

## Chapter 2

# Specifications and Deployment of SOA Business Applications Within a Configurable Framework Provided as a Service

Adam Grzech, Krzysztof Juszczyszyn, Grzegorz Kołaczek, Jan Kwiatkowski, Janusz Sobecki, Paweł Świątek and Adam Wasilewski

**Abstract** This chapter presents the concept of development and management of SOA applications within the configurable service platform which supports all phases starting from business process definition. The unique features of the platform include: business process compatibility, easy reconfiguration of composition schemes, visual support for requirements and service definition, QoS assessment (including communication services) and service execution control. Moreover, it illustrates how effective tools for SOA management may be developed within the SOA paradigm itself, and how this paradigm may be used to achieve their interoperability and flexibility.

## 1 Introduction

This chapter is devoted to presenting the main components and functionalities of PlaTel (platform for ICT planning and monitoring solutions), an integrated framework supporting business processes in distributed ICT (Information and Communication Technologies) environment based on the Service Oriented Architecture (SOA) paradigm. The key architectural assumptions of the PlaTel are: modular organization, reconfiguration capability, full compliance with the SOA paradigm. In result, the users are offered extended capabilities of building SOA applications starting from the definition of business processes which are translated to composite services' requirements. All components of PlaTel are implemented as services which allows to utilize the features of SOA paradigm in platform configuration and management. The framework scope of functionalities is divided into applications that cover the entire life cycle of business-oriented ICT applications. The framework's applications

---

A. Grzech (✉) · K. Juszczyszyn · G. Kołaczek · J. Kwiatkowski · J. Sobecki · P. Świątek · A. Wasilewski  
Faculty of Computer Science and Management, Institute of Computer Science,  
Wrocław University of Technology, Wrocław, Poland  
e-mail: adam.grzech@pwr.wroc.pl

and tools are responsible for business requirement analysis and comparison, service choices or the composition of services, efficient communication and computational resource provisioning and allocation in a distributed, virtualized environment as well as for the utilization of resources and the quality of monitoring and management services.

The presented framework covers service request processing, beginning with a service request's arrival until the completion of the request and evaluation of the delivered service. The platform's building blocks (modules) assure service request processing, composition of composite services—taking into account the functional and non-functional requirements, the provision for efficient communication and computational resources, the execution of services in a virtual distribution environment, and the evaluation of the quality of composite services. All of the aforementioned modules can be applied as independent data processing units or as sequences of such units that support selected sections of the requirements and service-matching processes for any given business process supported by SOA-based or legacy information systems. All the applications within the platform are designed and implemented as open, modular, scalable, and heterogeneous components (services) that are integrated according to the SOA paradigm. Such an approach allows for the integration of all or selected applications to support various decision-making processes.

So far, we can find many approaches to the problem of business process execution by means of available software services [25]. In work [26], it is mentioned that, rather than starting with a complete business process definition, the composition system could begin with a basic set of requirements and in the first step to build the entire process, whereas many approaches [26] require a well-defined business process to generate such a complex service. Current work often raises the topic of the semantic analysis of user requirements, service discovery (meeting the functional requirements), and the selection of specific services against non-functional requirements (i.e. execution time, processing and communication costs, security, etc.). However, the presented solutions have some disadvantages, i.e. these methods have not yet been successfully combined to jointly and comprehensively solve the problem of the composition of complex services that satisfy both functional and non-functional requirements. In many cases, only one aspect is considered. For example, the work [57] focuses on the selection of services based only on one functional requirement at a time. Other works show that non-functional requirements are considered to be of key importance; however, many approaches ignore the aspect of building a proper complex service structure that is key to the optimization of execution time, for example.

To date, the service composition problem has been approached from different perspectives. Some have presented specialized methods for selection services or composite service QoS-based optimization [26]. However, despite the importance of their contribution, those solutions are not widely used. Some propose complete end-to-end composition tools that introduce the concept of a two-stage composition: the logical composition stage to pare back the set of candidate services and then the generation of an abstract workflow [2]. METEOR-S presents a similar concept of binding web services to an abstract process and selecting services that fulfill the

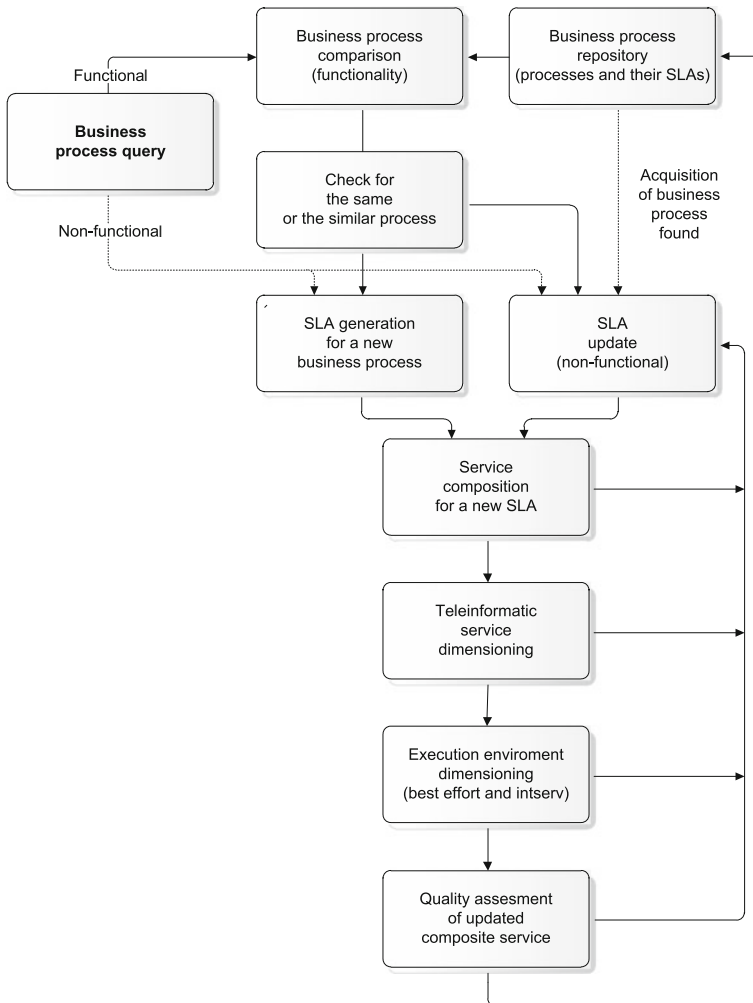
QoS requirements. Notions of building complete composition frameworks are also clear in SWORD, which was one of the initial attempts to use planning to create web services [47]. However, the proposed approaches are closed and do not support the implementation of other methods and, because of this, it is difficult to call them frameworks. Also, a framework-based approach is what is currently needed in the SOA field in order to create composition approaches that are suited to different domains and problems that are characteristic for them.

However, the service composition task is only a partial solution of the deployment of the SOA business applications problem. Numerous works point out those SOA business applications responsible for the realization of business processes should be constructed in accordance with business process description techniques [20]. Moreover, they should maintain the quality requirements imposed on the business process realization by means of composite services. It is especially important in the event that hardware devices and telecommunication services are involved.

Taking the conditions above into account, the PlaTel framework presented in this chapter offers a general solution for addressing all of the key issues involved in the deployment of SOA business applications: a process-level description of services, service composition, provisioning, dimensioning and execution, service validation and security assessment as well as the generation of user interfaces for semantically described services. All of these features constitute the generic PlaTel process for the deployment of SOA applications (Fig. 1) and are discussed in detail in the following sections.

It is started with the business process query, complemented by its associated non-functional requirements (cost, security, time constraints, etc.). The query is processed by means of comparing it with existing processes stored in the repository which contain associated Service Level Agreements (SLA), defining the requirements for composite services. In the event that there are no adequate processes, a new SLA is created. On the basis of SLA, an SOA application (composite service) is composed and dimensioned according to its functional and non-functional requirements. Composed and dimensioned services are executed by the native PlaTel service execution engine environment. The entire process undergoes validation and quality assessment in which the results are stored in a repository to be used to process future queries.

All stages of the SOA business application deployment are performed by dedicated PlaTel services that use common repositories. The PlaTel framework provides a multi-criteria (functional and non-functional) composition of Web services and QoS assurance for utilized ICT services (bez zmian) that allows for the construction of various service selection and composition scenarios. It also utilizes configurable composite services to provide for its functionality. In order to ensure the interoperability with external service and ontology repository specialized services, Mediators, responsible for database access and providing standardized interfaces for internal PlaTel services— had been created. As a result, external resources can be integrated within the PlaTel framework (which was equipped with access control functions). All of these combined features offer a flexible and extensible environment for communication and composition between the components of SOA applications. To the best



**Fig. 1** Deployment of SOA business applications within the PlaTel framework

of our knowledge, it is the first framework for service composition that implements this kind of holistic and flexible approach.

There are several unique features of the PlaTel framework, associated with the various stages of SOA business application deployment:

- All the key components of the PlaTel (namely: service composer, communication mapper, and execution engine) are composite services themselves, which implies that their execution schemes may be easily reconfigured or extended—and a specialized graphic user interface is provided to support such actions. This approach allows for the creation of repositories of dedicated composite services for the com-

position, mapping, and retrieval of the Web services, which may be used according to current needs. They may be stored and conditionally chosen from the repository.

- To address the non-functional requirements of Web services, an extensible description language (SSDL—Smart Service Description Language), which allows for the expression of arbitrary non-functional parameters of a composite service, is proposed.
- The execution engine interprets the SSDL definitions of composite services and maintains the non-functional parameters. It has two methods of execution—interpretation (the components of a composite service are explicitly executed) and composite service emulation (in this case, the execution engine reads the SSDL and communicates like a service described in the SSDL definition, thus supporting the automatic deployment of composite services). The framework also allows the use of external execution engines (in this scenario, the PlaTel engine serves as an SSDL-driven interface for them).
- The issue of communication between Web services is being addressed by providing a communication mapper—a dedicated service which supports a service composer and substitutes each node of the composite service scenario graph with a sub-graph representing the order of the execution of atomic ICT services (communication and computational) necessary to link domain atomic services (communication services provide an appropriate medium for the transmission of data between computational services, while computational services are responsible for data processing tasks such as: encryption, encoding, signal merging/splitting, etc.).

The main contribution of this work is at present a novel approach to service selection, composition, and execution, in which the abovementioned functionalities are provided as services. The PlaTel framework allows also the composition of communication services and supports flexibility via configuration mechanisms for basic processes (composition, mapping, and the execution of services).

In the following sections the details of the main activities required to complete a business process query request depicted in Fig. 1 are provided. The next section presents the basics of the business process description in the context of services and proposes an original PlaTel Event-driven Process Control notation that supports the service-oriented process description and SLA generation. Section 3 covers the composition and provisioning of composite services within the original architecture, allowing for a flexible approach to the composition task and the use of various communication and computational services which are used by the domain-oriented services. Section 4 is devoted to the PlaTel service execution environment and presents its original service-based execution engine and scalable execution environment. The next section discusses service validation within the layered model of SOA governance along with a method for composite service security assessment. Finally, Sect. 6 proposes an approach to the automatic generation of user interfaces for semantically-described composite services.

## 2 Service-Based Business Processes Description

This section is devoted to the presentation of the advantages and disadvantages of well-known and successfully applied business process modeling attempts and notations in terms of their suitability for the simultaneous description of the functional and non-functional requirements that are important for composite service composition. Results of a critical and comprehensive analysis of contemporary modeling approaches leads to the specification of invented modeling notation PEPC (PlaTel EPC), thus expanding the functionality of the well-known notations by adding new opportunities required in composite service composition processes.

Models of business processes are used as an input for service composition. A PlaTel customer expects a composite service that meets his/her functional and non-functional requirements. Functional requirements may be stated as a business process flow or a functional part of the SLA (Service Level Agreement). Non-functional requirements in typical approaches to BPM (ARIS, BPMN) have to be included in the SLA. In PEPC, notation for non-functional requirements can be set for each object in the business process model.

In the first step of the PlaTel workflow (Fig. 1), the customer's functional requirements are compared with the functional requirements defined for the business process models available within the business process repository. Next, the set of known business process models containing the expected functionality is compared due to the structural similarity with the customer's business process. If an acceptable model is found in the repository, then the non-functional requirements are updated. If there is not a suitable business process model in the business process repository, then required business processes have to be modeled using templates (patterns) from the business process repository or beginning with the blank model (PEPC modeler supports both options).

A new business process model provides information regarding functional requirements. A customer's non-functional requirements are added to the SLA (or to the PEPC model) and PlaTel is ready for service composition.

### 2.1 Business Process Modeling (BPM)

Each organization has its own formal or informal rules and patterns of conduct that result from the experience, knowledge, and competence of employees. Usually taken actions (activities) are arranged in a sequence with clearly-defined start and end triggers, targets, and human or system owners. Such sequences—business processes—require the involvement of many people from different organizational units. This means that the process actors do not have complete vision of the process and knowledge of the actions taken at all process stages. The solution to this problem is business analysis and then the mapping and modeling of business processes called Business Process Modeling (BPM). BPM supports standardization of the

management of business processes that pass through multiple data repositories, applications, departments, or even companies [41]. Business analysis lets for the understanding of how organizations work to accomplish their purposes and defining the capabilities the organization requires to provide products and services to external stakeholders [1]. On that basis, BPM allows the representation of enterprise processes and the capture of an ordered sequence of business activities and supporting information.

Key benefits of BPM are the following [21]:

- formalization of existing processes and spotting needed improvements,
- facilitation of an automated, efficient process flow,
- increase of productivity and decrease in the number of employees,
- opportunity for people to solve the difficult problems.

### 2.1.1 Basic Rules of BPM

Models of business processes may be designed to answer the question “Where are we now (“as-is” perspective) or “Where do we want to be?” (“to-be” perspective). The first option gives a common, baseline model which represents an accurate depiction of the actual situation within the organization. When the model is developed, it may be used to analyze and improve the business process. The second option lends itself to providing a vision of an organization’s development. Such diagrams present how the future processes might appear after incorporating the proposed improvements. Moreover, such models may be used to demonstrate and simulate the new processes before their implementation.

Before starting any business process modeling, it is important to be clear about several basic questions that can be extended by detailed questions (Table 1).

Comparing the ARIS methodology (especially EPC diagrams) and BPMN can draw the following conclusions:

- ARIS methodology enables a more accurate modeling of an organization’s information system, but modeling of the information system requires diagrams from every ARIS perspective,
- BPMN enables more a accurate modeling of the process flow,
- BPMN does not support modeling of the context of business processes, including the linkages between the processes.

In addition:

- The ARIS methodology is widely-used for business process modeling for the implementation of integrated information systems (in particular, SAP products for which ARIS offers dedicated diagrams)—this means that many organizations (especially large and medium size) prepare maps of processes using the ARIS methodology during the implementation of the ERP (Enterprise Resource Planning) system,

**Table 1** Questions to answer before business process modeling

	ARIS (EPC)	BPMN
Process flow	Yes	Yes
Decision points (connectors, logical operators)	Yes	Yes
Splitting/merging process flow	Yes	Yes
Sub-processes	Yes	Yes
Process flow including activities (functions) and events	Yes	Yes
Different types of events (start, mid, end)	No	Yes
Requirement of mid-events	No	No
Different sub-types of events (message, exception, timer, etc.)	No	Yes
Different sub-types of activities/functions (human, systems, etc.)	No	Yes
Participants and roles in the process	Yes (bound with Organizational Structure diagram)	Yes (as swim lanes)
Difference between process flow and message flow	No	Yes
Milestones	No	No
Exceptions' modeling	No	Yes
Definitions of inputs and outputs	No	No
Relationship between processes	Yes (Value-added diagram)	No
Relationship between participants (organizational units, roles, persons, etc.)	Yes (Organizational Structure diagram)	No
Data structure	Yes (ERM—Entity Relationship Model diagram)	No
Relationship with the environment	Yes (Organizational Structure diagram)	Yes (as pools)

- BPMN seems to be very useful for documentation of the organization, redesigning of the organization, using knowledge management, and supporting continuous process management,
- EPC diagrams are easier to understand by those in the organization [34],
- EPC and BPMN strongly support patterns to reduce the perceived model's complexity,
- transforming EPC diagram into the BPMN model is possible but with some limitations [22].



2.2 BPM for Service Composition

Service composition uses available services that meet the requirements of a specific user in order to deliver composite services. According to the SOA paradigm, services should fulfill business needs (in terms of functional and non-functional properties). If the business process model contains the necessary information, it may be the basis for service composition.

2.2.1 Requirements of Service Composition

The successful composition of composite services requires information about the business process, including:

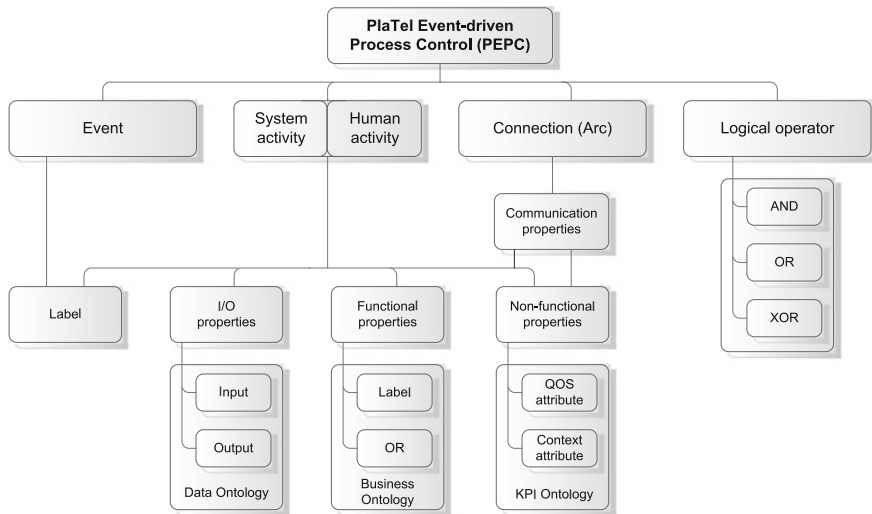
- sequence of activities to be performed (process flow),
- types of activities, including human-centered activities that need human-computer interaction,
- business logic (including the distribution of the process flow on different paths)
- functional and non-functional requirements,
- input and output data,
- the set of services that provide required functionality (by process, sub-process or activity)—this knowledge is available from the service repository.

