Decision as a Service (DaaS)

Decision as a Service (DaaS): A Service-Oriented Architecture Approach for Decisions in Processes

Faruk Hasić, Johannes De Smedt, Seppe vanden Broucke, and Estefanía Serral

Abstract—Separating decision modelling from the processes modelling concern recently gained significant support in literature, as incorporating both concerns into a single model impairs the scalability, maintainability, flexibility and understandability of both processes and decisions. Most notably the introduction of the Decision Model and Notation (DMN) standard by the Object Management Group provides a suitable solution for externalising decisions from processes and automating decision enactments for processes. This paper introduces a systematic way of tackling the separation of the decision modelling concern from process modelling by providing a Decision as a Service (DaaS) layered Service-Oriented Architecture (SOA) which approaches decisions as automated and externalised services that processes need to invoke on demand to obtain the decision outcome. The DaaS mechanism is elucidated by a formalisation of DMN constructs and the relevant layer elements. Furthermore, DaaS is evaluated against the fundamental characteristics of the SOA paradigm, proving its contribution in terms of abstraction, reusability, loose coupling, and other pertinent SOA principles. Additionally, the benefits of the DaaS design on process-decision modelling and mining are discussed. Finally, the DaaS design is illustrated on a real-life event log of a bank loan application and approval process, and the SOA maturity of the design is assessed.

Index Terms—Decision Modelling, Decision Model and Notation, DMN, Process Modelling, Integrated Modelling, Decision Mining, Integrated Mining, Service Discovery, Separation of Concerns, Service-Oriented Architecture.

1 Introduction

Recent business process management literature moves towards accommodating decision management into the paradigms of *Separation of Concerns (SoC)* [1]–[7] and *Service-Oriented Architecture (SOA)*. This implies externalising decisions and encapsulating them into separate decision models, hence implementing decisions as externalised services. Literature proposes several conceptual decision service platforms and frameworks [8]–[11], as well as ontologies [12]. Industry has adopted this trend, as several decision service management systems have appeared, e.g. SAP Decision Service Management [13]. This separation of concerns provides a plethora of advantages regarding understandability, maintainability, and flexibility of both the business process and the decision models [1]–[6], [14], [15].

A recently introduced decision modelling standard, the Decision Model and Notation (DMN) [16], has enjoyed significant interest in literature [5], [17]–[19]. DMN consists of two levels that are to be used in conjunction. First, the decision requirement level represented by the Decision Requirement Diagram (DRD) which depicts the requirements of decisions and the dependencies between elements involved in the decision model. Second, the decision logic level, which presents ways to specify the underlying decision logic. DMN aims at providing a clear and simple representation of decisions in a declarative form and offers no decision resolution mechanism of its own. Rather, the invoking context, e.g. a business process, is responsible for ensuring a correct invocation and enactment of the decision, as well as ensuring data processing and the storage and propagation of data and decision outcomes throughout the process. This makes DMN particularly interesting for a SOA, as DMN is independent of the applications and the invoking context.

However, there is no clearly and formally defined SOA design dealing with DMN decision services in business processes. This lead to violations against the SoC and SOA paradigms in previously produced models in research, where decisions tend to be embedded or hard-coded within the process, and where decision logic tends to be duplicated in both the process and decision models. This paper aims at bringing DMN to the service-orientation paradigm in order to exploit the benefits of SOA characteristics in terms of maintainability, scalability, understandability, and flexibility. Thus, the contribution of this paper is a layered design framework and a formalisation of its key concepts to aid in exteriorising the decisions and the decision logic from the process flow according to the SoC [1] and SOA paradigms. The layered architecture consists of a process layer, a service layer, and a decision layer. The two latter are connected to the former through a decision service interface. The evaluation of the contribution is fourfold:

- *F. Hasić and E. Serral are with the Department of Information Management, Modelling and Simulation, KU Leuven, Belgium. E-mail: faruk.hasic; estefania.serralasensio@kuleuven.be*
- *J. De Smedt is with the Management Science and Business Economics Group of the University of Edinburgh Business School, Scotland, UK. E-mail: Johannes.DeSmedt@ed.ac.uk*
- *S. vanden Broucke is with the Department of Decision Sciences and Information Management, KU Leuven, Belgium. E-mail: seppe.vandenbroucke@kuleuven.be*

Manuscript submitted on May 30, 2018; resubmitted on May 27, 2019.

1) The proposed DaaS design is assessed in terms of fundamental SOA principles, proving its merit in terms of service discoverability, loose coupling, standardisation, location transparency, abstraction, statelessness, longevity, reusability, and composability.
2) The implications of the DaaS design on process and decision modelling and mining are conferred in terms of scalability, maintainability, flexibility and understandability of both the processes and the decisions.
3) Additionally, we illustrate the DaaS design on an event log containing information on a real-life bank loan application and approval process. We show from the real-life event log that decisions are invoked as services, in compliance with the principles of SOA and that the DaaS design aids in conducting decision service compliance verification for processes.
4) Finally, we assess the maturity of the DaaS SOA design using a state-of-the-art SOA maturity model.

This paper is structured as follows. Section 2 constitutes a related work section while Section 3 outlines the methodology of the paper. Section 4 provides formal definitions for key concepts needed for the understanding of the layered SOA. In Section 5 the layered architecture design is established and elucidated and in Section 6 the proposed Decision as a Service architecture is evaluated against the core characteristics of SOA design. In Section 7 the implications of the proposed design for integrated process and decision modelling and mining are discussed in terms of advantages and disadvantages. Section 8 illustrates the DaaS approach on a real-life enriched event log, thus providing opportunities for decision service compliance verification. Next, Section 9 assesses the maturity of the proposed design according to a state-of-the-art SOA maturity model. Section 10 discusses limitations of the approach and finally, Section 11 concludes and provides future research directions.

2 Motivation and Related Work

The *SoC* paradigm is already well-established in the software modelling and design domains [1], [20], [21]. With the introduction of DMN, the paradigm is introduced in the Business Process Management (BPM) domain as well, effectively shifting the domain towards a SOA, by representing business decisions as externalised services. Most modelling and mining approaches in literature still breach the SoC between process control flow on the one hand, and data and decision aspects on the other. Consequently, issues concerning maintainability, scalability, reusability, and understandability arise [5], [22], given the fact that most decisions are hard coded within the process flow [2]. These issues can be avoided by externalising decision constructs to a dedicated decision model and setting decisions up as a service to be invoked by the process. The intersection between data and processes, classical data mining and process mining is rapidly gaining traction in literature [4], [23]–[27]. Separating multi-perspective modelling and mining tasks proves to be beneficial in multiple ways, as long as the separation and interaction between the models is conducted in a sound and consistent way [3]–[5], [22]. Some literature on decision service platforms exists. In [9] decisions are approached from an organisational process control flow perspective rather than a data and decision management perspective. Other works recognise that decision logic and process logic should indeed be separated, as application logic is specified in terms of processes while decision rules specify conditions to adapt the application behaviour [8], [11]. However, these works consider simple decisions and business rules, rather than holistic and intertwined decision models. Furthermore, the works discuss general requirements and solutions, rather than a standardised approach towards decision management. In [10] on the other hand, the authors advocate for DMN as a standard for decision services for the purpose of decision automation. However, a formalised framework on how to organise decision services for processes is still lacking. Decisions did receive attention in the process mining domains, as researchers have utilised DMN to automatically discover decisions from event logs that are compatible with the discovered processes. However, these works employ a rather narrow definition of a decision, confining decisions to specific locations within a process known as decision points, i.e. exclusive gateways splitting the control flow of a process [28], [29]. Seminal work on mining decisions independently from process control flow is presented in [25], where decisions can span across the entire process execution span rather than being embedded in a single decision point of the process. This leads to the discovery of a holistic decision model that is decomposable and reusable across the process, thus, introducing a form of decision modularity and composition that is inherent to service-orientation.

