A Lightweight Process Engine for Enabling Advanced Mobile Applications

Johannes Schobel, Rüdiger Pryss, Marc Schickler, Manfred Reichert

Institute of Databases and Information Systems, Ulm University, Ulm, Germany
{johannes.schobel, ruediger.pryss, marc.schickler, manfred.reichert}@uni-ulm.de

Abstract. The widespread dissemination of smart mobile devices offers new perspectives for timely data collection in large-scale scenarios. However, realizing sophisticated mobile data collection applications raises various technical issues like the support of different mobile operating systems and their platform-specific features. Often, specifically tailored mobile applications are implemented in order to meet particular requirements. In this context, changes of the data collection procedure become costly and profound programming skills are needed to adapt the respective mobile application accordingly. To remedy this drawback, we developed a model-driven approach, enabling end-users to create mobile data collection applications themselves. Basis to this approach are elements for flexibly defining sophisticated questionnaires, called instruments, which not only contain information about the data to be collected, but also on how the instrument shall be processed on different mobile operating systems. For the latter purpose, we provide an advanced mobile (kernel) service that is capable of processing the logic of sophisticated instruments on various platforms. The paper discusses fundamental requirements for such a kernel and introduces a generic architecture. The feasibility of this architecture is demonstrated through a prototypical implementation. Altogether, the mobile service allows for the effective use of smart mobile devices in a multitude of different data collection application scenarios (e.g., clinical and psychological trials).

Keywords: Mobile Process Engine, Mobile Data Collection, Smart Mobile Device, Mobile Process, Mobile Healthcare.

1 Introduction

Smart mobile devices are increasingly used in everyday life. In line with this trend, application domains for which large amounts of data need to be collected (e.g., clinical trials) will significantly benefit from the use of mobile applications and data collection procedures will change. Corresponding scenarios range from fitness trackers up to applications monitoring vital parameters of chronically ill patients. However, realizing such mobile data collection applications requires profound knowledge on the demands from real-world scenarios.

In various large-scale mobile data collection applications we realized (cf. Table 1), domain experts (e.g., medical doctors and psychologists) were provided with specifically implemented mobile applications. The electronic questionnaires used in these scenarios (so-called *instruments*) not only provide questions, but also comprise sophisticated features for guiding their processing (i.e., answering). For example, instruments require a proper navigation between questions taking already given answers into account. Moreover, instruments need to be well tailored to provide statistically valid results. Note that in many application scenarios (e.g., clinical trials) this is of utmost importance. Recent approaches aim to realize such instruments as smart mobile applications to reduce the overall workload for domain experts by digitally transforming paper-based ways of collecting data. Compared to traditional paper-based questionnaires, the collected data needs not be digitized anymore after completing an instrument, significantly reducing transcription errors.

To cope with these drawbacks, we propose a generic framework [25] that allows domain experts to rapidly create executable, robust data collection instruments. According to this end-user programming approach, an instrument can be defined using a high-level modeling language (cf. Fig. 1, ①). The resulting specification is then automatically transformed into an executable process model (cf. Fig. 1, ③) using a well-defined mapping (cf. Fig. 1, ②). Subsequently, this process model can be deployed to mobile process engines running on smart mobile devices (cf. Fig. 1, ④). Providing such a process engine running as a *mobile service* on heterogeneous smart mobile devices, raises additional challenges. In particular, a modular process engine architecture is required to enable the processing of instruments on smart mobile devices. As the latter are often limited with respect to resources, a lightweight, but robust process engine is indispensable. The following requirements were derived from various case studies and mobile application engineering projects conducted (cf. Table 1) and must be considered in this context:

R1 Enable offline execution. The mobile process engine shall allow for an offline execution of deployed process models as well as for the storage of the collected data on the smart mobile device. For example, in the Burundi project (cf. Table 1, #4), an international team of psychologists could not rely on stable Internet connection in rural areas.

#	**Data Collection Applications**	**Country**	**CN**	**Releases**	**Instances**
1	Tinnitus Research	World-Wide	○	3	≥ 20,000
2	Risk Factors during Pregnancy	Germany	○	5	≥ 1,000
3	Risk Factors after Pregnancy	Germany	○	1	≥ 100
4	PTSD in War Regions	Burundi	●	5	≥ 2,200
5	PTSD in War Regions	Uganda	○	1	≥ 200
6	Adverse Childhood Experiences	Germany	●	3	≥ 150
7	Learning Deficits among Medical Students	Germany	●	3	≥ 200
8	Supporting Parents after Accidents of Children	Switzerland	○	5	≥ 2,500
	Sum *Σ*			24	≥ 26,350