Table 2 shows how the ARIS methodology (EPC diagrams) and BPMN meet the needs arising from the composition of composite services.

Analysis of the ARIS methodology and BPMN notation shows that their application for composition of composite services is limited because most of the service composition requirements are not met. Sometimes IT tools or extensions of notations (e.g. for performance measurement [39]) enables one to solve some problems (e.g. allows the indirect indication of functional and nonfunctional requirements

Table 2 Fulfillment of service composition requirements

	ARIS (EPC)	BPMN
Sequence of activities (functions)	Yes	Yes
Activity types	No	Yes (partly, as swim lanes)
Business logic	Yes	Yes
Definition of functional requirements	No	No
Definition of non-functional requirements	No	No
Binding with ontology	No	No
Input and output data	No	No
Relationship between activities (functions) and services	No	No



**Fig. 2** Meta-model of PlaTel event-driven process control (PEPC) notation

and input/output data), but to get closer to the service composition requirements, expensive commercial tools are needed.

To fulfill the requirements of business process modeling for composite service composition, the concept of dedicated notation PEPC (PlaTel Event-driven Process Control) was developed. PEPC is based on the EPC diagram but has several extensions that lead to the modeling of business processes as a convenient addition for automated service composition.

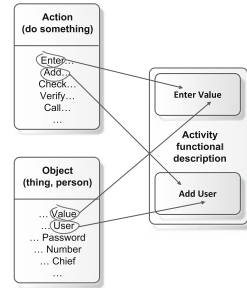
### 2.2.2 PEPC Notation

PEPC notation supports business processes modeling that provides models for the automatic composition and execution; these are used as composite services.

The key attributes of the proposed modeling PEPC concept are as follows: (Fig. 2):

- process flow modeling typical for EPC diagrams (activities are followed by events, logical operators—AND, OR, XOR—and are used similarly to the EPC notation),
- different graphic objects for system activities (without human-computer interaction) and human activities (with human-computer interaction),
- connections between activities, events, and logical operators,
- names of activities (labels) and key words based on business ontology—\$-the functional properties of activities,
- activities have defined inputs and outputs based on given data ontology,
- activities and connections have non-functional properties based on known KPI (Key Performance Indicator) ontology,

**Fig. 3** Using business ontology in PlaTel event-driven process control (PEPC)



- activities may be added optionally for information (WSDL file) with the definition of corresponding ICT service or services that provide functionality required in the activity.

PEPC notation uses the concept of Semantic Business Process Modeling (SBPM), a modern Semantic Web approach that adopts well-defined ontologies in the models of business processes [11].

PEPC includes data, business and KPI ontologies.

Data ontology stores information about a set of inputs and a set of outputs. Each input and output has a name and type that is necessary in order to link process activities with services. According to WSDL specifications, each service operation should be described by inputs and outputs so it is possible to find service or services and business process activities that correspond due to inputs and outputs automatically.

Business ontology includes set of terms that describe actions and objects. A combination of those terms is used to set the functional properties of the activity (Fig. 3). Functional properties are stored as activity labels or activity key words.

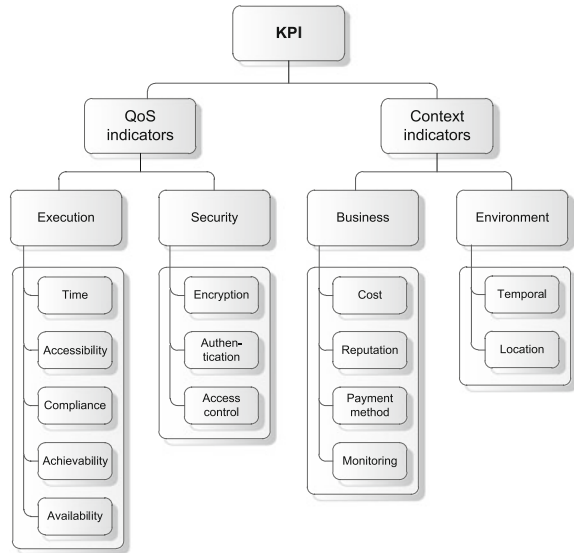
KPI ontology includes non-functional properties of the activities and connections in a hierarchical structure (Fig. 4).

Non-functional attributes of activities and connections allow for the mapping of SLA (Service Level Agreement) requirements for the business process or any of its components (sub-process, activity, connection).

PEPC optionally assigns an atomic or composite service or services to business process activities. It offers the possibility for one to indicate which services should be used in the composition of composite service (e.g. for reasons of safety or signed contracts with the provider of specific services) or precisely define the requirements for such services. Such services should be previously registered in the service repository to ensure the functional consistency of the composite services.

PEPC incorporates the functional description of the composite services with various QoS indicators thus defining a complete SLA and allowing for service composition and provisioning.

**Fig. 4** KPI ontology for PlaTel event-driven process control (PEPC)



### 3 Service Composition as a Service

On the basis of the SLA that results from the business process query, a service composition process begins. To generate a composite service means to find a set of atomic services and bind them together so that they, as a new service, fulfill all user functional and non-functional requirements. The services stored in repositories are functionally and non-functionally described with terms derived from the ontologies mentioned in the preceding section. A typically automated composition process requires a semantic query (description of a required composite service; often this description is referred to as a Service Level Agreement—SLA) and in PlaTel consists of two parts: service composition and service mapping. The first is responsible for the functional composition of the composite service, while the second adds the necessary communication services that connect the functionalities while preserving the QoS defined in the SLA.

Recently, developing information and communication technologies (ICT) have enabled entrepreneurs to develop monolithic structures into distributed ones. The entrepreneurs could focus on translating business processes into composite services; thus the Service Oriented Architecture (SOA) becomes a crucial paradigm in designing service-oriented systems (SOS) [65].

In SOS, the key element is a service that provides certain and well-defined functionalities and is characterized by parameters that describe the quality of a required and delivered service. Furthermore, services may be instantiated and assembled dynamically, leading to the changing structure, behavior and location of a software application in runtime. However, to ensure high satisfaction of clients and service

providers' profits, the services should be provided in order to guarantee a high quality of service (QoS) [60].

### **3.1 Service Description**

The PlaTel approach to the service description problem assumes the use of the native service description language, SSDL (Smart Service Description Language), proposed as a solution allowing a simple description of composite service execution schemes and the support for the functional and non-functional description of services—there exist similar solutions, like ROsWeL language, which utilizes a declarative, resource oriented approach [23]. Its functionality includes the Web Service Description Language (WSDL) but also offers important extensions. SSDL is dedicated to service execution support, including the service guarantees of QoS parameters and dynamic service composition at runtime.

#### **3.1.1 Languages**

WSDL applicability to service composition

Web Service Description Language (WSDL) is a format that aims to describe the functionality offered by Web services, mainly by describing their interfaces. It provides a machine-readable description of how to call Web Services, what parameters to forward, and what to expect from it in return. The WSDL file describes only a single Web Service and its functionality, whereas SSDL utilizes the WSDL as a base to describe composite services and requirements (or functionalities) of those services. Each SSDL file that describes a composite service consists of a set of nodes which, connected together in a graph-like structure, form a composite service execution plan determining a proper way to execute the services and to pass on the data between them. Each SSDL node contains a pointer to a WSDL file that describes how to access a particular Web Service in a composite service.

The main difference between WSDL and SSDL is not only the scope of description (atomic service vs. composite service) but also the fact that WSDL is a complete instruction on how to call a Web Service via HTTP (namely using SOAP protocol), while SSDL is an execution plan needing an engine to execute (interpret) it.

#### **SSDL**

When working with the service composition problem, it was noticed that there is a need for a language that allows for the definition of requirements and a service execution plan in a unified manner. Composition methods transform user requirements to the form of a composite service that fulfills those requirements. The literature has shown no description language designed to utilize this particular feature.

The most widely-known candidates considered were BPEL and OWL-S—both designed with different applications in mind. BPEL, a business standard adopted on many occasions, focused on execution of services and has no semantic description

of what the services actually do or should do. The latter is especially essential for the dynamic composition of semantically-described services. On the other hand, OWL-S, which offers a semantic description of web services, was designed for the description of atomic services; its offer for composite scenarios while preserving the non-functional QoS requirements is limited.

A decision was made to design a description language that would have the following design notions, have low complexity, allow for the definition of both functional and non-functional requirements of services for composition purposes, and at the same time the designer could combine the service requirements with specific services. In the process of development of this so-defined language, its other aspects became apparent. As a language defined for services requirement specifications, it had to allow for the distinction of various well-defined kinds of requirements and services. As a consequence, a specialized execution engine was designed whose primary purpose was to execute composite services described in such a manner. To fulfill the premise of being fully executable, the description language had to be fully interpretable. This interpretation is done in real-time by composition algorithms invoked by the engine.

The designed language was named SSDL after its elementary concept of a Smart Service. The idea of Smart Services extends the concept of a composite service. A Smart Service is, in fact, a type of a composite service; however, its elements do not necessarily have to be atomic services. In general, a Smart Service is an interpretable and simultaneous definition of atomic services and requirements of different types in one execution plan.

At the bottom of a Smart Service definition process stands the following usage case. Typically, users will want to define and then execute new composite services. In this case, with the use of the provided specialized graphic user interface (Sect. 6), they will input their requirements and appropriate composition mechanisms will be executed to produce an optimal composite service. All this can be saved in user profile and executed in well-defined situations when a composed service is requested.

However, sometimes a user will want his other service to be more flexible than just a standard composite service. Users can design their composite services to change with time or be composed in real-time. Saving parts of the Smart Service as requirements and not services will lead to dynamic interpretation at the point of execution later. The execution engine, inferring by the type of user requirements, will be able to perform different composition scenarios or discover and execute appropriate services for each requirement.

Technically, a Smart Service is represented by a combination of interconnected nodes (Smart Service Graph), some of which can represent concrete services: some sets of services and some various types of requirements. Nodes of a Smart Service are connected by identifying data sources—then they define data flow—and order of nodes defining control flow. Various types of nodes can be defined (ultimately in a specialized ontology) as long as the execution engine is configured to interpret and execute them. Note that it is the user who can define node types and their interpretation of what makes the SSDL language dynamic and highly susceptible to personalization.

A definition of SSDL node types contains all basic data types that allow for the functional and non-functional description of a service, its execution requirements, and the description of complex services with the conditional execution of their atomic components. The SSDL language was designed to enable the description of complex services which is directly interpreted and executed by the service engine as described in Sect. 4.1. SSDL allows a user to define alternatives from which dynamic composition can choose candidate services (in case the base one returns an exception) or introduce more levels of complexity of a service by introducing sub graphs. Furthermore, some control instructions could be added to further instruct the execution engine to interpret the Smart Service description.

### 3.1.2 Functional and Non-Functional Service Requirements in SSDL

A single SSDL node, which is used to describe a basic functionality requirement for a service has several important sections, which are extensively used during service selection, composition, and the final execution plan optimization:

- Physical description—used by every type of data that links to a specific Web Service, thus representing the optional description of service execution conditions (requirements leave this section empty),
- Functional description—used to semantically describe the capabilities of a service or required capabilities when defining a functionality-type node
- Non-functional description—used to describe the non-functional parameters of a service or requested services such as: time, cost, availability, etc.; non-functional parameters that can be requested for composition purposes are not limited in any way—external validation can be performed using user-defined ontology and rules, for example.

Each of them is associated with the number of sub-nodes allowing for precise description of a service. Below, we show an example of the information stored within the *nonFunctionalDescription* node:

```
<nonFunctionalDescription>
  <properties>
    <property>
      <name>Bitrate</name>
      <value>1.5</value>
      <relation>eq</relation>
      <unit>Mb/s</unit>
      <weight>1</weight>
    </property>
    <property>
      <name>Cost</name>
      <value>300</value>
      <relation>lt</relation>
```

```

        <unit>PLN</unit>
        <weight>1</weight>
    </property>
</properties>
</nonFunctionalDescription>

```

This allows the introduction of domain-specific parameters that could be defined according to the current needs with only one constraint—they should be also represented in the domain ontology, which assures the interoperability and semantic consistency of our framework.

## 3.2 *Service Composition Methods*

This section presents a general composition scenario and introduces the Service Composer as a specialized service that performs service composition based on the SLA agreement and contains both functional and non-functional requirements. The service composer is a configurable composite service itself, and therefore offers a flexible approach in which each of the stages of the composition process may be performed using different methods of service selection and optimization provided as services. This approach can be described as being utilized to ensure efficient service composition and provisioning while addressing the problem of QoS-aware communication between the components of composite services.

### 3.2.1 Overview

The composition of Web services permits the building of complex workflows and applications on the top of the SOA model [7]. There are two standard approaches to service composition, namely, workflow composition and Artificial Intelligence planning (AI planning). Besides the obvious software and message compatibility issues, good service composition should be carried out with respect to the Quality of Service (QoS) requirements; these, in turn, can be split into functional and non-functional [26] components. The first are considered to be of key importance; however, preserving the non-functional requirements (availability, performance, and security—to name only the most important) is a key factor as well as its importance rapidly grows in distributed environments where complex services are composed of atomic components [25, 33, 40].

### 3.2.2 SSDL-Based Service Composition

There are two standard approaches to service composition, namely, workflow composition and AI planning. In both approaches, the emphasis is placed on identifying



control and data flow. Generally speaking, functionalities of a composite service are represented as a Directed Acyclic Graph (DAG), also referred to as a functionalities graph, in which arcs denote interactions among functionalities and nodes determine functionalities (note that a service may be executed periodically or in a feedback loop—this mode is provided by the execution engine described in Sect. 4). Thus, the service composition task can be seen as a determination of a DAG structure, e.g., using AI methods and then a selection of atomic or composite services that provide required functionalities. The service selection is formulated as an optimization task in which an objective function reflects QoS attributes aggregation, e.g. response time or price, and constraints concerning structural dependencies in DAG. There are several known approaches to QoS optimization such as graph-based methods and mathematical programming (integer programming, stochastic programming). Recent trends also include semantic modeling and optimization via meta-heuristics, e.g. genetic algorithms.

The result of service composition is a composite service execution graph, which is, similarly to functionalities graph, a DAG, but its nodes are constituted not by functionalities but by services stored in the repositories. In fact, the composite service execution graph should have the same structure as the functionalities graph, and its nodes should contain services with the functionality described in the corresponding nodes of the functionality graph.

### Composite Services Structure Composition

In general, service composition process consists of two steps. First, the required functionalities and their interactions, i.e., control and data flow, are identified. Second, for the functionalities graph set, the appropriate candidate services are selected, and as a result a composite service execution plan is established. Candidate services are services that share a specific functionality defined in the functionality graph; in other words, they fulfill the functional requirement but are different with respect to the non-functional properties. Only one of the candidate services for each functionality within the functionality graph will be selected to replace this functionality in the composite service execution plan. The service selection is accomplished based on non-functional properties which, generally speaking, can be referred to as Quality of Service (QoS) attributes: service execution time, service execution cost, service availability, service execution success rate, service reputation, or service execution frequency.

Depending on the structure of the functionality graph, the non-functional parameters of the composite service will be calculated differently with respect to the functional parameter type: i.e., the cost of a composite service is a simple sum of all services that build it, whereas service execution time is a sum of the execution times whenever services are executed one after another, and a maximum of those composite service parts where services were executed in parallel. For each of the non-functional service parameters, the user places a constraint, determining the sufficient

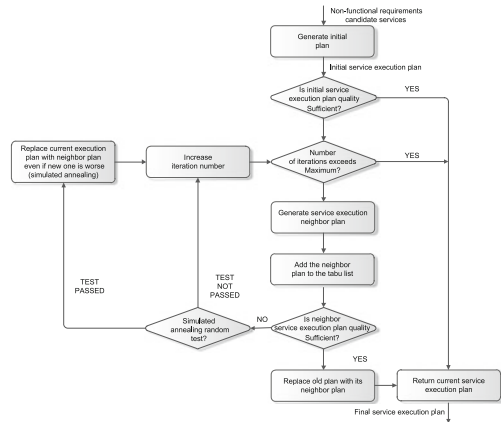
conditions for the service selection. This defines the goal to select such services (from sets of candidate services providing all of the requested functionalities) that their aggregated non-functional parameters—constituting the non-functional composite service parameters—meet the user-defined constraints.

Service selection consists of several stages:

- Inverted service indexing—inverted indexing of services, using a domain ontology describing the input and output service parameters annotated with service meta-data, allowed for faster selection of valid services, independent of the semantic filter being used (although semantic filters had to be redesigned to fit the approach).
- Semantic filtering—a method for service and requirement comparison that goes through a service requirement consisting of a set of required input and output parameters. Depending on the type of semantic filter and distance measure used, the method finds concepts that are identical to the requirements—or semantically close. Note that the definition of semantic closeness depends on the measure being used.
- Preliminary candidate service sorting—at this stage, a set of candidate services that fulfill the functional requirements (selected by semantic filters) can be sorted using various semantic filters. More strict or computation-heavy semantic filters could be applied to the candidate services sets to establish how well the functionalities were fulfilled. Some non-functional requirements could be checked locally at this stage as well.
- Composite service execution plan generation—this stage is defined as an optimization task from which only one service for each functionality is selected from sets of candidate services corresponding to nodes in the functionality graph. This occurs together with other selected services making up the composite service execution plan. Additionally, the inputs and outputs of the services included in the plan must be pair-wise compatible. The algorithm for this stage is the key component to service selection and is described in more detail below.

The service execution plan generation algorithm (Fig. 5) consists of several steps. First, based on a greedy approach, an initial plan is generated. If the initial plan does not meet the sufficient conditions then its neighbor plan is generated, then the neighbor plan is verified against sufficient conditions. If the neighbor plan is also not a valid solution, then it is put onto a taboo list and the next neighbor plan is generated. This is repeated until a valid solution is generated or the algorithm reaches the limit of iterations, which is a standard approach applied for service composition [61]. The taboo list ensures that the same composite service execution plan will not be generated repeatedly. Also, at certain times a simulated annealing mechanism accepts a neighbor plan that does not meet the requirements so that new solution spaces could be explored. This is more likely to happen due to the more iterations performed by the algorithm. The algorithm presented above is a general heuristic hybrid procedure based on simulated annealing and tabu search. It triggers more specialized methods such as the generation of an initial or neighbor service execution plan. Those methods are described below in more detail.

**Fig. 5** Algorithm for service execution plan generation



### 1. Method for generating composite service initial execution plan

The method is a greedy approach in that each functionality requested by the user selects one service from a set of candidates with the same functionality. Within each set of candidate services, a service with the highest fitness score is selected. This method does not take into account the structure of the functionalities graph but locally selects the best candidate service with a requested functionality.

