design an architecture based on an existing hybrid service provisioning approach (a combination of process and rule) and related architectural patterns. Then, we implement this approach using the commercially available process and rule engines. We demonstrate how a homecare application can be deployed, executed and how the application can adapt itself to the frequently occurring changes at runtime. We also demonstrated how a care-giver can modify the behaviour of the application to adapt the slowly occurring changes. Finally, we discuss the pros and cons of the approach and explain our future plan.**

***Index Terms*—service provisioning; application platform; business process and rules; adaptive and adaptable service**

I. Introduction

Population aging and demographic change is a global phenomenon for the near future in the industrialized countries. As a result of the increase in the proportion of old people, the industrialized countries will face new challenges [1]. One challenge will be to cope with the need of care for elderly (care-receiver) while there is not enough manpower (care-givers) to work in this domain. It will be difficult to support care-receivers, if the existing health care systems and processes remain as they are now [1]. A promising solution can be providing IT-based healthcare systems and supporting independent living for elderly in their own home [2].

A homecare system is "*a potentially linked set of services ... that provide or support the provision of care in the home*" [3]. With the emergence of Service-oriented computing, many of these services (such as vital sign monitoring) are provided by third-party organizations and can be integrated to provide a seamless homecare solution. Homecare applications can be built by composing several homecare services. To this end, service-oriented architecture and its advantage of service composition in a loosely-coupled manner are being considered as one of the approaches to promote such solutions [4] [5].

The realization of homecare services is difficult because of the dynamicity requirements and constraints that exist in this domain [6]. These requirements call for adaptivity and adaptability of the (composition of) homecare services in response to a) frequently occurring changes like change in the location or vital signs of a care-receiver, or b) slowly developing changes like extent of impairments of a care-receiver [7]. Existing homecare provisioning platforms do not (completely) address these requirements and constraints [8].

Our objective is to design and implement a homecare service provisioning platform which can support the dynamicity demands and constraints of the homecare domain. We propose a hybrid service composition approach that uses business processes and rules [9]. Having business rules beside processes, enables the provisioning platform to adapt the services execution based on runtime situations. In addition, it helps the care-givers to simply change the rules to satisfy individual requirements of care-receivers without changing the processes.

In this paper, we show the feasibility of a dynamic service provisioning platform for a homecare application scenario. As such, we design an architecture and accordingly implement the hybrid service provisioning approach. We explain how a homecare application can be deployed, executed and how the application can adapt itself to the frequently occurring changes at runtime. We also explain how the care-giver can modify the application behaviour to adapt the slowly occurring changes.

The rest of this paper is structured as follow. In Section 2, we explain the dynamicity requirements and constraints of the homecare domain, the overall view of our solution and our definition of a dynamic homecare provisioning platform. In Section 3, an application scenario will be presented. Section 4, elaborates the logical architecture of the platform and the architectural patterns which are employed. The deployment architecture, implementation and the technologies used are described in Section 5. In Section 6, we discuss the advantages and disadvantages of our approach in comparison with other related works and accordingly explain our future plan. Finally, in Section 7, we conclude the paper.

II. Dynamicity in the Homecare Domain

To design a dynamic service provisioning platform, first we need to identify the dynamicity requirements and constraints that exist in the homecare domain. Then we present the overall view of our ICT-based solution in the homecare environment to show how the provisioning platform interacts with its environment. Finally, we introduce our dynamic service provisioning platform with respect to the identified changes and constraints.

A. Requirements and Constraints

We define the dynamicity requirements in the homecare domain in two categories as follows:

- **Frequently occurring changes:** These changes occur when the context or the vital signs of the care-receiver changes during provisioning of the services (e.g., change in location, blood pressure of a care-receiver). Since such changes happen frequently and also during the service execution, they need to be handled at runtime.
- **Slowly developing changes:** These changes are about the care-receiver's needs and preferences which usually develop gradually over longer period of time, such as extension of the care-receiver's impairments. For such changes, the care-giver, in consultation with the care-receiver, should be able to tailor the homecare services to satisfy their current needs and preferences.

All of these changes should be addressed with respect to the constraints that exist in the homecare domain such as safety-critical situation of the care-receiver and lack of technical skills of the care-giver [6]. Due the safety-critical constraints, the applications behaviour should be accurate while facing the frequently occurring changes at runtime. The care-giver must be able to tailor the application behaviour without the need of advanced IT knowledge and with minimum cost like the man-power needed for the tailoring.

B. The Overall View

A homecare provisioning platform should support the homecare application to adapt its behaviour to the frequently occurring changes at runtime. If the runtime adaption is not applicable, the care-giver modifies the homecare services through a so-called tailoring platform and then (re)deploy them to the provisioning platform. In order to avoid any misunderstanding, we should explain our perception from the tailoring and provisioning platforms and their interaction with each other. Figure 1 shows the overall view of out ICT-based solution in the homecare domain.