CN = Complex Navigation; PTSD = Posttraumatic Stress Disorder

Table 1. Realized Mobile Data Collection Applications

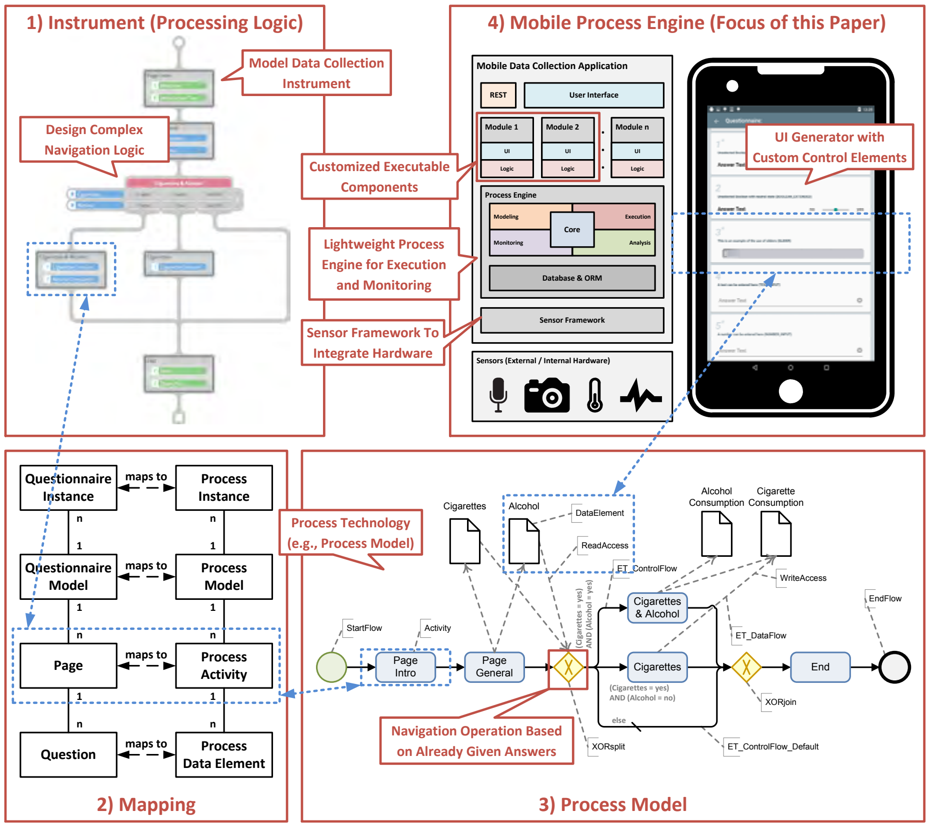
1) Instrument (Processing Logic)
Model Data Collection Instrument
Design Complex Navigation Logic
4) Mobile Process Engine (Focus of this Paper)
Mobile Data Collection Application
REST
User Interface
Module 1
Module 2
Module n
UI
Logic
Process Engine
Modeling
Execution
Core
Monitoring
Analysis
Database & ORM
Sensor Framework
Sensors (External / Internal Hardware)
Customized Executable Components
Lightweight Process Engine for Execution and Monitoring
Sensor Framework To Integrate Hardware
UI Generator with Custom Control Elements
Questionnaire Instance
maps to
Process Instance
Questionnaire Model
Process Model
Page
Process Activity
Question
Process Data Element
2) Mapping
Process Technology (e.g., Process Model)
Cigarettes
Alcohol
DataElement
ReadAccess
Alcohol Consumption
Cigarette Consumption
ET_ControlFlow
WriteAccess
EndFlow
StartFlow
Activity
Page Intro
Page General
Cigarettes & Alcohol
Cigarettes
End
ET_DataFlow
XORjoin
XORsplit
ET_ControlFlow_Default
(Cigarettes = yes) AND (Alcohol = yes)
(Cigarettes = yes) AND (Alcohol = no)
else
Navigation Operation Based on Already Given Answers
3) Process Model

Fig. 1. Overall Idea: 1) Modeling a data collection instrument, 2 & 3) Mapping the instrument to a process model, 4) Executing instances on smart mobile devices using a mobile process engine

R2 Enable flexible processing. The mobile process engine must support domain experts in changing (i.e., adapting) instruments during run-time. For example, the order of questions or labels often need to be flexibly changed in order to foster understandability of an instrument or to make it more convenient.

R3 Integrate sensors. The mobile process engine shall allow for the integration of sensors (e.g., heartrate sensors) in order to further enhance the value of the data collected. For example, [19] integrates the microphone of the smart mobile device to analyze the sound level of the environment during the processing of an instrument (i.e., during answering the questionnaire).

R4 Provide customizable user interfaces. The mobile process engine running on the smart mobile device shall dynamically create the user interface of the respective instrument based on its underlying model. For example, all information related to the structure, processing logic and design need to be taken into account by the rendering mechanism.

R5 Enable multilingualism. The mobile process engine shall enable domain experts to provide instruments in multiple languages. For example, three languages need to be provided for the instruments used in the Burundi project (cf. Table 1, #4).

R6 Support of different releases. The mobile process engine shall be able to cope with different releases of a data collection instrument. For example, one release of a particular instrument may only be suitable for underage subjects, while adults shall use another one.

R7 Enable real-time monitoring. The mobile process engine shall allow monitoring the current state of the running instances of an instrument. This allows, for example, to indicate the percentage of completion or to annotate the process model with run-time information.

R8 Enable multi-user support. The mobile process engine shall be able to handle multiple users as well as to distinguish between different roles. For example, when detecting risks during pregnancy (cf. Table 1, #2) certain questions may only be answered by a person with role `medical doctor`, while others may be answered by the `pregnant woman` itself.

This paper presents a lightweight mobile process engine for executing data collection instruments on smart mobile devices. In particular, this mobile process engine meets requirements R1 – R8. Moreover, a component for dynamically extending instruments is presented, which enables flexible adaptations of already deployed mobile applications during run-time. As opposed to hard-coded mobile data collection applications, changes of an instrument do not require its reimplementation and redeployment to respective smart mobile devices. In addition, data from multiple releases must not be merged manually in order to avoid inconsistencies. Finally, the validity of instruments can be ensured more easily.

Altogether, the approach enables flexibility regarding the design and execution of data collection instruments on smart mobile devices [24]. The remainder of the paper is structured as follows: Section 2 discusses fundamental requirements. Section 3 presents the architecture of the mobile engine, i.e., its *Execution* and *Analysis* components, whereas Section 4 illustrates their use in practice. Section 5 discusses related work and Section 6 concludes the paper and gives an outlook.

2 Background: The QuestionSys Framework

This section introduces fundamentals of the QuestionSys framework[1], focusing on the lifecycle phases related to mobile data collection. Both, the architecture of the framework and the mapping of paper-based instruments to mobile data collection applications are described.

To properly support domain experts in creating a mobile data collection instrument, all phases of its lifecycle need to be addressed. Note that related

[1] http://www.uni-ulm.de/en/in/dbis/research/projects/questionsys.html, accessed: July 13th, 2016