### 2. Method for generating local fitness function

The method establishes a local fitness function score according to the weighted QoS parameters of each service. The fitness function gives an aggregated, weighted value to all QoS parameters of a service, which is normalized relative to other QoS values of all candidate services with the functionality.

### 3. Method for calculating the composite service execution plan quality

The method calculates the quality of the composite service based on its aggregated QoS parameters compared to the user's non-functional requirements. The QoS parameters are calculated based on the composite service execution plan in the aggregation method below.

### 4. QoS parameters aggregation method for composite service quality estimation

The method establishes a vector of QoS parameters for any composite service provided with a composite service execution plan and with QoS parameters for each of atomic services in that plan. The method analyzes the functionalities graph structure: the sequence and parallel structures within the service execution plan and generate a vector of formulas for calculating the QoS parameters for a given functionalities graph. For any composite service execution plan with the structure of the functionalities graph, the generated formulas can be used to quickly estimate its QoS parameters. This is true because the generated formula is based on the functionalities graph and not any particular composite service execution plan.

## 5. Neighbor plan generation method

This method is based on the simulated annealing algorithm and, provided with an initial plan and a set of candidate services, finds a neighbor plan that is closer to satisfying the user's requirements. First, all QoS parameters from the previous composite service execution plan (either the initial or previous neighbor plan) that did not satisfy the user requirement are sorted in descending order according to the normalized difference between the QoS parameter and the appropriate requirement. The selected QoS parameters will be the basis for further modification of the service execution plan. For each of the dissatisfied QoS requirements, the atomic services in the service execution plan are sorted in either descending (if those QoS parameters are service execution time or service execution cost) or ascending (for other QoS parameters such as availability) order. Replace the first two atomic services from that sorted list with new services from the appropriate service candidate sets (services with the same functionality) in such a manner that a newly-selected service is not on the taboo list for that functionality; it is a service with the highest fitness function score among the other service candidates for that functionality. Then, add the selected services to the taboo lists for the appropriate functionalities. The composite service execution plan resulting in those service replacements is a neighbor plan to the previous one with respect to the minor changes made towards replacing the services with the most dissatisfying QoS parameters.

As the domain knowledge is represented in the form of ontology and the services are semantically described (in terms of associating their inputs, outputs, and functionalities with the concepts—or sets of concepts from the ontology), the composition is a process of transforming user requirements defined in the SLA into a fully-defined composite service that fulfills them. In the first stage of the composition process, user requirements are analyzed and assembled into a single structure that represents a final composite service structure by a graph where nodes contain user requirements and edges connect those requirements, determining the order in which a final composite service will be executed. Then, with the use of knowledge engineering, this structure is enhanced so it defines an execution scenario for particular atomic services in a composite service. In this stage, no requirement can be left disconnected from the other requirements. However, the services may differ in non-functional properties (as in execution time or cost), so in the last stage for each functional requirement, a single atomic service is selected so that all services that build a composite service jointly fulfill the non-functional requirements.

The key feature of the PlaTel's service composer is that it is a composite service itself; each of the components of the composition scheme (SSDL GUI, service matching, and service selection) are performed by the services. PlaTel's composition service is described in SSDL and may be freely reconfigured if there are more or better-suited services that could be used for a specific problem. This approach opens vast possibilities—each of the stages could be performed using different strategies. We may use different semantic discovery methods when searching for services that fulfill each of the requirements (or propose different distance measures for concepts in the ontology), or various optimization techniques could be used to produce the

composite service, fulfilling the non-functional requirements, not to mention that a variety of non-functional parameters could be requested and optimized by this.

Equipped with the above mechanisms, our framework allows composition service designers to incorporate various approaches, test them, and deploy—in a form of service—enabled composition tools well-suited to different domains and problems. The tool consists of two main parts: one is the front end of the application which allows a business client to define his/her domain by connecting to external service and knowledge repositories (here: ontologies) that will be used to construct composite services; the second part is the layer of specialized services which is extensible and can provide a variety of services but mainly composition services.

The PlaTel service composer execution plan may be passed directly to the Execution Engine, but in this work we want to refine our approach even further and introduce the communication (ICT) services, incorporating them to our framework. This is a unique feature of the PlaTel which will be described in detail in the following section.

### Composite Services Provisioning

Recently, enterprises tend to collaborate and integrate their business cores in Web markets in order to maximize both clients' satisfaction and their own profits [45]. Therefore, there is an increasing need to develop methods for combining existing services together to enrich functionalities and decrease execution costs. Hence, the old-fashioned way of developing composite services, i.e., human-based service designing, becomes very insufficient due to the enormous number of available services and a lack of computational resources. That is why the automatic or semi-automatic service composition method is a crucial element of any system that implements the service-oriented architecture (SOA) [45]. In general, service composition process consists of two steps [57]. First, the required functionalities and their interactions, i.e., control and data flow, are identified. Second, the execution plan is established, i.e., for sets of functionalities, an appropriate service version is selected. The service selection is accomplished basing on non-functional properties which, in general, can be referred to as Quality-of-Service (QoS) attributes.

In most of the proposed approaches for service composition, the domain services are considered apart from information and communication services (ICT services), which are understood as physical means of supporting the execution of domain services [9, 60, 63]. However, in order to allow Service-Oriented Architecture [45] to be applied properly, most of requirements need to be mapped to ICT services. Otherwise, it is ambiguous how the domain services should be executed physically. For example, a requirement for a building monitoring service consists of the following operations: signal acquisition, coding, sending, and decoding. All of these services are, in fact, the ICT services. Thus, the ICT service mapping is a process of mapping requirements to a combination of atomic ICT services which facilitates the delivery of requested domain-specific functionalities. In both approaches, the emphasis is put on identifying control and data flow. Generally speaking, functionalities of a composite service are represented as a directed acyclic graph (DAG) [19, 65, 67] in which arcs

denote interactions among functionalities and nodes determine functionalities. As a result, the service composition task can be seen as a determination of a DAG structure, e.g., using AI methods [48] and then as a selection of atomic or composite services that provide the required functionalities. The service selection is formulated as an optimization task in which an objective function reflects QoS attributes aggregation, e.g., response time, price, and constraints concerning structural dependencies in DAG [57].

There are several known approaches to QoS optimization such as graph-based methods [18, 19, 65, 58] and mathematical programming (integer programming [67], stochastic programming. Recent trends also include semantic modeling and optimization via meta-heuristics, e.g., genetic algorithms. However, the problem of ICT service mapping is usually omitted, and lately only a few papers have pointed out the necessity of considering ICT services in the process of service composition [17, 64]. Hitherto, due to the authors' knowledge, the problem of ICT service mapping has not been fully stated, and hence, no solution has been proposed.

The service composition in the telecommunications domain requires one to aggregate, update, maintain, and process knowledge about telecommunication services in order to analyze, plan, and manage—in an optimal way—telecommunication resources. Moreover, the frequent behavioral patterns in business processes must be discovered for the purpose of optimization of telecommunication service delivery scenarios. The workflow of the service composition and mapping process is presented in Fig. 6.

To compose the telecommunication services, we use three main modules in our framework:

- *Domain service composition*—based on the request a domain service is composed;
- *Service mapping*—the domain scenario is mapped with ICT services
- *Service composition*—in a telecommunication scenario, ICT services are composed and a final execution plan is returned.

and repositories:

- domain service ontology;

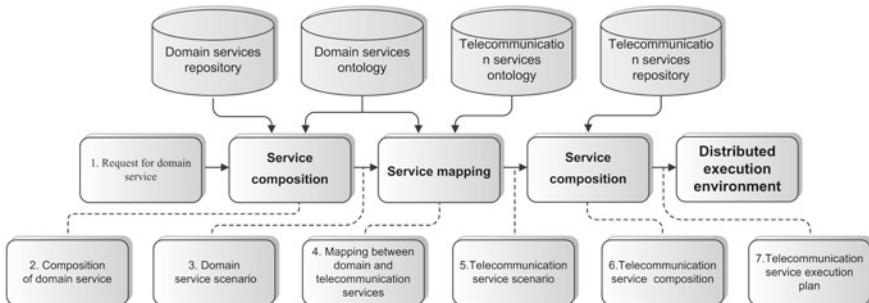


Fig. 6 Workflow of the services composition and mapping process

- ICT service ontology;
- domain service repository;
- ICT service repository.

Ontologies are representations for a seamless cooperation and knowledge flow between business processes and formal service descriptions in service-oriented systems. They play a crucial role in service mapping in translating domain service scenarios to ICT service scenarios.

Furthermore, two types of services could be distinguished, namely, computational and communication services. Ontologies, which are implemented in Protégé, are based on SIMS Ontology developed within a project of the Sixth Framework Programme [64]. SIMS has been extended for wireline telecommunication systems, QoS concepts, and parameters.

Generally speaking, computational services are used to process data streams, while communication services facilitate data transmission and end-to-end QoS guarantees. In the proposed system, there are multiple atomic or composite computational services, e.g.:

- signal merging and splitting;
- data compression;
- signal encoding/decoding;
- video/audio stream encoding;
- change detection in data stream.

The task is to compose composite ICT services which can deliver functionalities given in a business process describing all the required use-case scenarios for a particular business domain. The resulting composite ICT services are composed based on the available atomic ICT services (computational and communication) provided by ICT service providers and/or operators. Additionally, the above composite services must conform to the users' non-functional requirements concerning—among others—quality, security, and availability of the assumed business process.

The process of the composite ICT service composition consists of four stages: transformation of the business process into a set of composite services, composite domain service composition, domain and ICT service mapping, and composite ICT service optimization (see Fig. 6).

In the first stage, the business process provided by the user is transformed into a set of composite services which fully cover all use-case scenarios defined by the business process [59]. Each composite service is defined by the functionalities (atomic domain services) that need to be performed in order to fulfill the corresponding use-case scenario. Moreover, a composite service description includes non-functional requirements (e.g.: QoS, security, etc.) concerning the entire composite service and/or some of its functionalities. Each composite service is defined by a service-level agreement (SLA) including—among others—functionalities and values of non-functional parameters required to be fulfilled in the corresponding use-case scenario.

Next, each composite service defined by functional and non-functional requirements is composed of the available atomic domain services. The result of this stage



is a set of atomic domain services and the order in which they must be executed in order to provide the user with the requested functionality and conforming to the non functional requirements. Note that in order to deliver the requested composite functionality, some of the atomic functionalities may have to be delivered in parallel and/or in sequence. Therefore, the order in which the atomic domain services are generally performed is defined as a directed graph, the nodes of which represent atomic domain services, and the edges represent the precedence relationship between them. Such a graph, called a composite service scenario, is passed to the next stage of the service composition.

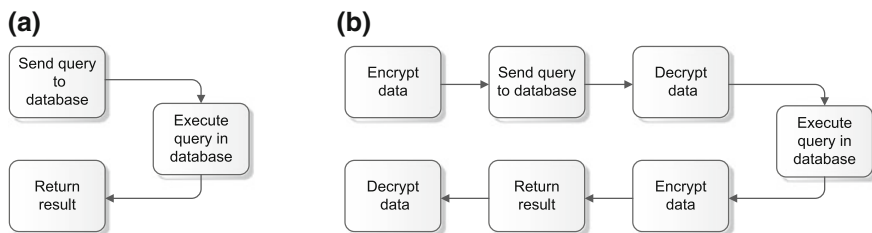
Each node of the composite service scenario graph represents a single atomic domain service that provides a well-defined functionality. In order for this functionality to be delivered, certain ICT (i.e. communication and/or computational) services must be executed. The task of the third (services mapping) stage of service composition is to substitute each of the nodes (atomic domain services) of the composite service scenario graph with a sub-graph representing the execution order of the atomic ICT services necessary to deliver the requested atomic domain functionality.

As an example, let us consider the atomic domain functionality consisting of sending a billing information record for a certain time period being the response to a query to a CRM application. This atomic domain functionality is delivered by execution of three atomic ICT services (see Fig. 7a), i.e.: transmitting a query to the database (communication service), execution of the query in a database (computational service), and returning the results to the user (communication service). Moreover, if a user requires a secure data transfer, two additional computational services (cipher and decipher) for each communication service should be executed (see Fig. 7b).

Note, that in general, two communication services, taking part in the delivery of the above exemplary atomic domain service, are functionally and non-functionally different. The first one consists in sending small message (e.g. SQL query) while the other may require the transmission of a large volume of data.

The result of the service mapping stage is a composite ICT service scenario graph where the nodes represent atomic ICT services and the edges define the precedence relationship between the services.

There are multiple approaches to accomplishing the mapping stage. First of all, the mapping for each atomic business service may be predefined, which is a simple



**Fig. 7** Exemplary mapping of a billing information acquisition atomic business service: **a** unsecure and **b** secure data transmission



and efficient—but not flexible—solution. On the other hand, the mapping task can be treated as a special case of a composition task. In this case, some known methods (e.g. semantic composition or GraphFold algorithm [50]) can be used. Another approach is to use pre-defined mapping rules and a reasoning engine to find the best possible mapping conforming to the non-functional requirements. The last composition stage is an optimization stage. The task performed at this point consists in finding such a version of the atomic ICT service available within the system: that non-functional requirements for the entire composite service are met. Additionally, certain composite service optimization tasks may be performed at this stage. These tasks may include, among others, finding the least expensive composite ICT service that satisfies the non-functional requirements or finding such a composition for which the usage of the ICT resources is balanced. The first optimization task may be stated by the user in the composite service request, while the latter is rather the concern of the ICT resource operator or service provider. The result of this stage, which is passed to the execution environment, is a composite service execution plan defining exactly which versions of ICT services will be used to fulfill each particular composite service request.

The first stage of service composition has to be accomplished during the ICT infrastructure design phase for each business process. The following stages of the services composition and delivery may be performed once along with the first stage during the ICT system development (off-line composition), or they can be performed on-demand each time when a certain use-case scenario is launched (on-line composition).

In the first approach (i.e. off-line composition), all composite services are predefined in the design phase and only these services can be executed during the lifetime of the system. This approach is very conservative; it does not make it possible to change or add new composite services and is very inefficient in terms of resource consumption, service reusability, and composition flexibility. It allows, however, for a very quick response to new incoming composite services requests, since the time-consuming process of composite service composition is performed only once.

In the second approach (i.e. online composition), on the other hand, each new incoming service request is handled separately, namely the best composite service conforming with the non-functional requirements is composed, taking into account the current state of the system (e.g. usage of communication and computational resources). This approach fully draws from the service-oriented architecture paradigm and allows for efficient resource usage, service reusability, and on demand composition of new services, scalability and flexibility. Additionally, it is possible to store pre-defined composite service execution plans or execution plans resulting from the compositions performed during the system's lifetime and apply them to the incoming composite service request when the response time is critical.

The composition process of an exemplary composite ICT service is presented in Fig. 8. The considered example concerns handling a physical alert in a business process of laboratory monitoring.

The business process of monitoring a laboratory consists of four use cases (see Fig. 8a): two concerning the monitoring of environmental security and two concerning the monitoring of physical security. Each use case is handled by the execution of

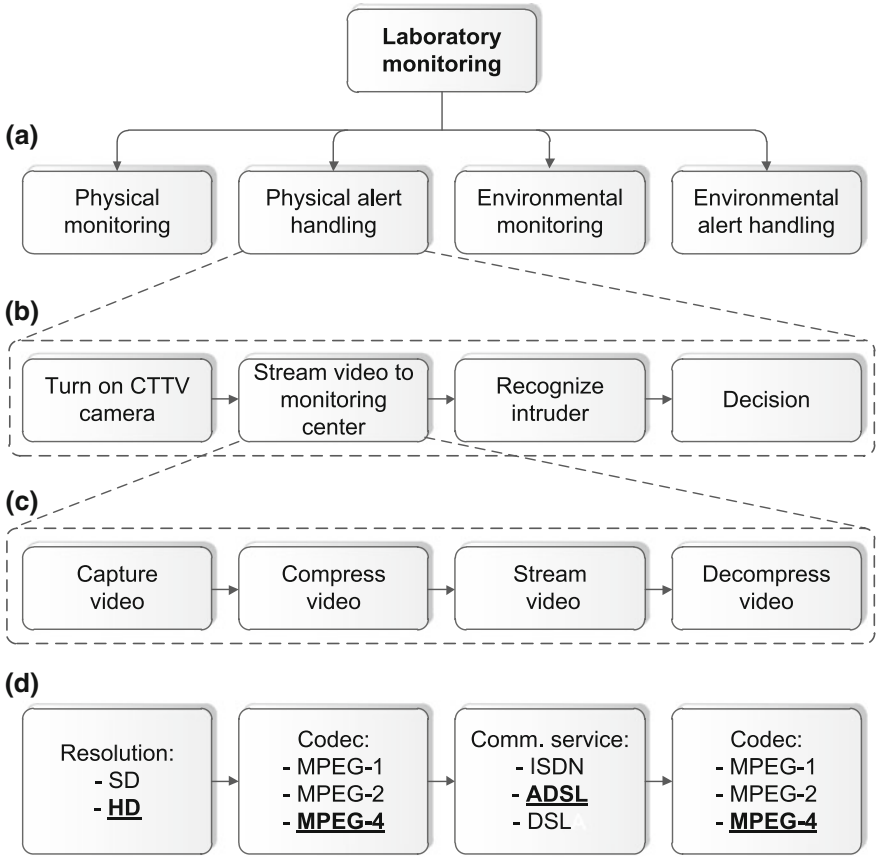


Fig. 8 Exemplary composition of telecommunication service

a separate composite domain service that has been predefined in a business process. In Fig. 8b, an exemplary composite domain service for a use-case scenario concerning the handling of a physical alert is presented. The definition of such a service is a result of the first stage of composite service composition, consisting of translating the functional and non-functional requirements of a business sub-process for physical alert handling into a sequence of atomic domain tasks which have to be performed to meet these requirements.

In the next stage (see Fig. 8c), each atomic domain service is mapped to a composite ICT service which guarantees the fulfillment of all the required functionalities. In this case, the atomic domain service stream video to monitoring center consists of a sequence of four atomic ICT services: video capturing, video compression, transmission of the compressed video stream, and video decompression.