Externalising decisions and setting them up as services is not the first example of adapting the SoC and SOA paradigms in the domain of processes: SOAs were already applied for business processes and for the interaction between business rules and processes [12], [13], [30]–[35]. With DMN, a comparable approach can be appied to decisions by implementing decisions as externalised services, which we call **Decision as a Service (DaaS)**. Processes, or other concerns, can invoke those decision services on demand by providing the relevant input data to the service through an interface. We call this **Decision on Demand (DoD)**. Such an approach aims at capitalising on the benefits of SOA, such as loose coupling, service standardisation, abstraction, and composability.

3 Methodology

This paper follows a design science approach [36], structured along three different cycles to obtain an artifact, being the SOA-based **Decision as a Service (DaaS)** design. Figure 1 provides an overview of the followed methodology.

First of all, during the *relevance* cycle we have identified the problem of inefficient use of decisions within processes and as a result the issues that arise regarding maintainability, scalability, flexibility, and complexity and understandability of both decisions and processes in Sections 1 and 2. We have argued that these are the relevant issues tackled when separating concerns in modelling endeavours through the use of the separation of concerns and SOA paradigms. Based on previous work of the authors, as outlined in the related work section above, and relevant literature - both academic and from industry - it was noted that there is

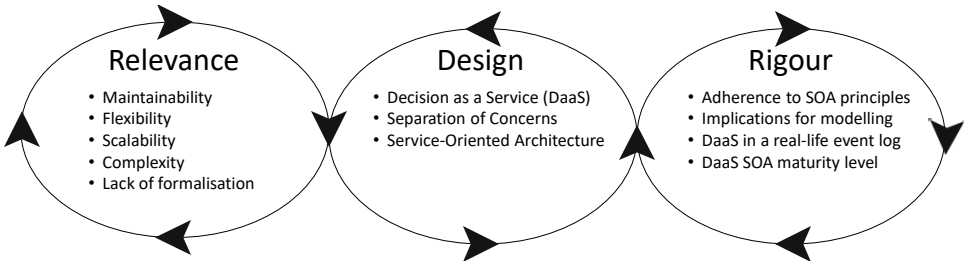

Figure 1: Overview of the followed cycles of the design science methodology.

no clearly and formally defined approach towards decision services in processes, and that from previously produced models in research multiple violations towards the separation of concerns and SOA paradigms were committed. Thus, we have developed the solution artifact in the form of a Decision as a Service (DaaS) SOA Design during the *design* cycle (Sections 4 and 5). Next, the artifact, i.e. the proposed *Decision as a Service (DaaS)* design, which will be outlined in Section 5, was validated according to the *rigour* cycle against the key principles of Service-Orientation (Section 6). Additionally, the effects of the proposed solution were evaluated against the key problems identified in the relevance cycle (Section 7). Furthermore, this work aims at bringing the proposed design to the body of literature on decision and process modelling and mining by exhibiting the artifact on a real-life enriched event log from industry, i.e., an enriched event log pertaining to a bank loan application and allocation process. It is demonstrated in Section 8 that decision data propagation within the log of a process indeed exhibits the invocation of decision logic through variable shifts throughout the process. As a final evaluation, the design was assessed in terms of its maturity in a state-of-the-art SOA maturity model.

4 Preliminaries

In this section, we provide a formalisation for key DMN concepts needed for the understanding of the **Decision as a Service** architecture, which will be discussed in the following sections. We adhere to the definitions provided in [22] and extend them to represent decision services as well. The DMN standard employs rectangles to depict decisions and subdecisions and ovals to represent data input. The decision logic is usually represented in decision table form.

Definition 1. A decision requirement diagram DRD is a tuple (D_{dm}, ID, IR) consisting of a finite non-empty set of decision nodes D_{dm}, a finite non-empty set of input data nodes ID, and a finite non-empty set of directed edges IR representing the information requirements such that $IR \subseteq (D_{dm} \cup ID) \times D_{dm}$, and ($D_{dm} \cup ID$, IR) is a directed acyclic graph (DAG).

According to the DMN standard, a decision requirement graph can be an incomplete or partial representation of the decision requirements in a decision model. The set of all DRDs in the decision model constitutes the exhaustive set of requirements. The information contained in this set can be combined into a single DRD representing the decision requirements level as a whole. The DMN standard refers to such a DRD as a decision requirement graph (DRG). We expand the notion of a DRG, in such a way that a DRG is a DRD which is self-contained, i.e. for every decision in the diagram all its requirements are also represented in the diagram.

Definition 2. A decision requirement diagram DRD is a decision requirement graph DRG if and only if for every decision in the diagram all its modeled requirements, present in at least one diagram, are also represented in the diagram.

The term *decision* can have a number of meanings. According to the DMN specification a decision is the logic used to determine an output from a given input. Meanwhile, in process modelling a decision is an activity or the act of using the decision logic, e.g. the business rule task in BPMN. Another common meaning is that a decision is the actual result, which we call the output of a decision, or simply the decision result. For the case of Decision as a Service, a decision is defined as follows:

Definition 3. A decision $d \in D_{dm}$ is a tuple (I_d, O_d, L), where $I \subseteq ID$ is a set of input symbols, O a set of output symbols and L the decision logic defining the relation between symbols in I_d and symbols in O_d.

In case of decision tables, I and O contain the variables of the input and output elements respectively, and L is the table itself, i.e. the set of decision rules present in the table.

In DRDs these decisions d_i are represented by the decision nodes $D_i \in D_{dm}$. We will use D to refer to both a decision and its representing node in a DRD. From the definition of DRGs, it is clear that every decision D in that model has a unique decision requirement graph DRG_D with D as its single top-level decision. A DRG contains exactly all information requirements of its top level decisions. Hence, only one DRG exists with D as its single top-level decision. We use DRG_D to denote this DRG. Furthermore, all the decisions in the DRG, except the top level decision, are consequently subdecisions of the top level decision. In other words, the top level decision *requires* these lower level subdecisions.

5 Decision as a Service (DaaS)

Separating the decision modelling concern from the process modelling concern implies modelling in two separate models or layers [5], [6], [22]. In Figure 2, the **Decision as a Service** layered architecture is presented through an example of a customer acceptance process with its corresponding

customer acceptance decision model. The bottom layer depicts the *processes layer*, while at the top the *decision layer* is represented. In the service-oriented approaches, the services are implemented offering a single decoupled point of entry to the services. That way, the bottom layer, i.e. the process layer, only needs the information regarding the point of entry, or more specifically the *interface*, in order to invoke the higher level layers. This single point of entry provides a plethora of advantages, such as flexibility, maintainability, automation and scalability [5], [11].

In Figure 2 the *service layer* is implemented as the connection between the *process layer* and the *decision layer*. The communication between the *process layer* and the *service layer* is bridged by the *interface*. Consequently, the processes are only aware of the *interface* and agnostic about the underlying *service layer* and *decision layer*. Thus, to invoke the services and the decisions the processes simply need to keep information regarding the *interface* and not regarding the higher level layers.

To formally define the interface and the decision services we first need to define the input requirement set of a decision as follows:

Definition 4. The decision input requirement set $dirs_D$ of a decision D is the set of all sets of input data which are sufficient to invoke D. $dirs_D$ contains sets of input data directly or indirectly required by D. The largest set in $dirs_D$ is the set of all input data nodes for which there exists a path to D in DRG_D. The smallest set in $dirs_D$ is D's input set I_D.
$dirs_D$ is constructed inductively by the following rules:

- $I_D \in dirs_D$
- For all $s \in dirs_D$ if there is an $i \in s$ such that $i \in O_{D'}$ for some D' in DRG_D, then $s \setminus \{i\} \cup I_{D'} \in dirs_D$.

Each decision in a DRD has its own output set, as formalised in Definition 3. As seen in Figure 2, a decision service is used to invoke a decision from the decision model.

