MODELING FUNCTIONAL REQUIREMENTS FOR CONFIGURABLE CONTENT- AND CONTEXT-AWARE DYNAMIC SERVICE SELECTION IN BUSINESS PROCESS MODELS[1]

Ales Frece, University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova ulica 17, SI-2000 Maribor, Slovenia /
ViRIS d.o.o., Smartinska cesta 130, SI-1000 Ljubljana, Slovenia, ales.frece@soa.si
Matjaz B. Juric, University of Ljubljana, Faculty of Computer and Information Science, Trzaska cesta 25, SI-1000 Ljubljana, Slovenia, matjaz.juric@fri.uni-lj.si

Abstract

In this article, we propose a meta-model for formal specification of functional requirements for configurable content- and context-aware dynamic service selection in business process models with the objective to enable greater flexibility of the modeled processes. The dynamic service selection can cope with highly dynamic business environments that today's business processes must handle. Modeling functional requirements for dynamic service selection in business process models is not well covered in literature. Some partial solutions exist but none of them allows modeling a complete set of functional requirements for the selection similar to the one we are addressing in this article. Our meta-model enables formal specification of service selection relevant data extracted from service request message, custom configuration data (e.g. thresholds), process and task definition/instance metadata, and service selection rules. The meta-model is configurable and content- and context-aware. Processes leveraging our meta-model can adapt to changing requirements without redesign of the process flow. Proposed meta-model allows users to additionally configure the models at run time (e.g. raising a threshold). Modeling can be divided into roles with different required competences. We implement our meta-model in BPMN 2.0 (Business Process Model and Notation) through specific extensions to the BPMN semantic and diagram elements. By measuring complexity of real-world sample process models we show that using our solution modelers can efficiently model business processes that need to address frequent changing demands. Compared to available alternatives, models using our solution have on average ~13% fewer activities, ~16% fewer control-flow elements and ~22% fewer control paths. By reading ~10% smaller models (by volume) model readers get more flexible process models that capture all functional requirements for the dynamic selection.

Keywords

Business process model, functional requirement, dynamic service selection, content/context awareness, BPMN

1 Introduction

The main objective of this article is to propose a meta-model to formally specify functional requirements for configurable content- and context-aware dynamic service selection in process models to enable greater flexibility of the modeled processes. We also show an implementation of our meta-model through a set of extensions of the de-facto industry standard BPMN 2.0 (Business Process Model and Notation 2.0 [1]).

Business processes must cope with highly dynamic business environments where requirements change rapidly [2]. In an ideal case, processes should be flexible enough to be able to adapt to changing requirements with minimal or no user intervention [2] (e.g. to satisfy the needs of different customers). This objective can be achieved with a dynamic service selection that is configurable at run time through user intervention [3]. Business process models are key artifacts in the development

[1] This article appeared in: Journal of Visual Languages and Computing 23 (2012) 223–247.

of process-oriented information systems [4]. Process models should include all requirements so that stakeholders can check if requirements are in fact correctly satisfied in the final software product [5]. For introduction of dynamic service selection to business process models to increase their flexibility it is necessary to include all requirements for the selection in the models. When we talk about dynamic service selection, we need to efficiently handle functional and non–functional requirements [6].

Modeling non-functional requirements for dynamic service selection such as QoS (Quality of Service) is well covered in literature. Solutions exist for BPMN [7] and WS-BPEL (Web Services Business Process Execution Language [8]) [9,10]. Researchers are also putting effort to support functional requirement modeling of dynamic service selection in BPMN [3] and other process meta-models [2]. However, none of the existing solutions provides modeling support for a complete set of functional requirements for configurable content- and context-aware dynamic service selection, neither in BPMN nor in any other process modeling language or notation. Functional requirements for dynamic service selection include content (e.g. customer payment preferences specified in the order submitted) and context (e.g. which user started the process by submitting the order) [11]. Users may change their preferences upon time and hence these preferences cannot be assessed statically (e.g. configuring new preferred channel for notifications on ordered items being shipped) [12]. Process modeling languages need to provide configurability of the process models to provide an adequate level of adaptation [13]. There is no holistic solution that would address formal specification of content, context, and rules for using them to select the most appropriate services inside process models.

Our solution fills this gap. It is presented as a language-/notation-independent meta-model. We show how to implement it in BPMN. The meta-model can also be implemented in WS-BPEL and other business process languages or notations. Our proposed meta-model enables formal specification of all functional requirements for the configurable content- and context-aware dynamic service selection in business process models. Using the proposed meta-model, processes become more flexible. They become content- and context-aware, so they can adapt to changing requirements with no or minimal user intervention. Our meta-model allows users to configure the models at run time. Modeling all functional requirements for the selection through our meta-model enables execution of the selection inside executable business process models. It also enables stakeholders to formally check if the requirements are correctly satisfied in the final software product before it is used.

This article is organized into nine sections. In Section 2, we explain why dynamic service selection requirements should be included in business process models. We show the importance of content and context data for the selection to work. We illustrate the advantages of our approach on an example process. We define use cases supported by our approach as a basis for evaluation of the results. We describe the problem and define and discuss the meta-model in Section 3. We describe BPMN elements to be extended and the extension approach in Section 4. In Section 5, we implement our meta-model by proposing extensions to BPMN. We present the extensions through XSD (XML Schema Definition [14]) diagrams. In Section 6, we extend BPMN diagram elements to visually represent each of our semantic extensions on BPMN diagrams. We demonstrate usage of the extensions and their visual representations on the example process in Section 7. We look at the effect of our approach on model complexity and model users' experience. We do this through measuring complexity of sample process models using our approach and compare it to alternatives. We discuss the results. We present related work in Section 8 and give conclusions in Section 9.

2 Motivation

2.1 Rationale of dynamic service selection and inclusion of functional requirements into business process models

The ability to efficiently design appropriate information systems and enable them to evolve over their lifetime depends on the extent to which knowledge can be captured [15]. To manage large and complex systems, a systematic engineering approach is required, typically one that includes modeling as an essential design activity [16]. Business process models are key artifacts in the development of process-oriented information systems [4,17]. It is important for the business users is to be able to understand business process models and apply model driven architecture, so that requirements are in fact correctly satisfied in the final software product [5]. Among the major issues in designing distributed information systems are dynamic configuration and continuous system change and evolution [18]. The dynamic nature of business processes impacts the development process. Which service to select, many times depends on the current situation and/or changing requirements. Therefore usage of static service binding in processes is not feasible. Changing requirements require redesign of the process flows [6]. Cases like these can be better handled by dynamic service selection. Dynamic service selection can cope with highly dynamic business environments [2]. It is one of the most often required necessities for managing complexity and unpredictability of business processes [19]. Generally speaking, service selection is always necessary when identical or similar functionality is provided by more than one service provider [20]. In this case, the most appropriate service should be selected. Policy (i.e. formal representation of business rules) usage in the service selection procedure helps service consumers (business processes) to use services which match best to their requirements [11]. A holistic approach on how to include configurable dynamic service selection and its requirements into business process models has not yet been proposed.

2.2 Why content, context and configurability are important for the selection

Service selection can be static (all parameters for the selection are known in the modeling time) or dynamic (some parameters for the selection are known only at run time). Static selection requires a contract, while dynamic selection requires additionally content and context [11,21,22]. Contract contains all the rules (i.e. policies) to select the appropriate service. Contracts most often contain QoS requirements in a form of Service Level Agreements [23,24,25]. Contracts can also contain policies that can be specified in advanced but finally evaluated only at run time. For example, if a customer's order amount is bigger than £500.00, manual credit check is required. These kind of dynamic policies depend on content and context. Content embraces parts of the request message that are important for the service invocation (e.g. order amount). Context embraces information about service invocation (e.g. caller process meta-data). Process modeling languages need to be configurable to provide adequate level of adaptation [13]. Thus, dynamic policies should be configurable at run time. This way additional process deploy cycle is not necessary when introducing changes (e.g. order amount threshold for manual credit check must be raised from £500.00 to £700.00).

2.3 Use cases and added value of proposed approach

Figure 1 shows the beginning and the end of the *Customer order handling* process. This example illustrates particular scenarios, where modeling functional requirements for dynamic service selection can considerably improve the amount of information relevant to stakeholders to be included in the process model. It also illustrates particular scenarios, where dynamic service selection may be even required to achieve some business goals, such as introduction of new features at run time without redesign of the process flow or process redeploy cycle.

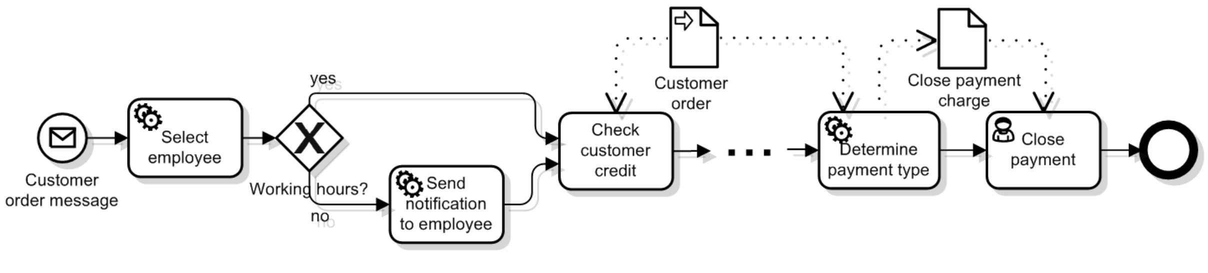

Figure 1: Part of BPMN diagram of *Customer order handling* process

Process is invoked when a customer submits an order. Through the *Select employee* service task, process finds the most suitable employee to handle the order. If the order was submitted outside working hours, the employee is notified about the new order. Notifications are part of the company's policy to encourage employees to work also outside working hours to achieve better business results. Employees have a chance to choose the notification channel that best suits them. Next, customer credit is checked. Currently all customer credits for orders under £500.00 are checked automatically. For all other orders customer's credit must be checked manually by the selected employee. Threshold between these two variants can change in the future. Process continues through different tasks, where ordered items are prepared, packaged and shipped (not shown in Figure 1). Process reaches the *Determine payment type* service task that determines customer's preferred type of payment. Customers have an option to select preferred payment type in their shopping profile. However, each customer can override his/her default behavior for preferred payment type when submitting a new order. For that reason, the *Determine payment type* service task determines the type of closing payment that must be performed in each individual case. Based on this information, the appropriate closing payment is performed by the employee. Credit card and cash post-payment are possible. Someday in the future mobile payment will be supported as well.

Added value of our approach reflects in enablement of a unified solution for modeling functional requirements for the following use cases:

- (A) *Handling specific behavior for individual process participants*: In the example process, notifying employees is achieved through two different services. The appropriate of the two available channels is determined in the dynamic service selection that operates on context data (task instance owner). Rules and data for the selection should be specified in the process model.
- (B) *Differentiation between different approaches to achieve the same business goal*: Customer credit is checked before order handling can continue. If order amount exceeds predefined threshold this task must be done manually by the employee. Otherwise it can be done automatically. This decision is reached through dynamic service selection. Request message data (order amount), configuration data (threshold) and selection rules should be specified as a part of the model.
- (C) *Dynamic configuration of business policies*: Threshold value in the previous use case can change through time. Process model should be configurable at run time without process redeploy cycle. Threshold should be modeled in the process model in a way it can be represented as a dynamically configurable named business parameter at run time.
- (D) *Adapting to changing demands*: The *Close payment* task is used to implement dynamic service selection, where service is selected based on the content (payment type data contained in the *Close payment charge* data object). This requirement should reflect in the model.
- (E) *Introduction of new service candidates to existing selection points*: Selection point is any type of atomic activity (i.e. task) in the process model that has at least two service candidates that can be invoked. Mobile payment will be offered as additional type of payment someday in the future. New service covering new type of payment closing will be introduced to the same selection point in the process at run time. This kind of at run time modifications should not compromise process model validity (they should continue to be "As-Is" models). If desirable by stakeholders, at a point in time "As-Is" models and run-time configuration information should be merged to produce enriched model representing all service alternatives at specific or all selection points.

Without our approach, requirements for the presented use cases would have to be captured outside the process model (for others similar alternatives please see Section 7.2). This would considerably reduce the power of process models to facilitate requirements validation in the final software product [5]. If the process model is used for execution, final product maintenance would be more expensive and more error prone [28]. A holistic approach to support described use cases has not yet been proposed.