In the last (optimization) stage (see Fig. 8d), particular versions of atomic ICT services are chosen. The choice is made based on given non-functional require-

ments and service availability. In this example, the streamed video is passed to the face recognition service. Therefore, a high definition (HD) video quality is chosen. Moreover, the MPEG-4 codec with the highest compression rate was chosen for video encoding. Since HD video encoded with MPEG-4 results in 16Mbps one-way data stream, Asymmetric DSL (ADSL) with 16Mbps upstream guarantees was chosen for a communication service. Finally, for the video decompression, the same codec as for compression must be used.

### Composition of ICT Services

In this section, we indicate the problem of ICT service mapping in the process of service composition and propose the solution. The presented solution applies the idea of decision tables [31] as an ICT service mapping tool. We utilize the decision table to associate ICT services with the corresponding requirements. The decision table for ICT service mapping consists of two columns. In the first column, a requirement is given and in the second one, a DAG of ICT services. In the service mapping step, the service selection through QoS optimization is performed. Our approach allows one to perform the execution plans on physical machines seamlessly because ICT services describe the computational and communication aspects of SOC. Furthermore, it is worth mentioning that the presented approach is domain independent and can be applied in any business area that requires utilization of ICT resources.

In general, the task of service mapping can be stated as follows. Find functionalities that are matched with ICT services and select such ICT services that perform matched functionalities in the best way, according to QoS attributes.

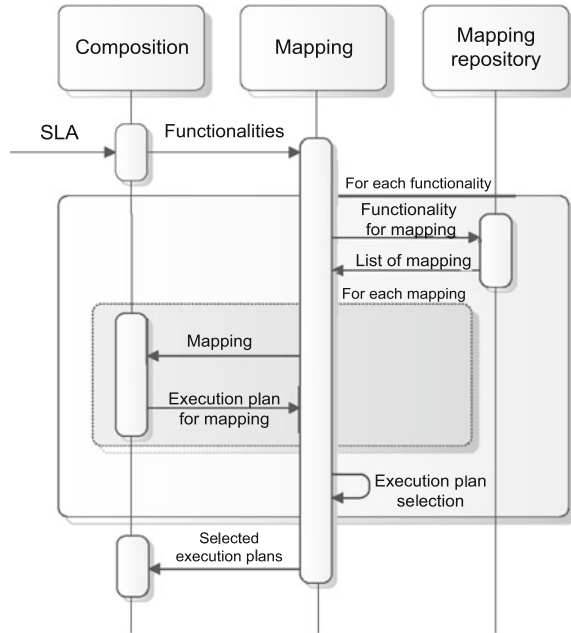
Thus, the mapping problem can be divided into three sub-issues. First, the mapping functionalities must be found. Usually, not all functionalities need to be mapped to ICT services. Second, the found functionalities must be matched with available ICT services. Third, one complex ICT service is selected.

Hence, the ICT service mapping can be seen as a service composition but performed only for the functionalities that are identified to be mapped. However, the crucial step is how to determine which functionalities are supposed to be mapped.

In order to solve this issue, the following two assumptions are made. First, functionalities and services are described in the same language in order to be identified by the same keys. For example, functionalities, as well as ICT services, are described by XML-based language and by the same tags that are used for their content comparison. Second, the mapping between functionality and a complex ICT service is given by an expert. For such assumptions we propose to apply decision tables for ICT service mapping.

It is worth noting that the problem of ICT service mapping is strongly connected with the service composition process. Namely, functionalities for matching are checked whether to be mapped to ICT services, or to be matched directly with services in a service repository. Therefore, the augmented service composition process can be represented by a sequence diagram presented in Fig. 9. The ICT service mapping consists of two steps. In the first step, all functionalities which are supposed

**Fig. 9** Sequence diagram of interactions between ICT service mapping and service composition. First, for all functionalities, only those are found which are supposed to be mapped. Next, for each mapping an execution plan is prepared. At the end, for each of the functionalities, an execution plan is chosen



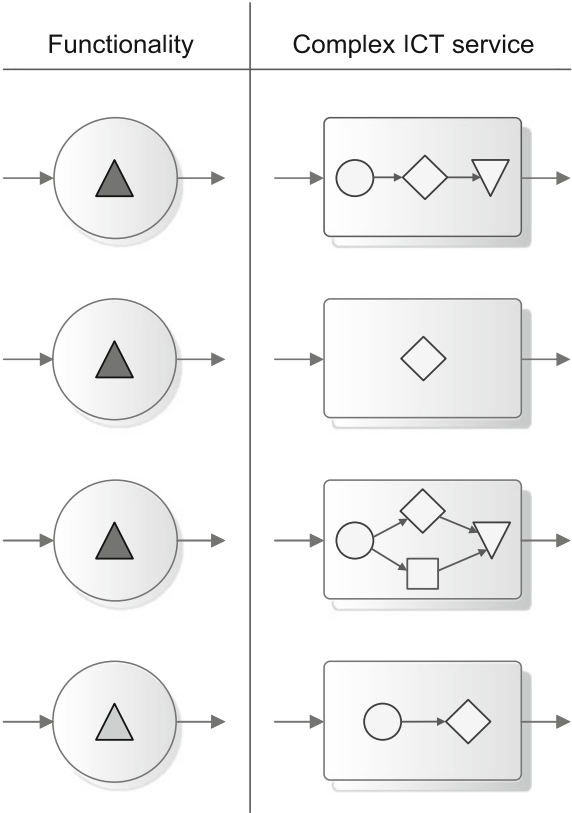
to be mapped are found (the light grey rectangle in Fig. 9). Next, for each of the functionalities identified to be mapped, a list of mappings is retrieved from a repository of mappings. For each mapping in the list, an execution plan is proposed by a service composition method according to given quality criterion (the dark rectangle in Fig. 9). At the end of the entire mapping process, one execution plan is selected according to QoS attributes.

There are two critical steps in the above process for ICT services mapping. The first step is how the repository is constructed; secondly, how the final execution plan is selected. Here, we focus only on the first problem as the second one can be solved using one of the methods known in the literature [48].

We take advantage of the earlier-mentioned assumptions, i.e., the mappings are given a priori and functionalities and mappings can be matched using universal keys. That is why, in this work, we decided to apply decision tables to represent the repository of mappings.

The decision table consists of two columns, i.e., the first column defines the condition and the second one—the decision. In the considered application, the condition is the functionality description, e.g., XML-based description, and the decision is the mapping. It is worth stressing that usually there are several possible mappings for one function. An exemplary decision table is presented in Fig. 10.

**Fig. 10** An exemplary repository as a decision table. *Dark and light gray triangles* in circles represent functionalities. *Circles, diamonds, and triangles* denote atomic ICT services and rectangles—complex ICT services. Notice that one function (*dark gray triangle*) corresponds to several complex ICT services



4 Composite Services Execution

Distribution of the request inside the Service Oriented Architecture yields a number of issues which involve virtualization management related to the management of virtual machines that represent the services offered. Automation of such a process requires well-defined handling requirements, policies, and a description of service features to be properly matched with the request and to be integrated with the service execution engine functionalities [PSS25]. As it was pointed out in this work, PlaTel is making such a capability real. Increasing the level of service awareness already at the lower (hardware and virtual resources) level can make the system more flexible and more adaptable to the changes in customers’ behavior.

## **4.1 Execution Engines**

### **4.1.1 Execution Engines Overview**

The execution of a composite service is performed by a Web Service Execution Engine (WSEE), which is responsible for invoking all its component atomic services and maintaining the QoS of their execution (if required). It needs to pass the input data to the services, acquire and process their output, and provide the procedures for any exceptions such as service execution failure, service blocking, or the lack of resources.

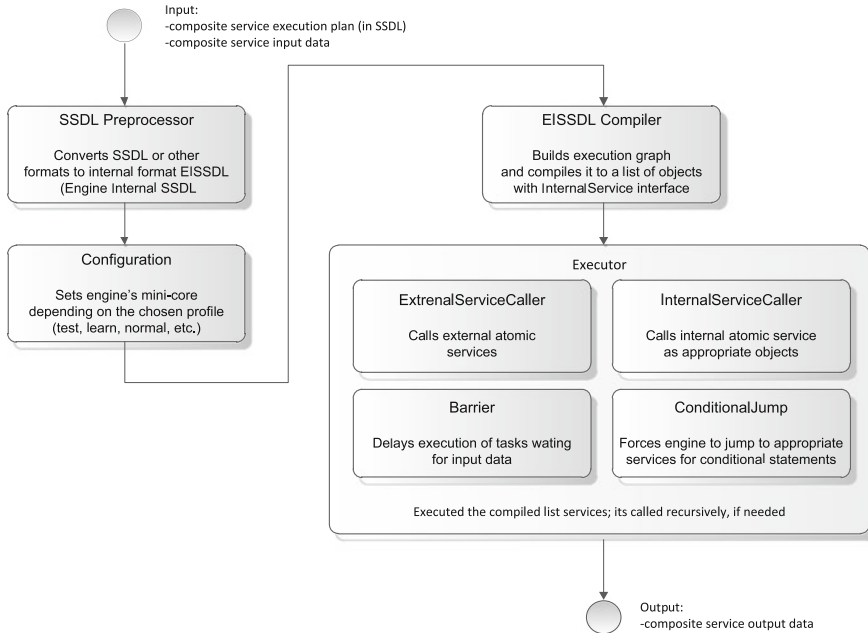
Execution engines which support the process-driven composite service execution were described in [61], while the engine support for execution of BPEL-defined processes was proposed in [20]. These solutions, however, assumed a fixed architecture of the service framework, which was based on certain devices and did not allow for the reconfiguration of the execution engine. An approach similar to that of the PlaTel framework (transition from business process definition to the composite service execution) was presented in [23], but it was based on the REST approach and was not an open solution that allowed for the use of multiple service repositories, the solution proposed in [23] is also comparable.

The PlaTel execution engine that supports the execution of any SSDL-described composite service and is provided as-a-service will be presented in the next section.

### **4.1.2 SSDL Execution Engine**

After the composition and mapping of the composite service are completed, the service is executed by the SSDL execution engine. It assumes an ‘engine-as-a-service approach’ and offers an execution engine as a configurable composite service, which acts as a SSDL language interpreter and supports dynamic interpretation of service description files, service configuration, and execution control. The SSDL execution engine is implemented as a lightweight virtual machine that may be duplicated and migrated upon the decision taken at the SOA infrastructure level. The core feature distinguishing the SSDL execution engine from other execution engines is its focus on composition mechanism and, together with the expressive nature of SSDL language it interprets, its ability to configure its own behavior.

The main role of the execution engine is to govern the process of services execution (and dynamic interpretation of requirements leading to service execution); however the configuration of the engine is in fact also something to distinguish it among other engines. We designed its core to be composed of several elements, quite like a composite service consists of atomic services. One of the elements is service interpretation and execution but we can add other elements or, as we call them, phases that introduce instructions like pre-processing, validation, full composition, and other actions such as a classification-based transformation of requirements to the composite service based on previous compositions. Actions focused on the



**Fig. 11** The Execution engine scheme and basic functionalities

SSDL performed in the preprocessing phase could be internal methods or, in the configuration-driven approach, composite web services. Such a situation is depicted in Fig. 11, where we distinguish a two-phase engine with its configuration managed in an external web application. Based on this fundamental application model, other phases could be added to further personalize the behavior and expand the engine's functionalities. Such phases could be final reporting (to specific applications via web services or a general report hub) or, as shown in Fig. 11, an even more complex approach where a pre-processor to the internal format is added and various work profiles are supported. This could ultimately lead to further flexibility of the engine, supporting various composite service definitions as modules and various behaviors, depending on identified situations (like debug, testing, return from halt, learning, normal, speed-low, etc.).

To better exploit the benefits of cloud computing, a presented execution engine was designed to work as a service. It was implemented in Java and supports multi-threading to process multiple service execution plans at the same time. It is capable of executing composite services defined in SSDL, but it can also automatically generate web interfaces to composite services stored inside the engine, broadcasting as those services. In this mode, the Execution Engine can emulate any composite service defined in SSDL. As a result of multi-threading, a single execution engine could act as multiple services or, in extreme cases, multiple engines could act as atomic services—hiding composite services behind a layer of abstraction. As a service, the proposed execution engine could be replaced by a different execution engine,

e.g. more specialized for a specific problem, such as a streaming services execution engine. For optimization purposes, not only could the engine be maintained in various localizations but also it could delegate parts of a composite service to other instances of the execution engine when some services in a series are closer to each other than others. Detection of clustered services and automated engine instance deployment, combined with service migration, is a good example of how automated composition could benefit SOA.

## ***4.2 Execution Environment***

Delivery of software services (computational, information, etc.) in the traditional form is characterized by the fact that applications are not open enough to follow the rapidly changing needs of business, had to be replaced with something more flexible. The idea to compose the processes from services publicly or privately available, mix and match them as needed, easily connect to business partners, seems like the best way to solve it. This involves functional requirements as well as assuring the changing needs of quality of processing. The proposed and discussed composite service execution environment is an attempt to solve the problem of traditional software's lack of flexibility. The environment takes into account the entire range of SOA advantages.

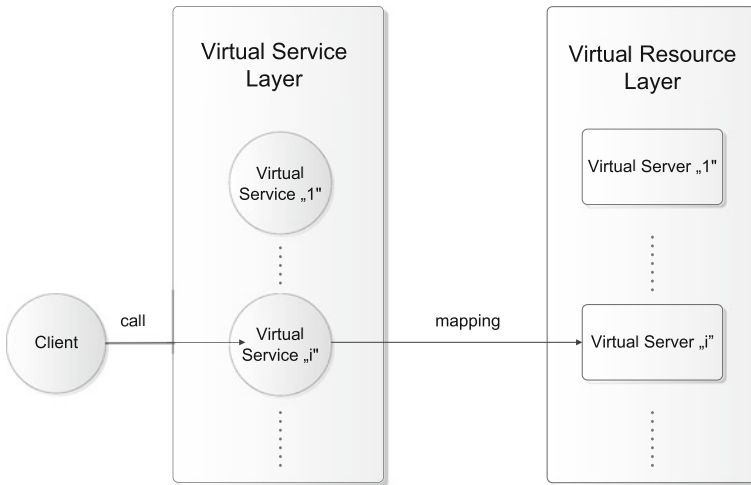
### **4.2.1 Infrastructure as a Service**

Flexible and efficient service delivery is a very important and difficult challenge in SOA-based systems. The quality of service delivery is also a very complex problem in such systems.

The new concept of effective and quality-aware infrastructures built in accordance with the SOA paradigm relies on the idea of a virtual service delivery system. The service delivery system consists of two layers (Fig. 12). The client of System calls a service visible to him/her at the Virtual Service Layer (VSL). The VSL virtualizes real services available at given service execution systems that are hidden from the client's point of view. Moreover, service execution systems can also be virtualized by service providers in a real execution system. As a result, the client deals with a virtual service that is mapped to a virtual server. Additionally, a virtual service can be instantiated in multiple virtual servers.

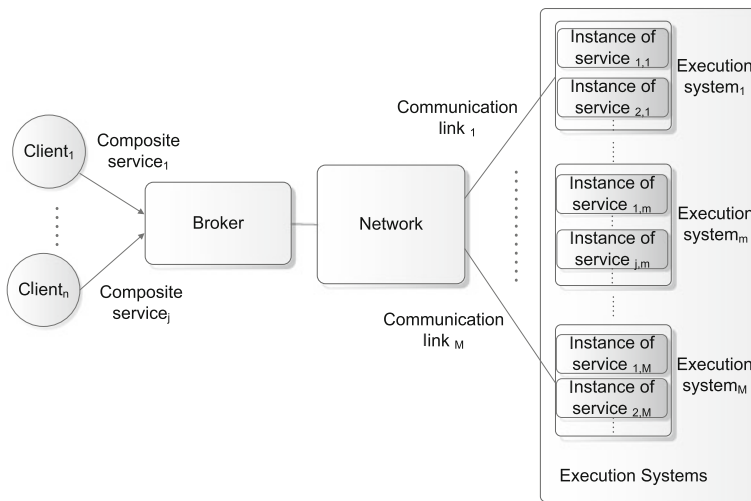
The composite service is put together with basic atomic services, i.e. the ones that cannot be partitioned afterward. The atomic services can be localized in different execution systems and executed by service instances running in these systems. The set of execution systems, atomic services, composite services, and service instances are constant for some time, but generally over longer periods, can change. The implementation of the Virtual Service Layer and Virtual Resource Layer must support the dynamic updates of these entities.



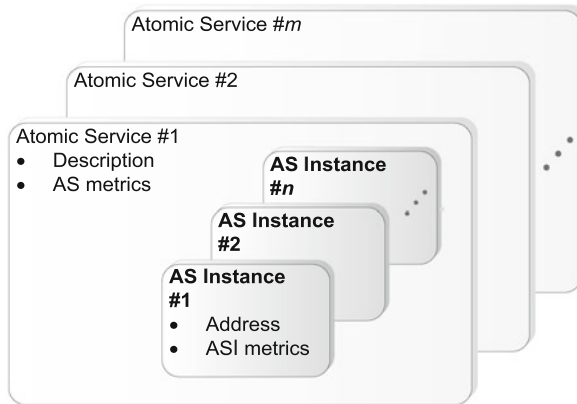


**Fig. 12** Two-layer service delivery system

Both layers form the Service Delivery and Request Distribution System (SDRDS). The Virtual Service Layer is implemented by a network service broker (hereinafter—‘Broker’) with an integrated Virtual Service and Resource Manager (VSRM) module that supports the virtualization of network services. Broker B maintains information about resources and handles atomic services client requests (Fig. 13). It hides request processing in the delivery system.



**Fig. 13** General infrastructure of Service Delivery and Request Distribution System



**Fig. 14** Service repositories

From the Broker's point of view, every component requesting services is a client—both the ordinary client application and any system component (e.g. service composer). The Broker acts as service delivery component, while in fact distributes requests for services to known service processing resources. Distribution is performed on the basis of formulated service quality criteria and values of service instance non-functional parameters (metrics), which are monitored.

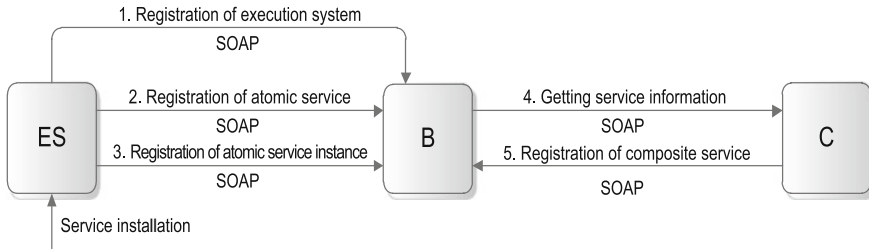
The Broker maintains a repository of all known services and components that support them (Fig. 14), i.e.: the set of atomic services (AS) available (and seen) for clients, the set of service instances localized at given execution systems, and the set of execution systems. It also maintains information about the available composite services.