Definition 5. A decision service DS_D of a decision D is a tuple (s_D, O_D), where $s_D \in dirs_D$ is a set of input data sufficient to invoke the decision D and O_D the output set representing the decision outcomes of D.

Note that multiple decision services can be defined for a single decision D, depending on which input data set s_D is used to access the decision layer. Consider the decision model in the decision layer of Figure 2. For Decision `Background Check` two decision services can be defined: DS_{BC1} with tuple $(\{O_{CIV}, pr\}, O_{BC})$ where O_{CIV} is the output of Subdecision `Customer Identity Verification`, which serves as the input for Decision `Background Check` and *pr* is the *Public Records* input file; and DS_{BC2} with tuple $(\{cid, pr\}, O_{BC})$ where *cid* is the *Customer ID* input data object required for the decision enactment of Subdecision `Customer Identity Verification` and consequently the `Background Check` decision, and *pr*, i.e. the *Public Records* input data object. Which decision service will be activated depends on the input data set provided through the interface by the process in the process layer. Hence, the interface will steer the information towards the suitable service that is able to invoke the required decision based on the input data set received from the process. Thus, a decision service's interface is the combination of its input requirement set and its output set. Decision interfaces can be defined as in Definition 6.

Definition 6. The interface IF_D of a decision service DS_D is defined as a tuple $(dirs_D, O_D)$, where $dirs_D$ is the input requirement set and O_D the output set of the underlying decision D.

Now we have formally defined the decision layer, the service layer and the interface layer present in Figure 2. What is left is to define how the process layer and the different interactions with the service interface. Decisions in processes do not surface solely as the driver of control flow. Rather, they both encompass the routing, i.e. because of decision outcomes that steer toward a certain activity tailored towards supporting its output, and the changes in the data layer of the process as well. The latter introduces numerous types of activities that are representatives of the *decision* model in the *process* model:

Definition 7. The input and output data variables of business activities are defined as follows:

- $I : A \rightarrow V$, function assigning activities which receive input of a certain variable,
- $O : A \rightarrow V$, function assigning activities which deliver output for a certain variable.

This enables the construction of the following activity types:

1) **Operational activities ((no) inputs, no outputs):** do not have any influence on the process' decision dimension and only act as a performer of a specific action that is tied to that specific place in the control flow. They might serve as the end of a decision. They are provided with the decision inputs needed, which are not used further in the process,
$A_o = \{a \in A \mid O(a) = \emptyset, \}$.
2) **Administrative activities (no inputs, outputs):** have the purpose to introduce decision inputs into the process,
$A_a = \{a \in A \mid I(a) = \emptyset \wedge O(a) \neq \emptyset\}$.
3) **Decision activities (inputs, outputs):** serve a true autonomous decision purpose as they transform decision inputs into a decision outcome,
$A_d = \{a \in A \mid I(a) \neq \emptyset \wedge O(a) \neq \emptyset\}$.
Note that it holds that $A_a \cup A_o \cup A_d = A$.

With the activity classification in mind, we can now make the connection with decisions in business processes and decision models. A decision in a business process can be defined as follows:

Definition 8. A decision in a process model, $d^a \in D_{dm}$ is a tuple $(I_{d_a}, O_{d_a}, L_{d_a})$, where $a \subseteq A_d$, $I_{d_a} \subseteq I(a)$, $O_{d_a} \subseteq O(a)$, and $L_{d_a} \subseteq L$.

This last definition connects a decision activity with a decision and it shows than one decision activity can be tied with multiple decisions. The latter implies that, within an event log, the same activity can make different decisions, i.e., changes in variable values, and can be represented as different decision nodes within a decision model, as well as different activity types. This interpretation of how activities

are present in process models is the main difference with other decision mining and modelling techniques, who keep the one-to-one mapping of activities and decisions.

We have formally defined elements from all three layers in the DaaS design in Figure 2: the decision layer, the decision service layer, the decision service interface, and the constructs from the process layer that are relevant for decision services and decision enactment in processes. Whether a process will correctly call upon a decision service depends on the information that the process itself provides to the decision service interface. In order to invoke a decision service successfully and unambiguously the input data objects needed for the invocation of the underlying decision need to be both complete and correct. This is defined in the decision **Service Adherence Criterion (SAC)** in Definition 9, which states when a process is compliant to the decision service it wishes to invoke.

Definition 9. The Service Adherence Criterion (SAC): A process fully adheres to the decision service DS_D of decision D if and only if at the time of invocation, the process has internally produced and/or externally received all $x \in s_D$ with $s_D \in dirs_D$ and provided all $x \in s_D$ to the decision service interface IF_D. Only then will the decision service DS_D of decision D be able to provide a decision outcome $o \in O_D$ to decision activity $d \in A_d$ that called upon the decision service DS_D.

In summation, a process can call upon a decision by providing a data input set that is required for the enactment of the decision to the decision interface. The interface will steer the information provided by the process towards the suitable decision service. Subsequently, the decision service will invoke the requested decision from the decision model. Consequently, the decision model will enact the decision, reach a decision outcome, and output it to the decision service. The decision service will forward the outcome back to the interface, and through the interface the outcome of the decision will finally reach the process layer. This mechanism is illustrated in Figure 2. The SOA design provided in the figure makes it possible to define decisions as services, which we call **Decision as a Service (DaaS)**. These decisions and services can be invoked on demand by information systems, e.g. processes, through a well-defined interface.

With the design provided in Figure 2, the SoC paradigm between processes and decisions can easily be respected, as the process is included in the bottom layer, while the decision model is situated in the top layer. Caution is still necessary when modelling the process and decision models: information regarding decisions and decision logic should not be implemented in the bottom process layer, but rather externalised and encapsulated in the decision layer [2], [3], [5], [22], [25].

We link the main elements present in the DaaS example in Figure 2 to the formalisation of the DaaS design and DMN constructs as elaborated upon in the current and the previous sections. We use the abbreviations of the names given to elements in Figure 2 in the subscript of the symbols to refer to those elements for a compact notation as well as the abbreviations of the data object elements (pertaining to both the process layer and the decision layer) in lower case letters to reference them.

Decision activities: $A_d = \{a_{VI}, a_{DRL}, a_{AC}\}$.
a_{VI} is a tuple $(cid, iv, L_{D_{CIV}})$;
a_{DRL} is a tuple $(\{fs, iv, fi, pr\}, raf, L_{D_{RL}})$;
a_{AC} is a tuple $(raf, cf, L_{D_{AC}})$.
Decision services: $DS = \{DS_{CIV}, DS_{RL}, DS_{CA}\}$.
DS_{CIV} is a tuple (cid, O_{CIV})
with $s_{CIV} = \{cid\} \in dirs_{CIV}$;
DS_{RL} is a tuple $(\{fs, iv, fi, pr\}, O_{RL})$
with $s_{RL} = \{fs, iv, fi, pr\} \in dirs_{RL}$;
DS_{CA} is a tuple (raf, O_{CA})
with $s_{CA} = \{raf\} \in dirs_{CA}$.
Service interfaces: $IF = \{IF_{CIV}, IF_{RL}, IF_{CA}\}$.
IF_{CIV} is a tuple $(dirs_{CIV}, O_{CIV})$;
IF_{RL} is a tuple $(dirs_{RL}, O_{RL})$;
IF_{CA} is a tuple $(dirs_{CA}, O_{CA})$.
Decisions: $D_{dm} = \{D_{CIV}, D_{BC}, D_{FPC}, D_{RL}, D_{CA}\}$.
D_{CIV} is a tuple (cid, O_{CIV}, L_{CIV});
D_{BC} is a tuple $(\{o_{CIV}, pr\}, O_{BC}, L_{BC})$
with $o_{CIV} \in O_{CIV}$;
D_{FPC} is a tuple $(\{fs, fi\}, O_{FPC}, L_{FPC})$;
D_{RL} is a tuple $(\{o_{BC}, o_{FPC}\}, O_{RL}, L_{RL})$
with $o_{BC} \in O_{BC}$ and $o_{FPC} \in O_{FPC}$;
D_{CA} is a tuple (o_{RL}, O_{CA}, L_{CA})
with $o_{RL} \in O_{RL}$.