3 Meta-model for modeling functional requirements for configurable content- and context-aware dynamic service selection

3.1 Problem description

There is no solution that would enable formal specification of functional requirements for the dynamic service selection use cases (A) to (E). We present a meta-model that will enable formal specification of these requirements. These requirements include rules for selecting the appropriate service and business relevant data used by these rules. We group business relevant data and selection rules into content, context and contract as follows:

- *Content* embraces business relevant data that should be considered in the service selection. This data can be inside service request message and pose changing demands. Business parameters (e.g. threshold) are used to configure process behavior in different scenarios.
- *Context* describes which process and task definition and instance metadata should be considered in the selection. For example, process starter name enables user specific behavior, while parent process name allows differentiation between different approaches to achieve the same business goal (if task T in process P is reached and P was invoked from process P1, service S1 is selected; but if P was invoked from P2, S2 is selected).
- *Contract* contains service selection rules. The most appropriate service is selected based on the content and context data and applying the rules contained in the contract.

We propose different concepts as parts of the meta-model that allow business process modelers to:

- Define which data from request message should be considered in the selection. Using this data is useful for process participants to influence the selection.
- Define custom configuration data that is used in the selection. The process designer can configure service selection in the modeling phase. Configuration can be altered at run time.
- Define which context information should be considered in the selection. Context information is useful to assure process flexibility based on the processes instance specifics.
- Describe, define and reuse service selection rules. These rules describe how all data mentioned above should be processed to select the most appropriate service.

The proposed approach enables modeling different kinds of data and rules that process this data to produce assertions. Assertion is a piece of metadata that describes a requirement or a capability related to a specific domain [26]. Produced assertions must describe each service that qualifies as a candidate for the selection (i.e. service capabilities must meet the requirements). For this to work, a service registry is needed. Each service must be described with the assertions (describing capabilities as its selection metadata) and other information needed to make the actual invocation (e.g. service endpoint).

3.2 Meta-model definition

As a part of the meta-model we define different *concepts*. We leverage a two-stage service selection approach [6] that differentiates between service types and their instances. *Service types* are groupings of similar services in terms of functionality. Formally, [6] denotes a set of web service types:

$$S = \{S_1, \dots, S_n\}, n \in \mathbb{N}$$

In [6], service types are defined with ontologies in OWL (Web Ontology Language [29]). In our approach, we simplify that by defining the service type to be determined by its interface specified in the process model. Using the same interface in two or more semantically different services digress from the expected behavior and advertised service contract (interface). Thus, it is considered as bad practice. Interface definition is supported in all business process languages and notations that enable data modeling. For example, in BPMN 2.0 an interface with a unique name and namespace is described by a WSDL (Web Services Description Language [30]) document. *Service instances* are actual deployed instances of a service type that can be invoked (they are implementing the same interface). Service instances are registered in a registry. We define a set of service instances as:

$$I = \{I_{11}, \dots, I_{1p}, \dots, I_{n1}, \dots, I_{nq}\}, p, q \in \mathbb{N}$$

The mapping of S into I is one-to-many (S_1 has p instances, S_n has q instances). The problem of a particular dynamic service selection is described in [6] as stitching together process flow elements (that define service type) and the desired functionality from the existing services (available service instances for defined type), while satisfying the non-functional requirements. The problem of the dynamic service selection in a particular step of the process in the context of starting points of this article can be adapted to selecting one particular service from the set of service instances of a specific service type that satisfies all the functional requirements. Our meta-model enables inclusion of these functional requirements in process models. We divide these *functional requirements* (R_F) into three sets: *context* (R_{Ctx}), *content* (R_{Ctn}) and *contract* (R_{Ctr}):

$$R_F = R_{Ctx} \cup R_{Ctn} \cup R_{Ctr}$$

3.2.1 Context

Service is always invoked when a specific task in the process is reached. Context embraces information about that service invocation that should be considered in the selection (e.g. caller process name, name of the process instance owner, etc.). For selecting the appropriate service, several task definition or task instance data can be important. For example, service can be selected based on task type (service task, human task, etc.), or actual task instance owner at run time, etc. Information about the caller process and its possible parents can also be important. For example, service can be selected based on process starter, parent process name, etc. All this kind of data represents service selection context. We define context as a union of *process definition* (P_D), *process instance* (P_I), *task definition* (T_D) and *task instance* (T_I) *metadata* sets:

$$R_{Ctx} = P_D \cup P_I \cup T_D \cup T_I$$

Process definition metadata that we define as part of the dynamic service selection context is the data that is included in the process model:

$$P_D = \{name, parentName, topLevelParentName\}$$

Where we define particular elements as follows:

- $name$ ::= Name of the process that started the invocation (e.g. "Customer order handling").
- $parentName$ ::= Name of the parent process (if exists).
- $topLevelParentName$ ::= Name of the top level parent process (if exists).

Parent process name and top level parent process name do not exist if the modeled process has no parent. However, if the modeled process has only one immediate parent, the parent and the top level parent process names have the same values. Process (internal context), parent process (immediate external context) and top level parent process (the broadest external context) names are useful for modeling requirements for a variant of use case (B) described in Section 3.1. Process instance metadata that we define as part of the context is the data that is not included in the process model but can be used as a part of the selection context when generated at run time:

$$P_I = \{starter, owner\}$$

Where we define particular elements as follows:

- $starter$ ::= Unique identification of a user that started the process instance.
- $owner$ ::= Unique identification of a user that is the owner of the process instance (e.g. the supervisor of the user that started the process instance).

Task definition metadata that we define as part of the context is the data that is included in the process model:

$$T_D = \{type, name\}$$

Where we define particular elements as follows:

- $type$ ::= Type of the task (e.g. human task, service call, business rules inferring, script interpretation, etc.). We do not limit the actual set of available types. Actual set of available types depends on the modeling language or notation chosen to implement the meta-model.
- $name$ ::= Name of the task (e.g. "Package ordered items").

Type is useful for modeling requirements that service instance candidates should be implemented with a certain technology (e.g. business rules). Name can be used to support per task definition specific requirements. Task instance metadata, we define as part of the context is the data that is not included in the process model but can be used when generated at run time:

$$T_I = \{owner\}$$

Where:

- $owner$ ::= Unique identification of a user that is the actual owner of the task instance (e.g. the user that should perform the human task instance).

Process instance starter and owner and task instance owner are important for modeling requirements for the use case (A).

3.2.2 Content

Content is the important data for the service selection in the request message (i.e. task input, like purchase amount) or other business relevant data outside the message. Other business relevant data is custom configuration data for the service selection, such as limits (e.g. purchase amount limit), thresholds (e.g. maximum number of repetitions), deadlines (e.g. until next week), etc. We define content as a union of *input data* (D_I) and *configuration data* (D_C) as follows:

$$R_{Ctn} = D_I \cup D_C$$

Input data is the data to be sent from process instance to task instance when dynamic service selection is to be performed (i.e. data in the request message). Input data that we define as part of the dynamic service selection content is the following:

$$D_I = \{d_1, \dots, d_m\}, m \in \mathbb{N}^0$$

Where:

- $d_i, i \in [1,m]$ ::= Reference to particular data value of an elementary type (i.e. number, string, date, etc.) inside the task instance input message. Our approach does not define a concrete set of elementary types. Which types are elementary depends on the language or notation.

These data values are important for modeling requirements for the use case (D). Configuration data includes business relevant configuration parameters (e.g. thresholds, deadlines, etc.). Configuration data that we define as part of the dynamic service selection content is the following:

$$D_C = \{c_1, \dots, c_t\}, t \in \mathbb{N}^0$$

Where:

- $c_i, i \in [1,t]$ ::= Particular configuration parameter with a value of an elementary type.

These configuration parameters are important for modeling requirements for the use case (C).

3.2.3 Contract

The most appropriate service is selected by applying the rules contained in the contract on the content and context. We define contract as a union of a set of possible *implementation assertions* (A) and a set of *selection rules* (F):

$$R_{Ctr} = A \cup F, A = \{a_1, \dots, a_u\}, F = \{f_1, \dots, f_v\}, u, v \in \mathbb{N}$$

Assertions (a_i) are represented as vectors with their *identification* (URI_i) and *value* ($value_i$):

$$a_i = (URI_i, value_i), i \in [1, u], a_i \in A$$

We define functions to get the URI (Uniform Resource Identifier) and the value from the implementation assertion vector respectively:

$$URI(a_i) = URI_i, value(a_i) = value_i$$

We define selection rules (f_j) as functions:

$$f_j: R_{jCtx} \cup R_{jCtn} \rightarrow A_j, f_j \in F, R_{jCtx} \subset R_{Ctx}, R_{jCtn} \subset R_{Ctn}, R_{jCtx} \cup R_{jCtn} \neq \emptyset, A_j \subset A, j \in [1, v]$$

We apply the following restrictions for assuring that R_{Ctx} and R_{Ctn} are minimal:

$$\bigcup_{j=1}^{v} R_{jCtx} = R_{Ctx}, \bigcup_{j=1}^{v} R_{jCtn} = R_{Ctn}$$

Applying each rule (f_j) produces a set of *required* implementation assertions (A_j). Applying all the rules (F) produces a set of all required implementation assertions (R):

$$\bigcup_{j=1}^{v} A_j = R, R \subset A$$

All assertions from R must describe service instance candidate for the selection (I_{xy}). In our case we restrain us from all non-functional assertions such as QoS, as for example described in [7]. Service instances must be listed in a service registry (such as UDDI – Universal Description Discovery and Integration [31] with proper extensions introduced like [32]). Each service instance (I_{xy}) in the registry must be described with the provided set (P_{xy}) of *provided* implementation assertions (p_k):

$$P_{xy} = \{p_1, \ldots, p_w\}, p_k = (URI_k, value_k), w \in \mathbb{N}, k \in [1, w]$$

I_{xy} qualifies as a candidate for the selection when and only when produced required set (R) is a subset of its provided set (P_{xy}). Equivalent definition: every implementation assertion from a required set (a) must be an element of the provided set (P_{xy}):

$$R \subset P_{xy} \Leftrightarrow a \in P_{xy}, \forall a \in R$$

Specific implementation assertion from a required set (a) is an element of the provided set (P_{xy}), when and only when it has the same URI as one of the assertions in the provided set (p_l) and the same value as that particular assertion. The only exception is, when an assertion has an empty value ($\emptyset$). An empty value is always treated as equal to any value. To formulate this:

$$a \in P_{xy} \Leftrightarrow \exists l \in [1..w]:$$

$$URI(a) = URI(p_l) \wedge (value(a) = value(p_l) \vee value(a) = \emptyset \vee value(p_l) = \emptyset), a \in R, p_l \in P_{xy}$$

I_{xy} also has to be of the right type (S_x). The use case (E) is supported by adding new properly described service instances of existing service type to the registry. How exactly the described selection of service instance candidates is implemented, is out of the scope of this article. In [6] it is demonstrated that such two-level selection is feasible. Which service instance will be selected when no or maybe even many candidates are qualified, it is also out of the scope of this article and briefly discussed in Section 3.3. In our meta-model, we represent each selection rule by *intents* (at least one) and *policies* (none, one or more). Intents describe selection rules in an abstract way, in natural language. Intent can *require* other intents to be met (rules to be applied) as its precondition. Policies specify these abstract descriptions in a formal format. Each policy *provides* formal representation of one or more intents.

Intent can have a human-readable name and description. Description holds the details in natural language, what the intent is. Intent can require other intents to be met. For example, intent that states "service must be ready to serve golden customers" can require the intent for silver customers. Intent for golden customers just supplements it with requirements that are different for golden customers. Intents serve to process designers that do not specify exact policies. Inside intents, designers fully describe in natural language, what exactly policies must be. Later, this kind of process model is reviewed by a policy designer. He creates corresponding policy documents. This way, process modeling for dynamic service selection can be divided into roles. This offers greater flexibility of modeling phase and easier iterative process model development. Policy can also have a human-readable name. Policy must name the intents it provides. The last part of the policy is policy content. The proposed approach does not restrict exact specification for policy content (for some possible specifications please refer to Section 7.3).

All meta-model's subsets with functional requirements for a specific service selection need to be attached to a relevant selection point in the business process model. The selection point is any task with interface of S_x in the business process model that maps to at least two I_{xy}. How this attachment is achieved is not part of the meta-model. It depends on the business process modeling language or notation selected for implementation of the proposed meta-model, which we will address in Section 5.

3.3 Discussion on meta-model

Our approach is extensible by design. Any other relevant metadata can be included into any of the context subsets, depending on the business process modeling language or notation selected for the implementation of the meta-model. For example, through this extensibility other roles of process participants can be included as part of the context.