The basic design feature that results in the service delivery system infrastructure as a service (infrastructure as a service, for short) is the BaaS (Broker as a Service) mode of the Broker's operation. This is accomplished by a set of Broker services available via a specified SOA-based interface. The Broker acts as a proxy for every client who requests network services but also delivers its own services that support integration for other service delivery system modules and the management of a virtual infrastructure. The services are WSDL compliant, accessed using SOAP protocol.

The BaaS services supply the following two main functionalities: registration and removing of entities of repository and delivering registered data, for example: `registerAtomicService`, `registerServiceInstance`, `getAtomicServiceList`, `getAtomicServiceMetrics`.

The services permit flexible management and integration of the main units of the service delivery system, i.e. the execution systems (ES), the composite service composer (C) and the Broker (B) itself. Figure 15 presents the complete scenario that includes basic steps for configuring and integrating the mentioned units.

After service installation in the execution system, the basic procedure that makes the network atomic service available includes three steps: registration of the execution



**Fig. 15** Basic steps of virtual service layer instantiation

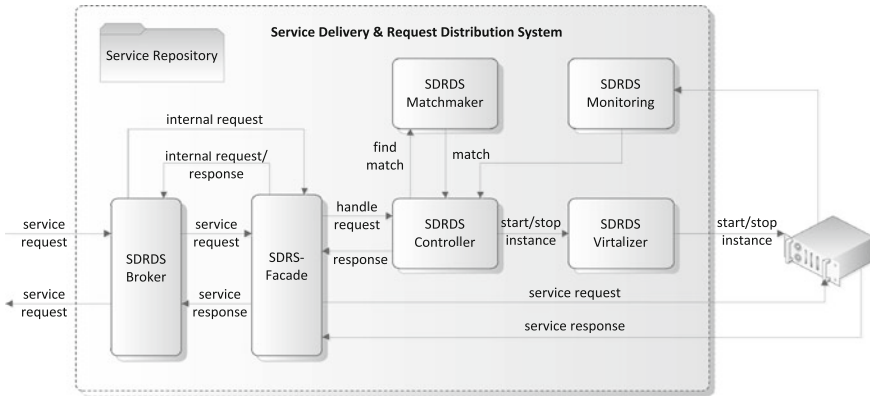
system by the Broker (if it is not registered yet), registration of atomic service (if it is already installed, it will be signaled to register), and registration of service instances. The order must follow the numeration of the presented steps.

If the client (e.g. service composer who wants to compose a composite service) does not know the available services, this information can be obtained with the use of a proper Broker service (Step 4). After that (or without the previous step, if the client is sure it knows the available atomic services), it can register a composite service for monitoring purposes (Step 5). The client can also get detailed information (e.g. monitored metrics) about each service, instance, or composite service.

### Service Delivery and Request Distribution System Architecture

The architecture of the Service Delivery & Request Distribution System (SDRDS) is presented in Fig. 16. It is composed of a number of independent modules that provide separate functionalities and interact with each other using specified interfaces. The main components are:

- **SDRDS-Broker**—handles user requests and distributes them to proper instances of virtualized execution resources. Decision making is based on specified criteria of the request distribution: in turn, based on non-functional requirements. It also performs internal requests to coordinate the operation of the system components as well as obtain some necessary information.
- **SDRDS-Facade**—separates the rest of components, supports communication with them, and collects necessary information for the Broker. It also provides special services for the Broker to test the current state of a processing environment (e.g. characteristics of communication links).
- **SDRDS-Controller**—manages all components behind the Facade. It is responsible for control processing according to the capabilities of the environment and its current state. It can also route the requests to the services (capsules) independently, taking into account computational resource utilization and performing decisions to start/stop another instance of service.
- **SDRDS-Virtualizer**—offers access to hypervisor commands. Uses libvirt to execute commands that give the project independence from a particular hypervisor.



**Fig. 16** Service delivery and request distribution system architecture

- **SDRDS-Monitoring**—collects information about particular physical servers as well as running virtual instances.
- **SDRDS-Matchmaker**—module responsible for properly matching the requirements of the request with capabilities of the environment and current state of it.

The instances of a given atomic service are functionally the same and differ only in the values of non-functional parameters such as completion time of execution or availability. SDRDS modules interact using two interfaces. Internal communication is XML-RPC based for components behind the Facade and uses SOAP messages for communication between the SDRDS-Broker and SDRDS-Facade. This allows for the flexibly to manage the distributed computational resources as well. Interaction with external components is based on the SDRDS-Broker services with SOAP messages. SDRDS delivers for clients' access to network services hidden behind it. Clients see network services at the Broker localization and do not know that services may be multiplied. From the client's point of view, services are described at the Service Repository using the WSDL standard and are accessible using standard SOAP calls. However, the Broker distinguishes individual execution resources, simultaneously coordinates activities with the SDRDS-Controller, and passes requests on to others. Each and every request is redirected to a proper instance based on the values of non-functional parameters of the requested service. Proper instance of service is either found from the ones that are working and available or the new one is started to serve the request. Such an approach gives the possibility to serve clients requests and manage resource virtualization and utilization automatically with minimal manual interaction.

The resource management is considered in the context of effective resource utilization and the delivery of network services with QoS. The effective resource utilization concerns performance issues as well as cost-related issues that embrace the financial cost of resource employment. The performance issues can concern such problems as load control (particularly load balancing) and the selection of suitable resources to

support given tasks. The cost of use of resources can be supervised by the control of their consumption with respect to time and capacity. Resource management is related to the quality of services. In fact, most of decisions related to resource utilization concern the quality of services. Both issues demand online monitoring as well as resource usage and resource conditions as values of the parameters of delivered services. Finally, we propose to distinguish two main policies for managing resources in the Service Delivery & Request Distribution System:

- control of resource allocation, taking into account supporting the quality of delivered services with the use of service virtualization, requesting distribution algorithms, and monitoring service execution as well as the execution environment;
- management of resource instances, capacity, and behavior with the use of resource virtualization.

Resource allocation can be performed in two primary areas: service process execution resources and communication resources. Communication resources management can be achieved on different levels with the use of different methods. With full control of communication links on a data link and network layer level within the entire network being used, the control can include establishing the route, new route creation, control of communication link saturation, link throughput, etc. The last two examples also refer to full control of dedicated links. However, on the Internet this is usually not the case. After all, the resource allocation at the application layer level, performed as resource selection (allocation of given set of resource), can be applied. The last case is considered in this work. The resource allocation that takes into account the quality of services is performed with a request distribution policy by the Broker.

## Communication Resources Management

Communication resources management is performed as a resource selection for every service request, i.e. allocation of a given communication link for the given request. At the same time, it works in tandem with the selection (optional instantiation) of computational resources. All is accomplished on the basis of the non-functional parameters of network service processing.

The request for atomic services can be labeled with non-functional attributes corresponding to the quality requirements of the request, such as response time, reliability, price, etc. In [52], a service selection was proposed in a way that maximizes user satisfaction expressed as utility functions over QoS attributes. However, this approach assumes that all service parameters are constant. In practice, it may be distinguished by two kinds of parameters: static—constant over a long period of time (e.g. price), and dynamic—variable in a short period of time (e.g. response time of service or throughput of service data transfer). For this case, we propose request control with a dynamic estimation of values of parameters characterizing network service instances. We distinguish two separate cases for considered approaches to request control:

- request distribution at the atomic service level only;
- request distribution at the composite service level.

The instance of a given atomic service is characterized by the values of non-functional parameters. The quality-aware service delivery can be performed using different approaches according to the constitution of requirements and the infrastructure of the delivery system. Taking into account communication infrastructure, we distinguish two general approaches:

- satisfying quality requirements for the request by direct control of consumption of resource (communication link) capacity for dedicated links from the Broker to execution systems;
- examining quality requirements for the request versus the values of non-functional parameters of service instances and the selection of proper service instances for service execution.

These two approaches essentially do the same thing: assign (or reject) the request to the communication link; however, they differ in the point of interest. The first one focuses on communication link parameters. The second focuses on network service parameters.

The Broker performs resource allocation decisions based on the actual values of the service instance parameters and chosen distribution strategy. Making control at the atomic service level, the problem of service request distribution can be expressed using criterion function on service instance non-functional parameters.

The criterion function can be expressed as different combination of more than one parameter and a formulated condition to fulfill the function requirements can be different as well, depending on the system's design assumptions. It may be formulated as guaranteeing the exact values of service parameters or, in this particular case, finding the extreme of the function and use the best effort, strategy, etc. An example of such distribution criterion can be minimizing the sum of data transfer time for given instance of atomic service and the completion time of its execution. It is a single request response time minimization. Assuming one communication link to one execution system, the selection of service instances is also the selection of communication link and selection of a virtual server.

To support distribution using dynamic parameters, its current values must be used. For the best effort, a distribution strategy following estimations of values of current parameters can be used:

- best last—the best (minimum) last monitored value of estimated parameter for all instances of service;
- best mean—the best (minimum) average value in k-window for all instances of service;
- best max—the best (minimum) maximum value of in estimated parameter for all instances of service;
- best prediction—the forecasted value derived using any forecasting method, e.g. artificial intelligence approach.

Considering something other than a best effort strategy, the same estimations can be used.

For most important dynamic parameter, request response time, the Broker monitors and registers transfer time and execution time of each request and response. These times are used to estimate response time.

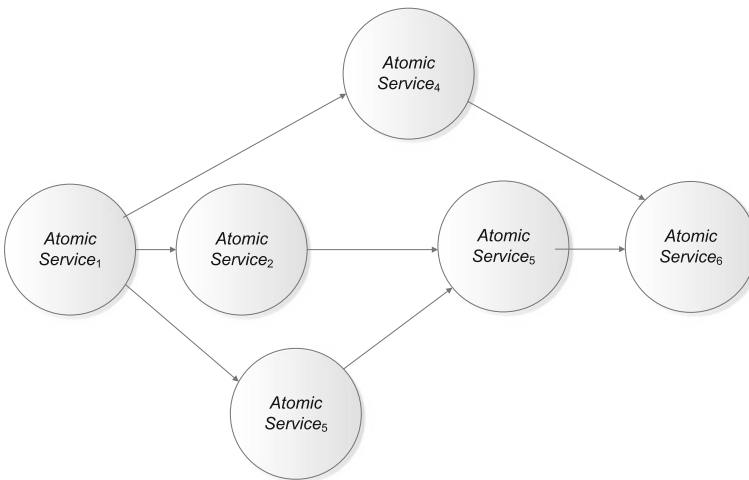
For forecasting transfer times and execution times of the request for atomic service served by the given instance, the fuzzy-neural controller [15] is used. Each service instance and each communication link for each service instance are modeled as a two stage fuzzy-neural network with two inputs characterizing the current conditions of modeled entity and the estimated time as an output.

For execution time, the model refers to the execution server. The two inputs are processor loads and the number of requests actually served. For communication links, the two parameters can be chosen from the following:

- the estimated (using time series analysis) latency (the exact TCP Connect time is proposed);
- the estimated link load derived from a regular test of the link—download of test portion of data;
- the aggregated intense of transfer of all requests directed to a given link.

Making control on the composite service level is more complex. The composite service scenario (composite service plan) can be described as a directed acyclic graph (see example in Fig. 17 where nodes represent atomic services to be executed and the edges define dependencies between atomic services—precedence relationships).

For the composite service scenario graph of the service requested at the given moment, there is an equivalent graph in which nodes are sets of instances of atomic



**Fig. 17** Composite service scenario graph—an example

services that constitute composite services. This equivalent graph defines different execution graphs (i.e. graphs that refer to concrete instances of atomic services).

The request for composite service at the given moment can be characterized with requirements on values of the non-functional service parameters. Every non-functional parameter of composite service is a function on the non-functional parameters of atomic service instances composing this service. The form of the function depends on the kind of parameter as well as the structure of the composite service implementation graph.

Knowing all non-functional parameters of instances, it is possible to perform different ways to fulfill the composite service non-functional requirements. If we consider assuring requirements for each request for composite service separately, the goal of service request distribution can be expressed with criterion functioning similarly; as for atomic services, however, it is formulated with use of a set of parameter vectors instead of a set of parameters and is more complex.

The task to select such instances so that the criterion function is satisfied (minimized in this particular case) is equivalent to the determination of the composite service proper execution graph. The determination of proper instances is performed in two ways. The first approach is to specify all instances the moment the request for composite services is received. The dynamic approach assumes that every execution of a subsequent atomic service that makes-up the composite one runs the procedure for the determination of the service instance that fulfills the specified criterion. The SDRDS broker supports queries for the composite services execution plan registered with the Broker.

## Computational Resources Management

The main driver of SDRDS' internal communication was the condition of independence. Each and every module was meant to be operable without the others. This leads to the implementation of communication among internal modules using the XML-RPC protocol. It is the ancestor of the SOAP protocol and as such has limited functionalities in comparison to its descendant. Nonetheless, they are more than sufficient for the requirements implied by SDRDS. Moreover, the protocol itself is widely supported in various languages plus it is well documented. The lack of some possibilities that have been adopted in SOAP is not a problem in this case; on the contrary, the simplicity of use and the maturity of the protocol mean no problems coming from various implementation of the protocol in various languages, further promoting the independency of internal modules.

Communication with hosted services is conducted with the SOAP protocol. Each of these services is also described using WSDL and has the advantage for the execution engine (part of the service composer) that it is capable of dynamically generating requests and creating composite services based on the existing atomic ones.

The WSDL document describing the service is linked with a URL address that allows one to call among a number of other resources available on the Web. Yet, due to the dynamic nature of the entire process—in this case, during the registration



phase—it is modified in such a way that the URL is no longer connected with the original URL (e.g. a local address if a document was generated in this manner), but rather with the Broker that serves as a proxy to access the service. This hides the true service situation from its customer who does not need (or even should not) know how many instances there are of a given service and to which one his/her request would be directed.

The independence of all of the modules actually led to some overhead in the registration process which is now conducted in three phases. Those were realized automatically and come from the logical consequence of the actions. First, a new service is being registered in the SDRDS. It is responsible for managing virtual machines; thus, nothing can carry on without informing this system that there is a new service that it will incorporate into this process. During that step, some information concerning the execution details (e.g. required RAM, number of processors) is given. The next step consists of the registration of the service in the SDRDS-Broker and the beginning of the first instance of the service. During this phase, the provided WSDL is modified by the SDRDS-Broker and made available publicly as if it was his/her own. The final phase is connected with the registration of the service in the external service repository that belongs to the service composer. The last of the steps requires some additional parameters that cannot be found in the WSDL to be provided. Those are non-functional parameters to be used later by the service composer in the composition process.

Hiding the real address of the service has deep meaning in a system like this. It allows SDRDS to manage the services and their instances in an independent manner for the execution engine in which only the address is important and that it remains constant. On the other hand, information about services might be used by the SDRDS-Broker. It is a semi-internal module of SDRDS that presents the access point to the services. The SDRDS-Broker possesses information about the instances and the realization times regarding them, and can therefore suggest where the request should be directed. Its role in this process is basically limited to forwarding the request and appending the suggestion of where should it go. There is also an alternative path for processing the request that takes into consideration more conditions, including all that might have been already enclosed in the request. In that case, the SDRDS-Matchmaker module is used to find the best match.

The SDRDS-Broker module is an element that remains outside the others—in the way that is publicly available for service clients to access services. It provides an API called Broker as a Service (BaaS) that, in basic terms, is used to register and execute the services. The Broker is not, however, only a mere proxy between the service client and the provider but also influences the internal processing of the message by giving extra information regarding the proposed instance in order to process a particular request with the goal of reducing time spent. The most important functionalities of the Broker are: Execution system registration, Service registration in broker, Service instance registration with Broker.

During the composite service registration, SDRDS predicts the service execution time and the necessary resources needed for service execution. The time estimation made is the SDRDS based on earlier realizations of the requests. This allows for

the prediction of the realization time either simply by averaging the times or by an approximation based on just the last requests. In both instances, we first need to register the composite service with the Broker to be able to trace not only a single request, but also a composite one as a whole. The time of the composite service realization is computed based on historical data from previous runs of the services. Those times are simply totaled to give the overall time of the composite service realization. In the case of time, it is important to note that the algorithm performing the computation assumes a full sequential of the composite service. It does not take into account the fact that some of the computation could be done in parallel, for example, and thus reducing the total time of the composite service.

The estimation of the resources required by the composite service execution is not a trivial task. Currently, it is done with few simplifications. It is assumed that the request is realized with the worst case scenario, in which all of the resources assigned to the service will be used while performing the request. Used resources are estimated assuming this negative scenario. For each atomic service involved in the composite service, it is assumed that all of the resources available for the instance will be utilized.

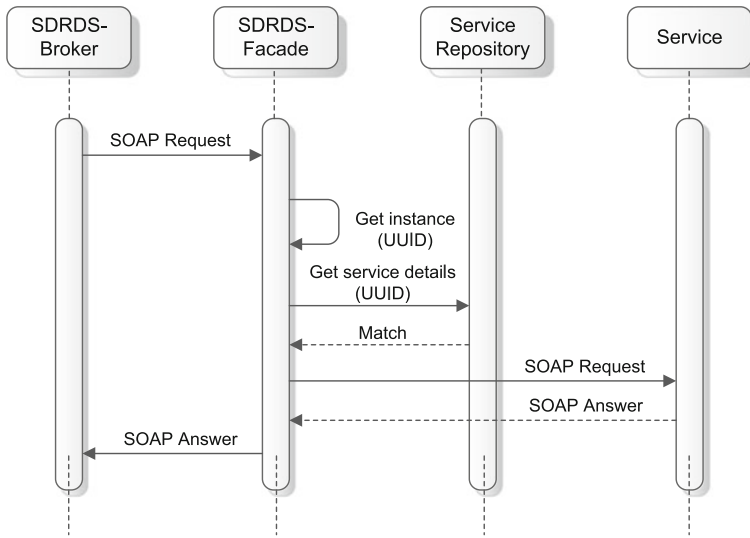
### ***4.3 Execution Engine-Execution Environment Interaction***

Communication between the Execution Engine and Execution Environment was done with independence in mind. In any event, neither of those two will depend on the other. Furthermore, the Execution Environment will preserve transparency for other modules, including the Service Composer and Execution Engine. On the other hand, composition and execution tasks performed by the Execution Engine will not depend on the location of a particular service or how it is hosted but solely on the provided functional and nonfunctional description. This leads to the usage of generic registration mechanisms offered by the Service Composer in order to communicate the presence of new services to this module.