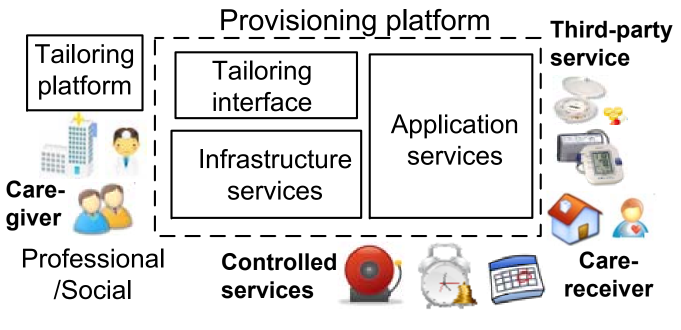


Fig. 1. The overall view of our ICT-based homecare solution

With respect to the requirements and constraints that exist in the homecare domain, we choose a hybrid service composition, a combination of process and rules, to design the provisioning platform [6]. A *service plan* refers to one or more service building blocks (SBBs) and describes the configuration and orchestration of these SBBs as well as decision rules required to specify runtime behaviour. Since these rules are used in the service plan to specify the runtime behaviour of the application, we call them decision rules instead of business rules, which are mostly used to define or constrain high level business goals. The SBBs, like a medicine dispenser or reminder, are the smallest manageable services from the care-giver point of view. Configuration parameters of SBBs allow the care-givers to specify different aspects of the SBBs such as service operations and user interface modalities. Orchestration schemes determine how SBBs are composed. Decision rules determine the possible adaptation at runtime, based on evaluation of the rules with runtime data (e.g., context values). For example, decision rules can be used to choose between alternative implementations one SBB or between alternative data and control flows among the SBBs, based on specific runtime circumstances.

By service provisioning, we mean the execution of the service plans by employing the functionality offered by available application services at runtime. The provisioning platform binds the abstract SBBs used in the service plan to the *application services*. As such, the *infrastructure services* of the provisioning platform must be able to match an abstract SBB with the several available application services at runtime. Then the most suitable application service is binded to this SBB based on predefined decision rules. The application services are provided either by the *third-party service providers* located outside a care home or the *controlled services* installed inside a care home. The care homes are either private homes located outside of a care center or units located inside a care center. Although the ownership of these two types of application services are different, they are treated by the provisioning platform in the same way. For instance, a blood pressure measurement service is provided and owned by a third-party organization while a reminder service is managed and owned by the platform.

The service plans, coming from the tailoring platform will be deployed to the provisioning platform through its *tailoring interface*. The provisioning platform interacts with the care-receiver and needs to be installed per each care home to execute its own services. In contrast, the tailoring platform interacts with the care-giver and one tailoring platform can be employed for a care center in charge of several care homes.

C. The Dynamic Service Provisioning Platform

With respect to the aforementioned dynamicity requirements and constraints, we define dynamic service provisioning platform as an *adaptive* and *adaptable* service-oriented application platform.

a) By *adaptive* homecare provisioning platform, we mean that the platform supports the applications to adjust their behaviour according to the relevant frequently occurring changes at runtime with minimal or no manual care-receiver intervention. It is done through adjusting the configuration and orchestration of a set of available application services at runtime. The adjustment is constrained by the service plan which is created by the care-giver.

b) By *adaptable* homecare provisioning platform, we mean that the deployed service plans, on top of the platform, can be updated and instantiated to support the slowly developing changes, respectively, with the minimum cost. The cost can be related to the man-power needed for the tailoring or the effect of updating and instantiation of an application on the platform and other running applications.

III. Application Scenario

To motivate the need for our proposed provisioning platform, we illustrate a homecare application for blood pressure monitoring (BPM) and explain its corresponding service plan.

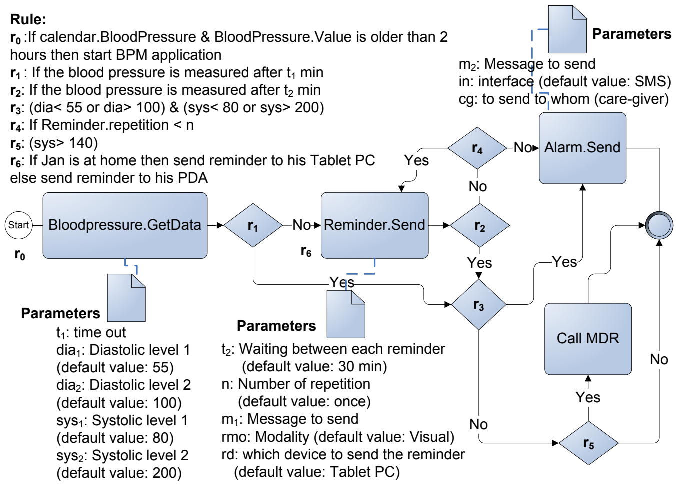

Rule:
r0 :If calendar.BloodPressure & BloodPressure.Value is older than 2 hours then start BPM application
r1 : If the blood pressure is measured after t1 min
r2: If the blood pressure is measured after t2 min
r3: (dia< 55 or dia> 100) & (sys< 80 or sys> 200)
r4: If Reminder.repetition < n
r5: (sys> 140)
r6: If Jan is at home then send reminder to his Tablet PC else send reminder to his PDA
Parameters
m2: Message to send
in: interface (default value: SMS)
cg: to send to whom (care-giver)
Start
r0
Bloodpressure.GetData
r1
No
Reminder.Send
r6
r2
r4
Yes
No
Alarm.Send
Yes
r3
Yes
Yes
Call MDR
Yes
No
r5
No
Parameters
t1: time out
dia1: Diastolic level 1 (default value: 55)
dia2: Diastolic level 2 (default value: 100)
sys1: Systolic level 1 (default value: 80)
sys2: Systolic level 2 (default value: 200)
Parameters
t2: Waiting between each reminder (default value: 30 min)
n: Number of repetition (default value: once)
m1: Message to send
rmo: Modality (default value: Visual)
rd: which device to send the reminder (default value: Tablet PC)