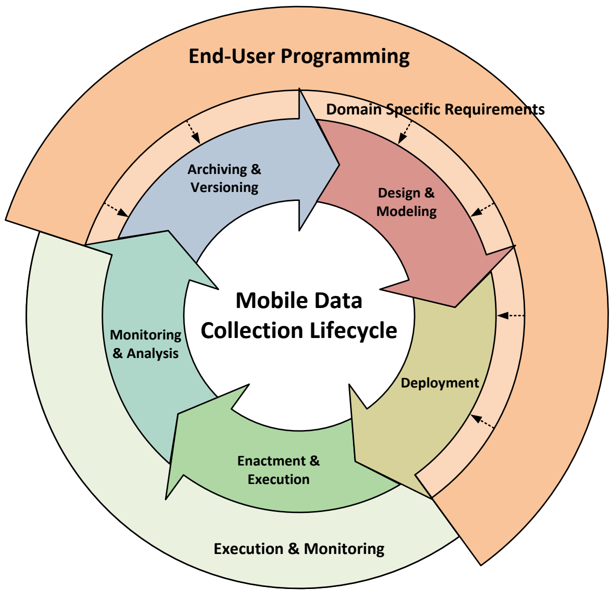

Fig. 2. Mobile Data Collection Lifecycle

approaches do not provide such an explicit consideration of the different phases for mobile data collection. In the light of a generic framework for mobile data collection applications, lifecycle management is crucial. Fig. 2 depicts the lifecycle of a mobile data collection application, which consists of five phases. The *Design & Modeling* phase shall enable domain experts (i.e., end-users) to create sophisticated mobile data collection applications with a complex logic (i.e., end-user programming). The *Deployment* phase deploys the latter on smart mobile devices. During the *Enactment & Execution* phase, multiple instances of the respective data collection instrument may be created and executed in a robust manner on the smart mobile devices. The *Monitoring & Analysis* phase, in turn, deals with the real-time analysis of the data collected on the smart mobile device. Finally, the *Archiving & Versioning* phase enables release management for mobile data collection instruments.

The QuestionSys framework we developed, provides an architecture supporting all phases of this lifecycle. As depicted in Fig. 3, the designed instrument model ① as well as rules for analyzing the data collected ② are mapped to XML documents. The latter are then automatically deployed to the respective smart mobile devices ③ capable of executing this model. Log files capturing execution information are stored using an XML structure to allow for their subsequent analysis ④. In this context, security ⑤ is ensured based on state-of-the-art data

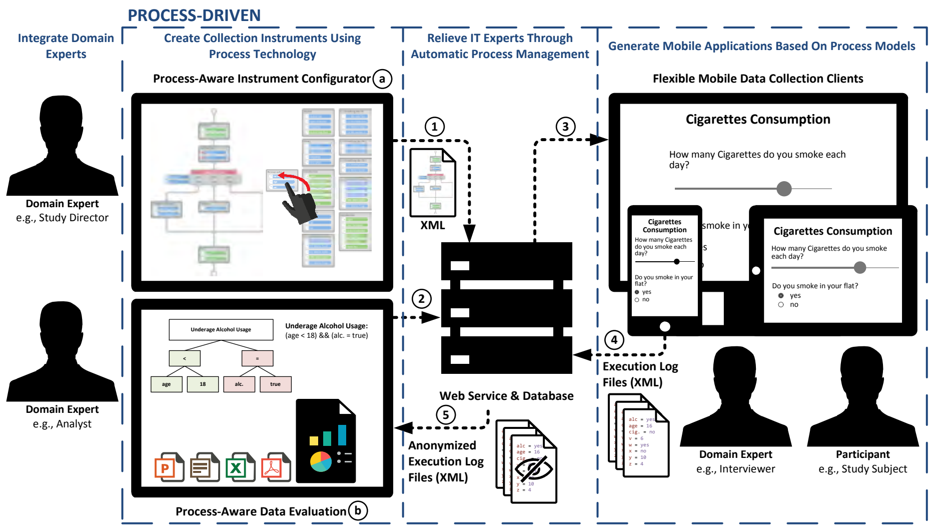
PROCESS-DRIVEN
Integrate Domain Experts
Create Collection Instruments Using Process Technology
Relieve IT Experts Through Automatic Process Management
Generate Mobile Applications Based On Process Models
Process-Aware Instrument Configurator (a)
Flexible Mobile Data Collection Clients
Domain Expert e.g., Study Director
(1) XML
(2)
(3)
Cigarettes Consumption
How many Cigarettes do you smoke each day?
Do you smoke in your flat?
yes
no
Underage Alcohol Usage
Underage Alcohol Usage: (age < 18) && (alc. = true)
age
18
alc.
true
Domain Expert e.g., Analyst
(4) Execution Log Files (XML)
Web Service & Database
(5) Anonymized Execution Log Files (XML)
Domain Expert e.g., Interviewer
Participant e.g., Study Subject
Process-Aware Data Evaluation (b)

Fig. 3. QuestionSys Architecture: Supporting Flexible Mobile Data Collection

encryption techniques. Note that the communication required for steps ① – ⑤ relies on Web Services [26]. Based on this automation, many challenging requirements of mobile data collection application projects are mitigated. For example, when releasing new versions of already existing instruments, IT experts are no longer required. Note that release management constituted the main cost driver in the context of the aforementioned mobile data collection projects (cf. Table 1). Finally, changes solely affecting the XML documents require implementation adaptations to be performed by IT experts. For example, new legal regulations may cause changes of the used data encryption algorithm.

In QuestionSys, the structure of an instrument is directly mapped to an executable process model, which then can be enacted by a lightweight process engine running on smart mobile devices. Using this model-driven approach, a separation of the processing logic of an instrument from actual application code [22] of the data collection application becomes possible. Thereby, a process model acts as the schema for executing instrument instances. This model, in turn, consists of process steps (i.e., activities) and edges expressing the control and data flow between them. Additionally, gateways (e.g., AND and XOR-splits) allow for more complex control flow structures.

Both the logic and the content of a paper-based instrument may be mapped to a process model using the described approach. In particular, *pages* of a questionnaire are mapped to *process activities*, whereas gateways matches respective navigation logic. Furthermore, *questions* correspond to *process data elements* connected to activities, which, in turn, may be used to store given answers (cf. Fig. 1, ② `Mapping`).

The data collection instruments may be created by domain experts using a process-aware configurator component (cf. Fig. 3, ⓐ). The latter provides an abstract and comprehensible modeling notation to specify the flow (i.e., processing) logic of the mobile data collection instrument. Navigation operations (e.g., decisions based on already given answers) as well as the data elements of instruments are modeled. Data elements, in turn, are connected to pages. Note that the latter are important for rendering instruments as they represent single screens on the smart mobile device and allow thematically structuring a questionnaire. In the context of questionnaire instruments, data elements represent questions, whereas navigation allows skipping questions (or even pages) depending on previously given answers. Finally, the configurator component allows defining rules for the automated evaluation of gathered data (cf. Fig. 3, ⓑ).

The work presented in this paper focuses on the *Enactment & Execution* as well as the *Monitoring & Analysis* phases. In this context, a mobile service providing a lightweight process engine for executing data collection instruments is developed. Furthermore, an approach for dynamically extending the logic of the already running smart mobile application is presented.

3 QuestionSys Mobile Service

This section presents the overall architecture of the realized mobile process engine. Furthermore, insights into the *Execution* and *Analysis* components are provided.