6 Compliance with the Principles of SOA

In this section, we will evaluate the adherence of the proposed DaaS design to the key Software-Oriented Architecture characteristics as defined in fundamental literature on SOA design [37]–[43], both in theory and through the example proposed in Figure 2.

6.1 Discoverable and Dynamically Bound Services

A component, i.e. a client, calling upon a service must be able to discover that service based on information provided at runtime. After discovering the service dynamically, the component that called the service, or the client, must be able to interpret the service outcome. Thus, the client must at runtime be able to discover the service it needs, to invoke it, and to receive a sensible answer from the service. In the DaaS design, the services are dynamically discovered at runtime by the process. This happens by providing the correct data input set $s_D \in dirs_D$ to the interface IF_D. Given the decision D that the client wants to invoke and the set s_D, the interface IF_D will guide and discover the relevant decision service DS_D. After enacting the decision D, the service DS_D will return the decision outcome $o \in O_D$ to the interface IF_D, after which the interface will pass it on to the invoking context, i.e. the client, for interpretation. In short, the process will inform the interface about the decision it wants to invoke. However, given the modular hierarchy of the decision model, multiple decision services, i.e., with a different input requirements set, can be defined that invoke the same underlying decision. Which decision service will be bound to the invocation of the process depends on the data input set provided by the process to the decision service interface. Note that according to basic SOA architectures [41], the decision model functions as the service provider, while the process model fulfils

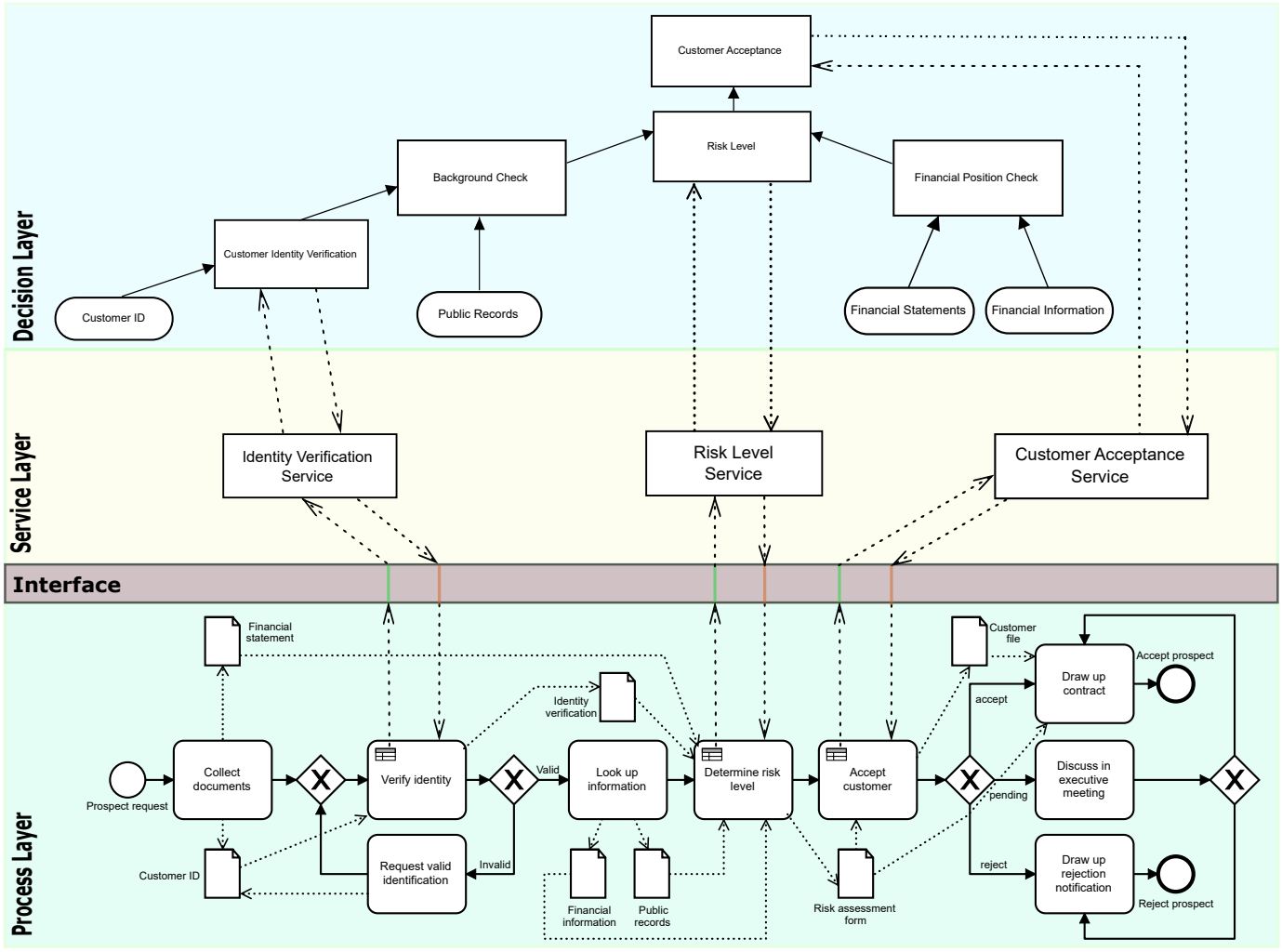

Figure 2: Decision as a Service (DaaS) layered architecture

the role of service consumer. The decision service interface corresponds to a service directory where the decision service provider can publish information concerning the services that can be invoked. The decision service consumer can address the service directory in order to invoke the required decision services.

In the example in Figure 2, decision activity `Determine risk level` provides the input set $\{fs, iv, fi, pr\}$, i.e. the four input data objects of the activity, to the interface. The interface guides, given the input set and the decision requested by the decision activity, the invocation of the correct decision service, i.e. the `Risk Level Service` (DS_{RL}). This leads to the invocation of the `Risk Level` decision in the decision layer. Through the service and the decision service interface, the outcome of the decision will be returned to decision activity `Determine risk level` which will produce the *Risk assessment form* containing the decision outcome.

6.2 Standardised Service Communication and Loose Coupling

Communication between the invoking context, e.g. a business process, and the services, e.g. decision services tied to a decision model, should be standardised and addressed systematically. Besides, a loose coupling between clients and services is preferred above a tight coupling. In a loose coupling there are a few well-defined dependencies between the different modules, while a tight coupling produces a vast array of dependencies that might not all be observed. In the DaaS design, the communication between clients from the process layer and the decision service DS_D of a certain decision D is standardised through a well-defined interface IF_D. The interface is the sole channel of communication between the clients and the services, and thus serves as a loose coupling mechanism between the two. Ergo, the clients are aware of the existence of a service, given that the clients can witness and access the interface IF_D of a decision service DS_D. However, the coupling does not go beyond the simple interface connection and the client's awareness of the existence of a decision service.

In the example in Figure 2, the communication between a decision activity, e.g. `Determine risk level` and the decision service of `Risk Level Service` is standardised and loosely coupled through the decision service interface IF_{RL}, defined as a tuple $(dirs_{RL}, O_{RL})$. The decision activity can only invoke its underlying decision, i.e. `Risk Level` by providing the input data relevant for the invocation of the decision service, i.e. $s_{RL} \in dirs_{RL}$. Likewise,

the decision model will render the outcome of the `Risk Level` decision $o_{RL} \in O_{RL}$ to the invoking decision activity through the interface.