Fig. 2. The service plan of blood pressure monitoring (BPM) application

The service plan of the BPM application should be created and tailored by Nancy (a care-giver) for John (a care-receiver) to help him to measure his blood pressure on time. The application starts based on a predefined calendar event and reminds Jan, possibly several times, to measure his blood pressure. If he does not measure or his blood pressure is not in the normal range, the application sends an alarm to Nancy. If his blood pressure is still in the range but the systolic level is higher than 140, the application calls medicine reminder (MR) application to remind him to take his medicine.

Figure 2 shows the service plan of the BPM application which consists of an orchestration of SBBs as well as decision rules to specify the behaviour of the application at runtime. For instance, rule r_0 defines when the application starts, rule r_4 determines how many times to send the reminder and rule r_6 determines to which application service the reminder SBB should be mapped, based on Jan's location at runtime. To support the rules, there are several configuration parameters which are assigned to the SBBs.

The BPM application will be executed for one year. Due to the extension of Jan's heart disease, Nancy decreases the maximum systolic level to 130 to remind him to take his medicine. Furthermore, due to his movement difficulties, Nancy increases the reminder time, so Jan has more time to do his reminded tasks.

IV. Provisioning Platform Architecture

To present our provisioning architecture, first we explain the architectural patterns which are employed by our provisioning platform. Then we describe how the infrastructure services interact with each other and the other platform services to deploy and execute the applications. Finally, we explain the steps to deploy a service plan on top of the provisioning platform as a homecare application.

A. Architectural Patterns

Based on our definition, the provisioning platform should address both the adaptivity and adaptability properties. For these properties, there are several architectural and design patterns which allow us to reuse of solutions proposed by experienced practitioners for the common problems [10]. The patterns which are employed by our proposed logical architecture are explained as follows:

(1) Adapter: Adapter is a pattern to enable heterogeneous software components interact with each other by providing compatible interfaces. This allows us to replace a service by another service both at design and runtime without considering their implementation details. As shown in Figure 3, we have several adapters to provide uniform transportation protocols as well as interfaces for *third-party* services. Since our proposed *process engine* is interacting only trough SOAP protocol, these adapters provide web service interfaces out of any communication protocols such as JMS and HTTP, used by different service providers. For instance, the blood pressure (BP) measurement service has an interface to retrieve the last value of Jan's BP measurement through a web service which can be binded to different third-party service providers. In case of controlled services, since they are implemented by the platform, there is no need for the adapters. For instance, the calendar is implemented as an application to provide a web service to add events and accordingly to be notified. For sake of simplicity, by application services, we also mean adapter services.

The host of application services, the *application server*, has a service repository to maintain the binding ports and WSDL interfaces of the application services. The application server can be located inside a care home, a.k.a home gateway, or at back office like a care center. This design choice depends on whether the devices at home are able to communicate with the *application server* at back office. In section V, we explain why our home gateway is located in the back office. The internal mechanism of the *application server* is not the focus of this paper. Instead, we emphasize on the infrastructure services as the core of our logical architecture.

(2) Event-Control-Action: Due to our definition, contexual changes is part of the frequently occuring changes which should be addressed by the dynamic provisioning platform. We chose Event-Control-Action pattern for our dynamic provisioning platform [11]. By using this pattern, we aim to decouple context concern from reaction by means of Event-Control-Action (ECA) rules. The context-related application services such as the location determination service, provide

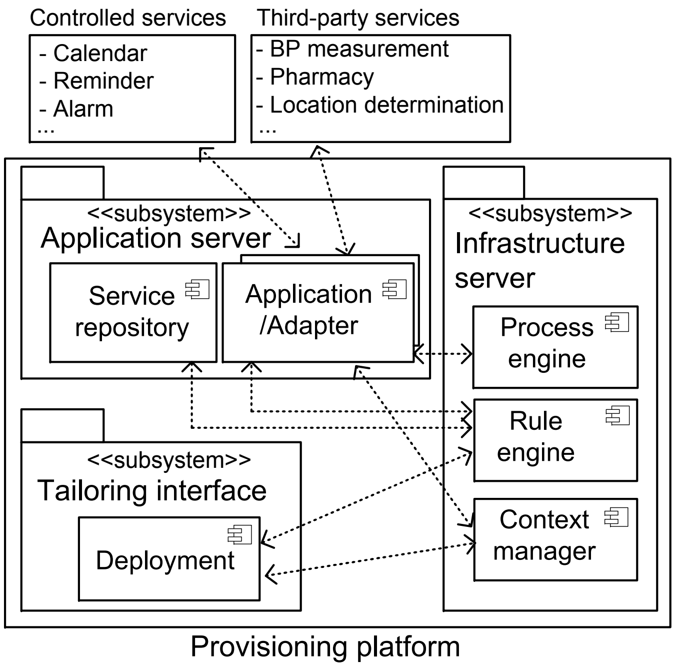


Fig. 3. The proposed logical architecture of the provisioning platform

publish-subscribe interfaces. For each contextual event, one or several of these application services have been subscribed by the context manager. The context manager notifies the rule engine if any contextual event happens. The rule engine can also query the context manager about the current contextual conditions.