The only dependency here comes from the requirement to obtain the address of the repository and to adjust communication to the interface offered accordingly. Registration is offered again using the XML-RPC protocol. Registration is simply a call to the AddService method that takes a list of information: name, class, address, inputs, outputs, location, cost, response time, and availability. After registration, the service can be immediately used by the Service Composer in its processes.

Realization of atomic service means usage of the SOAP protocol just like it is used when there is no interaction between the provider and client. Although there are no extensions to the regular message, they might be introduced to fine tune the realization of a particular request.

Extra parameters, which are then processed by the Execution Environment, are described with more detail further. Usage of those parameters is one of available paths of processing a request. The latter is connected with the Broker who can extend the

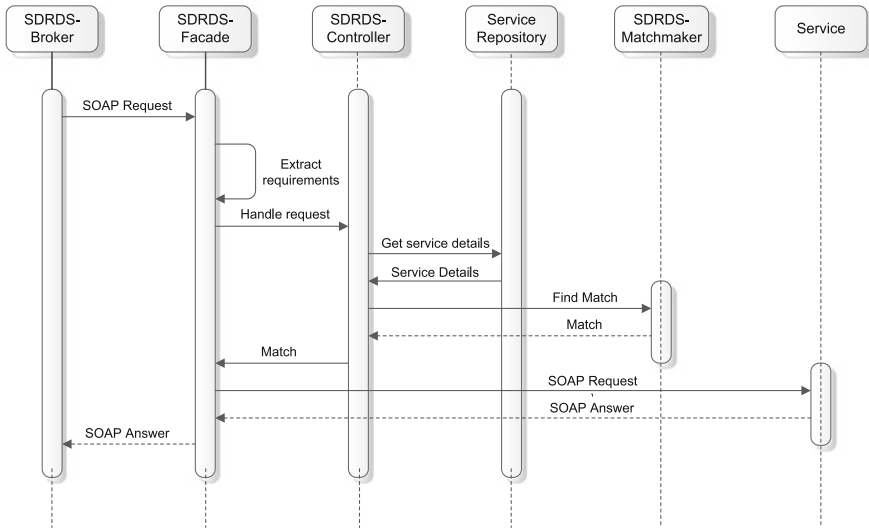


**Fig. 18** Sequence diagram—request processing with broker suggestion

message with explicit information about the instance to the one who will process the incoming request.

The Broker, as it was already mentioned before, gathers some data on request performance in particular instances. Having those data, the Broker is able to suggest which of the available instances shall be used to perform a certain request. Such information is simply placed in a SOAP message header, and then the modified message goes to the SDRDS-Facade module. It is then the responsibility of the SDRDS-Facade form to get the details about the instance (IP address) and forward the message there. The answer is then sent back to the Broker and back to the client (in this case to the Service Composer). Figure 18 presents the simplified scenario for service request processing.

In cases where the Broker does not send the information about the instance upon which the request shall be directed, a longer path would be used to find the destination address. The SDRDS-Matchmaker module would then be fed the data describing the request and system status. First of all, the request is described by the name of the service: secondly, by all of the constraints passed in the header of the SOAP message. It is specified in XML and then translated into the form understood by the SDRDS-Matchmaker. Next, the matching process has two steps. In the first step, matching is done only among the already running instances, and only in the case when no matching instance is found the second step happens. The second step means checking the capabilities of the servers in order to begin a new instance that would handle the request according to the request submitted. The entire process is described in Figs. 19 and 20. Figure 19 shows a request processing when there is an active service instance



**Fig. 19** Sequence diagram—request processing with successful matchmaking

that satisfied the request requirements and Fig. 20 when there is a need to activate the new service instance to fulfill the request requirements.

In some cases, access to internal data is required to work with an execution environment. Those rare situations are exhibited to the outside world using SOAP services—just like all of the services made publicly available. Each of the exposed actions extends the address of the system with certain keywords. The list of available actions is as follows:

- **system\_load**—access to information about the number of requests finished and being processed;
- **link\_load**—allows measurement of the communication with SDRDS and as a response returns the requested amount of data;
- **echo**—simply sends back received message;
- **monitoring**—access to system logs; access can be filtered out using following parameters:
  - **system.requests\_age**—age of requests to be filtered;
  - **system.requests\_since**—exact point in time when the requests will be given;
  - **system.requests\_until**—complementary to the parameters above and making it easy to mark the range in time from which requests will be given;
  - **system.requests\_to**—provides name of the service to get the logs from.

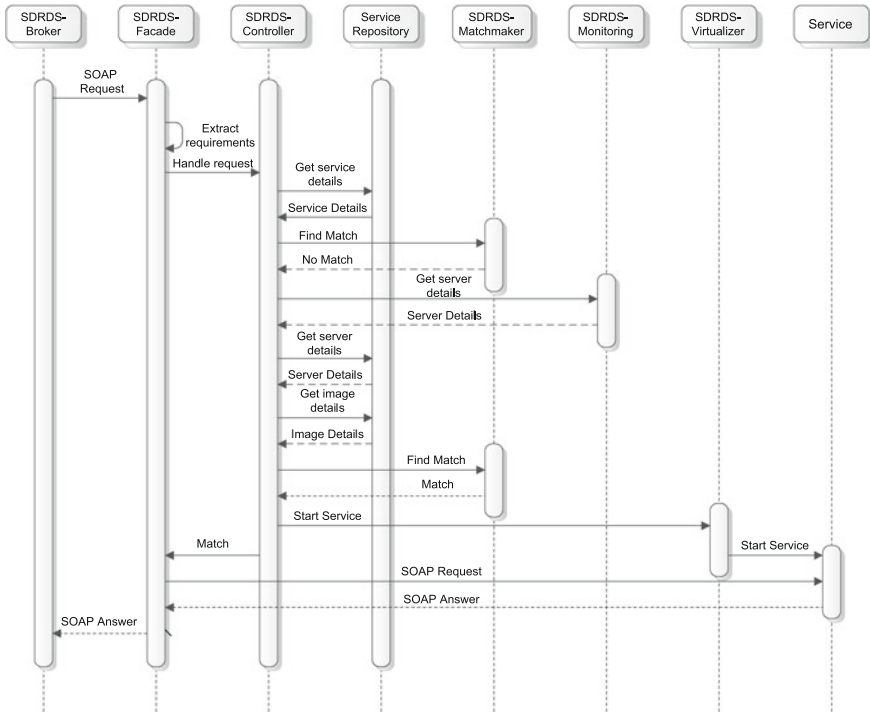


Fig. 20 Sequence diagram—request processing with unsuccessful matchmaking

## 5 Composite Services Validation

This section focuses on the method of composite service validation in SOA-based systems. The proposed method to measure the security level of services uses a layered approach derived from the SOA governance model. Based on the security level estimates of the atomic services, an information fusion scheme—utilizing Subjective Logic formalism—is proposed to evaluate the security level of a composite service.

The composition of Web services, which has been presented in detail in earlier sections of this chapter, allows the building of composite workflows and applications on top of the SOA model [62]. The main aim of service composition is providing a new functionality to the end-users but apart from this, composition should be also done with respect to the Quality of Service (QoS) requirements. Preserving the non-functional requirements defined by QoS parameters is a key factor in which its importance rapidly grows in distributed environments where composite services are generated from atomic components [49]

In order to fulfill the non-functional requirements, the Service Level Agreements (SLAs) are defined or negotiated between service providers and their clients. The SLAs must be precise and unambiguous; they should also be obeyed regardless of the

current system's state and the complexity of services being provided [4]. The QoS characteristics may be expressed in many ways; there are also several QoS specification languages including QML and WSLA [28]. While composite services consist of atomic services which are often available in multiple versions (e.g. implementations using different programming languages or supported by a different hardware layer), the assessment of QoS properties is complex task, and there appear to be several optimization problems [24]. In some cases, sophisticated frameworks are used to deal with the QoS of composite services.

Among other QoS proprieties, the security-related elements have attained special importance as the number of service-oriented systems grows. The literature related to SOA security focuses on problems with threat assessment, techniques, and functions for authentication, encryption, or verification of services [13], and [42]. Some other works focus on a high level modeling processes for engineering a secure SOA [54] with trust modeling, identity management, and access control. Many studies focus on secure software design practices for SOA, with a special interest in architectural or engineering methodologies as the means to create secure services [30]. On the other hand, one could notice a rapid development of semantic technologies which benefit from ontologies and semantic similarity assessment methods for service composition and orchestration. The example approaches to the composition of semantically-described services are presented in [30]. The key issue resulting from the use of semantics is the automation of service management tasks, especially service composition. In most cases, building composite services converts into a satisfaction problem regarding constraint—there can be many candidates (atomic services) for building blocks of a composite service (process), and it is necessary to select the optimal execution plan. So, the constraints defined in the optimization process are selected to satisfy the chosen QoS parameters. There are many approaches to the task of QoS parameter assessment for composite services, such as service-oriented systems. However, all they miss the general approach to the composite service security level estimation problem. The proposed solution in this section fills this gap by defining a formal model of composite security level service evaluation.

## ***5.1 SOA Security Governance***

The general flexibility of SOA based systems results in problems related to information assurance. These types of problems, which are quite natural in most distributed environments that have been designed to provide inter-operability and availability, become more challenging in a context of a service-oriented paradigm. SOA-based systems assume a scenario with multiple service consumers interacting with multiple providers, all potentially without prior human confirmation as to the validity of either the consumer request or the provider response. According to application of knowledge-based systems supporting the process of a security level evaluation, it is possible to reduce direct human intervention or interpretation to seek or acquire "valid" information about the QoS of composite and atomic services. This idea is

based on the automatic processing of well-defined resource description standards and the high availability of such resources.

The specific problems of service-oriented systems validation related to the system environment are as follows:

- **Identity management**—the identity needs to be decoupled from the services. All entities like users, services, etc. may have their own identities, and these identities need to be properly identified so that appropriate security controls can be applied.
- **Seamless connection to other organizations on a real-time basis**—organizations are providing services which should be integrated during the composition process. To enable execution of composite services where atomic services come from different providers, the connection between organizations must be seamless.
- **Proper security controls management**—there is a need to ensure that, for composite services, proper security controls are enacted for each atomic service and when services are used in combination.
- **Security management sovereignty**—SOA needs to manage security across a range of systems and services that are implemented in a diverse mix of new and old technologies.
- **Protection of data in transit and at rest**—execution of composite services is associated with complex data flow. Because of the weakest link security paradigm, this data flow must be secured at each intermediate stage.
- **Compliance with a growing set of corporate, industry, and regulatory standards**

These security issues arise in the context of the following SOA characteristic features:

- high heterogeneity of resources (both software and hardware),
- distribution of resources,
- the intensity and variety of communication,
- the different level of resources and tasks granularity,
- resource sharing,
- competition for resources,
- the dynamics of resources,
- the business level requirements related to security.

There are several standards and mechanisms that have been elaborated to provide and to maintain the high quality level of SOA-based systems. The basic solutions address the problems of confidentiality and integrity of data processed by an SOA-based system. Because of the network context of SOA and the multi-level security risk related to the layers of ISO/OSI network model, there are several solutions that offer data protection mechanisms at the corresponding level to each network layer.

In this context, the most commonly used and described are standards and protocols coming from the application layer that are maintained by the OASIS consortium. These solutions have been worked out to support the development of Web services and so also SOA-based systems. The other type of protection methods, mechanisms,

**Table 3** The validation requirements for SOA functional layers (selection)

SOA layer	Evaluate/verify/test
Business process	Policy consistency, policy completeness, trust management, identity management
Composite service	Identification of the composite services, authentication of the composite services, management of security of the composite services
Service description	Description completeness, availability, identification, authentication
Service communication	Confidentiality, integrity non-repudiation
Service execution	Availability, protection from attacks

and protocols are common for all network applications and can be used also in SOA-based systems as well in any other type of software.

As the SOA-based system can be defined by its five functional layers, the corresponding definition of SOA security requirements for validation of composite services should address the specific problems within each of the defined layers.

A selection of elements defining security requirements for the five layers of SOA systems has been presented in Table 3. The complete list can be found in [30].

The security level estimated with respect to the requirements, which have been enumerated in the Table 3, are typically constant. For example, the implementation of an atomic service does not change during operation, so its security level of authentication, trust management methods, etc. remains constant. However, a practical impact of a particular service on a security level of a composite service may vary in time. The fluctuations of the security level are caused by how the services are executed in a system. The security level should especially reflect the states of identified attacks against the services. Only then can the validation procedure bring up-to-date information about the security level of services that can be used during the composition phase.

Both elements required by composite services validation: the evaluation of service security properties for SOA layers and the service security level evaluation accordingly to an observed service execution history have been presented below.

## 5.2 Service Security Properties for SOA Layers

The proposed framework for composite service validation benefits from the model of trust and opinions and is called Subjective Logic [27]. According to this model, the security level of the particular layer of the SOA system is expressed using Subjective Logic opinions and Subjective Logic operators. The aim is to assign quantitative values to the measured security of the SOA architectural levels. Opinions which have been used are defined in Subjective Logic as tuples composed of three values



<belief, disbelief, uncertainty>, for which the following interpretation has been defined:

- The **Belief** component of the Subjective Logic's opinion which reflects trust in the security level of the service under consideration. Its value close to one literally means that one perceives the service as secure.
- The **Disbelief** component reflects the opinion that a service is not secure.
- The **Uncertainty** component is to express that the knowledge is partial or incomplete and the security assessment does not give a definite result.

### 5.2.1 Business Processes Layer

The Business Processes Layer defines the requirements for SOA-based systems that come directly from the business process definition. The most important elements pertaining to this layer and their influence for the process of services composition and execution have been discussed in detail in Sect. 2. One of the most important results obtained was a conclusion that the standard approaches to business modeling, such as ARIS or BPMN, have received too little expressiveness for services composition (e.g., they lack the ability to define the functional and non-functional requirements of composite service). As the security-related requirements are typically non-functional, this also limits the application of these approaches to the task of security level evaluation or generally to composite services validation. Introduced in Sect. 2, PEPC notation overcomes this problem and offers the well-defined framework that can be successfully used to express security-related requirements.

For example, PEPC can be used to define the service requirements for security policy, trust and identity management, etc. After that, during the service validation process, subjective opinions describing the level of how user requirements have been satisfied by the particular service implementation are calculated.

The Subjective Logic opinion about the Business Processes Layer is calculated as the combination of opinions regarding policy consistency, policy completeness, trust management, and identity management.

### 5.2.2 Composite Service Layer

For proper validation of composite services, two elements are required. The first one is a result of the corresponding atomic service validation. The second is the execution plan defining the composite service. The first element is performed using information provided by the Service Description Layer (described below). The results of the service validation are represented as Subjective Logic opinions calculated in analogous way as it has been described for the Business Process layer (a combination of opinions about the service authentication method, service authorization, identity management, etc.). The second element, which is required for the composite services validation, is provided in the form of an SSDL file. The SSDL file contains a

description of a composite service which has been defined by a set of nodes connected together in a graph-like structure. The details of the proposed method for enabling composite services validation has been presented in the next sections.

### **5.2.3 Service Description Layer**

The Service Description Layer in the context of the services validation process focuses on analyzing the functionality offered by services (e.g., about authentication, input and output provided by services, etc.). Corresponding information used during the Subjective Logic opinion calculation can be obtained from appropriate WSDL files, for example [30]. However, Sect. 3 of this chapter presents the fact that the WSDL file can describe only a single Web Service and its functionality. Additionally, this description offers a limited level of precision. The approach to service composition presented in earlier sections benefits also from the knowledge representation in a form of ontology. Ontology contains many of the domain-specific parameters which may be defined according to the current needs; therefore, during the service composition process, several specific domain ontologies (e.g. business service ontology, telecommunication service ontology, etc.) have been used. In this context, domain ontology has become an important source of information that is used while calculating Subjective Logic opinions about services. For example, at this layer, the required information about what is implemented is provided by service authentication or authorization methods.

### **5.2.4 Service Communication Layer**

This layer addresses elements that are responsible for communication between services. This is a critical element when we consider the composite services (services must be able to send/receive data to/from each other), and this is also an important element to be taken into account during the composite service validation process. The calculation of an appropriate Subjective Logic opinion about security at the Service Communication Layer is a result of the analysis of several elements influencing communication security, e.g. communication confidentiality, integrity, or authentication.

### **5.2.5 Service Execution**

The proposed method for a service's security level estimation using a service execution history actually benefits from an anomaly detection approach toward finding attacks and security breaches in ICT systems. The main assumption of this method is that attacks and security breaches are related to abnormal system behavior. While the typical behavior of most ICT systems shows some periodicity, e.g. number of processes executed during the daytime or data transferred, it can be assumed that

also the data rate, volume, etc. characterizing the services executed will show this type of dependency. Therefore, the anomaly detection approach should be also valid to estimate not only the system but also the security level service. The purpose of anomaly detection is to identify so-called anomalous states of the systems. To achieve this, a tuple of characteristic values describing the system's state is observed. Next, using pre-defined methods and criteria, the current values of that tuple are classified into one of the two following classes: normal behavior and anomalous behavior. An observation might be an anomaly in a given context, but an identical data instance (in terms of behavioral attributes) could be considered normal in a different context. Contextual anomalies have been most commonly explored in time-series data. Other popular methods frequently implemented to detect anomalies in ICT systems are outlying detection methods which can be divided between univariate methods and multivariate methods or parametric (statistical) methods and nonparametric methods that are model-free.

The analysis performed to detect an anomalous state of services can be done in four steps. The first step is a feature selection. A selection algorithm chooses among a set of candidate feature functions. The second step is parameter estimation, and in a general approach, a new feature function is added to a model and the weights of all feature functions are updated. Collected historical data about service behavior (features values) are compared with the current feature value. After this step the model of service behavior is constructed. This model is built by iterating Steps 1 and 2 until a pre-defined stopping criterion is met. Finally, the last step is anomaly detection. A large difference between the distribution of the selected feature value and a baseline distribution derived from training data indicates an anomaly.