6.3 Service Standardisation

In order to provide transparency in design and to avoid issues concerning redundancy, services should, in analogy with service communications, be standardised and systematised. The notion of a service should be unambiguous and coherent. This characteristic is closely related to the service encapsulation characteristic, which states that services that were not part of the service-oriented design from the beginning, can later be defined and become a part of the design, as long as they follow the designated service standardisation. In the DaaS design, a decision service DS_D of a decision D is a tuple (s_D, O_D), with $s_D \in dirs_D$ a set of input data sufficient to invoke D and O_D the set of decision outcomes of D. Thus, all new decision services should be designed according to these well-delineated concepts in order to ensure the correct invocation of those new services by potential clients.

In the example in Figure 2, the `Risk Level Service` is designed as a tuple $(\{fs, iv, fi, pr\}, O_{RL})$ with $s_{RL} = \{fs, iv, fi, pr\}$ and O_{RL} being the output set of the decision `Risk Level`. Thus, the decision service follows the standard design regarding input requirement and output sets.

6.4 Service Location Transparency

The invoking context of a service should not be burdened with the knowledge of the service location within the network, yet the client should be able to invoke a service at any time, regardless of the service location. This service location transparency allows the service location to change within the network without impairing service availability to clients. In the DaaS design, the process layer is agnostic about the location of a decision service DS_D. Once a correct data input set $s_D \in dirs_D$ is provided to the interface IF_D, the interface will, given the decision D that the client wants to invoke and the set s_D, discover the location of the relevant decision service DS_D and consequently invoke that service. This all happens without any client in the process layer knowing the location of the invoked decision service within the network. Hence, the decision services can be moved around the network and stored anywhere in the network without compromising the availability and accessibility of a decision service.

In the example in Figure 2, the `Risk Level Service` is designed as a tuple $(\{fs, iv, fi, pr\}, O_{RL})$. Thus, given the input requirement set $s_{RL} = \{fs, iv, fi, pr\} \in dirs_{RL}$, the interface will discover the location of decision service DS_{RL}, without the client, i.e. decision activity `Determine risk level` being aware of the location of the `Risk Level Service` DS_{RL}.

6.5 Service Abstraction

The service abstraction characteristic denotes the proposition that services should be conceived as a black box by the invoking context or the clients. Thus, the clients invoking the service are alleviated from the burden of understanding the underlying decision logic and mechanisms the services pertain to. In the DaaS design, the decision logic is encapsulated in the decision model in the top layer, i.e. the decision layer, given that a decision D was defined as a tuple (I_D, O_D, L), where I_D is a set of input symbols, O_D a set of output symbols and L the decision logic defining the relation between symbols in I_D and symbols in O_D. Hence, the decision logic L is encapsulated in decision nodes in the decision model and the services merely invoke the decisions from the decision model and pass the output, through the interface, to the clients in the process layer. Therefore, the clients are agnostic about the underlying decision logic and experience the decision services as a black box. All the clients need to worry about is providing the relevant inputs to the interface of the decision services in order to ensure a sound decision enactment.

In the example in Figure 2, the decision logic invoked by the `Determine risk level` decision activity (a_{DRL}) through DS_{RL} is encapsulated in the decision node `Risk Level` (D_{RL}) of the DRD in the decision layer. Hence, the `Determine risk level` decision activity is agnostic about the decision logic underpinning its execution, and the decision activity conceives the `Risk Level Service` DS_{RL}, and consequently the `Risk Level` decision, as a black box that merely returns the decision outcome $o \in O_{RL}$ when provided with an input set $s_{RL} \in dirs_{RL}$.

6.6 Service Statelessness

Services should be implemented as simple mechanisms that either return the relevant outcome or throw an exception if no conclusion can be reached, i.e. services should be stateless and consequently resource efficient. Thus, the state of the service should be separated from the service itself, providing the service with more flexibility and reusability. In the DaaS design, a decision service DS_D of a decision D is defined as a tuple (s_D, O_D), where $s_D \in dirs_D$ is a set of sufficient input data and O_D the output set representing the decision outcomes of D. Hence, decision services in the DaaS design will return an outcome from the set O_D or an empty set serving as an exception if no conclusion can be reached within a reasonable time frame.

In the example in Figure 2, the `Risk Level Service` DS_{RL} simply invokes the `Risk Level` decision at the request of the `Determine risk level` decision activity, and passes the decision outcome outcome $o \in O_{RL}$ back to the invoking decision activity. The `Risk Level Service` itself does not keep a state, but simply serves as a bridge between the process and the decision.

6.7 Service Longevity

Service longevity denotes the objective that a service should remain unchanged and in existence during a considerably long time. Services should change only when utterly necessary in order to avoid that clients must adapt time and again to the newly changed service. Hence, services should be designed carefully with the intent to last long. In the DaaS design, it is clear that a decision service DS_D of a decision D will only change if the underlying decision model in the decision layer changes as well. Since a decision service DS_D of a decision D is defined as a tuple (s_D, O_D), the decision

service DS_D will only need to change if the input set s_D and/or the output set O_D of the underlying decision D are subject to change. Once changes to a decision service occur, clients in the process may need to adapt to ensure a proper invocation of the underlying decisions. Note that if the input sets and output sets of the decision services remain the same, while the underlying decision logic that relates the inputs to the outputs undergoes adaptation, the decision services remain the same from the perspective of the clients, i.e., the processes, since the processes experience the decision logic as a black box.

In the example in Figure 2, the `Risk Level Service` DS_{RL} is designed as a tuple $(\{fs, iv, fi, pr\}, O_{RL})$ with $s_{RL} = \{fs, iv, fi, pr\}$ and O_{RL} being the output set of the decision `Risk Level`. Hence, the defined service will only need to undergo change if s_{RL} and/or O_{RL} exhibit any changes. In all other changes to either the decision layer or the process layer, the `Risk Level Service` DS_{RL} remains valid.

6.8 Service Reusability

This charecterisitc is cosely related to the charactersitic of service longevity. Services should be reusable and invocable repetitively and their results should be reusable. Additionally, the underlying logic should be divided into modules and linked to different services, thus enhancing the reuse of the underlying logic. In the DaaS design, the logic L is stored in the decision model as part of the decision layer across multiple decision nodes D, given that a decision D was defined as a tuple (I_D, O_D, L) with the decision logic L being part of the decision D. Ergo, the decision logic is stored in modules represented by the decision nodes. For every decision node D multiple decision services DS_D can be created, as discussed in Section 5. Hence, the underlying decision logic L can be accessed in smaller modules linked to a number of decision services DS_D, allowing for the reusability of the decision logic through repetitive invocations of the decision logic L by the decision services DS_D pertaining to a decision D.

In the example in Figure 2, the logic contained in the `Risk Level` decision node in the decision model can be reused by multiple clients as long as the clients provide the necessary input requirement set $s_{RL} = \{fs, iv, fi, pr\}$ to the `Risk Level Service` DS_{RL}.

6.9 Service Composability

Several services might be composed into a single larger service thanks to the modular design inherent to the SOA paradigm. This modular design allows to assemble smaller services into coherent larger ones or even into an application. This characteristic is closely related to the characteristic of service granularity, or more specifically: the degree of modularity that the services should have. Decision service composability is inherently enabled by the modular and hierarchical structure of the DMN model, i.e., a higher-level decision is composed of its lower-level decisions. As such, the higher-level decision service is composed of these lower-level decision services as well.