(3) Process vs. rule engine: Since we chose the hybrid service composition approach, the provisioning platform employs the rule engine pattern to manipulate the rules. These rules are fired based on the happening contextual events which are managed by the context manager or non-contextual events which are triggered directly by application services, for instance, the calendar service triggers a rule to start a medication reminder application. In contrast with rule engine, the platform employs the process engine to manage the orchestration of the services which is more static compared to the decision rules.

The rule engine can either trigger a process in the process engine or be queried by the process engine at the decision points of the running processes. The rule engine has several components such as rule repository to maintain all the rules for execution and pattern matcher to decide which rule should be fired based on the event and contextual conditions. In this paper, since we emphasize on the interaction between the rule engine and the process engine, the internal components of the rule engine are not explained.

B. The Infrastructure Services

In our architecture, we have two types of services: application services and infrastructure services. The application services are scenario-dependent and implemented by adapters or applications to support a specific type of application scenario in the platform. For instance, the BP measurement service is employed by the BPM application. In contrast, infrastructure services are generic and scenario-independent.

As explained before, the service plan consists of the process (i.e., orchestration) and the decision rules to specify the behaviour of its corresponding application. The rules can be fired based on a set of predefined contextual and non-contextual events. To manipulate the process, rules and contextual events, as shown in Figure 4, our platform has three infrastructure services: *process engine*, *rule engine* and *context manger*. These services have several inner and outer interfaces. The outer interfaces can be used by the application services, service repository and deployment component. To execute an application, first its service plan must be deployed.

We assume all the possible orchestrations of the service plans have been deployed on the process engine with unique Identification (ID) and during the deployment, an orchestration ID is assigned to a care-receiver by a rule. As such, the process engine does not have deployment interface. The rule engine and context manger have deployment interfaces to deploy a rule and contextual event, respectively (R8,C2). Later in this section, the deployment process will be explained.

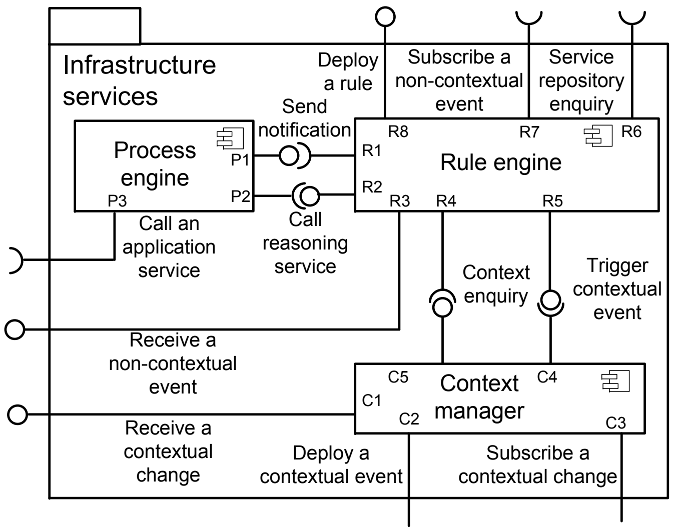

Infrastructure services
Deploy a rule
Subscribe a non-contextual event
Service repository enquiry
Send notification
Process engine
P1
P2
P3
R8
R7
R6
R1
R2
R3
R4
R5
Rule engine
Call an application service
Call reasoning service
Context enquiry
Trigger contextual event
Receive a non-contextual event
C5
C4
C1
C2
C3
Context manager
Receive a contextual change
Deploy a contextual event
Subscribe a contextual change

Fig. 4. The infrastructure services inner and outer interfaces

The rule engine sends notification to the process engine either to start an orchestration of a service plan or to inform the running orchestration about new happening event(R1-P1). In addition, the process engine can call a reasoning service, provided by executing a rule set on top of the rule engine(P2-R2). It can happens whenever the notification received by the process engine or at the decision points of the orchestration. Since, we use Event-Condition-Action pattern to decouple contextual concern from the actions done by the services, there is no direct communication between context manager and process engine. The process engine can directly call an application service in order to coordinate the orchestration, for instance, call a reminder service to send reminder to a care-receiver(P3).

Based on the events defined in a service plan, the rule engine or the context manager subscribes to a set of adapters (R7,C3). The contextual events are defined on the context manager during the deployment process. For each of the contextual event, the context manager subscribes to one or several application services. Therefore, the context changes are sent to the context manager through its interface (C1). If a contextual event happens, the context manager trigger the event through the rule engine interface (C4-R5). In addition, if the rule engine needs more contextual information, it can query the context manager (R4-C5). The non-contextual events, like calendar events, are directly defined in the rule engine and the corresponding application service can notify the rule engine through its interface (R3). During the execution, the rule engine contacts the service repository to know what are the available application services for a specific SBB(R6) to be binded.