The anomaly detection methods usually classify the system's state into one of two well-known classes: anomalous and normal. However, there are also approaches to anomaly detection which uses fuzzy methods, allowing for the presentation of a wider scale of a system's states. These fuzzy methods make it possible to define a more fine-grained representation of the service's security level. In the proposed method of evaluation in this section, opinions about service security expressed in Subjective Logic correspond to the fuzzy approaches to anomaly detection.

Subjective Logic opinion about a service's security level is calculated using the service execution history. Data describing the history of the execution is provided by a monitoring subsystem, which is an integral part of the composite service execution environment described in detail in Sect. 4. This data allows for the calculation of the values of Subjective Logic opinions in a form of three previously-defined values  $\langle \text{belief}, \text{disbelief}, \text{uncertainty} \rangle$ . The disbelief value, in estimation of the security level for anomaly detection, is proportional to the value returned by a distance function, which calculates the distance between the current observation and the typical observation (e.g. in some implementations it may be a simple difference between the current value and the mean value of the observed parameter).

The disbelief value varies from 0 to 1. When the detected anomaly is relatively small (near the average value), the value of the disbelief component of the Subjective Logic opinion will be near 0. While the high deviation from the earlier observed values has been observed, the disbelief value is equal to 1.

The uncertainty value in the Subjective Logic opinion about security level of the monitored service is related to the variance level of the observed parameter values. This means that when the observed parameter has a small variance, the uncertainty level is equal or near to 0. While the variance increases, the uncertainty level of the Subjective Logic opinion also increases until it reaches its maximum value.

### 5.3 Aggregation of Security Properties for Composite Services

A composite service execution plan is a graph that (apart from indicating which atomic services are used in a composite service) represents the structure of a composite service. Execution plans consist of serial, AND-parallel, or XOR-parallel execution substructures. After having assessed the security level of atomic services, the security of a composite service may be evaluated. In general, it is assumed that the security of a serial execution plan—where there is a chain of services executed one-by-one, is defined by the security of the “weakest point” (or “link”) in the chain (indicating the possibility that any security breach may occur during the execution of composite service). The same thing concerns the AND-parallel execution plan. In the case of the XOR-parallel plan—where it is not determined which service will be executed, the security level of all parallel connections should be taken into account.

In the presented approach, an execution plan of composite service is defined by the SSDL file. After obtaining the graph structure from a corresponding SSDL file, a security-level estimate for each connection between services connected by one edge is defined in the form of a Subjective Logic opinion. According to the—*weakest point*—paradigm, if for example, a non-secure service is followed by a set of services executed in parallel, then its insecurity will greatly influence each following security level estimate. To find a security level estimate of a composite service, all opinions about links between services are aggregated independently to find the—*IJ**weakest link-I* of the composite service—that is, a connection with the lowest security estimate. According to this, a composite service *general security level estimate* has been defined as a probability expectation that projects security level estimates onto a one-dimensional probability space.

As the information used for security level evaluation comes from the sources which are fundamentally different by origin and technical nature, there has been a second strategy also defined for producing opinions about composite services—the *layer-dependent security level estimate*. The *Layer-dependent security estimate* has been based on layer-specific opinions which have been described in Sect. 5.2. In this case, the rules for generating opinions about the security of the execution plan apply separately to opinions concerning specific SOA layers. Calculation of a composite service *layer-dependent security estimate* for Service Communication Layer is performed in the following steps:

- calculate the Subjective Logic opinion about the Service Communication Layer security level for each service constituting the composite service of a given execution plan;
- calculate the composite service security estimate for the Service Communication layer using the formula that defines a probability expectation that projects security level estimates onto a one-dimensional probability space (in the similar way that it is done while calculating *the general security level estimate*).

The third possible estimation of composite service security level also takes into account the results of that anomaly detection process as it has been described in Sect. 5.2.5. The estimation of this type of composite security level is performed in the following steps:

- calculate service security estimate for each service from composite service execution plan for each SOA layer, also including security estimates concerning anomalies detected during service execution (as it has been defined by the *layer dependent security estimate*),
- aggregate those estimates into one security level estimate of a particular service using the Subjective Logic consensus operator form ;
- calculate *the general security level estimate* as it has been defined earlier in this section.

This flexible strategy allows the generation of three types of opinions about a composite service security level with required granularity and the depth of security analysis of a composite service in service-oriented systems.

As it has been presented in the Table 3, a security layer interacts with all other SOA layers. The first type of this interaction is data acquisition for SOA system validation at the corresponding level of description (Business Policy, Composite Service, Service, etc.) The main idea of SOA-based system validation, with respect to elements defined by particular layers, has been presented in Sects. 5.2.1 through 5.2.5. While the atomic service security level has been evaluated, the composite service can be validated using Subjective Logic operators, the execution plan defined by the SSDL file, and Subjective Logic opinions about atomic services (Sect. 5.3) On the other hand, the obtained results of the services validation are available for corresponding elements from specific SOA system layers. This type of feedback can be used to reorganize or reconfigure SOA system to fit predefined security requirements. For example, the results of the Business Policy validation may be used to redefine some policy elements that are responsible for the final security level of an SOA-based system. Similarly, the information provided by the service validation module may be used by a module responsible for service composition to select services, providing the required security level of composited service.

## 6 User Interface for Services

The problem of design and implementation of the personalized and automatically generated user interface (UI) for services in SOA-oriented systems is quite new. We can try to find similar solutions in the area of user interface adaptation that have been addressed in many papers before, for example [53] and [46]. The user interface is usually adapted to the personal user's needs and/or environment settings.

The SOA paradigm was proposed because of the discontent of the software community over interoperability, reusability, and other issues of traditional software development best practices and standards [6]. The service-oriented best practices are shaped to deliver strategic solutions for enterprise or other types of organizations that are focused on overcoming different shortcomings appearing at the tactical level. The SOA framework delivers a general guide on how to conceptualize, analyze, design, and structure organizational service-oriented assets [6]. It is believed that one of the most important features of the SOA-based systems is the composition of services. The composite service is built from at least two different reusable services. The consequence of this feature is that the particular user interface differs depending on the composition (sequence of services) and the context of use of the particular service.

In today's implementations of the SOA-based system, the user interface for each composite service and for each user role is designed and then implemented manually or semi-automatically using CASE-tools, however then the precise information flow has to be specified. A consequence of this is considerable time and money consumption, which is rising with the number of different user roles and the system environments to be programmed. These problems may be overcome by the application of the automatic user interface authoring procedures, which may be additionally enhanced by the application of the ontologies.

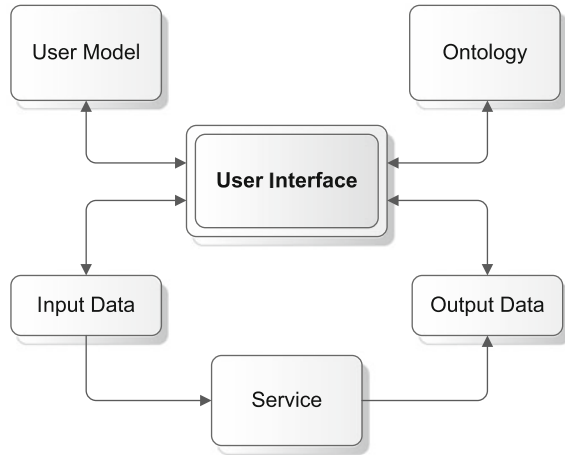
The UI implementation for PlaTel required the implementation of a specialized graphic tool—a Web-based SSDL GUI interface that was developed for supporting the Service Composer configuration. As a composite service, the service composer has its own execution plan, which may be redefined according to the needs and available services. In this way, it is very simple to construct different composition schemes just by dragging services from the repository and setting the necessary parameters.

In the following sections, we present composite service UI formal definition, UI automatic generation, user interface adaptation, and finally UI quality assurance. As an example, we take the delivery of UI for the applications of the PlaTel platform.

### 6.1 User Interface Definition

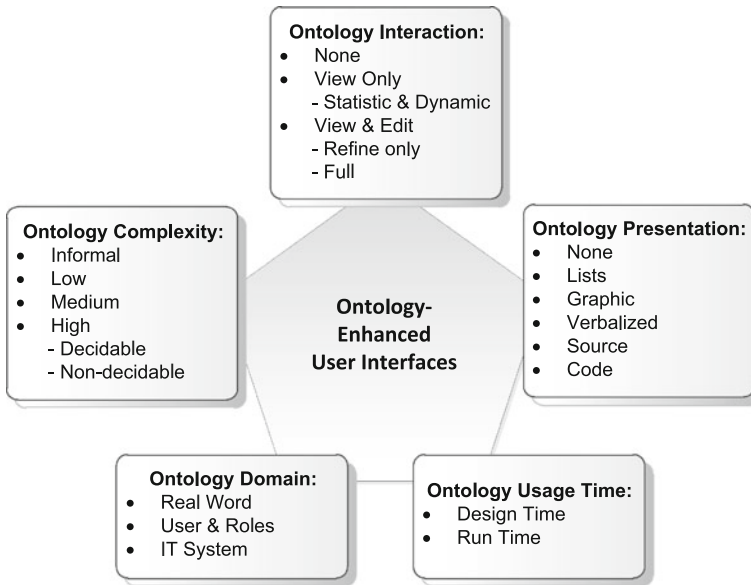
Automatic user interface generation for services in the SOA-based systems is possible according to the formal service description and the user model. The formal description given in the SSDL defines functional and non-functional description,

**Fig. 21** General architecture for UI generation in SOA system



service input, and service output. The most important element that is necessary for UI construction is the service input attribute names and the type of values as well as output attribute names and values. However, to present the input and output properly, we should possess the proper user model that decides, for example, what language is appropriate for the given user and how the specified input values should be entered (i.e. directly or using a select box) and how the output values should be presented (i.e. using a table of values or charts). In some cases, we should also apply ontologies to resolve some complex concepts into simple ones (i.e. address could be divided into street, city, state and zip code—in the case of an address in the U.S.). The general framework for UI generation for the service is given in Fig. 21. The user interface first enables the user to enter the input data, the input data labels, as well as their form and is worked out according to the user model; the ontology as well as the output data is presented according to the same elements.

Ontologies are applied in many areas of information systems design and development, such as database integration, business logic or Graphical User Interfaces (GUI). An ontology-enhanced user interface is defined as a user interface, which visualization capabilities, interaction possibilities, or development processes are enabled or (at least) improved by the employment of one or more ontologies [46]. In the user interface enhancement, we may use different types of ontologies. We can distinguish the following ontologies that concern: the real world, the IT system, users, and roles [46]. The real world ontology characterizes its part, especially the area of the system application (i.e., e-commerce, travel, etc.), in order to identify the central concepts and relations among them. The IT system itself delivers formalized ontology by containing categories such as Software Module, Web Service, etc. We may further divide the IT ontology into hardware and software ontologies. Finally, users and roles ontology characterizes the users, their preferences, and their roles, which have an influence on the rights and possibilities they have in using a system. In the work [46] and also other categories of the ontology-enhanced user interfaces have



**Fig. 22** Ontology-enhanced user interfaces characterization schema [46]

been distinguished, these are: ontology complexity, ontology interaction, ontology presentation, and ontology usage time (see Fig. 22).

Usually in order to formally define the UI, specialized language is used. Today in this area the most popular are languages based on XML; some of the most popular are XUL and XIML, but other UI description languages are also used such as: UsiXML, UIML, Maria, LZX, WAI ARIA, and XForms. It has been discussed, however, that the application of ontology will not replace the application of user interface description languages, but rather deliver its valuable enhancement.

In the implementation of the PlaTel user interface, we decided to choose the quite popular JavaScript library jQuery, which simplifies client-side scripting in HTML. We also applied different types of ontologies in the UI design and implementation. First, we adopted domain ontology, which is also used in the description of functional and non-functional parameter description of services.

In the work [68], we presented a method for Web-based user interface construction for SOA-oriented systems enhanced by ontology, where we postulate to extend the WSDL service file in order to define a user interface in SOA architecture for atomic as well as composite services. The proposed extension is based on populating WSDL nodes, which describe Web Service method parameters and child nodes describing the metadata.



## 6.2 *User Interface Automatic Generation*

User interface automatic generation is a process where the input is a formal definition of the UI, i.e. in the form of a file written in the specified user interface language (for example, XUL or XIML), and the output is usually in the form of a working GUI. PlaTel is a Web-based application that it is run in the Web browser, so in fact the final GUI rendering and control is made by the browser itself. The general idea of generating user interfaces for Web Services is based on the assumption that we can serialize user interaction in HTML form. Therefore, to generate a user interface for Web Services, we should be able to produce a Web form that would deliver all the necessary inputs from the user and make the interaction with a Web service possible [68].

The input of data in Web form should be validated by means of validation metadata. Some basic validation may be done using HTML5 Forms, which enable one to force enter a valid e-mail address or other pattern matching string. However, more complex validation, i.e. verification if given a credit card number, necessitates the application of more sophisticated methods needed.

There could be several approaches to the problem of input validation: one of which is the restriction of the user input to some pre-defined set of acceptable values, which are specified for each non-free text input, and practically limiting input into a select list. However, depending on the number of values and their types, we can distinguish the following restrictions [68]:

1. Size of the domain values set: if the select list contains more than 20 elements, applications of scroll list are very inconvenient.
2. Conditional nature of some inputs: if selecting one input value in one field determines set off acceptable values in another field, i.e. when in one field we select a certain country, then in another field the set of acceptable cities depends on the former choice.
3. Multilingual and internationalization support: the domain definitions should support multilingual and convention values, i.e. distances measured in kilometers or miles.
4. Multiple selection: Multiple selection: there are inputs that require a single value, some others might need entering a specified number (greater than one) of values entered from a defined domain, i.e. a single choice we have in the case of the student's faculty, and the multiple choice in the case of the list of enrolled courses.

The first problem could be solved using specialized input widgets that allow filtering domain values and shrinking the select list while the user enters the following characters contained in the value [68]. This problem could be also solved using an autocompleting mechanism.

Usually, the user interaction with a system is some kind of user-system conversation, and the accessible functionalities and possible user inputs depend on the current state of the system. As a result, some business logic has to be embedded in the user interface regarding user inputs in order to avoid invalid user input. For each set of

domain values, we should be able to define the context in which these values are valid.

The following problem is natural language that is used during the interaction. It is obvious that most of the text presented to the user is language-dependent and also the input value domains are specific for each language. In the case of a simple text field type input, a label should be language sensitive, but in a more complex situation it should support localization and internationalization.

Finally, in the case of multiple selection support it is becoming more difficult the longer the selection list is. We can consider several different cases: when the values to enter are known to the user, then a simple text field with auto-complete may be sufficient; however, in the opposite case, a filtering application or a shortening of the list is needed.

In order to verify our method that is based on the WSDL extension, a UI generator has been built. It generates a jQuery widget Web-based user interface according to the description given in WSDL that is enriched with necessary metadata. The generated Web-based user interface may be localized for the specified language or date format, as well as pre-selected values in the select list, ratio, or checkboxes. So far, we have implemented several types of jQuery widgets, and the generator could be further extended by the application of graphical visualization, i.e. geo-data presentation using Google Maps API [68].

In the present stage of PlaTel user interface development, the user interface definition is specified manually. However, it is ready for the user interface adaptation that is described in the following section and also for dynamic generation in the event of compound services. It must be noted that HTML 5 gives its own solutions for generating user interfaces for Web-services; however, there is a problem in work flow of the user interface and service calls. This problem is being solved by HTML5 Web Workers, and thus in our future works we will concentrate on its application in PlaTel.

### ***6.3 User Interface Adaptation***

User interface adaptation is a process that results in delivering the personalized UI for the given user. The key element of UI adaptation is the proper user model [56]. The user model usually contains the user data, usage data, and environment data. The user data contains: demographic data and users' knowledge, their skills, interests and preferences, and also their plans and goals. The usage concern selective operations that express users' interests, unfamiliarity or preferences, temporal viewing behavior, ratings concerning the relevance of these elements, as well as different interactive events, such as opening a page, purchasing a product, sending feedback information to the system are stored. The environment data contains information about software and hardware platform data and data about usage conditions. The user model may

be initialized and maintained in many different ways [56]. Additionally, in the case of PlaTel, we record many different usage data that can be used in UI adaptation.

Having the proper user model, we may utilize it by applying, for example, the methods developed within the recommender systems or more specific user interface adaptation [56]. They are based on the following filtering methods: demographic (DF), collaborative (CF), content-based (CB), and hybrid approach (HA).

To date, we have implemented a simple user interface adaptation based on demographic filtering; this implements stereotypical reasoning based mainly on the user's demographic data and is very easy. The adaptation concerns personalization of the user interface language, date format, some other default data, etc. More sophisticated adaptation methods may be applied when we gather more information about the user interface usage by a considerably large number of users.

## 6.4 User Interface Quality Assurance

The key notion of quality of user interface is usability. Usability, and also other connected concepts, are defined within many different international standards and guidelines such as: ISO 9126, ISO 9241 and ISO 13407 standards, W3C WAI (Web Accessibility Initiative), or design guidelines such as the Sun Java Look & Feel, Gnome 2.0, Apple, or MS Windows Guidelines. Considerable comprehensive usability definition is given in ISO 9241-11 standard: *Usability is an extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.* Where *effectiveness* is defined as: *accuracy and completeness with which users achieve specified goals*, *efficiency* is defined as: *resources expended in relation to the accuracy, and completeness with which users achieve goals*, *satisfaction* is defined as: *freedom from discomfort, and positive attitudes towards the use of the product* and *context of use* is defined as: *users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used.*

Many different methods of usability testing have been developed. They may be applied at a specified phase of the interactive system development: from the planning and feasibility studies to post release. In practical cases, we should apply at least several methods to achieve acceptable results. However, usability is not the only quality parameter we should verify in SOA systems quality assurance. We should also generate the following types of tests: functional, automatic, and sensitivity. Below, we present the quality assurance tests we have made for the PlaTel prototype verification. The functional tests verify if the application is able to return the results that have been assumed by the designers. We verify this by checking each single function, possibly with several sets of representative input data. When we verify the user interface we should test it manually, using the specified input devices such as: keyboard, mouse, touch-screen, etc. PlaTel is implemented using an agile programming methodology—SCRUM, which is iterative and incremental, and the following steps are made within so-called sprints. Each sprint is restricted to the particular

length of time (i.e. one week) and brings specified requirements which should be implemented. These requirements must be and are recorded in the sprint backlogs, so then it is quite easy to verify if the specified functionalities are completed.