In the DaaS design example in Figure 2, multiple decision services or modules are implicitly part of larger decision services or modules. Take for instance the `Risk Level` decision node in the decision layer: for this decision, multiple decision services can be defined. Previously, we defined the `Risk Level Service` DS_{RL} as $(\{fs, iv, fi, pr\}, O_{RL})$ with $s_{RL} = \{fs, iv, fi, pr\}$ and O_{RL} being the output set of the decision `Risk Level`. Another Risk Level Service can be defined as well: DS_{RL_2} as a tuple $(\{o_{BC}, o_{FPC}\}, O_{RL})$ where $o_{BC} \in O_{BC}$ and $o_{FPC} \in O_{FPC}$ are the outputs of the `Background Check` decision and the `Financial Position Check` decision respectively. For both the `Background Check` and the `Financial Position Check` decision, invocable decision services can be defined as well, e.g. DS_{BC} as a tuple $(\{o_{CIV}, pr\}, O_{BC})$ and DS_{FPC} as a tuple $(\{fs, fi\}, O_{FPC})$. Hence, it can be argued that the `Risk Level Service` DS_{RL_2} is a composition of the `Background Check Service` DS_{BC}, the `Financial Position Check Service` DS_{FPC}, and some additional decision logic encapsulated in the module of the `Risk Level` decision node.

7 Implications of DaaS for Process and Decision Integration

The advantages of the separation of modelling and mining concerns are emphasised in literature [1]–[3], [5], [22], [25], [28]. These works especially highlight the scalability, maintainability, flexibility, and understandability of decisions and processes. However, they do not provide a clear design or framework on how to systematically address these issues and how to guarantee a sound SoC by design. In this section we will discuss these advantages of separating the concerns and we will relate them to the DaaS design.

7.1 Scalability

A straightforward advantage of separating the process and decision modelling concerns is scalability. Given the fact that, when separating concerns, the decision logic is not modelled together with the process, the decision logic in the decision model can be concurrently invoked by many clients. This promotes the reusbility of the decisions and the underlying decision logic and thus creates scale advantages.

In the DaaS design in Figure 2, it is clear that multiple processes from the process layer can call upon a number of services simultaneously and that all the services can access the decision model at the same time. In cases where the modelling concerns are not separated, e.g. a typical way of modelling the process and decision concerns in one model is using an intricate setup of gateways, the decisions and the decision logic are embedded in the process, thus providing no opportunities for the reuse of logic and decisions and no possibilities for parallel invocation and enactment of the decisions. The DaaS design in Figure 2 clearly avoids these convoluted situations, as the decision logic is stored in the decision model as part of the decision layer, while the control flow is part of the process layer. The two do not convolute each other, however, the decisions can easily be invoked by any process as long as the process is able to provide the right input set to the interface of the decision service. That way, multiple decisions can access the decision

model and make use of the decision logic concurrently. Hence, the DaaS design provides the suitable circumstances for the scalability of decision enactment and logic reuse.

7.2 Maintainability

Another clear advantage of SoC is maintainability. Take for example the situation of a convoluted process-decision model where decisions are hard coded into gateways and the control flow of the process: if either the process or a decision changes, the convoluted model needs to be adapted. This creates a need for constant maintenance of the convoluted process-decision model. When adopting the SoC paradigm, this issues of maintainability and convolution of models are circumvented.

In the DaaS design in Figure 2, the decisions and the processes are not convoluted as they are separated into different modules of their respective layers. If parts of a process change in the process layer, the decisions in the decision layer are not affected. This corresponds to a decision-first approach, as opposed to the process-first approach that is present a convoluted process-decision model with cascading gateways. The maintenance of the process does not affect the underlying decisions. On the other hand, if parts of the decision model change, some services and processes will need to adapt to the changes in order to be able to invoke said decisions correctly. However, not all the services and processes will need to undergo adaptations; only the ones that are directly affected by the decisions that were modified. Other services and processes that do not pertain to the adjusted decision will still function properly without any need for adaptation. Noteworthy is that in the case of the DaaS design, where the modelling concerns are separated, all the decision logic is concentrated in the decision model as part of the decision layer. If anything concerning the decisions needs to change, the adaptation happens only in one place, i.e. the decision model. However, if decisions change in the process-first approach, every process containing those decisions will need to adapt as well. Hence, DaaS advances the maintainability of both processes and decisions.

7.3 Flexibility

Flexibility refers to the ad-hoc reuse of decisions and decision logic, as well as the flexibility in adapting and changing the underlying decision logic. Clearly, the process-first approach where modelling concerns are not separated does not support flexibility in any way: no reuse of logic is possible, since the logic is embedded in the control flow of the process and not stored in a separate module; and adapting the decision logic is rather cumbersome, since the logic is dispersed across multiple processes and hidden in convoluted process paths. Therefore, the process-first approach shows little flexibility in general.

In the DaaS design in Figure 2, the concerns are separated into their respective layers, and as explained earlier, the separated decision logic can be invoked ad-hoc by any client from the process layer conforming to the interface needed to invoke a decision service and consequently the necessary decision. Besides, changing the underlying decision logic is less of a burden, since the logic is concentrated in a single model, rather than dispersed across a process or even across multiple distributed or collaborating processes. Hence, the DaaS design offers a higher level of flexibility, both in terms of reusability of decisions and decision logic, as well as in terms of flexibility in logic adaptation.

7.4 Complexity and Understandability

The complexity and understandability of models is of particular importance. When modelling concerns are not separated the process quickly becomes overly complicated due to the cascading gateways. This is especially the case in knowledge-intensive processes where a lot of decisions need to be made based on a certain underlying logic. Often the term *spaghetti-like processes* [44] is used to refer to this phenomenon of intricate and convoluted control flows.

When opting for a DaaS design, the decisions and the decision logic are externalised and encapsulated in a separate layer as part of the decision model. Therefore, the control flow of the process is alleviated from the burden of representing the different decision paths. As a consequence, the actual process becomes visible and more understandable. However, because of the fact that decisions have been externalised, the process is still burdened with data management and data propagation, i.e. the process is responsible for the collection and propagation of the data needed for the invocation of a decision service. The decision model does not concern itself with these issues of data propagation and data management. It simply transforms the input data, obtained from the process through the decision service, into a decision output which is sent back to the invoking process. Hence, in the DaaS design, process complexity in terms of control flow will decrease, however, the complexity of data management is likely to increase. Thus data management and data propagation within processes becomes of paramount importance. Besides, when separating the decisions from the process, the overall view of the entire problem might be clouded, as processes take abstraction of decisions and simply approach decisions as a black box that answers to input, without concerning themselves with the underlying decision logic. Therefore, when applying the DaaS design, the emphasis should be put on the decisions and on a decision-first approach. The process might take abstractions from decisions and conceive them as a black box, however, everything stands or falls with the correct definition of the decisions, as both the decision services and the processes need to heavily rely on the decisions to function properly.

8 Evaluation of DaaS Design on a Real-Life Enriched Event Log

In this section, the DaaS approach is illustrated with automatically discovered decision services from an enriched event log. Literature on automatic discovery of DMN decision models from event logs has seen a considerable surge over the past few years [25], [28], [29]. The work in [25] is particularly interesting as it addresses decisions within a processes over multiple activities or even across the entire process execution span, rather than containing decisions to local decision points in the process. This **P**rocess **M**ining **In**tegrating **D**ecisions **(P-MInD)** framework provides interesting insights in the interaction between processes on the

one hand, and data, rules, and decisions on the other. We capitalise on the findings in [25] by adapting the technique in order to unveil decision services within processes, which will better explain the interaction between the processes and the decisions. In the remainder of this section we explain the changes applied to P-MInD needed to acknowledge decision services, thus rendering the discovered models consistent with the SOA paradigm. We call this approach the **S**ervice-**O**riented **A**rchitecture **P**rocess **M**ining **In**tegrating **D**ecisions **(SOAP-MInD)**. This section is concluded with a real-world example of DaaS-compatible decision services derived from an enriched event log containing information on a bank loan application and approval process.