C. *Service Plan Deployment*

As explained in Section II, the service plan consists of the orchestration of the SBBs and a set of decision rules. We explain the deployment process of a service plan in three steps as follow:

(1) Process deployment: An orchestration of a service plan will be deployed as a process to the process engine and can not change without redeploying the process. In our approach, the orchestration are fixed for each homecare application and if there is any need to change the orchestration, the service plan should be redeployed. In order to improve the adaptability, based on our interview with care-givers, each application has several alternative orchestrations with unique ID. So during the deployment, by mapping the care-receiver to one of the orchestration ID, the desired orchestration will be specified.

(2) Rule deployment: The decision rules will be deployed to the rule engine by defining their Events-Conditions and corresponding Actions. The rules can change without redeployment as far as their input and output parameters remain unchanged. To have more adaptability, instead of having several rule sets for a service plan, we define a rule set for each service plan. The rule set have all possible input and output parameters for a service plan, even if all of the parameters are not currently used by the decision rules. Therefore, changing the rules does not change the input and output parameters, unless a new parameter needs to be defined. Each rule set, after being deployed on the rule engine, can be accessed by a web service.

(3) Subscription: Based on the events and conditions of the rule set of the deployed service plan, the rule engine or the context manager subscribes to several application services. For instance, the service plan of Jan's BPM application has a calendar event, so the rule engine directly subscribes to the calendar service with Jan's ID and event type of blood pressure. It also needs to know whether Jan's blood pressure is measured. So the rule engine also subscribes to BP measurement service with Jan's ID. The service plan also has an event for change of Jan's location as a contextual event which is defined on the context manager. As such, the context manager subscribes to the location determination service with Jan's ID and will be notified whenever Jan arrives at or leaves the care home.

V. Deployment and Implementation

To demonstrate our implementation, first, we show the deployment architecture to describe the mapping of the logical architecture to the physical environment and technologies. Then, to demonstrate our implementation, using the application scenario presented in Section III, we explain how the infrastructure and application services interact with each other to execute the corresponding service plan.

A. *The Deployment Architecture*

We have employed several tools and technologies which can be installed either on one or separate servers. As Figure 3 shows, in our architecture, we have two types of services: a) the scenario-independent infrastructure services and b) scenario-dependent application services. Therefore, in our deployment architecture, we have allocated two different servers for them, the *infrastructure server* and the *application server*.

The infrastructure server is running on a PC with Windows 2008 server operating system. As a process engine, we use WebSphere Lombardi Edition[1] and as a rule engine we use WebSphere ILOG JRules[2]. We assume that all the application services, are accessible by the infrastructure services through SOAP protocol, therefore, this server can be located at our back office and this eases the maintenance of the server. The proposed rule engine has a GUI to edit the rules called *rule editor* which can only be installed on Windows platform. In addition, interoperability among different platforms is not an issue because of using web services. As such, we chose Windows server for the infrastructure server. Figure 5 depicts how the provisioning platform has been deployed.

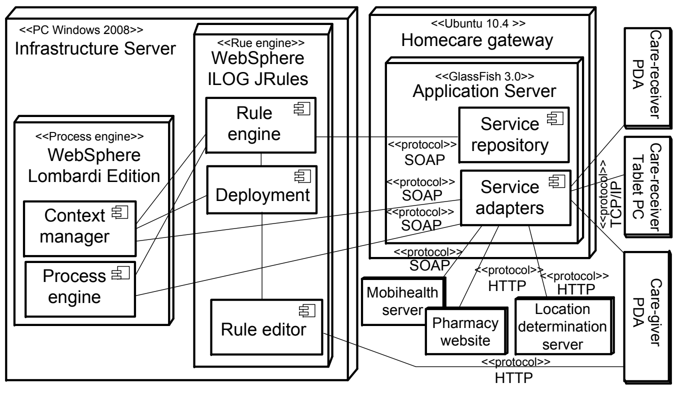


Fig. 5. The deployment architecture with respect to our application scenario

The application services on the application server must be able to communicate with their corresponding heterogeneous hardware and software components. In the homecare domain, some of the hardware components such as location sensors are resource-constrained and therefore, only capable to communicate with limited types of networks such as WiFi. On the other hand, it is not financially feasible to put a server inside

[1]http://www-01.ibm.com/software/integration/lombardi-edition/
[2]http://publib.boulder.ibm.com/infocenter/brjrules/v7r1/index.jsp

each care home. With respect to our application scenario, all the devices are accessible trough TCP/IP network and thus, although our application server can be located inside the care home, it is also located in the back office. As such, we setup the application server on a Ubuntu (version 10.4) machine. The adapters and their web services are implemented in an open source application server called GlassFish[3]. Its Openness and support of all Java EE API specifications such as web services, XML and JMS (Java Message Service), seems promising in order to have heterogeneous types of adapters.

As previously explained in Section II-B, we have two types of application services: controlled and third-party services. As one of our third-party services, the blood pressure monitoring device at home is connected to a server in Mobihealth company (one of our third-party service providers), through their own infrastructure. The Mobihealth server is connected to our application server to provide web service interface which will be employed by the infrastructure services. The medicine dispenser service, as another third-party service, is managed by a pharmacy through a GPRS network. The medicine dispenser adapter retrieves the data from the pharmacy website to inform the infrastructure services when the medicine is taken by mean of its web service interfaces. The location determination server, as our last third-party service, employs its RFID sensors at the care home to determine the location of the care-receiver. Then, the location information can be retrieved from its server via HTTP protocols.