When selecting an appropriate service instance, a possible solution for cases where more than one suitable service instance is found, is to perform an advanced selection algorithm to find the most appropriate one, as for example described in [21,22]. Similar algorithms include but are not limited to selection algorithms based on QoS attributes alone [33,34] and QoS attributes enhanced with trust and reputation management [35]. If there is no suitable service instance found, this probably indicates that some situations were not considered when designing the process. There might also be a system administrative problem (e.g. some services are not yet implemented or properly described in the registry). In these cases a system exception is raised.

From a business perspective, one of the most important aspects of software solutions in dynamic and flexible environments is to be able to configure the solutions [36]. Our proposed approach enables configuration of executable processes models not only at design but also at run time. In the business process deploy phase it is possible to extract custom configuration data expressed with proposed approach (D_C set) from the core process model. This enables manipulation with all of the extracted definitions independently of the rest of the process model. Thus, this supports use case (C). If the process model is an executable one and used for execution, it is the authoritative one. For the stakeholders to use the latest version of the configuration data they have to use the model from the executional process repository. If the process model is not an executable one or is not used for execution, every change in process configuration at run time (e.g. raising the threshold) should reflect in the D_C set of the model and vice versa. This can be achieved with proper synchronization techniques. A more advanced business process platform would enable similar extraction of all meta-model subsets and their similar modification at run time. This would enable modification of policies and all the rest parts of content and context (e.g. it would be possible to add new rules).

4 Overview of BPMN 2.0 and WS-Policy in aspect of meta-model implementation

Wide support is given to the OMG's adoption of the BPMN specification [1] for business process models [37]. Since version 2.0, BPMN formalizes business process execution semantics and diagram exchange (among others irrelevant to the scope of this article) [1]. Selecting BPMN enables us to show feasibility of our proposed approach. We can check the impact of our approach for the stakeholders. With BPMN 2.0 executable business process models we can demonstrate execution semantics of our proposed approach.

BPMN 2.0 introduces extensibility mechanisms to enrich process semantic description and its diagram representation. Elements can be extended in two different ways. BPMN models that use XSD as an exchange format can leverage the `xsd:any` and the `xsd:anyAttribute` elements. If the exchange follows XMI (XML Metadata Interchange [38]) mapping, the `xmi:extension` element can be used. BPMN 2.0 specification provides two-way XSLT transformation between XSD and XMI exchange formats. Consequently, extending BPMN 2.0 through one of the two ways suffices to implement the extensions in the other format. For the definition of extension elements, XSD is preferred [1]. The majority of the BPMN elements are extensible, including `bpmn:process`, `bpmn:task` and all subtypes of the latter. This means that BPMN tasks can be extended to support definition of functional requirements for configurable content- and context-aware dynamic service selection. We specify proposed extensions as additional XSD elements to the BPMN semantics elements. BPMN 2.0 leverages widely adopted standards like XSD for data type definition (data objects), WSDL for interface definition and XPath [39] for formal expressions [1]. We use XPath formal expressions and reference BPMN data objects from the proposed extensions. BPMN 2.0 describes a process modeling conformance and a process execution conformance (among others irrelevant to the scope of this article). The process modeling conformance addresses altering look-and-feel of the diagrams. The original look-and-feel must not be altered in a way modelers could not easily understand the extended diagrams. Permitted extensions include (1) markers or indicators to the existing graphical representation of elements to highlight a specific attribute of the corresponding element, (2) new shapes that do not conflict with existing ones, and (3) use of colors and change of line style that does not conflict with the existing line style to add or stress (new) semantic meaning of the represented elements. The process execution conformance states that execution semantics of existing elements can be enhanced. It can be enhanced in a way core semantics is not altered by the extensions [1].

Since version 2.0 BPMN is also transferable to the software community in a format that is rich enough to be executed [5]. BPMN 2.0 prefers web services as an implementation technology [1]. BPMN executable processes are web service consumers. For these reasons, we leverage WS-Policy (Web Services Policy) for the definition of policies. However, alternative specifications can also be used (for details please refer to Section 7.3). WS-Policy - Framework [26] specification defines an abstract model and an XML-based language for expressing policies of entities in a web services-based system. Policy is basically a set of assertions. Assertion represents a requirement, a capability, or other property of a behavior. WS-Policy is therefore a specification that allows web service consumers and web services themselves to specify their requirements for interaction in an XML based language. We leverage WS-Policy to formally specify policies in the contract. WS-Policy allows assertions relevant to service selection to be expressed and evaluated in a consistent manner [26]. This is very important for our extensions expressing the contract to be executable. We propose BPMN extensions that enable definition of policies inside BPMN process models (internal policies). We also describe how to bring in externally defined policies to the models. Policies are attached to service selection points in the model. We implement the attachment through usage of complementary specification called WS-Policy - Attachment [27]. WS-Policy - Attachment defines mechanisms for associating policies with the subjects to which they apply [27]. A policy subject is an entity (e.g., a web service endpoint) with which a policy can be associated. Policies may be defined as a part of existing metadata about the subject or they may be defined independently and associated through an external binding to the subject [27]. WS-Policy - Attachment defines a mechanism (among others) for associating policies with arbitrary XML elements. We leverage this mechanism to attach internal and external policies to selection points (tasks) in the BPMN process models. This way, service selection of the most appropriate service based on the content and context data and applying the contract (policies) becomes a part of the (executable) business process models and feasible at run time.

5 Implementation of meta-model with BPMN 2.0 extensions and WS-Policy

To implement our proposed meta-model we extend the BPMN 2.0 specification. We leverage the BPMN extension mechanism and describe proposed extension elements through XSD. We use the `bpmn:extensionElements` element to include extension elements into the BPMN models. The `bpmn:extensionElements` element is contained in `bpmn:tBaseElement`, a common parent of all of the elements we extend. We divide extensions into the three groups that correspond to the three subsets of functional requirements defined in the meta-model (context, content and contract, in short CCC). We use `ccc` as the XML namespace prefix of the proposed extensions namespace `http://soa.si/bpmn/extensions/ccc`. Each group of the requirements is covered by its own extension element as shown in Figure 2.

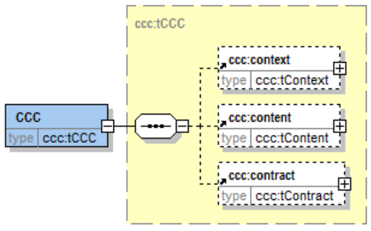

Figure 2: Root extension element with context, content and contract extension elements

We leverage WS-Policy - Framework for the definition of policies in the contract. Policy definitions can be a part of the process model, where they are described and formally specified. Their usage scope is limited to the containing business process model. Policies can also be brought in from definitions outside the process model. They are formally specified in external models and imported to the business process model. This approach offers reuse of policies. Same external policies can be imported into different process models. We leverage WS-Policy - Attachment for attaching policies to service selection points in the process model.

5.1 Context extensions

For the purpose of modeling context requirements (R_{Ctx}) we propose assertions as shown in Figure 3.

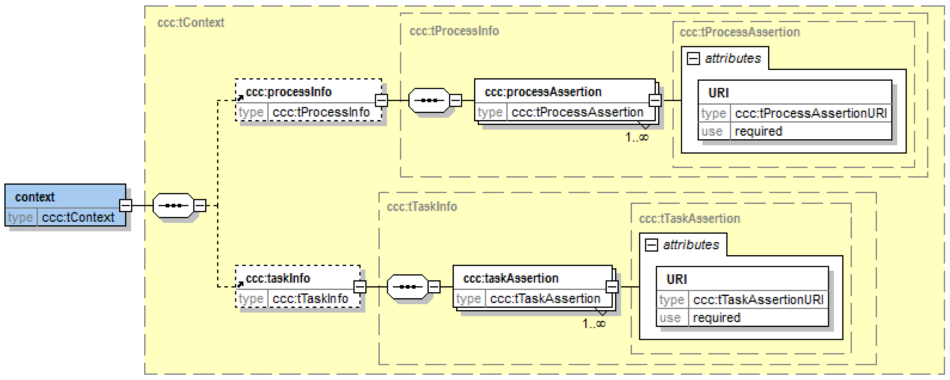

Figure 3: Context extension element with process and task assertions

We can request usage of context data at specific selection points in BPMN model by using process or task assertions. These assertions are proposed extension elements with URIs as shown in Table 1.

Table 1 lists all assertions that represent all elements of all context (R_{Ctx}) subsets (shown in the first column of the table). URI as identification is necessary to reference the assertions from the policies. Process assertion with the `http://soa.si/assertions/context/processDef#name` URI indicates that the process name will be used in the policies.

Table 1: Context assertions identifiers

Set	*Type*	*Identifier (URI)*
P_D	`ccc:tProcessAssertionURI`	`http://soa.si/assertions/context/processDef#name`
	`ccc:tProcessAssertionURI`	`http://soa.si/assertions/context/processDef#parentName`
	`ccc:tProcessAssertionURI`	`http://soa.si/assertions/context/processDef#topLevelParentName`
P_I	`ccc:tProcessAssertionURI`	`http://soa.si/assertions/context/processInstance#starter`
	`ccc:tProcessAssertionURI`	`http://soa.si/assertions/context/processInstance#owner`
T_D	`ccc:tTaskAssertionURI`	`http://soa.si/assertions/context/taskDef#type`
	`ccc:tTaskAssertionURI`	`http://soa.si/assertions/context/taskDef#name`
T_I	`ccc:tTaskAssertionURI`	`http://soa.si/assertions/context/taskInstance#owner`

At run time, context data is derived from different locations as follows:

- The process name (also the parent and the top level parent names) from `bpmn:process/@bpmn:name` and the task name from `bpmn:task/@bpmn:name` (both of type `xsd:string`).
- The process instance starter, the process instance owner and the task instance owner are fully qualified usernames from the identity management system.
- The task type in a form of `xsd:QName`. It can have a value of "`bpmn:tTask`" or one of the `bpmn:tTask` subtypes (`bpmn:tBusinessRuleTask / bpmn:tManualTask / bpmn:tScriptTask / bpmn:tSendTask / bpmn:tServiceTask / bpmn:tUserTask`). If a global task (`bpmn:tGlobalTask`) is used, task type can be global task itself or one of its subtypes (`bpmn:tGlobalBusinessRuleTask / bpmn:tGlobalManualTask / bpmn:tGlobalScriptTask / bpmn:tGlobalUserTask`).

Assertions can be used globally (process level scope) or locally (task level scope). If used globally, that specific assertion is used at every single selection point in that process. On the other hand, local usage indicates that the assertion is used only at that particular selection point.

5.2 Content extensions

Content requirements (R_{Ctn}) are represented as input data assertions (D_I) and custom data assertions (D_C). Request message (i.e. task input) is specified in the `bpmn:ioSpecification/bpmn:dataInput` array that holds elements of type `bpmn:tDataInput`. Data inputs contain reference to the exact structure of the data described through the XSD type definition. With input data assertions, we can specify which data input is relevant to the service selection. We achieve that by specifying data input identification (`bpmn:dataInput/@id`) in the assertion's `ccc:dataInputId` attribute (shown in Figure 4). We can also address specific elements inside that data input if it is of complex type. We achieve that with an XPath expression stored in the `ccc:localXpath` attribute.

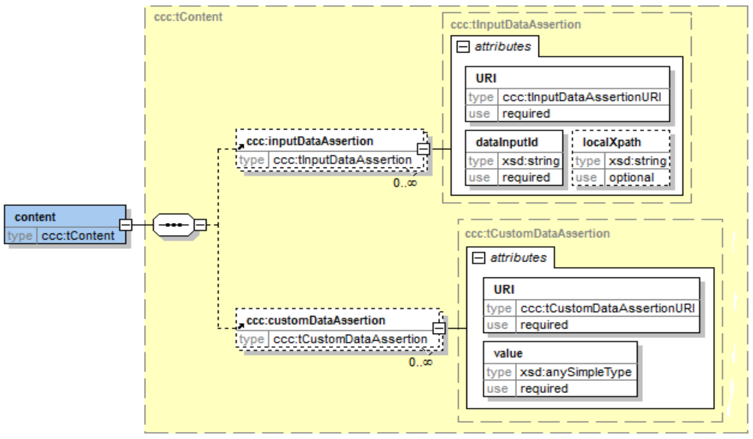

Figure 4: Content extension element with input and custom data assertions

Custom configuration data is specified in the `ccc:value` attribute of a custom data assertion. This attribute can hold any XSD simple type (i.e. an elementary type as described in the meta-model). We specify the URI for unique identification of every input and custom data assertion. The URI is composed of a predefined part (separates input and custom assertions) and per assertion specific name. We provide the URI in conformance with regular expressions presented in Table 2.