The lightweight mobile process engine we developed applies a service-driven approach. The engine comprises five components (cf. Fig. 4, left part): The most important one constitutes the core of the engine providing the data model. The latter represents the process model as well as components enabling robust interactions with process instances (e.g., start or stop activities). Although the process model relies on the ADEPT2 framework [21], other process meta-models may be used as well. For this purpose, the core provides functions to import process models. Furthermore, it comprises operations to map one model to another. The other components provide functions to support the different phases [27] of enacting process models locally on smart mobile devices. Note that these components only interact with the core itself and may be used as standalone functions as well (i.e., not all components are required). For example, the *Monitoring* component uses data provided by the *Execution* component to visualize the current state of the process instance or to provide information on upcoming process activities (e.g., delays or insufficient data). This loose coupling of the components (e.g., no other dependencies between components exist) allows for a customizable, but still lightweight mobile process engine.

The engine itself strictly follows the *Manager* pattern (cf. Fig. 4, right part). Each component corresponds to one specific manager, providing high-level APIs for interacting with the process engine. The main idea is to manage multiple entities of the same type. As an example, consider the *Execution* component, which offers an `ExecutionManager` for accessing functions related to

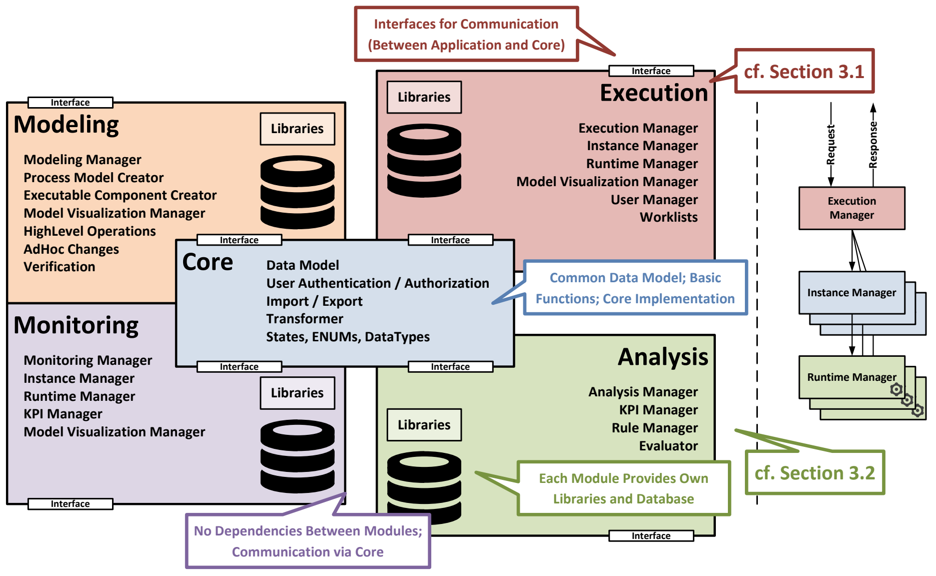

Fig. 4. Components of the Mobile Process Engine

the execution of process instances. Note that each running instance has its own `InstanceManager`, which manages control and data flow. Furthermore, the `RuntimeManager` is used to execute a specific activity of an instance.

As shown in Fig. 4, several components provide similar functions. Consider, for example, the `ModelVisualizationManager` provided by the *Modeling*, *Execution* and *Monitoring* component. In general, these components require different functions of the respective managers (e.g., various notations) and, therefore, must be implemented several times. For example, the *Modeling* component needs to provide all elements of the process meta-model (e.g., process activities, data elements, control and data flow), whereas the *Execution* component may only provide information regarding the current and upcoming activities to be executed. The interface shared for this manager, however, is defined by the core of the mobile process engine. In addition, each component contains its own persistence layer. For example, the *Execution* component stores information about the current state of the enacted process instance (including user information, timestamps, data produced and consumed), whereas the *Analysis* component stores evaluation rules as well as corresponding results for each process instance. These separated databases, in turn, foster the modular design of the process engine. Data between components, however, is shared through the core. Furthermore, each component may provide additional libraries to enhance functionality. For example, the *Analysis* component uses the Java Expression Language (JEXL) [4] for dynamically evaluating data elements of process instances.

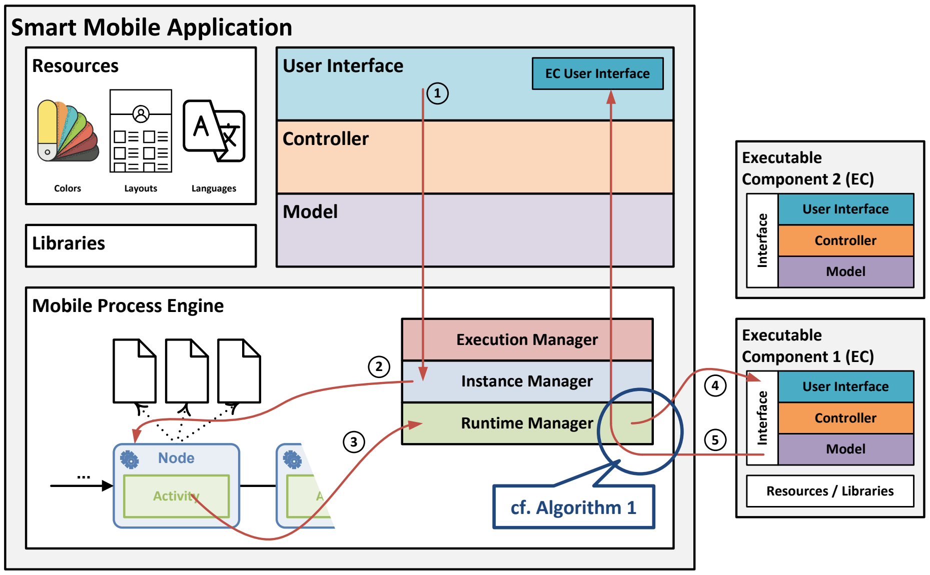

Fig. 5. Enacting Executable Components (`EC`s) During Run Time

3.1 Mobile Execution Component

Recall that the mobile process engine runs as a service and may be embedded into another application based on well-defined communication interfaces. The overall interaction between the mobile data collection application and the lightweight mobile process engine is shown in Fig. 5.