8.1 SOAP-MInD

While P-MInD [25] discovers holistic decision models from event logs, it does not clearly illustrate how the process model communicates with its underlying decision model. In order to clarify the decision invocations by the process it is necessary to explain which decisions the process invokes at which specific points in the process. Additionally, it is required to know how the process invokes the desired decision, i.e. with which input data objects. In other words, for each invoked decision D the decision services DS_D that are called upon by the process need to be identified. P-MInD abstracts from the idea of services and is only concerned with the decision layer and process layer in Figure 2. However, to understand the interactions between the process and decision layers, the specific invocations of the process layer, i.e. the service calls, need to be acknowledged. Thus, we expand P-MInD to **S**ervice-**O**riented **A**rchitecture **P**rocess **M**ining **In**tegrating **D**ecisions **(SOAP-MInD)**, a framework for mining integrated decision services and the respective process traces where they are invoked. SOAP-MInD builds further on the principles of P-MInD and we refer to that seminal work on holistic decision modelling for details [25]. In what follows, we briefly describe the adaptation of P-MInD to render SOAP-MInD. SOAP-MInD was made compatible with the ProM framework as it is available as a ProM plugin, thus making the code open source[1].

The basis for both the P-MInD and SOAP-MInD approaches lies in the classification of activities according to Definition 7. This classification aides in understanding the interaction between the decision services and the decision service interface on the one hand, and the process itself on the other. Clearly, the set of operational activities (A_o) is not involved in decision making and is therefore irrelevant for the decision service layer and the decision service interface. Administrative activities have the purpose to introduce input data objects for decisions and are therefore relevant for the decision service layer and interface. An administrative activity $a \in A_a$ will provide an output set $O(a)$ which can be used as inputs for future decision activities in the process. A decision activity $d \in A_d$ will receive an input set $I(d)$ which is composed of the outputs of previously executed administrative activities and/or outputs of previously executed decision activities. The decision activity $d \in A_d$ will call upon the decision service DS_D through the decision service interface IF_D in order to invoke underlying decision D.

1. https://svn.win.tue.nl/repos/prom/Packages/PMinD/

Algorithm 1 captures how both the the decisions are stored and possible decision services are identified. First, the influence of activities over variables is identified through checking which shifts exist in an event log, i.e., whether the value of a variable v changed during an activity a compared to its previous occurrence in a trace. This is taken as evidence that the activity influences v, and both are stored as potential decisions (lines 2-3). Next, all potential decisions are checked for their occurrence, to see whether a certain decision happens (enough times, see $mintraces$) before another ($a_i < a_j$), and hence whether it might have an influence over each other's variables (lines 4-5). Note that this allows for decisions in which a_i serves as input to a_j, and vice versa. For all traces where this is the case, the correlation between both variables is calculated to see whether there is a link between both decisions (line 6). The variable/activity pair (VP) occurring the latest ((v_j, a_j)) is stored as a potential decision with inputs and outputs including the variable(s) of the earlier occurring activity a_i. This happens until all these connections are made, and the full input set of a_j is established. Next, all different decision sequences that happen in all traces are distilled, depending on what decision-relations were established through the correlations. After this, all possible decision services are discovered as well, in set DS. For each cluster of decisions, the values attached to the corresponding variable/activity pair are used to train a predictive model L_d, completing the decision (lines 10-13). More details on the extraction of the shifts, as well as correlation and thresholds are discussed in [25].

Algorithm 1 Decision service detection in an event log

1: **procedure** FIND_DECISION_SERVICES(event log $\mathcal{L}$)
2: Retrieve all shifts in the event log, i.e., all activities $a \in A$ for which a variable's v value changes during its execution
3: $VP \leftarrow (v, a)$, a potential candidate decision D where $v \in O_{d_a}$ and $a \in A_d$
4: **for** $(v_i, a_i), (v_j, a_j) \in VP \times VP$ **do**
5: **if** $|\{t \in \mathcal{L} \mid a_i, a_j \in t \wedge a_i < a_j \wedge v_i \in I(v_j)\}| > mintraces$ **then**
6: **if** $corr(v_i, v_j) > corrthres$ **then**
7: $D \leftarrow d = (I_d \leftarrow v_i, O_d = v_j, L = \emptyset)$ ▷ See Def. 3
8: $DS_d \in DS \leftarrow (I_d, O_d)$
9: Cluster traces in set C_T where the same subsets $DS_{C_T} \subseteq DS_d$ are present
10: **for** $c \in C_T$ **do**
11: **for** $DS_d \in DS_{C_T}$ **do**
12: train predictive model L_d over I_d to predict O_d
13: $D \leftarrow d = (I_d, O_d, L_d)$

Hence, the trace clusters are grouped based on the decision services they invoke. Note that in Definition 5, a decision service DS_d of a decision d was defined as a tuple (s_d, O_d), where $s_d \in dirs_d$ is a set of input data sufficient to invoke the decision d and O_d the output set representing the decision outcomes of d. The traces in this step of the SOAP-MInD approach are clustered based on the set of input data $s_d \in dirs_d$ needed to invoke a decision d and to obtain a decision outcome $o \in O_d$. Hence, the clusters present different execution sequences in which the decision services were invoked through the interfaces.

8.2 Decision Service Compliance Verification

SOAP-MInD renders two models per trace cluster of variable shift sequences: a process model and a decision service

model. This offers opportunities of decision service compliance verification, i.e. per trace cluster it can be investigated whether the discovered process model is able to correctly invoke the services pertaining to that same trace cluster.

Naively it can be stated that the process model either complies with the decision service or it does not comply with the service, as defined in the decision **Service Adherence Criterion (SAC)** in Definition 9. If the process model complies with the decision service, it inherently provides the necessary decision input requirements set $s_D \in dirs_D$ needed for the invocation of decision service DS_D pertaining to decision D. Clearly, this input requirements set s_D must be readily available to the process before the point in the process where the decision service DS_D is invoked. That way, the process can correctly provide the decision service interface IF_D with its corresponding input data. Consequently, the decision service DS_D will invoke decision D and an outcome $o \in O_D$ will be returned to the process through the interface IF_D. However, if no valid decision input requirements set $s_D \in dirs_D$ is available in the process at the time when the process invokes decision D by calling upon its decision service DS_D through the interface IF_D, the process is not complying with the decision service. Hence, no crisp decision outcome $o \in O_D$ of decision D will be returned to the process by the decision model through the decision service DS_D and the decision interface IF_D.

This is a rather naive approach towards decision service compliance verification, as decision service invocations are only considered to enact an underlying decision if the input data provided is both complete and correct. However, decision reasoning on incorrect and incomplete data can be applied as well. If for instance one input data element is missing from the decision input requirements set $s_D \in dirs_D$, the decision service DS_D might still be able to invoke its underlying decision model and provide all possible decision outcomes, given the known values is s_D and all possible values for the missing data element in s_D. The process would then, instead of one decision outcome, be provided with a set of possible decision outcomes $O_{D_M} \in O_D$. As a result, the process could still be able to continue properly after the decision enactment and reach a sound conclusion, given the correct interpretation of the decision outcomes in the process flow. Hence, the situation is more nuanced as it can be argued that a process can partially comply with a decision service as well, i.e. the process does not fully comply with the decision service in terms of input requirements sets, data hierarchies, and data propagation, however, given the data that the process provides to the decision service, a set of decision outcomes might still be reachable and useful for further process enactment. In that case, the decision Service Adherence Criterion (SAC) of Definition 9 can be relaxed towards a weaker version of decision service compliance.

8.3 Illustration and Discussion

In this subsection, we illustrate the DaaS design by applying the SOAP-MInD framework to a real-life enriched event log containing information on a bank loan application and approval process, made available for the 2017 BPI Challenge[2]. The log was filtered to a time window containing data between 26 August 2016 and 28 October 2016, i.e. 9 weeks of data. The filtered log contains 77,317 events over 4,382 cases. No additional pre-processing was performed and the unaltered log was used to create the output. Figure 3 provides two decision services that were extracted from the log. The decision services pertain to two trace clusters. Figure 4 depicts a part of the mined trace cluster relating to the rightmost decision service from Figure 4. Due to page restrictions, the full mined process models representing the trace clusters have been made available online[3].