Table 2: Content assertions identifiers

Type	*Identifier (URI)*
`ccc:tInputDataAssertionURI`	`http://soa.si/assertions/content/input#([a-zA-Z0-9/])+`
`ccc:tCustomDataAssertionURI`	`http://soa.si/assertions/content/custom#([a-zA-Z0-9/])+`

At run time, input data assertion gets its value from the location in the model that can be addressed with an XPath expression. For data inputs of complex type the `ccc:dataInputId` and the `ccc:localXpath` attributes are specified. In this case, the expression is as shown in Listing 1.

```
/bpmn:definitions/bpmn:process/…/bpmn:task/bpmn:ioSpecification/bpmn:dataInput
[@id="{DATA_INPUT_ID}" and @id=../bpmn:inputSet/bpmn:dataInputRefs]/{LOCAL_XPATH}
```

Listing 1: Location in the BPMN model for input data assertion to get its value from

The expression states that the selected input must have desired *DATA_INPUT_ID* and must be a part of at least one input set. The local XPath expression that is appended at the end, addresses particular simple type value inside the data input. If the data input is of simple type, there is no `ccc:localXpath` attribute defined. In this case, the local XPath part of the expression is omitted from the expression in Listing 1. Optional inputs are specified in the `bpmn:inputSet/bpmn:optionalInputRefs` list. They can be addressed with the XPath expression shown in Listing 2.

```
/bpmn:definitions/bpmn:process/…/bpmn:task/bpmn:ioSpecification/bpmn:dataInput
[@id="{DATA_INPUT_ID}" and @id=../bpmn:inputSet/bpmn:optionalInputRefs]/{LOCAL_XPATH}
```

Listing 2: Location in the BPMN model for optional input data assertion to get its value from

If optional input is unavailable or there is no value present at the location `ccc:localXpath` addresses, assertion gets an empty value (∅). Empty values are denoted with `xsi:nil="true"` attribute from the XSD Instance namespace.

5.3 Contract extensions

Overview of the contract (R_{Ctr}) representation is shown in Figure 5. Selection rules in the contract are represented by intents and policies as described in Section 3.2.3.

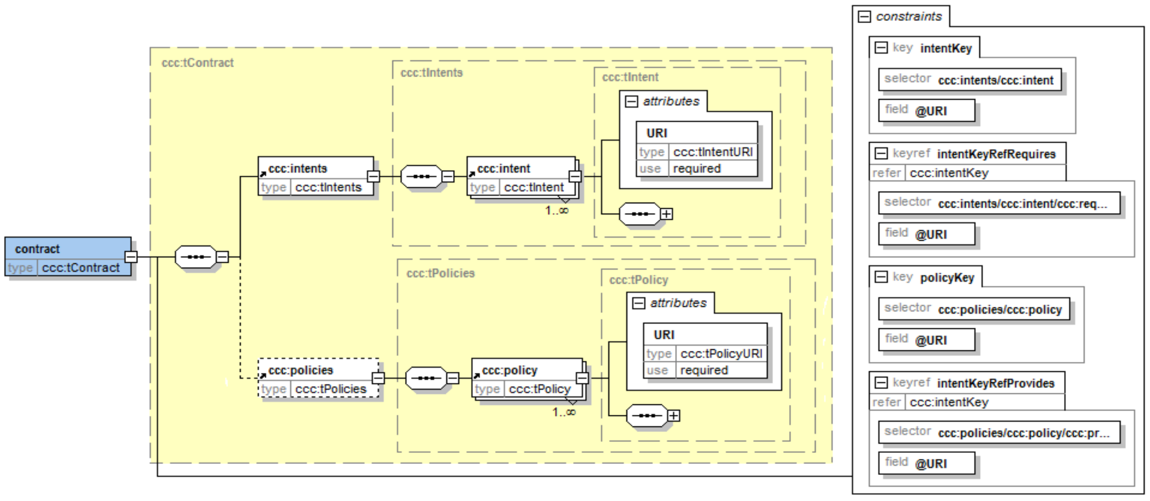

Figure 5: Contract extension element with intents and policies

Each intent and each policy is uniquely identifiable and referencable. When intents are referenced through the URI (relations *requires* and *provides*), reference integrity is enforced through the `xsd:key` and the `xsd:keyref` elements. Each intent has the URI attribute specified in accordance with a regular expression for `ccc:tIntentURI` as shown in Table 3.

Table 3: Contract elements identifiers

Type	***Identifier (URI)***
`ccc:tIntentURI`	`http://soa.si/assertions/contract/intents#([a-zA-Z0-9/])+`
`ccc:tPolicyURI`	`http://soa.si/assertions/contract/policies#([a-zA-Z0-9/])+`
`ccc:tImplementationAssertionURI`	`http://soa.si/assertions/contract/implementation#([a-zA-Z0-9/])+`

Intent can have a human-readable name (shown in Figure 6) to be displayed on the process model diagram as label. Description holds the details in natural language, what the intent is. Intent can require other intents to be met by referencing them through the requires list (the `ccc:requires` element). Each intent is referenced at most one time since more than one reference would mean an unnecessary redundancy. This means no double references are allowed as shown in Figure 6 (the `xsd:unique` constraint).

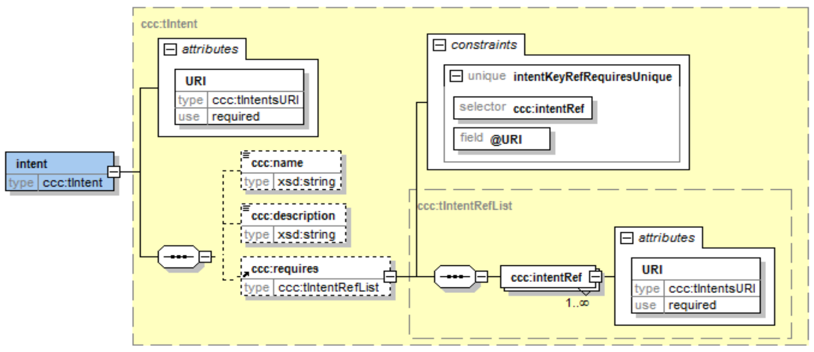

Figure 6: Intent extension element with requires relation

Each policy has URI specified for unique identification inside the process model. URI restriction for policies is as defined with the regular expression for `ccc:tPolicyURI` in Table 3. Policies can also have a human-readable name to be displayed on the process model diagram as label (shown in Figure 7). Policy firstly names the intents it formally specifies (provides). This is achieved by referencing intents in the list of references (the `ccc:provides` element). An unnecessary redundancy of referenced intents is avoided (the `xsd:unique` constraint).

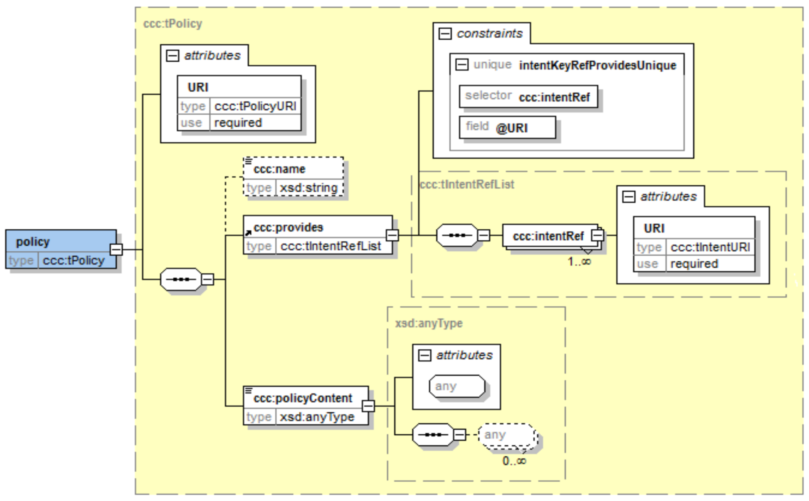

Figure 7: Policy extension element with provides relation and policy content

The last part of the policy is policy content. Proposed extensions do not restrict on specification for policy content, so any element is allowed beyond this point (`xsd:anyType`). We use WS-Policy for definition of policies content. We use policy assertions to express required implementation assertions (*a* from meta-model). These assertions fit in the `xsd:any` location in the `wsp:Policy` element. Each policy assertion states through the `wsp:URI` attribute, which assertion from context or content is considered in the policy evaluation (shown in Figure 8).

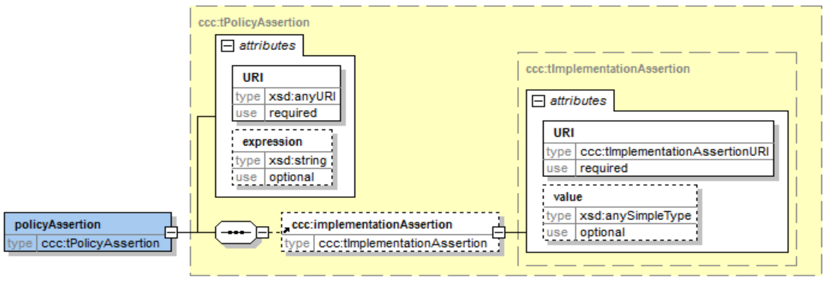

Figure 8: Policy assertion extension element with implementation assertion extension element

When content or context assertion is only referenced through the URI attribute, it is included in the required set (*R* from the meta-model) as an implementation assertion without modifications. For example, policy assertion in Listing 3 states that the `orderType` content input assertion is included into the required set as an implementation assertion including its value regardless what it is.

```
<ccc:policyAssertion URI="http://soa.si/assertions/content/input#orderType"/>
```

Listing 3: Simple policy assertion for including content assertion in the required set

If checking assertion's value is necessary before including the assertion into the required set, we propose usage of formal expressions. XPath is the default language for formal expressions in BPMN 2.0 [1], but others are permitted. So, we recommend using XPath for formal expressions in policy assertions (for alternative languages please refer to Section 7.3). The dot (.) operator (denotes current or "this" node in XPath) is used to reference the assertion's value. For example, to check if a task name is equal to `Task1`, the policy assertion would be as shown in Listing 4.

```
<ccc:policyAssertion URI="http://soa.si/assertions/context/taskDef#Name" expression=". = 'Task1'"/>
```

Listing 4: Policy assertion with simple condition expression

We propose an extension XPath function `ccc:get-assertion(xsd:anyURI)` to enable inclusion of values of other assertions in the expression. This function receives the URI of any of the content or context assertions and returns its value. If an invalid URI is specified the function raises an exception. With this function, comparison of two or more assertion values is possible. For example, the policy assertion in Listing 5 checks if the value of the `Number` input assertion is equal to the value of the `WinningNumber` custom assertion.

```
<ccc:policyAssertion URI="http://soa.si/assertions/content/input#Number"
  expression=". = ccc:get-assertion('http://soa.si/assertions/content/custom#WinningNumber')"/>
```

Listing 5: Policy assertion with condition expression comparing its value to value of another assertion

If the policy assertion expression evaluates to true or no expression is provided, the assertion is included in a non-changed form (same URI and value) as an implementation assertion in the required set. In a case shown in Listing 5, the `Number` implementation assertion with the unchanged value (equals `WinningNumber`) is included into the required set. If the expression evaluates to false, the assertion is not included. Instead of the original assertion, another one as a replacement can be included. This is achieved by specifying an implementation assertion (shown in Figure 8). The implementation assertion has an URI and a value. The value is optional and if not specified, the assertion must only be present in the provided set, regarding its value (as described in the meta-model). The implementation's assertion URI is in accordance with regular expression restrictions for `ccc:tImplementationAssertionURI` defined in Table 3.

Policies that are defined inside a BPMN element can be used only in that element. Scope of these kinds of a policies is limited to only that specific element. It cannot be reused. Policy can also be defined in a separate XML document and can be imported into the BPMN process model. This is achieved through the /`bpmn:definitions/bpmn:import` element. When a policy document is imported, it is ready to be attached to BPMN elements. Policies defined inside process models are also attached. We leverage WS-Policy – Attachment to achieve that. We use the `wsp:PolicyAttachment` element with the `wsp:AppliesTo` child element to attach a policy. Element `wsp:AppliesTo` uses `xsd:anyType` for the definition of the target the policy applies to. For the `wsp:AppliesTo` target definitions we propose usage of an empty extension element `ccc:this`. This element attaches policies to the enclosing task. If the attachment is done on the whole process scope, the attachment target can also be any of the `bpmn:tTask` subtypes. This means that the policy attachment is done on every instance of a `bpmn:tTask` subtype of that type in that particular process. For example, to attach a policy to every service task in the process, we include XML fragment shown in Listing 6 in the contract part of our extensions inside the `bpmn:process` element.

```
<wsp:AppliesTo>
   <bpmn:serviceTask />
</wsp:AppliesTo>
```

Listing 6: Attaching policy to every service task in a process

Additionally, `bpmn:tTask` itself can be used. This means that the referenced policy is attached to every single task in the process, i.e. to the whole process scope. In this way we can model a BPMN business process that has configurable dynamic service selection enabled on every single task in the process. We use the `wsp:PolicyReference` element for referencing the policy we want to attach. Instead of the policy reference we can use the `wsp:Policy` element to attach a policy that is defined only in that particular element. For details on importing, defining, referencing and attaching policies please refer to Section 7.1.