First, the user, interacting with the smart mobile application, starts a new instance of an instrument. The mobile data collection application, in turn, directly interacts with the process engine, which then provides access to the `ExecutionManager` ①. The latter offers functions that allow users to control a particular process instance (i.e., move to the next page of the instrument). *Second*, the `InstanceManager` validates whether or not the current node may be executed (e.g., if the user has appropriate access rights); all needed data elements are then provided ②. The node becomes activated and handed over by a *third* step to the corresponding `RuntimeManager`, which is able to call the linked *executable component* (`EC`). The latter covers several aspects. Its main functionality is to extract the main class file of the implementation of the `EC` as well as to create a list of all required *input* and *output variables* for the component to be called ③. *Fourth*, the `RuntimeManager` calls the respective component by invoking its main method ④ and passing both input and output lists to the `EC`. Algorithm 1[2] specifies how an `EC` is flexibly loaded and instantiated during run time. As an `EC` can be seen as a *Micro Service* [16], it may provide sophisticated

[2] Due to lack of space we only illustrate the algorithm for the Android platform.

Algorithm 1: Dynamically Loading an Executable Component

```
Data:
ctx: The current context of the application
nodeEC: Node containing meta information and the executable component
Result:
newEC: The instantiated EC to be used within the application
begin
    ExecutableComponent ec = null;
    String classPath = nodeEC.getClassPath();
    String packagePath = nodeEC.getPackagePath();
    String dexPath;
    DexClassLoader dexLoader = null;
    /* load the EC dynamically from an installed APK file                        */
    /* get Android PackageManager to find the installed package                  */
    final PackageManager pm = ctx.getPackageManager();
    /* search for ApplicationInfo of installed application with specified package path
       (Android 5.0 and higher: application id = package path)                   */
    ApplicationInfo appInfo = pm.getApplicationInfo(packagePath,
      PackageManager.GET_META_DATA);
    /* get filepath for installed .APK file of application                       */
    dexPath = appInfo.sourceDir;
    /* create DexClassLoader for APK file, where optOutPath is a cache directory for
       optimized loaded source files                                             */
    String optOutPath = ctx.getDir("random_name",
      Context.MODE_PRIVATE).getAbsolutePath();
    dexLoader = new DexClassLoader(dexPath, optOutPath, null,
      this.getClass().getClassLoader());
    /* instantiate EC with given classPath and initialize its environment (specified in
       the nodeEC)                                                               */
    newEC = (ExecutableComponent) dexLoader.loadClass(classPath).newInstance();
    newEC.setEnvironment(nodeEC.getEC(), packagePath, ctx);
    return newEC;
end
```

logic as well as an user interface for interaction. Note that the `EC` may contain its own resource files as well as libraries. In a *fifth* step, the `EC` user interfaces are passed back to the `ExecutionManager` and the respective data collection application. This allows the latter to embed it as UI fragment inside the main user interface ⑤. Note that interactions with the UI fragment of the `EC` (e.g., clicking a button) are handled by the logic of the `EC` itself and not by the *surrounding* mobile data collection application.

If the respective `EC`, which is executed as a mobile service, satisfies certain conditions (i.e., all mandatory elements are correctly filled in), it produces the `canBeFinished` event. The latter indicates that the coordinating `RuntimeManager` will safely terminate the `EC`. Furthermore, all *output variables* of the terminated `EC` are transferred back to the `InstanceManager`, which stores them in the corresponding *data elements* of the process instance. Log files containing the data collected during the execution of a specific instrument instance may be accessed by other components using the `ExecutionManager`.

3.2 Mobile Analysis Component

The mobile analysis component allows for the analysis of instrument instance collections following a rule-based approach. Fig. 6 describes the corresponding

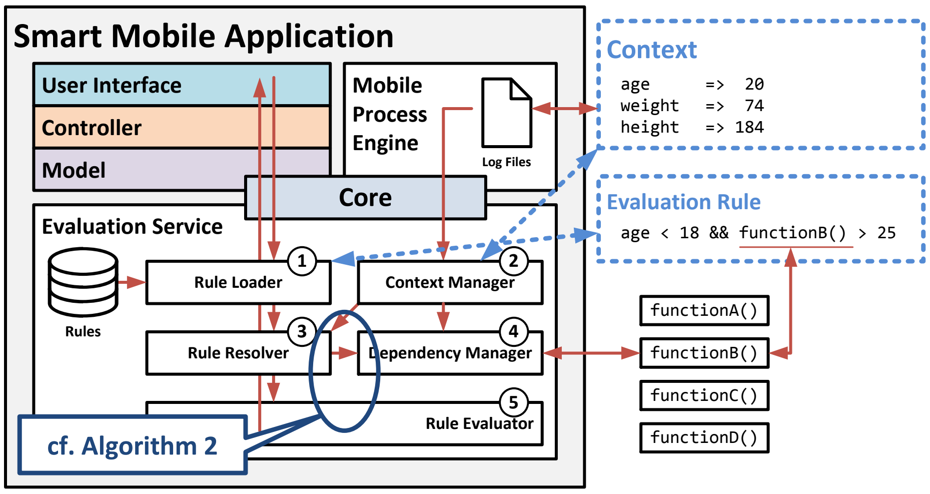

Fig. 6. Analyzing an Instrument Instance Using Evaluation Rules

procedure. The rules, which are created by domain experts, are stored locally in the database of the *Analysis* component. *First*, the user starts the analysis by selecting a specific process instance (i.e., instrument instance). This request is then sent to the manager coordinating the further steps. For example, all relevant rules must be loaded ①. *Second*, for the specific process instance, execution log data is requested from the *Execution* component in order to create the `Context` ② of the analysis. Both, the selected rules and the analysis context are then sent to the `RuleResolver`, which then replaces variables with the respective data values collected ③.

Note that an evaluation rule, in addition to variables and simple comparisons, may comprise user-defined functions specific to the respective application scenario. For example, the `calculateBodyMassIndex(int weight, int height)` function may be used to detect obesity of the subjects interviewed. However, as user-defined functions depend on the respective use case of the mobile data collection application as well as the instrument, the latter are not directly integrated with the analysis component. Instead, a `DependencyManager` ④ invokes the function needed using an approach similar to the one for calling executable components. Note that these components only need to provide the logic realizing the required function (e.g., calculate the `BodyMassIndex`), but need not to provide a user interface. Finally, the `RuleEvaluator` ⑤ checks for the satisfiability of the respective rule, the given context, and the functions needed. The result is returned to the smart mobile application, which may provide additional information to the user. For example, contact details of a physician may be displayed if a specific behavior is detected when analyzing the subject's data. Algorithm 2 shows, how a specific instrument instance is analyzed using a given rule.