The second type of the verification tests that we have made is called automatic testing. PlaTel is a Web-based interactive application that is run in an Internet browser. Today, this type of application is tested automatically by a software tool that enables repeatable tests to be conducted. Test automation has many advantages over manual testing; they are mainly related to a test's repeatability and speed. There are many such tools available commercially and free of charge. In the automatic testing of PlaTel, we have used a free tool called Selenium IDE (Integrated Development Environment), which is a prototyping tool for building test scripts. This is a Firefox plugin, which provides an easy-to-use interface for developing automated tests with a recording feature of user interactive actions. These actions are exported to a reusable script that can be later executed.

The last type of test we have conducted is a sensitivity test, which verifies PlaTel behavior by using different software and hardware platforms. PlaTel is a Web-based application, so the most obvious verification is in using different Web browsers: Firefox, Opera, IE and Chrome while using standard a PC with Windows 7. As mobile apps are becoming more and more popular, we also verified PlaTel running on iOS using both the iPad and iPhone and also Android (versions 2.3 and 4.0) running on the smartphone and an emulator. The tests have shown that PlaTel is working pretty well on most PC browsers, iOS, and Android 4.0-based systems.

We have also conducted two types of quality tests: the first usability tests using heuristic evaluation and the second application code validation using different automatic validators. Heuristic evaluation is a usability testing method where usability experts judge whether the verified application follows a list of established usability heuristics (i.e. Nielsen's Ten Usability Heuristics). In this test, usually 3–5 experts are making independent judgments.

Automatic code validators are becoming more and more popular; they are easy to use and give quick results by locating specific problems in HTML or JavaScript codes. We have used the following validators: syntax validator W3C Validate by URI, CSS W3C validator, W3C link checker, JuicyStudio Readability test, and color visibility Vischeck.

However, making all the tests after completing each sprint would be very time consuming. We must admit that automatic code validators and automatic testing may be applied pretty often, even several times within the single sprint. Functional tests should be performed after each sprint. However, sensitivity and usability tests are time consuming because we should engage at least three experts for two working days; so, these tests should be performed only when we have available resources, but at least twice during the entire project.

## 7 Summary

The presented platform offers a set of unique applications, tools, and techniques which provide an integrated approach to composite service composition, service execution in a distributed environment, and quality of service monitoring. Originally developed service description language allows the inclusion of QoS requirements and composite service execution-related parameters in service description, which is directly used by the Workflow Engine. All the parameters are measured during service execution and may be used when needed by the Service Composer, which feature forms feedback between composition and service execution.

The invented framework is an extensible solution in terms of adding new methods, algorithms, and techniques for business processes analysis, composite services composition, as well as resources allocation, provisioning, and utilization monitoring purposes. The results of first experiments of the developed framework's components functionality, scalability, openness, and reusability with are very promising.

The presented platform is designed to gain support in decision-making tasks, which may be distinguished in the process of the matching required (business process) and available (computer-based information system) services. The business process and ICT systems' services integration scenario, as well as the required and available services matching process, is divided into several, well-defined steps which cover all activities that should be undertaken between requesting an arriving time until service delivery time (end-to-end working prototype). The scope of functionalities obtainable at distinguished steps may be easily extended by adding new procedures, methods, and algorithms.

The general concepts, implemented in the platform as a set of cooperating applications and tools, is based on the component-oriented software development idea; required business process information services may be delivered performing predefined (an available form services repository) or composed on-demand from service components (service on demand) distributed IT systems services. The presented platform is an attempt to offer various functionalities delivered as services: data processing as a service, composition as a service, security as a service, composition as a service, infrastructure as a service, monitoring as a service, etc. Such an assumption means also that the functionalities available in the presented platform above are reusable at different steps within the services matching process.

Functionalities of the presented modules are based on extensive data, information and knowledge gathering activities, and processing. It is evident, especially in the case of service composition, where ontology matching and prediction algorithms are used to select proper services or to obtain the optimal services composition of components available in various instances within a space-distributed environment.

The presented platform is still under development. The general framework idea assures that all of the presented functionalities may be easily extended by adding general-purpose and specific-oriented data in addition to information and knowledge gathering activities, and processing units. Moreover, it is assumed that both the available and planned processing capabilities are reusable in many parallel service

delivery processes as well as in various steps of the same distinguished service delivery process.

The functionality of the described tool also presents a multi-step methodology for semantically annotating services regarding matching, selection, composition, and execution. There are several innovations in the presented attempt. First of all, the services matching (management) is available in three versions: service selection, service level agreement negotiation, and service composition. The second is that the services composition (service on demand) process is decoupled into stages that allow for the utilization of various kinds of knowledge to obtain optimal composite service and its execution.

**Acknowledgments** The research presented in this paper has been partially supported by the European Union within the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

## References

1. A guide to the business analysis body of knowledge, International Institute of Business Analysis. [www.teiiba.com](http://www.teiiba.com) (2009)
2. Agarwal, V., Chafle, G., Dasgupta, K., Karnik, N., Kumar, A., Mittal, S., Srivastava, B.: Synth: A system for end to end composition of web services. *World Wide Web Conf.* **3**(4), 311–339 (2005)
3. Aguilar-Saven, R.S.: Business process modeling: review and framework. *Int. J. Prod. Econ.* **90**, 129–149 (2004)
4. Anderson, S., Grau, A., Hughes, C.: Specification and satisfaction of SLAs in service oriented architecture. In: 5th Annual DIRC Research Conference, pp. 141–150 (2005)
5. Badr, Y., Abraham, A., Biennier, F., Grosan, C.: Enhancing web service selection by user preferences of non-functional features. In: 4th International Conference on Next Generation Web Services Practices, IEEE Computer Society Washington (2008)
6. Bell, M.: Introduction to Service-Oriented Modeling. *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley, Hoboken (2008)
7. Blanco, E.: Techniques to produce optimal web service compositions. In: *IEEE Congress on Services*, pp. 553–558 (2008)
8. Borzowski, L., Zatwarnicka, A., Zatwarnicki, K.: Global distribution of HTTP requests using the fuzzy-neural decision-making mechanism. In: *Proceeding of 1st International Conference on Computational Collective Intelligence. Lecture Notes in AI*, Springer (2009)
9. Brzostowski, K., Drapała, J., Świątek, J.: system analysis techniques in eHealth systems: a case study. *Lecture Notes in Computer Science. Lect. Notes Artif. Intel.* **7196**, 74–85 (2012)
10. Brzostowski, K., Tomczak, J.M., Rekuć, W., Sobecki, J.: Service discovery approach based on rough sets for SOA systems. In: Nguyen, N.T., Zgrywa, A., Czyżewski, A. (eds.) *Advances in Multimedia and Network Information System Technologies*, pp. 131–141. Springer, Heidelberg (2010)
11. Cardoso, J., van der Aalst, W.: *Handbook of Research on Business Process Modeling*, Information Science Reference, ISBN: 978-1-60566-288-6 (2009)
12. Chynał, P., Szymański, J.M., Sobecki, J.: Using eyetracking in a mobile applications usability testing. *LNCS/LNAI* **7198**, 178–186 (2012)
13. Department of Homeland Security: National Vulnerability Database of the National Cybersecurity Division. <http://nvd.nist.gov> (2009). Accessed 20 March 2009

14. Fraś, M.: The architecture of complex service requests broker. Grzech, A. (eds.) *Information Systems Architecture and Technology: Networks and Networks' Services*, pp. 369–379. Wrocław University of Technology Publishing House, Wrocław (2010)
15. Fraś M., Zatwarnicka A, Zatwarnicki K.: Fuzzy-neural controller in service request distribution broker for SOA-based systems. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) *Proceeding of International Conference Computer Networks 2010*, pp. 121–130. Springer, Berlin (2010)
16. Fraś, M., Grzech, A., Juszczyszyn, K., Kołaczek, G., Kwiatkowski, J., Prusiewicz, A., Sobecki, J., Świątek, P., Wasilewski, A.: Smart Work Workbench : integrated tool for IT services planning, management, execution and evaluation. *LNCS/LNAI* **6922**, 557–571 (2011)
17. Grzech, A., Świątek, P., Rygielski, P.: Dynamic Resources Allocation for Delivery of Personalized Services. In: *I3E 2010, IFIP AICT 341*, pp. 1728 (2010)
18. Grzech, A., Świątek, P.: Modeling and optimization of complex services in service-based systems. *Cyb. Syst. Int. J.* **40**, 706–723 (2009)
19. Grzech, A., Rygielski, P., Świątek, P.: Translations of service level agreement in systems based on service-oriented architectures. *Cyb. Syst. Int. J.* **41**, 610–627 (2010)
20. Hackmann, G., Haitjema, M., Gill, C., Roman G.: Sliver: A BPEL workflow process execution engine for mobile devices, *LNCS* 4294 pp. 503-508 (2006)
21. Havey, M.: *Essential Business Process Modeling*, O'Reilly, ISBN: 0-596-00843-0 (2005)
22. Hoyer, V., Bucherer, E., Schnabel, F.: Collaborative e-Business Process Modeling: Transforming Private EPC to Public BPMN Business Process Models, *Business Process Management Workshops*, pp. 185–196 (2008)
23. Brzeziński, J., Danilecki, A., Flotyński, J., Kobusińska, A., Stroiński, A.: ROsWell Workflow Language: A Declarative, Resource-oriented Approach. *New Gener. Comput.* **30**(2 & 3) (2012)
24. Jaeger, M.C., Rojec-Goldmann, G., Muhl. G.: QoS aggregation in web service compositions. In: *IEEE International Conference on e-Technology, e-Commerce and e-Service*, pp. 181–185 (2005)
25. Jinghai, R., Xiaomeng, S.: A Survey of Automated Web Service Composition Methods, Semantic Web Services and Web Process Composition. In: *First International Workshop, SWSWPC, San Diego, CA, USA*, pp. 43–54 (2004)
26. Jong Myoung, K., Chang Ouk, K., Ick-Hyun, K.: Quality-of-service oriented web service composition algorithm and planning architecture. *J. Syst. Softw.* **81**, 2079–2090 (2008)
27. Josang, A.: A Logic for uncertain probabilities. *Int. J. Uncertainty Fuzziness Knowl Based Syst.* **9**(3), 279–311 (2001)
28. Keller, A., Ludwig, H.: The WSLA framework: Specifying and monitoring service level agreements for web services. *IBM Research Report*, May 2002
29. Klush, M., Fries, B., Sycara, K.: OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. *Web Seman. Sci. Serv. Agents World Wide Web* **7**, 121–133 (2009)
30. Kołaczek, G. Opracowanie koncepcji specyfikacji metod i modeli szacowania poziomu bezpieczeństwa systemów SOA i SOKU, WUT, (in polish) (2009)
31. Kohavi, R.: The power of decision tables. *Proc. ECML LNCS* **912**, 174–189 (1995)
32. Korherr, B., List, B.: Extending the EPC and the BPMN with Business Process Goals and Performance Measures, *9th International Conference on Enterprise Information Systems*, pp. 287–294 (2007)
33. Kozik, A., Rudek, R., Świątek, P., Grzech, A.: Resource allocation problems in network processors for the Future Internet. In: Grana, Manuel, et al. (eds.) *Advances in knowledge-based and intelligent information and engineering systems /*, pp. 1509–1520. IOS Press, Amsterdam (2012)
34. Kruczyński, K.: Business process modeling in the context of SOA – an empirical study of the acceptance between EPC and BPMN. *World Rev. Sci. Technol. Sustain. Dev.* **7**(1/2), 161–168 (2010)
35. Kwiatkowski, J., Fraś, M., Pawlik, M., Konieczny, D.: Request distribution in hybrid processing environments, *Lecture Notes in Computer Science*, vol. 6067. Springer, Berlin, pp. 246–255 (2010)



36. Kwiatkowski, J., Papkala, G.: Service aware virtualization management system. In: Grzech, A. (eds.) *Information Systems Architecture and Technology: Service Oriented Networked Systems*, pp. 317–326. Wrocław University of Technology Publishing House, Wrocław (2011).
37. Kwiatkowski, J., Pawlik, M., Konieczny, D.: *Efficient Computational Resources Allocation for Service Request, Application of Systems Science*. Academic Publishing House EXIT, Warszawa (2010)
38. Kwiatkowski, J., Pawlik, M., Fraś, M., Konieczny, D., Wasilewski, A.: *Design of SOA-Based Distribution System, SOA Infrastructure Tools. Concepts and Methods*, pp. 263–288. Poznań University of Economics Publishing House, Poznań (2010)
39. Lodhi, A.: An extension of BPMN meta-model for evaluation of business processes. *Sci. J. RTU* 5. series 46, 27–34 (2011)
40. Milanovic, N., Malek, M.: Current solutions for web service composition. *IEEE Internet Comput.* **8**(6), 51–59 (2004)
41. Minoli, D.: *Enterprise Architecture A to Z, Frameworks, Business Process Modeling, SOA, and Infrastructure Technology*. CRC Press, Boca Raton. ISBN: 978-0-8493-8517-9 (2008)
42. Nakamura, Y., Tsubori, M., Imamura, T., Ono, K.: Model-driven security based on web services security architecture. *IEEE Int. Conf. Serv. Comput.* **1**, 7–15 (2005)
43. Nguyen, N.T., Sobecki, J.: Determination of user interfaces in adaptive systems using a rough classification-based method. *New Gener. Comput.* **1**(4), 377–402 (2006)
44. O'Brien, L., Merson, P., Bass, L.: Quality Attributes for Service-Oriented Architectures. In: *Proceeding of the International Workshop on Systems Development in SOA Environment*. IEEE Computer Society, Washington, DC (2007)
45. Papazoglou, M.P., Georgakopoulos, D.: Service-oriented Computing. *Commun. ACM* **46**(10), 25–28 (2003)
46. Paulheim, H., Probst, F.: Ontology-enhanced user interfaces: a survey. *Int. J. Semant. Web Inf. Syst. (IJSWIS)* **6**(2), 36–59 (2010)
47. Ponnekanti, S.R., Fox, A.: SWORD: a developer toolkit for web service composition. In: *11th World Wide Web Conference*, pp. 97–103 (2002)
48. Prusiewicz, A., Zięba, M.: The Proposal of Service Oriented Data Mining system for solving real-life classification and regression problems. *Technol. Innov. Sustain. IFIF series* **349**, 83–90 (2011)
49. Rao, J., Su X., A Survey of Automated Web Service Composition Methods, *Semantic Web Services and Web Process Composition, SWSWPC*, San Diego, CA, USA, pp. 43–54 (2004)
50. Rygielski, P., Świątek, P.: Graph-fold: an efficient method for complex service execution plan optimization. *Syst. Sci.* **36**(3), 25–32 (2010)
51. Rygielski, P., Tomczak, J.: Context change detection for resource allocation in service-oriented systems. *Lecture Notes in Computer Science. Lect. Notes Artif. Intell.* **6882**, 591–600 (2011)
52. Schmietendorf A., Dumke, R., Reitz, D.: SLA management—challenges in the context of web-service-based infrastructures. In: *Proceeding of the IEEE International Conference on Web Services*, San Diego, California (2004)
53. Shahzad, S.K.: Ontology-based user interface development: user experience elements patterns. *J. Univers. Comput. Sci.* **17**(7), 1078–1088 (2011)
54. SOA Reference Model Technical Committee. *A Reference Model for Service Oriented Architecture*, OASIS (2006)
55. Sobecki, J., Żatuchin, D.: Knowledge and data processing in a process of website quality evaluation. In: Nguyen, N.T., Katarzyniak, R.P., Janiak, A. (eds.) *New challenges in computational collective intelligence*, pp. 51–61. Springer, Heidelberg (2009)
56. Sobecki, J.: Ant colony metaphor applied in user interface recommendation. *New Gener. Comput.* **26**(3), 277–293 (2008)
57. Strunk, A.: QoS-Aware Service Composition: A Survey, In: *Eighth IEEE European Conference on Web Services*, pp. 67–74 (2010)
58. Szpala, A., Rutkowska-Kucharska, A., Drapała, J., Brzostowski, K., Zawadzki, J.: Asymmetry of electromechanical delay (EMD) and torque in the muscles stabilizing spinal column. *Acta Bioeng. Biomech.* **12**(4), 11–18 (2010)



59. Świątek, P., Juszczyszyn, K., Brzostowski, K., Drapała, J., Grzech, A.: Supporting content, context and user awareness in Future Internet applications. *Lect. Notes Comput. Sci.* **7281**, 154–165 (2012)
60. Świątek, P., Rygielski, P., Juszczyszyn, K., Grzech, A.: User assignment and movement prediction in wireless networks. *Cybern. Syst.* **43**(4), 340–353 (2012)
61. Świątek, P., Stelmach, P., Prusiewicz, A., Juszczyszyn, K.: Service composition in knowledge-based SOA systems. *New Gener. Comput.* **30**(2&3), 165–188 (2012)
62. Tari, Z., Bertok, P., Simic, D.: A dynamic label checking approach for information flow control in web services. *Int. J. Web Serv. Res.* **3**(1), 1–28 (2006)
63. Tomczak, J., Cieślińska, K., Pleszkun, M.: Development of service composition by applying ICT service mapping. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) *Computer Networks*, pp. 45–54. Springer, Berlin (2012)
64. Xu, D., Wang, Y., Li, X., Qiu, X.S.: ICT Service Composition Method Based on Service Catalogue Model, In: *Proceeding of AIAI'2010*, pp. 324–328 (2010)
65. Yu, T., Lin, K.-J.: Service selection algorithms for Web services with end-to-end QoS constraints. *Inf. Syst. E-Bus. Manage.* **3**, 103–126 (2005)
66. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for Web services composition. *IEEE Trans. Softw. Eng.* **30**(5), 311–327 (2004)
67. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. *IEEE Trans. Soft. Eng.* **30**(5), 311–327 (2004)
68. Kopel, M., Sobiecki, J.: Web-based user interface for SOA systems enhanced by ontology. In: Aleksander, Z., Kazimierz, C., Andrzej, S. (eds.) *Multimedia and Internet Systems: theory and practice*, pp. 239–247. Springer, Heidelberg (2013)

Advanced SOA Tools and Applications

Ambroszkiewicz, S.; Brzeziński, J.; Cellary, W.; Grzech, A.;  
Zieliński, K. (Eds.)

2014, VII, 323 p. 114 illus., Hardcover

ISBN: 978-3-642-38956-6