Beside the third-party services, we implement ourselves three controlled services: reminder, alarm and calendar services. The application components of the reminder and alarm services provide interfaces to send reminder using Jan's PDA and Tablet PC. The application component of alarm service provides an interface to send alarm using Nancy's PDA. The calendar service has an interface to set an appointment and to trigger an event when that appointment approaches. In the BPM application scenario, the calendar service, used by all the third-party service providers such the pharmacy, to set an appointment to remind the care-receiver of doing a task. As such, the calendar service sends an event to the rule engine to start the BPM application.

B. The Implementation

To implement the BPM application scenario, we need to deploy its service plan. As we explained in section IV-C, the deployment consists of three steps: process deployment, rule deployment and subscription. With respect to the process deployment, the assigned orchestration of BPM application is shown in Figure 6(A). It uses a BPMN-like language used by Lombardi software to show how the infrastructure and application services interact with each other.

The BPM application is triggered by the rule engine through its starting message event. The orchestration conforms with the service plan which is explained in Section III but with more details for being executable. For instance, to implement the r_0, the process engine calls the reasoning service implemented by the rule engine to check whether the the latest Jan's BP measurement is still valid. The orchestration has a simple split to define a parallel subprocess to handle the events sent by the rule engine at runtime. For each event, the reasoning service will be called to update the values of the configuration parameters. As an example for the BPM application, if Jan leaves the care home, the context manager is triggered by the location service and accordingly the rule engine is triggered by the context manager. Then the rule engine queries the context manager if it needs more contextual information. The rule engine matches the contextual information and the event with r_6 and sends an event to trigger the parallel subprocess of the orchestration. The subprocess calls the reasoning service to update the binding port of reminder service from Jan's Tablet PC to his PDA, even if the first reminder is already sent.

As Figure 6(B,C) shows the rule engine and the context manger have their own processes which are implemented as constantly running processes on top of the process engine. These two processes are scenario-independent and can be employed by different homecare applications to subscribe to different application services for any types of events. For instance, the rule engine process subscribes to BP measurement service to receive the latest value of Jan's BP measurement. It supports the asynchronous interaction between the third-party service providers and the platform, and thus, Jan can measure his blood pressure even before the orchestration of BPM application starts on the process engine.

As mentioned before, there is one rule set, i.e., reasoning service for the BMP application which is called at different points of the orchestration with different input values. To implement the reasoning service, we define a rule set on top of the rule engine. Each rule set has all the input and output parameters which can be transfered between the process and rule engine to execute a specific application like BPM. All these parameters are defined as the same data model by both the process engine and the rule engine.

The data model of the rule engine, which is called XOM (Executable object model), can be used by its corresponding BOM (Business object model) and rules templates in the rule editor to modify the behaviour of BPM application. According to our application scenario, Nancy can decrease the maximum systolic level and increase the reminder time by changing the rules which are represented in natural language without redeployment of the orchestration. Figure 7 shows how Nancy can edit the BPM application rules in the rule editor. For instance, the *diastolic* and *blood pressure* are the business objects and "*the diastolic of The Blood pressure is more than*" is a rule template. By clicking on the plus/minus symbols, Nancy can change the objects or their values.

VI. Discussion

In this paper, we investigate the feasibility of using our hybrid service composition approach to achieve adaptivity and adaptability for our homecare service provisioning platform. Initial results shows that the non-technical care-receiver can

[3]http://glassfish.java.net/

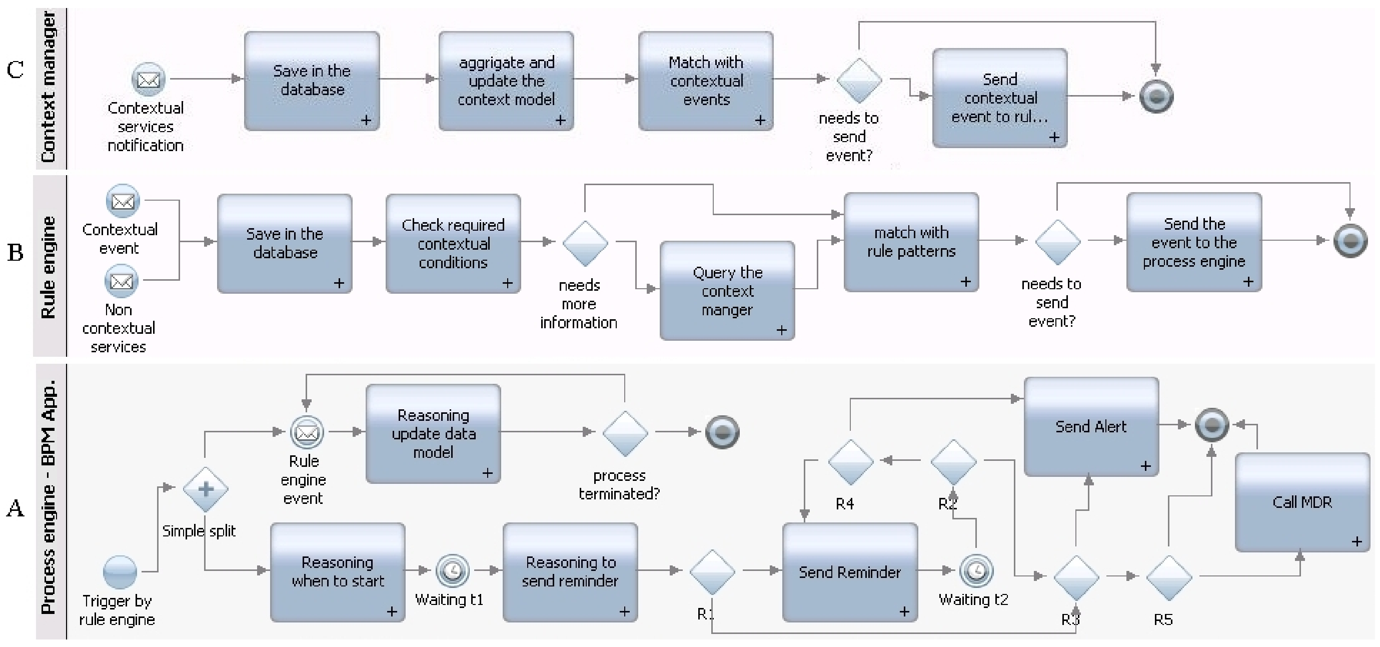
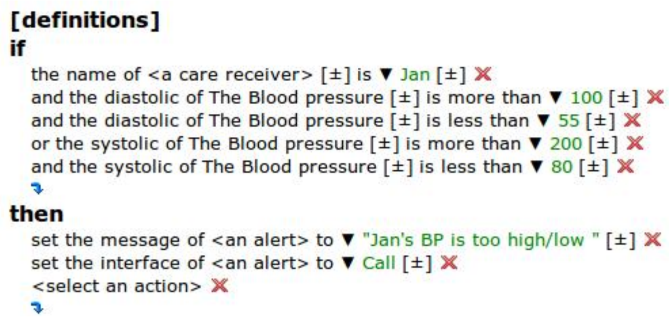