6 Visualization of BPMN 2.0 extensions

BPMN provides a standardized visualization of business processes. In addition to formal restrictions mentioned in Section 4, easy readability must be respected when extending BPMN visualization model [1]. This includes clear distinction between existing and newly introduced or extended diagram elements. Existing BPMN's basic concepts must also be respected (e.g. a sheet of paper symbolizes data). Taking all these aspects into consideration, we propose the following visual extensions.

6.1 Context visualization

We propose visual extensions that change border style of the task symbol to visualize the context requirements (content of the `ccc:context` extension element). This is in accordance with (3). When the `ccc:context/ccc:taskInfo` element is specified on a task, the corresponding task symbol is encircled with an additional dashed border as shown in Figure 9 (a). The surrounding dashed border symbolizes task context (context is "around the task" or "additional information about the task"). This line style is similar to the one of the BPMN symbol for a transaction (a double solid border line). However, the transaction is always a sub-process and `ccc:taskInfo` is always defined on a task, so on the diagram there is enough clear distinction between the two visualizations. When `ccc:context/ccc:processInfo` is specified on a task, it is also represented with an additional dashed border. This time the border is thick as shown in Figure 9 (b). The line thickness is used to distinct between `ccc:taskInfo` and `ccc:processInfo`. In BPMN 2.0 thick lines are used in the global sense (e.g. global tasks have thick border to distinguish them from the local ones). From the task point of view, `ccc:processInfo` embraces the global context (information about the containing process), while the `ccc:taskInfo` embraces the local context (information about the task). When both these elements are defined on a task, both types of dashed border are used as shown in Figure 9 (c). The thick border symbolizing the process context is the outermost one, because processes are of a broader scope than tasks.

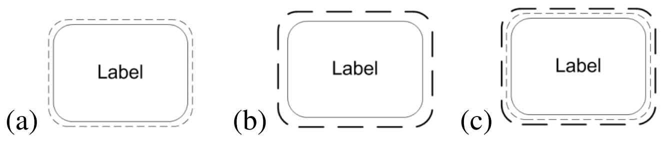

Figure 9: Visualization of context extension element in BPMN diagrams

6.2 Content visualization

To visualize the content requirements (content of the `ccc:content` extension element), we propose inclusion of indicators to the graphical representation of BPMN data object (shown in Figure 10 (a)) to highlight a specific content assertion. This is in accordance with (1). A task can have `ccc:inputDataAssertion` specified only when it already has data input specified through a data object association. This is because `ccc:inputDataAssertion` must reference the input data object indirectly through the data input. The proposed marker consists of four horizontal lines inside the symbol as shown in Figure 10 (b). The lines represent content of the input data object. They are arranged in a way that there remains space at the top of the symbol for a possible BPMN input marker (Figure 10 (c)) and at the bottom for a possible BPMN collection marker (Figure 10 (d)). The proposed marker is used if there is at least one input data assertion defined on a task.

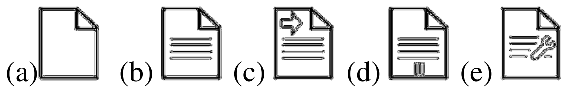

Figure 10: Visualization of content extension element in BPMN diagrams

A task can have `ccc:customDataAssertion` elements specified inside the `ccc:content` extension element. For that purpose, proposed input data assertion marker is altered in a way that a "wrench" symbol is added on the top of the four lines as shown in Figure 10 (e). The "wrench" alone or in a company with other similar symbols for tools is widely accepted symbol in information technology world for the term of customization [40]. Custom data assertions are not bound to input data objects of tasks. They are rather data object themselves. When custom data assertions are defined on a task, they are added to the process diagram. They are connected to the parent task through an association in the same way BPMN data objects are. In this way BPMN 2.0 concepts are preserved. When more than one custom assertion is defined on a single task, only one custom data assertion symbol is used (as for input data assertions).

6.3 Contract visualization

We propose visual extensions that introduce new shapes to visualize the contract requirements (content of the `ccc:contract` extension element). These shapes do not conflict with existing ones. This is in accordance with (2). Every `ccc:contract` extension element has at least one intent defined. In this case, the new "contract" symbol as shown in Figure 11 is used. The symbol represents a contract that is being signed by a pencil. This symbol composition is widely used for visualization of contracts [41]. If not all intents are provided by policies, the symbol in Figure 11 (a) is used. Its dashed border symbolizes that the contract is not complete, since it contains at least one intent without any policy that would provide it. When `ccc:contract` has all intents provided by policies, a similar symbol with a solid line is used as shown in Figure 11 (b). When a contract is referenced from a specific task, an association is created on the process diagram to visualize this reference (as for the custom data assertion). When it is referenced from a process, an association is attached to the BPMN group shape that embraces all of the process flow elements (processes are formed by many flow elements that do not have a common visual container on the BPMN process diagram).

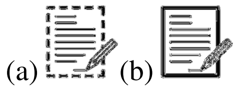

Figure 11: Visualization of contract extension element in BPMN diagrams

7 Proof of concept and discussion on the BPMN 2.0 extensions

We presented the *Customer order handling* process in Section 2.3. Process model can be enriched by inclusion of functional requirements for content- and context-aware dynamic service selection by using our proposed meta-model and corresponding BPMN extensions. Part of the presented process diagram with included functional requirements is shown in Figure 12.

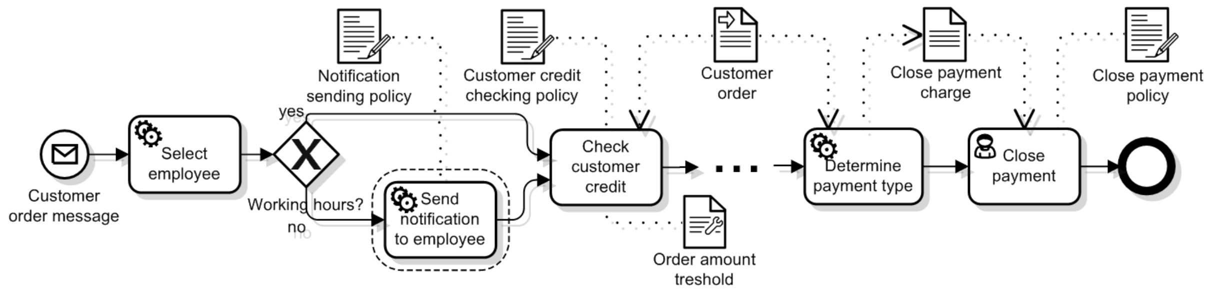

Figure 12: Part of BPMN diagram of *Customer order handling* process using our extensions

When the *Send notification to employee* service task is reached, appropriate service is selected based on the task context (employee's preferred notification type; the task's border changed) and applying the rules in *Notification sending policy* (a symbol added to the diagram). For the *Check customer credit* task, the appropriate service is selected based on the input and the custom content (if the order amount from *Customer order* exceeds *Order amount threshold* or not). For the *Close payment* user task, the appropriate service is selected based on the input content (the payment type contained in *Close payment charge*). *Customer credit checking policy* (a symbol added) operates with the *Customer order* input data assertion that is derived from the existing *Customer order* data object (a visual marker added) and the *Order amount threshold* custom data assertion (a symbol added). The *Close payment charge* data object is extended to the level of an input data assertion (a visual marker added) that is used in *Close payment policy* (a symbol added).

7.1 Realization of presented use cases through proposed extensions

Listing 7 shows top overview of the *Customer order handling* example process model with the proposed extensions in XML. The extensions are imported to the process model in the first `bpmn:import` element. This is necessary for the model to be aware of the extensions definitions [1]. Import type is XSD, because proposed extensions are formally specified in the XSD format. Each particular extension element that will be used directly inside BPMN elements must be announced [1]. For that purpose, the `bpmn:extension` element is used. It declares that our root extension element `ccc:CCC` is mandatory for this model to be complete as we can see in Listing 7 (the `bpmn:mustUnderstand` attribute). The second `bpmn:import` element shows how external policies can be imported into the model. In this case import type is WS-Policy.

```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions xmlns:bpmn="..." xmlns:ccc="http://soa.si/bpmn/extensions/ccc" xmlns:wsp="..." xmlns:ns1="..." xmlns:ns2="..."
    xmlns:tns="..." targetNamespace="..." id="CustomerOrderHandlingProcess" xmlns:xsi="..." xsi:schemaLocation="...">
    <bpmn:import namespace="http://soa.si/bpmn/extensions/ccc" location="..."
        importType="http://www.w3.org/2001/XMLSchema" />
    <bpmn:import namespace="http://soa.si/policies#NotificationSendingPolicy" location="..."
        importType="http://www.w3.org/ns/ws-policy" />
    <bpmn:extension mustUnderstand="true" definition="ccc:CCC" />
  <bpmn:process processType="Private" name="Customer order handling" id="CustomerOrderHandlingPrcs">
    ...  <!-- start and end events, sequence flows, tasks, gateways, data objects ... -->
  </bpmn:process>
  ...  <!--messages, item definitions ... -->
</bpmn:definitions>
```

Listing 7: Top overview of Customer order handling process model with proposed extensions in XML

In the example process model with extensions, adapting to changing demands (D) is covered in the dynamic service selection for the *Close payment* user task. For this task, the selection is made based on the *PaymentType* input data assertion as we can see in Listing 8.

```
...
<bpmn:userTask name="Close payment" id="ClosePaymentUsrTsk">
    <bpmn:extensionElements>
        <ccc:CCC>
            <ccc:content>
                <ccc:inputDataAssertion URI="http://soa.si/assertions/content/input#PaymentType" dataInputId=
                    "ClosePaymentChargeDOInput" localXpath="/ns1:ClosePaymentCharge/ns1:paymentType" />
            </ccc:content>
            <ccc:contract>
                ...    <!-- policy intents, policy name, description ... -->
                    <ccc:policyContent>
                        <wsp:PolicyAttachment>
                            <wsp:AppliesTo>
                                <ccc:this />
                            </wsp:AppliesTo>
                            <wsp:Policy Name="ClosePaymentPolicy">
                                <wsp:All>
                                    <ccc:policyAssertion URI="http://soa.si/assertions/content/input#PaymentType"/>
                                </wsp:All>
                            </wsp:Policy>
                        </wsp:PolicyAttachment>
                    </ccc:policyContent>
                ...
            </ccc:contract>
        </ccc:CCC>
    </bpmn:extensionElements>
    ...    <!—input/output specification (data inputs/outputs, input/output sets), data input/output associations -->
</bpmn:userTask>
...
```

Listing 8: Overview of BPMN model of Close payment user task and its extensions in XML

The input data assertion has a unique identifier (`URI`). The `ccc:dataInputId` attribute states that the assertion's value is gathered from the `ClosePaymentChargeDOInput` task input. This input is of a complex type, so the `ccc:localXpath` attribute specifies exact location of the value. In the contract part, there is one policy defined. This policy is attached to the enclosing task (the `ccc:this` extension element). Inside `ClosePaymentPolicy`, we can see one policy assertion with the same URI as the input data assertion. This means that this input data assertion with its value gathered from the task input is included as an implementation assertion in the required set without modifications. If `/ns1:ClosePaymentCharge/ns1:paymentType` in the task input has a value of `"CREDIT_CARD"`, service described with the implementation assertion `("http://soa.si/assertions/content/input#PaymentType","CREDIT_CARD")` will be selected (i.e. the service that handles credit card payment closings will be selected).