Algorithm 2: Evaluating an Instance Using Rules

```
Data:
instance: The instance to be analyzed
rule: The rule to be evaluated
ctx: The current context of the application
Result:
report: The report containing information about the evaluation
begin
    EvaluationReport report = new EvaluationReport();
    RuleEvaluator ruleEngine = new RuleEngine();
    Expression ruleExpression = ruleEngine.createExpression(rule.getExpression());
    RuleContext ruleContext = new RuleContext();
    /* Load and instantiate dependencies from external JAR file                */
    for ClassDependency cd : rule.getDependencies() do
        Object op =
          loader.loadClass(cd.getDependency()).getConstructor(Instance).newInstance(instance);
        /* Add operation object to ruleContext                                 */
        ruleContext.set(cd.getDependency(), op);
    end
    /* Add data to ruleContext. ruleContext is basically a key-value-store     */
    for DataValue dv : inst.getDataValues() do
        ruleContext.set(dv.getElementName(), dv.getValue());
    end
    /* Evaluate Expression on created ruleContext                              */
    if (Boolean)ruleExpression.evaluate(ruleContext) then
        report.setResultText(rule.getPositiveText(), ctx.getLocation());
    else
        report.setResultText(rule.getNegativeText(), ctx.getLocation());
    end
    return report;
end
```

4 Evaluation

In order to demonstrate the feasibility of the approach, the QuestionSys mobile service we implemented is applied to real-world scenarios. Furthermore, we discuss selected issues.

4.1 Mobile Service in Practice

In order to validate the presented architecture, a mobile application supporting scientists in collecting trial data was realized and applied in practice.

Fig. 7 presents screenshots of this mobile data collection application. Note that the overall user interface as well as resources (e.g., icons, color schemes) are provided by the main application itself. The user interface of the respective data collection form, however, is provided and rendered by the executable component (`EC`) independently. This executable component may contain additional resources (e.g., to overwrite existing styles) or add component-specific images. The Android-specific *floating button* for proceeding to the next screen of the application (i.e., next page of the instrument), in turn, is provided and rendered by the main application. This button, however, may only be displayed if the interviewer has answered all mandatory questions. The executable component, therefore, produces the `canBeFinished` event indicating that it may be terminated.

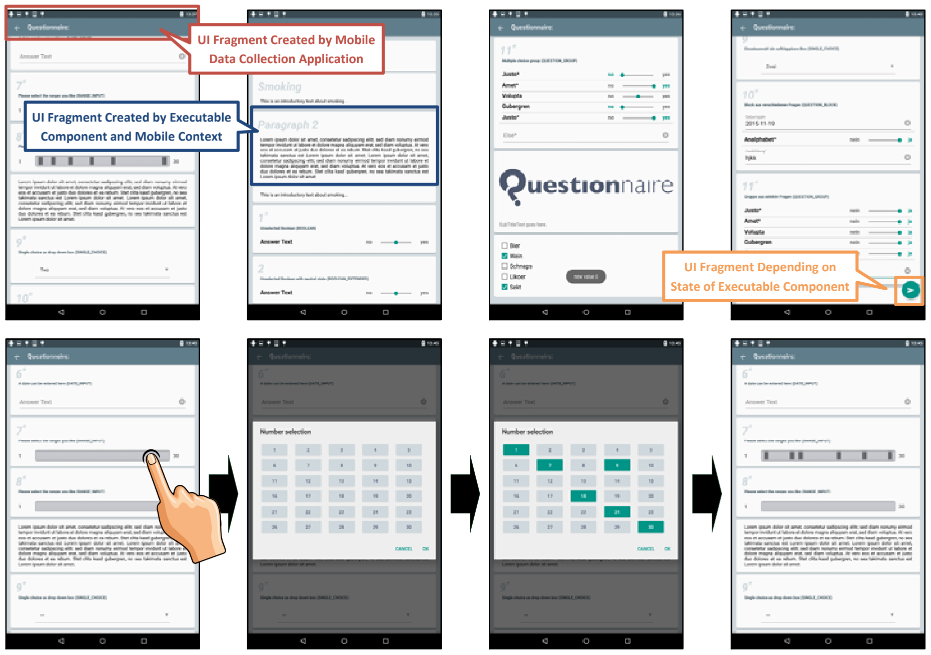
UI Fragment Created by Mobile Data Collection Application
UI Fragment Created by Executable Component and Mobile Context
UI Fragment Depending on State of Executable Component
Questionnaire
Number selection

Fig. 7. Realized Mobile Data Collection Application

Clicking the floating button will try to terminate the currently running executable component and persist all collected data. Finally, the `InstanceManager` evaluates the next activity in the respective process instance to be started.

The bottom row of Fig. 7 presents a custom input element (i.e., it is not available by default) that may be used to enter multiple non-contiguous ranges. Consider, for example, the question *"Select the pregnancy weeks where complications have occurred?"* from an instrument used in the context of pregnancies (cf. Table 1, #2, #3). If the woman interacting with the smart mobile device selects the range input field, a modal dialog is displayed, providing available values (i.e., values from 1 to 30, depending on the actual pregnancy week). When closing the dialog, values are directly marked within the input field indicating the selection.

4.2 Discussion

This section discusses selected issues of the QuestionSys mobile service with respect to the requirements R1 – R8 (cf. Section 1).

A1 Implementation challenges. When designing the architecture of the mobile engine, extensibility was a particular goal. Specifically, the concept of executable components fosters the service-driven approach as the latter may be controlled by the engine itself. Altogether, an easy adaptation of data collection applications becomes possible, as only these components have to be adapted when new application-specific requirements arise.

A2 Concept evaluation. Section 4.1 introduced an advanced mobile data collection application from the psychological domain. In order to get valuable information regarding the acceptance of different users involved in the *process of data collection*, several pre-studies were conducted. Furthermore, a field study is currently on its way. Although mobile data collection applications have already proven feasibility in different settings [15,2], this user-driven approach needs to be evaluated more deeply in additional studies.