Fig. 6. The assigned orchestration of BPM application on the process engine (A), the process of the rule engine (B) and the process of context manager (C)



```
[definitions]
if
    the name of <a care receiver> [±] is ▼ Jan [±] ✗
    and the diastolic of The Blood pressure [±] is more than ▼ 100 [±] ✗
    and the diastolic of The Blood pressure [±] is less than ▼ 55 [±] ✗
    or the systolic of The Blood pressure [±] is more than ▼ 200 [±] ✗
    and the systolic of The Blood pressure [±] is less than ▼ 80 [±] ✗
then
    set the message of <an alert> to ▼ "Jan's BP is too high/low " [±] ✗
    set the interface of <an alert> to ▼ Call [±] ✗
    <select an action> ✗
```

Fig. 7. The business objects and rule templates in the rule editor

modify the rules relatively easily provided that a comprehensive list of rule templates, which can be edited, is given. Currently, we conduct an experiment in a care center to decide on this list of rule templates required for provisioning dynamic homecare services. The increase in the number of rules increases the complexity of the tailoring process beyond a manageable level. In order to decrease the number of rules, for each service plan, we propose to use a number of alternative orchestrations as the basis. We believe that by deploying several alternative orchestrations for a service plan, we can have an acceptable level of dynamic service provisioning and by providing a number of rule templates, we enable the care-giver to tailor the services to individual requirements and preferences of care-receivers.

One of the limitations of our approach is the use of alternative orchestrations. Since the alternative orchestrations of a service plan are static and cannot be changed after its deployment, the adaptability and adaptivity aspects have to be handled using rules. However, the rules cannot change the orchestration which limits the adaptation capability of our approach. A fully rule-based service plan would make it easier to support adaptability. However, it makes the tailoring process more difficult for non-technical care-givers and increases the possibility of miss-planing in addition to making it difficult to manage. This requires an approach that makes a balanced use of rules and processes in the service plan, which is planned as our future work.

Alternatively, the hybrid service composition can already be used at tailoring level to generate an abstract service plan. The tailoring platform then can generate a fully rule-based executable service plan, automatically and deploy to the provisioning platform. However, this can complicate the conformance validation. It would be difficult to guarantee that the generated rule-based service plan actually conforms to the initial service plan. Keeping the hybrid service provisioning both in the tailoring and provisioning, makes the service plan refinement more straightforward and can possibly provide a basis for an automatic service plan refinement from the tailoring to the provisioning level. On the other hand, the hybrid service provisioning approach makes conformance validation simpler in comparison to a fully rules-based approaches.

In our approach, we use only one rule set per each service plan. Therefore, whenever the process engine calls the rule engine, all the rules of the rule set can be executed. Their execution depends on the input parameters which are sent by the process engine. It has a negative impact on the performance of the rule engine because some of the rules might be executed while they are not required. Although classifying rules, like before and after interceptor as in [12], improves the performance of the rule engine, it increases the complexity of tailoring a service plan. In our approach, the care-giver can edit the rule templates for a specific care-receiver in any order and at once without specifying where (before or after a specific task) they should be executed. We need to do a performance evaluation of our approach which is planed as our future work.