Introduction of new service candidates at run time to existing selection points (E) is always supported on every single selection point in the process model (please refer to Section 3.3 for details). Most notably this is useful for the `Close payment` user task where an additional `Close mobile payment` service can be added at run time. There is no need to change the process model. Specified input data assertion just extracts the payment type from the task input and policy assertion requires that service candidates must be able to handle exactly that type of payment. What is required is to add a new service with the `("http://soa.si/assertions/content/input#PaymentType","MOBILE")` implementation assertion in its provided set to the service registry. When the `Close payment charge` data object will hold `ns1:paymentType` with a value of `"MOBILE"` the new service will be selected without any further human intervention.

In the example process model with extensions, differentiation between different approaches to achieve the same business goal (B) is supported with the dynamic service selection for the `Check customer credit` task. The `http://soa.si/assertions/content/input#OrderAmount` input data assertion declares to extract the order amount from the order summary as we can see in Listing 9.

```
...
<ccc:content>
    <ccc:inputDataAssertion URI="http://soa.si/assertions/content/input#OrderAmount"
        dataInputId="CustomerOrderDOInput" localXpath="/ns2:CustomerOrder/ns2:summary/ns2:amount" />
    <ccc:customDataAssertion URI="http://soa.si/assertions/content/custom#OrderAmountThreshold" value="500" />
</ccc:content>
<ccc:contract>
    ...
        <wsp:Policy Name="CreditCheckingPolicy">
            <wsp:All>
                <ccc:policyAssertion URI="http://soa.si/assertions/content/input#OrderAmount"
                        expression=". &gt; number(ccc:get-assertion('http://soa.si/assertions/content/custom#OrderAmountThreshold'))'
                    <ccc:implementationAssertion URI="http://soa.si/assertions/contract/implementation#Type" value="MANUAL" />
                </ccc:policyAssertion>
                <ccc:policyAssertion URI="http://soa.si/assertions/content/input#OrderAmount"
                        expression="not(...)"> <!-- argument to not() function is the whole above expression -->
                    <ccc:implementationAssertion URI="http://soa.si/assertions/contract/implementation#Type" value="AUTOMATIC" />
                </ccc:policyAssertion>
            </wsp:All>
        </wsp:Policy>
    ...
</ccc:contract>
...
```

Listing 9: Overview of BPMN model of Check customer credit task extensions in XML

The `http://soa.si/assertions/content/custom#OrderAmountThreshold` custom data assertion with a value of 500 is specified. This time the policy assertions have expressions defined. The expression in the first policy assertion compares the value of the `OrderAmount` input assertion with the value of the `OrderAmountThreshold` custom data assertion. Comparison is done by the `"&gt;"` string. It represents an XML entity for the ">" symbol (comparative operator "is greater than"). The value of the `OrderAmount` assertion is addressed with the dot (.) mark. The value of `OrderAmountThreshold` is retrieved with the proposed `ccc:get-assertion()` XPath extension function. If `expression` evaluates to `true`, an implementation assertion `("http://soa.si/assertions/contract/implementation#Type","MANUAL")` is added to the required set. The expression of the second policy assertion has a negation (the XPath `not()` function) of the first expression. This is a replacement for if-else clause in conventional programming languages. When the first expression evaluates to `false`, the second expression evaluates to `true`. In this case the `("http://soa.si/assertions/contract/implementation#Type","AUTOMATIC")` implementation assertion is added to the required set.

Dynamic configuration of business policies (C) is always enabled on every selection point in the process model (please refer to Section 3.3 for details). Most notably, this is useful for the `Check customer credit` service task, where the `OrderAmountThreshold` custom data assertion can be changed at run time (e.g. raised from £500.00 to £700.00).

Handling specific behavior for individual process participants (A) in the example process model with extensions is covered with the dynamic service selection for the `Send notification to employee` service task. `NotificationSendingPolicy` was imported to the process model as shown at the beginning of this Section. To reference this policy from the contract, the `wsp:PolicyReference` element is used as we can see in Listing 10. The `wsp:PolicyReference` element is used instead of the `wsp:Policy` element that contains the policy definition. Attachment of the policy is achieved by using the `ccc:this` extension element. Imported policy `http://soa.si/policies#NotificationSendingPolicy` expects the task instance owner identification when making the selection (each employee has its own preferred notification channel). The preferred channel is resolved based on the task instance owner identification from the identity management system and his/her properties. The identity management system must be already integrated with the process runtime engine. This is usually already the case, because only authenticated users can administer process instances and work on tasks based on their roles. To use the task instance owner identification in the policy, a task assertion with the task instance owner identification is included in the context element as we can see in Listing 10.

```
...
<ccc:context>
    <ccc:taskInfo>
        <ccc:taskAssertion URI="http://soa.si/assertions/context/taskInstance#owner" />
    </ccc:taskInfo>
</ccc:context>
<ccc:contract>
    ...
        <ccc:policyContent>
            <wsp:PolicyAttachment>
                <wsp:AppliesTo>
                    <ccc:this />
                </wsp:AppliesTo>
                <wsp:PolicyReference URI="http://soa.si/policies#NotificationSendingPolicy" />
            </wsp:PolicyAttachment>
        </ccc:policyContent>
    ...
</ccc:contract>
...
```

Listing 10: Overview of BPMN model of Send notification to employee service task extensions in XML

On the model of the *Customer order handling* process, we showed how proposed BPMN extensions which implement our proposed meta-model can be included into business process models and how they work. Presented use cases and described dynamic service selection value-added scenarios are a small portion of what can be carried out by the approach proposed.

7.2 Model complexity

The only alternative to our approach, as presented in this article, is to model all service alternatives explicitly inside the process model. This kind of model (like the *Customer order handling* process shown in Figure 13) supports use cases (B) and (D) only. In [3] authors have combined business rules and BPMN through new "rule" gateway as a BPMN extension. Using this extension gateway instead of BPMN exclusive gateway, use case (C) can be supported as well. However, explicit modeling of service alternatives makes support for use case (E) impossible. Use case (A) is also not supported in BPMN since actual users cannot be modeled (only roles can). To the best of our knowledge this use case is not supported in any other solution involving BPMN. Use cases (A) to (E) are partly supported in other solutions that leverage other modeling languages or notations (please refer to Section 7.3 for details).

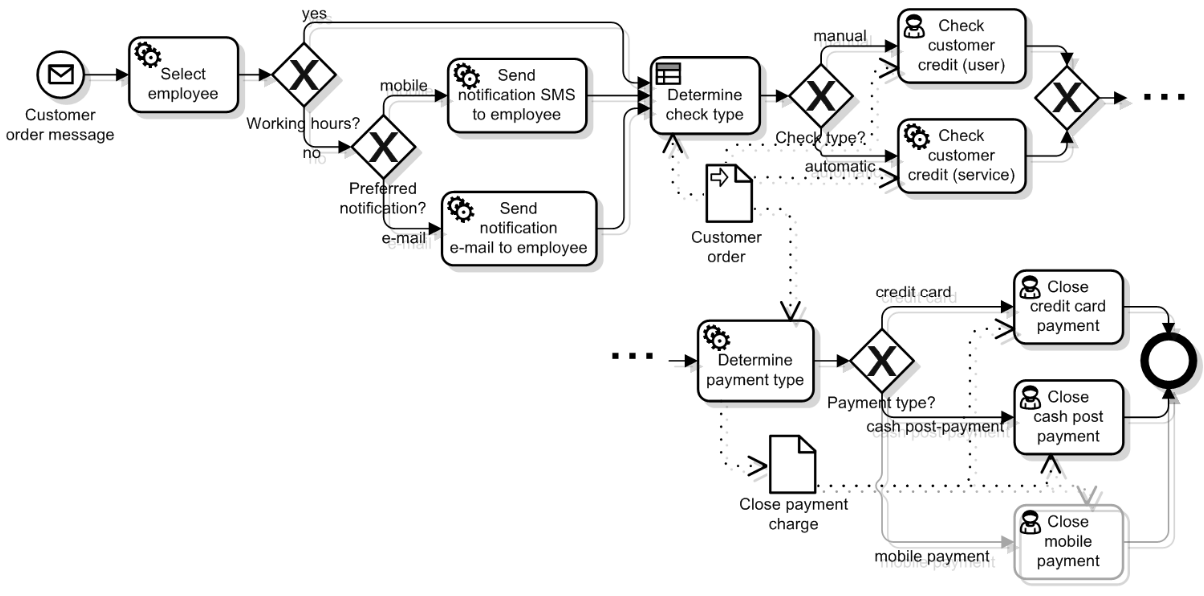

Figure 13: *Customer order handling* process model with modeled service alternatives

With explicitly modeled service alternatives, there are three more exclusive gateways in the model representing a service selection point each. Using extensions presented in [3] at these selection points (gateways) it enables support for use case (C). There is the *Determine check type* business rule task with rules that replaces *Customer credit checking policy*. Use case (E) is not supported. When enabling an additional payment type (mobile payment), the process model with modeled service alternatives needs to be modified. Consequentially, executable process models needs to be redeployed. With our approach model does not need to be changed.

How do newly introduced extensions influence model complexity comparing to the available alternatives? This question is very important for the stakeholders and other users of the models. A significantly bigger model complexity would indicate a reasonable doubt in the usability of the extensions. We answer the question with measuring and comparing complexity of both alternatives with different metrics. A good overview of the appropriate metrics is presented in [42]. Traditional software metric Lines of Code is adapted to the business processes specifics. It is substituted with the following metrics:

- Number of activities in a process metric (NOA) counts activities in the process. It can show a bad process design when processes are not properly structured.
- Number of activities and control-flow elements in a process metric (NOAC) is basically NOA that takes process control-flow nodes (gateways in BPMN) into account. NOAC is used in well-structured processes (each splitting of the process flow with control-flow node must be closed with the corresponding control-flow node).
- Number of activities, joins, and splits in a process metric (NOAJS) is NOAC used for processes that are not well-structured (there is at least one splitting of process flow that is not closed).

These metrics are widely used [43]. Others still simple, but even more meaningful metrics measuring business process complexity overviewed in [42] are:

- McCabe's cyclomatic complexity metric (MCC) that measures the number of control paths through the process. It is independent of language and language format [44]. MCC is defined to be $e - n + 2$, where e is the number of edges and n number of nodes in the control flow graph. Edges represent transfer of control between nodes (number of sequence flow elements in BPMN). Nodes represent computational statements or expressions (BPMN activities, events and gateways).
- Control-flow complexity metric (CFC) that is on MCC based metric which takes into consideration differences between gateways [45]. It is especially meant to be used to measure mental states that need be considered when a designer is modeling a process. It is defined to be $C_{XOR} + C_{OR} + C_{AND}$, where C_{XOR} is complexity of XOR-split (equals to number of branches that can be taken), C_{OR} complexity of OR-split (equals to number of states that may arise from the execution of the split), and C_{AND} complexity of AND-split (always equals 1).
- Halstead-based process complexity metrics (HPC) that are based on a set of primitive Halstead measures [46]. They include measuring data complexity while other metrics above do not. The primitive measures are n_1 for number of unique flow elements (activities, gateways, events, sequence flows), n_2 for unique data objects, N_1 for number of n_1 elements, and N_2 for number of n_2 elements. Metrics based on these primitive measures calculate as follows:
  - Halstead-based process length metric (HPC-N) is defined to be $n_1 * \log_2(n_1) + n_2 * \log_2(n_2)$.
  - Halstead-based process volume metric (HPC-V) is defined to be $(N_1+N_2) * \log_2(n_1+n_2)$.
  - Halstead-based process difficulty metric (HPC-D) is defined to be $(n_1/2) * (N_2/n_2)$.

We have measured the complexity of the example process model with modeled service alternatives and its version using our extensions. Results are gathered in Table 4. We have considered both models as if mobile payment is already supported. In the case of NOAC, we have considered that both models are structured even though not all XOR gateways are explicitly closed (the ones that are not are closed implicitly). This is also the reason NOAJS metric is not measured.

Table 4: Complexity of example process and its version using proposed extensions

Metric	*Example process*	*With extensions*	*Difference (in % of example process)*
NOA	10	5	-50
NOAC	14	6	-57
MCC[a]	6	2	-67
CFC[b]	9	2	**-78**
HPC-N[c]	21,7	35,2	62
HPC-V[c]	126,8	81,4	-36
HPC-D[c]	3,5	3,5	0

[a] e is 21 for the example process and 8 for the one using the extensions; n is 17 and 8.

[b] C_{XOR} is 9 for the example process and 2 for the one using the extensions; C_{OR} and C_{AND} are 0 for all cases

[c] n_1 is 7 for both of the processes; n_2 is 2 for the example process and 6 for the one using the extensions; N_1 is 38 and 16; N_2 is 2 and 6.