A3 Alternative approaches. Apparently, there exists other approaches to implement mobile data collection applications instead of mapping instruments to process models. For example, hard-coding applications may still be acceptable, depending on the respective application scenario. However, this would limit domain experts in flexibly adjusting their instruments. To realize these applications, profound programming skills are necessary.

A4 Multi-user and role support. We need to evaluate, which approach for supporting multiple users interacting within respective mobile data collection scenarios may be applicable. On the one hand, a dedicated user management handled by the lightweight process engine itself might be suitable. On the other, Android offers a sophisticated user management on its own. In order to use this approach, however, the engine must be installed as service accessible for users running on respective smart mobile devices.

On one hand, the process-driven modeling supports domain experts to create mobile data collection instruments on their own. On the other, process technology enables the flexible execution of instruments on smart mobile devices. Therefore, a framework enabling such advanced mobile data collection applications on smart mobile devices is indispensable.

5 Related Work

Two categories of related work need to be discussed in the context of this paper.

5.1 Mobile Process Engines

Executing business processes on mobile devices has been addressed by several papers. Some of them provide proprietary execution languages specifically designed for the respective application, whereas others provide middleware services or frameworks enabling developers to create process-aware mobile applications.

In [12], a context-aware execution language for business processes is presented. As one of the core concepts of this language, specific aspects of the mobile device are considered when executing a process instance. For example, the battery status or the current location of end-users might affect the execution (e.g., decisions within XOR gateways). [8] presents extensions for WS-BPEL when integrating mobile devices into business processes. In certain scenarios the number of available mobile devices to be coordinated is unknown. In order to cope with this issue, *Partner links* bound to multiple endpoints are introduced.

In [18], an iPad application supporting medical staff during ward rounds is presented. Besides reviewing medical records, the staff may add further information during rounds. However, if certain keywords are documented, a corresponding process is started (e.g., the keyword *blood test* may trigger process *laboratory analysis*) using a lightweight process engine. Although the concept of automatically invoking processes based on user input data is promising, the functionality of the respective engine is quite limited, as gateways are not supported, but only sequences of activities. Besides this limitation, only simple tasks may be executed, which need to be directly implemented in the iPad application.

A workflow engine that can run on PDAs is introduced in [17,9,6,28,13]. All approaches use WS-BPEL to specify the business processes to be executed. Furthermore, they rely on Web Service standards (e.g., WSDL and SOAP) to specify the activities to be called. Some approaches use HTML for displaying a user interface. In order to execute specific activities, some use own extensions for WS-BPEL, whereas others ship with an Apache Server in order to execute scripts. However, all approaches provide core activities, like a browser for displaying user forms, maps, a calendar and basic messaging services.

In the ROME4EU project [23], a scenario for disaster management, where no stable Internet connection is available, is addressed. Particularly, the coordination of emergency teams must be controlled and organized in the field, which is provided by a process engine running on the smart mobile device of the team leader. Note that this work relies on WS-BPEL regarding the definition and execution of the processes (supporting activity sequences, conditional branches, and parallel routing). The disaster management leader may assign tasks to other team members (i.e., they can work on respective tasks) or automatic services in order to complete the process. As all mobile devices are connected to the leader (e.g., via ad-hoc network), this device acts as orchestrator within the service-oriented architecture, posing a critical point-of-failure.

In [20], a process engine for smart mobile devices is presented. It allows defining process models on a server component, which are then fragmented and deployed on the respective mobile devices for execution. However, the authors only consider the *Execution* phase to be enacted on the smart mobile device itself. Other phases of the BPM lifecycle need to be covered by the server.

5.2 Model-Driven Approaches

Obviously, there is a lot of related work on model-driven approaches. However, in the context of this paper, we focus on model-driven architectures and software development with respect to smart mobile devices.

In [10], an approach for model-driven software development for mobile applications is presented. The authors argue that model-driven development is well understood in regular desktop and server scenarios, but not discussed in detail for smart mobile devices so far. Therefore, the authors introduce a cross-development approach for the latter. Software developers are able to describe the problem (e.g., the business scenario, data types or device features needed) using a meta-programming language, which is then translated to respective platform-specific native code (e.g., Java for Android). Furthermore, it allows automatically generating backend code for a server component offering common `CRUD` operations for the interaction with respective generated mobile applications.

Similar to the previously mentioned approach, [5] introduces a WYSIWYG editor to create mobile applications. Furthermore, the approach borrows techniques known from Apple's Storyboard. The modeled application is mapped to a platform independent model and finally compiled to a native language and deployed to respective platforms.

[7] discusses a framework that allows specifying services that run on smart mobile devices. The authors rely on the XForms standard, which holds information about the model and user interface in order to manipulate the latter. The model is then transferred to the *Service Broker* component for the actual processing. In order to allow for an easy development of respective XForms and services, the presented approach provides a graphical editor implementing a well-defined domain-specific language. The model derived from this language may then be transformed to a graphical user interface running on smart mobile devices. Using this graphical notation, decisions may be modeled as well, allowing adapting the user-interface and mobile application during run time.

[11] introduces an approach for enabling domain experts to model care plans for people suffering from chronic diseases. A code generator then transforms this plan to an Adobe Flash or DHTML application which can be installed on smart mobile devices. This enables physicians to provide mobile applications specifically tailored to one patient, triggering reminders or asking about his physical wellbeing.

6 Summary and Outlook

Based on the insights we gained in several data collection scenarios, this paper advocates the need for sophisticated mobile services running on smart mobile devices. In order to mitigate the efforts between IT and domain experts, a sophisticated framework allowing domain experts to model data collection instruments themselves was proposed. In this context, a mobile service became necessary to process instances of instruments directly on smart mobile devices. For this purpose, we present a flexible and modular architecture of a lightweight process engine. In particular, this architecture allows extending the functionality of already installed mobile data collection applications during run-time based on the concept of `executable components`. These components allow providing domain-specific logic as well as dynamically generated user interfaces for activities executed by the process engine. Furthermore, the *Analysis* component enables comprehensive analysis of the data collected during run time based on (user-defined) rules. Along an application scenario, benefits of using presented mobile services were discussed.

To further validate the presented approach, a study for evaluating the user interface as well as the user experience working with the realized mobile data collection application is currently conducted. In particular, differences between the latter and paper-based questionnaires considering complex navigation features are evaluated. In addition, the novel interaction elements (e.g., slider with no initial state) need to be evaluated. In order to leverage the overall functionality of the proposed lightweight mobile process engine, *process mining* algorithms [1] and approaches known from *business intelligence* [3] may be added to the *Analysis* component. Also, *Monitoring* may benefit from *complex event processing* techniques [14]. Finally, `executable components` using sensors may be realized allowing domain experts to collect additional information during enactment.