Some of the rules are crosscutting concerns and defined by specifying where (i.e., after or before an activity) to be executed. This types of issues are handled through use of *aspects* from aspect-oriented-programming in the filed of service composition [13]. However, this requires modification of the existing process engines, which we like to avoid. We believe that the task-dependency of rules can also be implemented by existing process and rule engines. Thus, as another future work, we plan to define a set of tasks similar to the SBBs which can be called by the rule engine and then the result will be returned to the process engine. For instance, currently our service plan does not let the care-giver to add a rule to execute MR application within the orchestration of BPM application. But if we define MR application as an application service, then it can be invoked from a rule engine (e.g., WebSphere ILOG allows us to call external services). After its execution, the control and data flows will be returned to the process engine and thus from the care-giver's point of view, it seems that the MR application is added to the orchestration of BPM application.

To address adaptivity, we employ a parallel subprocess in the orchestration of the service plan. As discussed in Section V, the runtime contextual events are sent to the rule engine by the context manger and accordingly the rule engine sends the events to the process engine. Finally, the rule engine updates the data model of the service plan through its reasoning service. This data model, which is defined on both rule and process engines, is the essential element for both adaptivity and adaptability of a service plan. In order to have the desired application behaviour, the data model of a service plan must be consistent in both process and rule engines. If a new rule needs a new parameter, the data model should be updated on both the rule engine and the process engine. As such, the process must be redeployed which interrupt the service provisioning process. To alleviate this problem, we plan to externalize the data model from the process engine and thus, a change in the data model does not cause the process to be redeployed.

VII. Conclusions

In this paper, we investigate the feasibility of a dynamic service provisioning platform by using a hybrid service composition approach. The platform is capable of handling the frequently occurring changes (both contextual and vital signs changes) at runtime through a parallel subprocess which is triggered by the rule engine. By exacting the rules embedded in the process and providing them separately as services, the care-giver can modify the homecare application behaviour even at runtime. The platform is implemented using commercially available process and rule engines to evaluate the feasibility of our approach. The initial evaluation reveals that our approach is promising for providing a realistic solution to address dynamicity requirements in the homecare domain.

Although we show the feasibility of our approach, there are still several domain-dependent and domain-independent aspects which require further research. The domain-dependent aspects such as to what extent the deployed orchestrations of the homecare applications remain unchanged over a longer period of time needs further investigation. Furthermore, the conformance validation between the service plan and the executable orchestrations and rules is crucial because of the safety-critical nature of homecare environment. The domain-independent aspects include, for example, balanced use of rules and processes in the service plan. This requires further investigation on how to achieve this requirement and whether a guideline can be defined.

Acknowledgements

This work is part of the IOP GenCom U-Care project(http://ucare.ewi.utwente.nl) which is sponsored by the Dutch Ministry of Economic Affairs under contract IGC0816.

References

[1] European Commission, "Ageing well in the information society - an i2010 initiative - action plan on information and communication technologies and ageing," EU, Tech. Rep., 2007.

[2] K. Gabner and M. Conrad, "ICT enabled independent living for elderly, A status-quo analysis on products and the research landscape in the field of Ambient Assisted Living in EU-27," prepared by VDI/VDE Innovation und Technik GmbH, 2010.

[3] M. R. McGee-Lennon, "Requirements engineering for home care technology," in *26th Annual SIGCHI Conf. on Human Factors in Computing Systems*, 2008, pp. 1439–1442.

[4] P. D. Gray, T. McBryan, N. Hine, C. J. Martin, N. Gil, M. Wolters, N. Mayo, K. J. Turner, L. S. Docherty, F. Wang, and M. Kolberg, "A scalable home care system infrastructure supporting domiciliary care," Tech. Rep., 2007.

[5] M. Mikalsen, S. Hanke, T. Fuxreiter, S. Walderhaug, L. Wienhofen, S. ICT, and N. Trondheim, "Interoperability services in the MPOWER Ambient Assisted Living platform," *Stud Health Technol Inform*, vol. 150, pp. 366–70, 2009.

[6] A. Zarghami, M. Zarifi Eslami, B. Sapkota, and M. van Sinderen, "Toward dynamic service provisioning in the homecare domain," in *5th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, 2011, pp. 292–299.

[7] T. McBryan, M. R. McGee-Lennon, and P. Gray, "An integrated approach to supporting interaction evolution in home care systems," in *1st International Conf. on Pervasive Technologies Related to Assistive Environments*. New York, NY, USA: ACM, 2008, pp. 1–8.

[8] A. Zarghami, M. Zarifi Eslami, B. Sapkota, and M. van Sinderen, "Service realization and compositions issues in the homecare domain," in *6th International Conf. on Software and Data Technologies (ICSOFT)*, vol. 1, 2011, pp. 347–356.

[9] A. Charfi and M. Mezini, "Hybrid web service composition: business processes meet business rules," in *2nd International Conf. on Service Oriented Computing*. ACM, 2004, pp. 30–38.

[10] *Pattern-oriented software architecture: On patterns and pattern languages*, 2007.

[11] P. Dockhorn Costa, L. Ferreira Pires, and M. van Sinderen, "Architectural patterns for context-aware services platforms," in *2nd International Workshop on Ubiquitous Computing*. INSTICC Press, 2005, pp. 3–18.

[12] F. Rosenberg and S. Dustdar, "Business rules integration in bpel - a service-oriented approach," in *7th IEEE International Conf. on E-Commerce Technology*, 2005, pp. 476–479.

[13] A. Charfi and M. Mezini, "Ao4bpel: An aspect-oriented extension to bpel," *World Wide Web*, vol. 10, pp. 309–344, 2007.