We can see that in the example process model using extensions the complexity is smaller by 50% and more for majority of the metrics compared to the process with explicitly modeled service alternatives. Model using extensions has only half the activities (NOA) and even fewer control-flow elements (NOAC). It has only a third of the control paths (MCC) and less than a quarter of control-flows (CFC). It is more than a half "longer" (HPC-N). This is due to increased value of n_2 Halstead primitive measure that counts unique data objects. This value is maximized since we have used all newly introduced extensions to demonstrate their added value (e.g. input and custom data assertions). It has less than two thirds of the volume of the model with explicitly modeled alternatives (HPC-V). Models have exactly the same difficulty (HPC-D).

We have also measured complexities of real world process models. We have a repository of process models from different projects, mainly carried out in the telecom and electro distribution areas. We have chosen two sets of process models from each of these two industries. We have measured complexities of 22 process models with more than 400 activities. We have calculated complexity difference for all models in the same way as for the example process model above. Aggregated results are shown in Table 5.

Table 5: Complexity differences for real world process models[a]

Metric	*Telecom customer orders*	*Telecom service provisioning*	*Electro location acts and directives*	*Electro distribution client lifecycle*	*Average[b] (by metric)*
NOA	-14%	**-19%**	-8.0%	-11%	**-13%**
NOAJS	-17%	**-20%**	-12%	-13%	**-16%**
MCC	-25%	**-39%**	-4.2%	-20%	**-22%**
CFC	**-22%**	-8.5%	-8.1%	-4.2%	**-11%**
HPC-N	4.8%	5.7%	4.2%	**2.6%**	**4.3%**
HPC-V	-11%	**-18%**	-2.0%	-8.5%	**-10%**
HPC-D	**-1.2%**	5.3%	0.39%	5.9%	**2.6%**
Average[c] (by project)	-12%	**-13%**	-4.2%	-6.9%	**-9.2%** [d]

[a] Difference is calculated as: ("our extensions" – "alternative") / "alternative".

[b] Based on rounded values. Data from individual projects used in calculation is not weighted.

[c] Based on rounded values. Data for individual metrics used in calculation is not weighted.

[d] Based on rounded values of all metrics in all projects. Data used in calculation is not weighted.

We have achieved the best results regarding complexity of the models in the second telecom project. This is due to the nature of business processes in this set where diverse set of post-paid services are being configured and activated through different service instances of the same service type (depending on the subscription plan selected). We have achieved the least promising results in the first electro project. This is due to the project's specific integration of two distinct systems. One of the systems is a legacy system containing older data and the other is a newer system containing all the new data. The only points in the process models where we were able to introduce dynamic service selection are decision points where selection of the appropriate system for reading or writing takes place.

Using the proposed extensions for modeling functional requirements for dynamic service selection instead of explicitly modeled service alternatives positively affects most of the metrics in all of the process sets. Models using extensions have on average ~13% fewer activities (NOA) and ~16% fewer control-flow elements (NOAJS). They have ~22% fewer control paths (MCC). This fact is very important for isolated unit testing of the executable processes. It implies that around three quarters of effort is enough to cover all of the control paths while testing the processes in isolation. Process models using the extensions have ~11% fewer control-flows (CFC). This means that a process modeler must put only ~89% of initial effort to comprehend all the states that are possible inside processes. Models with extensions are less than 5% "longer" (HPC-N). The reason for the increase of HPC-N is similar as in the example process: the number of unique data objects (n_2) is increased because extended models use newly introduced extensions (e.g. data assertions). However, increase is smaller than with the example process because real world processes use a more variegated set of unique flow elements (n_1), which reduces influence of n_2. This means that models with extensions have almost insignificantly bigger variegated set of information that must be thought over by a model reader. Models with extensions have ~10% less volume (HPC-V). This implies that a process model reader needs to absorb only ~90% of information to understand the models with extensions. This fact is very important for the stakeholders. They have to know a little more variegated set of symbols, but by absorbing less information they get process models which capture functional requirements for dynamic service selection. Models using our extensions are less than 3% more "difficult" (HPC-D). This is due to an increased value of n_2. However, its influence is limited by an increased number of n_2 elements used (N_2). A smaller value of HPC-D (~3%) compared to HPC-N (~5%) implies our extensions are on average used more often than an average BPMN construct. This implies that there are no major differences between effort for creating process models with or without extensions (not including mind effort). On average complexity of process models from all of the sets is smaller by almost 10% when using our extensions instead of alternatives.

7.3 Discussion on proposed extensions

Stakeholders can see alternative service instances on the BPMN business process diagram using our extensions. For example, a stakeholder sees *Send notification to employee* task and is interested in knowing that notification can be sent by e-mail or SMS. In the model, he can only see all possible implementation assertions that can be included into a required set. Without knowing the provided sets of service instances in the registry he cannot determine selection candidates. To address this requirement, service selection emulation is provided on the BPMN diagram. The selection emulator has access to the service registry to match the set of possible required assertions (model) with a provided set of assertions (registry). Stakeholder can then expand the task with dynamic selection enabled (like sub-process in BPMN). When he does that, service candidates are explicitly rendered on a process diagram. An exclusive gateway with parallel alternatives is encircled with a group symbol (with a name of the expanded task) to differentiate from ordinary gateways.

As described earlier in the article, BPMN 2.0 prefers web services as implementation technology. It demands it for process execution conformance. It prefers XPath for formal expressions [1]. Our solution respects these guidelines. We use XPath for formal expressions inside policies. BPMN 2.0 allows usage of alternative formal expression languages. This is seamlessly reflected in our extensions. We use WS-Policy for definition of selection policies. The extensions we propose also allow usage of alternative policy definition languages. Policies defined in other definition languages can be used inside the `ccc:policyContent` element in the contract. Attachment can be achieved in the same way we have proposed (using WS-Policy Attachment). Instead of WS-Policy, other policy content definition languages based on XML can be used (like XACML - eXtensible Access Control Markup Language [47]). The alternative to policies is business rules. XML languages for definition of business rules could also be used in the contract (like XRules [48]). Alternatively, external business rules definitions can be imported to the model. Other non-XML formats can be used. For example

OMG's specifications PRR (Production Rule Representation [49]) or SBVR (Semantics of Business Vocabulary and Business Rules [50]) can be used to define the contract. The best way to include these contracts is by import to the process model. Another way to include business rules into the contract is by referencing existing business rules inside the business rules engines.

8 Related work

To the best of our knowledge, an approach to include functional requirements for configurable content- and context-aware dynamic service selection into business process models comparable to the one presented in this paper has not yet been proposed.

Several authors have identified the benefits of dynamic service selection in business processes. Additionally, some have identified benefits of inclusion of its requirements into business process models. Akram et al. [19] have identified dynamic service selection as one of six most often required necessities for managing complexity and unpredictability of workflows. They have evaluated WS-BPEL for these necessities. They have discovered that the only dynamism regarding service selection supported in WS-BPEL is to invoke alternative services in exception handler if primary service fails. This means WS-BPEL does not support any of the use cases (A) to (E) supported in our approach. Casati and Shan [2] have showed that dynamic service selection can cope with highly dynamic business environments. Service selection rules in their solution called eFlow are defined in a service delivery platform specific language and are simply transferred to the platform for execution. eFlow does not understand the selection rules and the selection rules are unaware of the process definition and instance data. Therefore eFlow does not support use cases (A), (B) and (D). Recker et al. [13] have identified the need for process modeling languages to be configurable to provide an adequate level of adaptation. They have presented configurable process models that support human decisions for the transformation of reference models from configuration time to build time. In their solution, a user can individualize the model by selecting from alternative options before model instances will be derived from it. Their solution does not support any of the use cases (A) to (E). However, use cases (B) and (D) could be conditionally supported with their approach: one process model that is individualized to all possible variants before run time. The condition is that all the data to make the selection must be present before process instance is started, so that the right variant of the process is selected. Milanovic et al. [3] have combined business rules and BPMN through new "rule" gateway as a BPMN extension. This approach does not provide dynamic service selection per se but rather controls flow of the process, because all service alternatives must be modeled explicitly in the process model. Their solution is an alternative to our approach regarding the use cases (B), (C) and (D). However, it does not support use cases (A) (no access to process instance metadata: actual users cannot be modeled, only roles can) and (E) (explicitly modeled service alternatives). Additionally, process models using their approach are more complex (please refer to Section 7.2 for details). Above solutions only partly address use cases (A) to (E) which we have fully addressed with our proposed approach.

Other authors have identified benefits of inclusion of dynamic service selection non-functional requirements into (extended) business process models. Hwang et al. [10] have proposed dynamic service selection to WS-BPEL in a failure-prone environment. They are using composite service as a mediator between process and services. Composite service contains selection rules ensuring reliability and delegates invocations from process to most reliable services. Composite service is not part of a process model but is rather another model itself, so requirements for the selection are not part of the core process model. Ardagna and Pernici [9] have proposed a meta-model for QoS requirements inclusion into business process models. They have specified an algorithm and a negotiation protocol for selecting best set of services available at run time based on predefined QoS constraints in the process model. They have used WS-BPEL as an operational language for process implementation. Saeedi et al. [7] have extended BPMN to incorporate time, cost and reliability quality requirements. Requirements can be specified on entire process or on specific activities. Extensions are introduced on two different levels in a similar way as ours. Solution includes visual representation of proposed extensions as ours does. All above presented solutions address inclusion of non-functional requirements for a dynamic service selection into the process models. None of them addresses inclusion of functional requirements. Our approach is therefore complementary to these solutions.

Some authors have addressed content- and context-aware dynamic service selection through contract specification. Laliwala et al. [11] have described architecture and provided implementation results for dynamic service selection based on event (context), policy (contract) and semantics (content). Its focus is the procedure of actual service selection and corresponding architecture to perform it. They suppose that requirements for the selection will be captured by business processes. They stop at this point. A similar approach is described by Goering and Louie [22]. They present Dynamic assembler component that uses content, context and contract to dynamically select service endpoints. Dynamic assembler is designed to be part of WSPA (Web Services Platform Architecture [51]) realized SOA (Service Oriented Architecture [52]). However, their solution does not describe where requirements for content and context are captured. Content and context must be filled through a programming code as input for their solution. Our approach does exactly what is missing in the above solutions to be able to specify functional requirements for the selection. We propose a formal specification of requirements inside business process models. Our approach is integrable with architecture of Laliwala et al. [11] and Goering and Louie [22]. We expect this is also true for other SOA and its de-facto realization technique WSPA solutions (such as other extensions of web service-oriented BPMN).

Momotko and Nowicki [53] have addressed visualization of BPMN process instances. Their solution helps process users to better understand execution of process instances. They have proposed visual extensions to BPMN to present process model and the current state of process instance in a coherent way. These extensions can be extended further to visualize the details of a dynamic service selection. We believe that our solution would benefit greatly if combined with the above solution. For example, users could see which services qualified as candidates, which did not, and which one was selected.

9 Conclusion

In this article, we have proposed a meta-model for formal specification of functional requirements for configurable content- and context-aware dynamic service selection in business process models to enable greater flexibility of the modeled processes. We have implemented our meta-model in BPMN 2.0. For that purpose, we have proposed specific extensions to the BPMN 2.0 semantic and diagram elements. Proposed extensions for modeling content, context and contract enable full definition of functional requirements for configurable dynamic service selection inside (executable) BPMN business processes. There is no meta-model, formal or informal business process language or notation that would support such a holistic solution for modeling of functional requirements for dynamic service selection. No other solution offers formal specification of all these requirements neither for diagram representation nor execution.

Our solution leverages content- and context-aware dynamic service selection. By using this solution, modeled processes become more flexible. They can adapt to changing requirements with minimal or no user intervention because they become aware of the content and context. They can handle specific behavior for individual process participants. They can achieve the same business goal through different approaches and can adapt to users changing demands. User intervention at run time is supported through meta-model configurability. Formal specification of all the functional requirements for the selection through our meta-model enables execution of the selection inside executable business process models. New service selection candidates can be introduced at run time. Configuration parameters can be altered (e.g. threshold can be raised). All this is achieved without redesign of the process flow and process redeploy. Using models with our solution stakeholders can check if the requirements are correctly satisfied in the final software product.

Our meta-model enables specification of data that is observed when selecting services. It allows specification of data that is extracted from service request messages. It also allows specification of other custom configuration data; such are limits, thresholds, deadlines, etc. It enables specification of process definition/instance and task definition/instance metadata to be used in selection rules. It also enables definition of service selection rules (contract). The proposed meta-model allows informal (to be used by stakeholders) and also formal definition of contract (understandable to computer). This enables iterative development of the models. Modeling can be divided to roles with different required competences (only describing the policies through intents or formality specifying them through policies). Our BPMN extensions implementing the meta-model allow specification of complex formal expressions to address complex scenarios of service selection.