Altogether, the presented approach will significantly change the way instruments may be used in practice (e.g., clinical trials). Moreover, due to its flexibility, the proposed architecture may be suitable for other life domains relying on collecting and processing data in mobile scenarios.

References

1. Van der Aalst, W.M., Weijters, A.: Process mining: a research agenda. Computers in Industry 53(3) (2004)
2. Ainsworth, J., Palmier-Claus, J.E., Machin, M., Barrowclough, C., Dunn, G., Rogers, A., Buchan, I., Barkus, E., Kapur, S., Wykes, T., et al.: A comparison of two delivery modalities of a mobile phone-based assessment for serious mental illness: native smartphone application vs text-messaging only implementations. Medical Internet Research 15(4) (2013)
3. Anandarajan, M., Anandarajan, A., Srinivasan, C.A.: Business Intelligence Techniques: A Perspective from Accounting and Finance. Springer Science & Business Media (2012)
4. Apache Commons: Java Expression Language (JEXL). `http://commons.apache.org/proper/commons-jexl/`, last visited: July 10, 2016
5. Balagtas-Fernandez, F., Tafelmayer, M., Hussmann, H.: Mobia Modeler: Easing the Creation Process of Mobile Applications for Non-Technical Users. In: Proc of the 15th Int'l Conf on Intelligent User Interfaces. pp. 269–272. ACM (2010)
6. Baresi, L., Maurino, A., Modafferi, S.: Workflow Partitioning in Mobile Information Systems. Proc. IFIP TC8 Working Conf on Mobile Information Systems pp. 93–106 (2004)
7. Dunkel, J., Bruns, R.: Model-Driven Architecture for Mobile Applications. In: Int'l Conf on Business Information Systems. pp. 464–477. Springer (2007)
8. Hackmann, G., Gill, C., Roman, G.C.: Extending BPEL for interoperable pervasive computing. In: Pervasive Services, IEEE Int'l Conf on. pp. 204–213. IEEE (2007)
9. Hackmann, G., Haitjema, M., Gill, C., Roman, G.C.: Sliver: A BPEL workflow process execution engine for mobile devices. In: Service-Oriented Computing (ICSOC 2006), pp. 503–508. Springer (2006)
10. Heitkötter, H., Majchrzak, T.A., Kuchen, H.: Cross-Platform Model-Driven Development of Mobile Applications with md^2. In: Proc of the 28th Annual ACM Symp on Applied Computing. pp. 526–533. ACM (2013)
11. Khambati, A., Grundy, J., Warren, J., Hosking, J.: Model-Driven Development of Mobile Personal Health Care Applications. In: Proc of the 23rd IEEE/ACM Int'l

Conf on Automated Software Engineering. pp. 467–470. IEEE Computer Society (2008)

12. Kocurova, A., Oussena, S., Komisarczuk, P., Clark, T.: MobWEL - Mobile Context-Aware Content-Centric Workflow Execution Language. In: 3rd Int'l Conf on Advanced Collaborative Networks, Systems and Applications. pp. 61–70 (2013)
13. Kunze, C., Zaplata, S., Lamersdorf, W.: Mobile Processes: Enhancing Cooperation in Distributed Mobile Environments. Journal of Computers 2(1), 1–11 (2007)
14. Luckham, D.C.: Event Processing for Business: Organizing the Real-Time Enterprise. John Wiley & Sons (2011)
15. Mudano, A.S., Gary, L.C., Oliveira, A.L., Melton, M., Wright, N.C., Curtis, J.R., Delzell, E., Harrington, T.M., Kilgore, M.L., Lewis, C.E., et al.: Using tablet computers compared to interactive voice response to improve subject recruitment in osteoporosis pragmatic clinical trials: feasibility, satisfaction, and sample size. Patient Preference and Adherence 7, 517 (2013)
16. Newman, S.: Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, Inc. (2015)
17. Pajunen, L., Chande, S.: Developing workflow engine for mobile devices. In: 11th Int'l Conf Enterprise Distributed Object Computing. pp. 279–279. IEEE (2007)
18. Pryss, R., Mundbrod, N., Langer, D., Reichert, M.: Supporting medical ward rounds through mobile task and process management. Information Systems and e-Business Management 13(1) (February 2015)
19. Pryss, R., Reichert, M., Langguth, B., Schlee, W.: Mobile Crowd Sensing Services for Tinnitus Assessment, Therapy and Research. In: IEEE 4th Int'l Conf on Mobile Services. IEEE Computer Society Press (June 2015)
20. Pryss, R., Tiedeken, J., Kreher, U., Reichert, M.: Towards Flexible Process Support on Mobile Devices. In: Proc. CAiSE'10 Forum - Information Systems Evolution. pp. 150–165. No. 72 in LNBIP, Springer (2010)
21. Reichert, M., Dadam, P.: Enabling Adaptive Process-aware Information Systems with ADEPT2. In: Cardoso, J., van der Aalst, W. (eds.) Handbook of Research on Business Process Modeling. Information Science Reference, Hershey, New York (March 2009)
22. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer, Berlin-Heidelberg (2012)
23. Russo, A., Mecella, M., de Leoni, M.: ROME4EU - A service-oriented process-aware information system for mobile devices. Software: Practice and Experience 42(10), 1275–1314 (2012)
24. Schobel, J., Pryss, R., Schickler, M., Reichert, M.: Towards Flexible Mobile Data Collection in Healthcare. In: 29th IEEE Int'l Symp on Computer-Based Medical Systems. IEEE Computer Society Press (June 2016)
25. Schobel, J., Pryss, R., Schickler, M., Ruf-Leuschner, M., Elbert, T., Reichert, M.: End-User Programming of Mobile Services: Empowering Domain Experts to Implement Mobile Data Collection Applications. In: IEEE 5th Int'l Conf on Mobile Services. IEEE Computer Society Press (June 2016)
26. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR (2005)
27. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer Science & Business Media (2012)
28. Zaplata, S., Hamann, K., Kottke, K., Lamersdorf, W.: Flexible Execution of Distributed Business Processes Based on Process Instance Migration. Journal of Systems Integration 1(3), 3–16 (2010)