With our approach, modelers can efficiently model business processes that need to address frequent changing demands. We have proved that by measuring complexity of real-world process models. We have compared the complexity of process models using our solution with available alternatives. We have shown that in most cases model complexity is smaller when using our solution. Models using our solution have on average ~13% fewer activities and ~16% fewer control-flow elements. The model reader must put only ~89% of initial effort to comprehend all the states that are possible inside processes. Only ~78% of initial effort is needed for isolated testing of executable processes. There are no major differences between the efforts needed to create the models using our solution or models using the alternatives. Compared to alternatives, model readers have to know ~5% more variegated set of symbols, but by reading ~10% smaller models (by volume) they get more flexible process models (especially compared to initial models) which capture all functional requirements for the dynamic selection that assures that flexibility.

Acknowledgements

Operation part financed by the European Union, European Social Fund. Operation implemented in the framework of the Operational Programme for Human Resources Development for the Period 2007-2013, Priority axis 1: Promoting entrepreneurship and adaptability, Main type of activity 1.1.: Experts and researchers for competitive enterprises.

References

[1] OMG, Business Process Model and Notation (BPMN) Version 2.0, 2011, http://www.omg.org/spec/BPMN/, accessed on: 16 March 2011.
[2] F. Casati, M.-C. Shan, Dynamic and Adaptive Composition of E-services, Information Systems, 26/3 (2001) 143-163.
[3] M. Milanovic, D. Gasevic, G. Wagner, Combining Rules and Activities for Modeling Service-Based Business Processes, in: L. F. Pires, M. Steen (Eds.), 12th Enterprise Distributed Object Computing Conference Workshops, IEEE, Munich, 2008, pp. 11-22.
[4] H. A. Reijers, J. Mendling, A Study into the Factors that Influence the Understandability of Business Process Models, IEEE Transactions on Systems, 41/3 (2011) 449-462.
[5] A. Fouad, K. Phalp, J. Mathenge Kanyaru, S. Jeary, Embedding Requirements within Model-Driven Architecture, Software Qual Journal, 9/2 (2010) 411-430.
[6] V. Agarwal, G. Chafle, K. Dasgupta, N. Karnik, A. Kumar, S. Mittal, B. Srivastava, Synthy: A System for End to End Composition of Web Services, Journal of Web Semantics, 3/4 (2005) 311-339.
[7] K. Saeedi, L. Zhao, P.R.F. Sampaio, Extending BPMN for Supporting Customer-Facing Service Quality Requirements, in: S. S. Yau, E. Damiani (Eds.), IEEE International Conference on Web Services, IEEE, Miami, 2010, pp. 616–623.
[8] OASIS, Web Services Business Process Execution Language Version 2.0, 2007, http://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm, accessed on: 16 March 2011.
[9] D. Ardagna, B. Pernici, Adaptive Service Composition in Flexible Processes, IEEE Transactions on Software Engineering, 33/6 (2007) 369-384.
[10] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, C.-H. Chen, Dynamic Web Service Selection for Reliable Web Service Composition, IEEE Transactions on Services Computing, 1/2 (2008) 104-116.
[11] Z. Laliwala, A. Desai, S. Chaudhary, A. Allam, Why Context, Content and Contract are Key for Dynamic Service Selection, in: L. J. Zhang, P. Hofmann (Eds.), IEEE Congress on Services, IEEE, Honolulu, 2008, pp. 281-288.
[12] X. Li, S.-K. Chang, User Profiling in the Chronobot/Virtual Classroom System, International Journal of Software Engineering and Knowledge Engineering, 17/2 (2007) 191-206.
[13] J. Recker, M. Rosemann, W.V.D. Aalst, M. Jansen-Vullers, Configurable Reference Modeling Languages, in: M. Potter, J. Neidig, S. Reed, K. Roth, and M. Goldberg (Eds.), Reference Modeling for Business Systems Analysis, Idea Group Publishing, Hershey, 2006, pp. 22-46.

[14] W3C, XML Schema, 2001, http://www.w3.org/XML/Schema, accessed on: 16 March 2011.
[15] S.J. Greenspan, J. Mylopoulos, A. Borgida, Capturing More World Knowledge in the Requirements Specification, in: Y. Ohno, V. R. Basili, H. Enomoto, K. Kobayashi, R. T. Yeh (Eds.), Proceedings of the 6th International Conference on Software Engineering, IEEE Computer Society Press, Tokyo, 1982.
[16] A. Malizia, P. Bottoni, S. Levialdi, Generating Collaborative Systems for Digital Libraries: a Model-Driven Approach, Information Technology and Libraries, 29/4 (2010) 171-186.
[17] A. Awad, M. Weidlich, M. Weske, Visually Specifying Compliance Rules and Explaining Their Violations for Business Processes, Journal of Visual Languages & Computing, 22/1 (2011) 30-55.
[18] Y. Deng, S.-K. Chang, A Framework for the Modeling and Prototyping of Distributed Information Systems, International Journal of Software Engineering and Knowledge Engineering, 1/3 (1991) 203-226.
[19] A. Akram, D. Meredith, R. Allan, Evaluation of BPEL to Scientific Workflows, in: S. J. Turner, S. Matsuoka (Eds.), Sixth IEEE International Symposium on Cluster Computing and the Grid, IEEE, Singapore, 2006, pp. 269-274.
[20] M.W. Mutka, L.M. Ni, Service Discovery in Pervasive Computing Environments, IEEE Pervasive Computing, 4/4 (2005) 81-90.
[21] S. Cuddy, M. Katchabaw, H. Lutfiyya, Context-aware Service Selection Based on Dynamic and Static Service Attributes, in: F. Leymann, M.-C. Shan (Eds.), IEEE International Conference on Wireless And Mobile Computing, Networking And Communications, IEEE, Montreal, 2005, pp. 13-20.
[22] A. Goering, L. Louie, Demystifying WebSphere Business Services Fabric Policy Evaluation and Dynamic Endpoint Selection, IBM Technical library, 2008, http://www.ibm.com/developerworks/websphere/library/techarticles/0802_goering/0802_goering.html, accessed on: 16 March 2011.
[23] Y. Caseau, Self-adaptive Middleware: Supporting Business Process Priorities and Service Level Agreements, Advanced Engineering Informatics, 19/3 (2005) 199-211.
[24] S. Hudert, T. Eymann, H. Ludwig, G. Wirtz, A Negotiation Protocol Description Language for Automated Service Level Agreement Negotiations, in: K.-J. Lin, C. Huemer (Eds.), IEEE Conference on Commerce and Enterprise Computing, IEEE, Vienna, 2009, pp. 162-169.
[25] Y. Jun, R. Kowalczyk, J. Lin, M. Chhetri, S. Goh, J. Zhang, Autonomous Service Level Agreement Negotiation for Service Composition Provision, Future Generation Computer Systems, 23/6 (2007) 748-759.
[26] W3C, Web Services Policy 1.5 – Framework, 2007, http://www.w3.org/TR/ws-policy, accessed on: 16 March 2011.
[27] W3C, Web Services Policy 1.5 – Attachment, 2007, http://www.w3.org/TR/ws-policy-attach, accessed on: 16 March 2011.
[28] O. Holschke, J. Rake, P. Offermann, U. Bub, Improving Software Flexibility for Business Process Changes, Business & Information Systems Engineering, 2/1 (2010) 3-13.
[29] W3C, OWL Web Ontology Language, 2004, http://www.w3.org/TR/owl-features, accessed on: 16 March 2011.
[30] W3C, Web Services Description Language (WSDL) Version 2.0, 2007, http://www.w3.org/TR/wsdl20/, accessed on: 16 March 2011.
[31] OASIS, UDDI Version 3.0.2, 2004, http://uddi.org/pubs/uddi-v3.0.2-20041019.htm, accessed on: 16 March 2011.
[32] M.B. Juric, A. Sasa, B. Brumen, I. Rozman, WSDL and UDDI Extensions for Version Support in Web Services, Journal of Systems and Software, 82/8 (2009) 1326-1343.
[33] T. Yu, K.-J. Lin, Service Selection Algorithms for Web Services with End-to-end QoS Constraints, Information Systems and e-Business Management, 3/2 (2005) 103-126.
[34] T. Yu, Y. Zhang, K.-J. Lin, Efficient Algorithms for Web Services Selection with End-to-end QoS Constraints, ACM Transactions on the Web, 1/1 (2007) 6-es.
[35] R. Meersman, Z. Tari, L.-H. Vu, M. Hauswirth, K. Aberer, QoS-based Service Selection and Ranking with Trust and Reputation Management, in: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems, Springer, Berlin-Heidelberg, 2005, pp. 466-483.
[36] R. Roeller, P. Lago, H. Van Vliet, Recovering Architectural Assumptions, Journal of Systems and Software, 79/4 (2006) 552-573.
[37] S. Mazanek, M. Hanus, Constructing a Bidirectional Transformation Between BPMN and BPEL with a Functional Logic Programming Language, Journal of Visual Languages & Computing, 22/1 (2011) 66-89.
[38] OMG, XML Metadata Interchange (XMI), 2007, http://www.omg.org/spec/XMI/2.1.1/, accessed on: 16 March 2011.
[39] W3C, XML Path Language (XPath) 2.0, 2010, http://www.w3.org/TR/xpath20/, accessed on: 16 March 2011.
[40] Google, Image search for term "customize icon" with restriction of image size "icon", 2011.
[41] Google, Image search for term "contract icon" with restriction of image size "icon", 2011.

[42] J. Cardoso, J. Mendling, G. Neumann, H.A. Reijers, A Discourse on Complexity of Process Models, in: J. Eder, S. Dustdar (Eds.), Business Process Management Workshops, Springer, Berlin-Heidelberg, 2006, pp. 117-128.
[43] M. Azuma, M. David, Software Management Practice and Metrics in the European Community and Japan: Some Results of a Survey, Journal of Systems and Software, 26/1 (1994) 5-18.
[44] J. McCabe, A Complexity Measure, IEEE Transactions on Software Engineering, 2/4 (1976) 308-320.
[45] J. Cardoso, Control-flow Complexity Measurement of Processes and Weyuker's Properties, in: C. Ardil (Eds.), 6th International Enformatika Conference, Transactions on Enformatika, Systems Sciences and Engineering, Prague, 2005, pp. 213–218.
[46] M.H. Halstead, Elements of Software Science, Elsevier, New York, 1977.
[47] OASIS, eXtensible Access Control Markup Language (XACML) Version 2.0, 2005, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, accessed on: 16 March 2011.
[48] W.K. Abdulla, XRules, 2005, http://www.xrules.org/, accessed on: 16 March 2011.
[49] OMG, Production Rule Representation (PRR) Version 1.0, 2009, http://www.omg.org/spec/PRR/1.0/, accessed on: 16 March 2011.
[50] OMG, Semantics of Business Vocabulary and Business Rules (SBVR) Version 1.0, 2008, http://www.omg.org/spec/SBVR/1.0/, accessed on: 16 March 2011.
[51] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D.F. Ferguson, Web Services Platform Architecture, Prentice Hall, Upper Saddle River, 2005.
[52] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall, Upper Saddle River, 2005.
[53] M. Momotko, B. Nowicki, Visualisation of (Distributed) Process Execution Based on Extended BPMN, in: V. Marik, O. Stepankova, W. Retschitzegger (Eds.), 14th International Workshop on Database and Expert Systems Applications, IEEE Computer Society, Heidelberg, 2003, pp. 280-284.

Vitae

Ales Frece, B.Sc., is a researcher preparing his doctoral dissertation. His research is focused on business process management and service oriented architecture. He participates in several research and applicative projects, such as business process consolidation and optimization, development of proof-of-concepts, pilots and blueprints. He co-authored WS-BPEL 2.0 for SOA Composite Applications with IBM WebSphere 7 book. He is an IBM SOA Solution Designer and SOA Associate.

Matjaz B. Juric, Ph.D., is Full Professor at the University of Ljubljana and the head of SOA and Cloud Computing Competence Centre. He has authored 15 SOA and Java books, such as Business Process Driven SOA using BPMN and BPEL, SOA Approach to Integration, Business Process Execution Language, BPEL Cookbook (award for best SOA book in 2007), etc. Matjaz has been SOA consultant for several large companies. He has contributed to SOA Maturity Model and performance optimization of RMI-IIOP, etc. He is also a member of the BPEL Advisory Board, an Oracle ACE Director and a Java Champion.