The leftmost decision service in Figure 3 is invoked in the trace cluster containing 1,125 traces. In these traces, the `Complete application` decision has to be made. In order for the service to invoke this decision, the outcomes of two subdecisions need to be provided by the process to the decision service. The outcomes of the `Call incomplete files` and `Call after offers` decisions form the input requirement set needed for the correct invocation of the `Complete application` decision. Hence, this DMN model represents a decision service that is invoked in the trace cluster it pertains to. Note that in this model the information requirement arrows are coloured black, indicating that the trace cluster conforms to the discovered decision service, i.e. the **Service Adherence Criterion (SAC)** in Definition 9 is adhered to in terms of input requirements for the decisions that are invoked. Thus, the order of the decision activities in the process model conforms to the hierarchy as depicted in the decision service model.

On the other hand, the rightmost decision service in Figure 3 is not adhered to by the trace cluster which invokes that decision service. Notice that the arrows in this model are coloured red to indicate that the order of decision activities in the process model, representing the 625 traces cluster, violates the decision hierarchy demanded by the decision service model. Namely, the service requires the outputs of the `Application cancelled` and `Call after offers` decisions in order to invoke the `Complete application` decision. As shown in the corresponding process model in Figure 4, and the full process model that is provided online, this hierarchy is not respected and at the time of invocation of the decision service in the process, the necessary input requirement set is not available to the process. As a consequence, the process can not call upon the decision service in a proper manner and hence the decision model can not enact the invoked decision without the relevant input data. Thus, no sound decision outcome will be provided to the process, and the process might not be able to resume.

Next to the input requirement arrows in Figure 3 the number of traces that invoke the decision service are depicted. These traces are clustered according to the decision input and output propagation, as discussed in Section 8.1. Hence, the data propagation within the trace indicates which input requirement sets are passed on within the process and thus which decision services are invoked at specific points in the process. Thus, service-orientation and SoC provide a view into the intersection and interplay

2. https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

3. https://feb.kuleuven.be/public/u0111379/TSC/

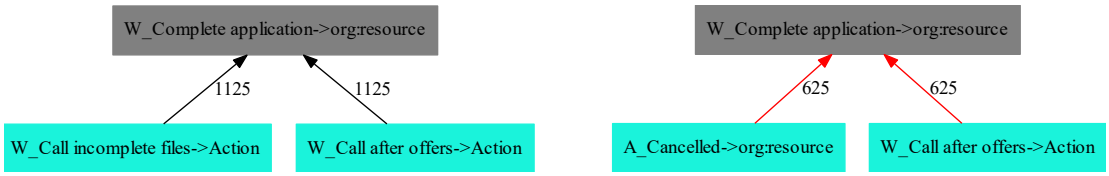

Figure 3: Discovered decision services.

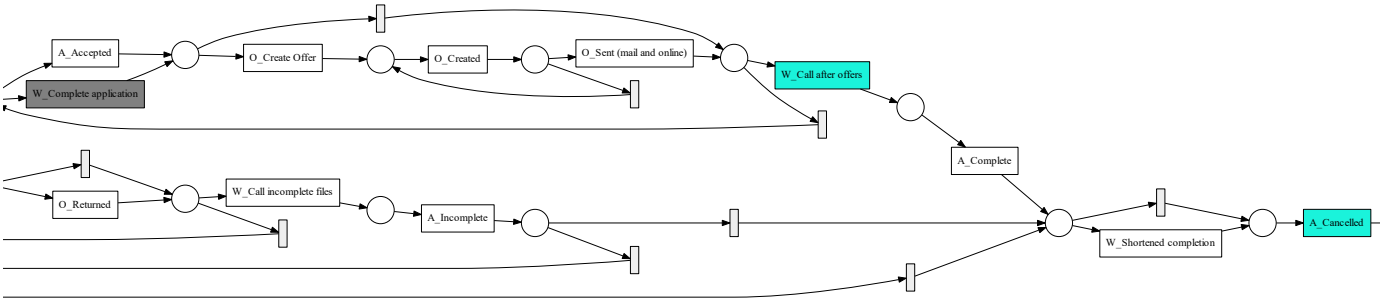

Figure 4: Fragment of the trace cluster not adhering to the decision service.

between data and processes, process modelling and decision modelling, process mining and classical data mining. Capitalising on the DaaS design, the advantages inherent to service-orientation can be exploited when separating multi-perspective modelling and mining tasks, such as the process perspective and the decision perspective.

9 Evaluating DaaS SOA Maturity

Maturity models can be used to assess the maturity of a system and thus to provide a roadmap towards a successful implementation of the system. For the purpose of SOA adoption, SOA maturity models have been proposed in literature. However, [45] point out that existing SOA maturity models are in most cases developed by vendors of SOA solutions and that they are therefore dependent on the specific products they are designed for. Hence, they propose an independent SOA Maturity Model (iSOAMM), i.e., a SOA maturity model that is independent of the used technologies and products. They develop SOA maturity model levels which are oriented at the capability of an SOA to support business processes. This means that a SOA with higher maturity possesses more features, which are useful within business processes. An overview of the iSOAMM maturity levels for the architectural viewpoint is provided in Figure 5.

Level 1: Trial SOA: This level recognises the existence of services. However, different services use incompatible technologies and standards. Hence, the services exhibit a lack of standardisation and they form a collection of unconnected service islands and not a true service-oriented architecture.

Level 2: Integrative SOA: This maturity level introduces a standardised service interface such that a high-level application, e.g., a business process, can use the interface to access the different services that are provided by the system.

Level 3: Administered SOA: This maturity level introduces service orchestration, i.e., it allows for composing several existing fine-grained services into a single higher order composite service. This modular service approach promotes the reuse and manageability of service components.

Level 4: Cooperative SOA: This level supports the choreography of processes, i.e., cooperation between processes. Additionally, human users are often vital to process execution support. Hence, choreography can be employed to close the gap between services on the one hand, and human users and (external) business processes on the other.

Level 5: On Demand SOA: In this maturity level, the static binding of the services is replaced with a dynamic binding of services at runtime, i.e., the service is looked up by name at runtime.

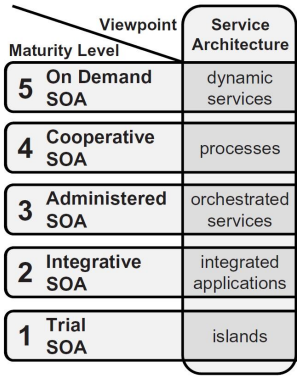

Figure 5: Independent SOA Maturity Model (iSOAMM) [45]

In what follows, we briefly explain the adherence to each maturity level and we apply the iSOAMM to the DaaS design proposed in this paper. The DaaS design conforms to maturity level 1 as it provides services. Since the design provides a standardised service interface in the form of Definition 6, and since services are developed according to Definition 5, using the same standard, i.e., DMN, the DaaS design conforms to maturity level 2 as well. Given the modular design of DMN decision models, as defined in Definitions 1 and 2, the designed decision services exhibit

modularity as well as they provide opportunities for decision service composition and decomposition as explained in Subsection 6.9. This ensures conformance to maturity level 3. Adherence to maturity level 4, i.e., cooperation between processes and human user support of processes, is not explicitly included in the DaaS design, as collaborative business processes were considered out of scope in this paper. However, the DaaS design allows for adding another layer that is concerned with communication between the process in the DaaS design and human actors or external processes. Furthermore, the process layer inherently allows for choreography between process and human users by employing pools and lanes within the process models. Finally, the DaaS design adheres to maturity level 5 as well, since it provides a decision on demand service to business processes, i.e., the services are invoked at runtime by the process by providing the decision service name that the process wants to invoke, together with the decision service input requirements set necessary for the invocation of the service.

10 Limitations of the DaaS Design

Organising process-aware information systems (PAIS) according to the SOA DaaS design requires system redesign. This induces limitations and obstacles as the processes in the legacy system did not follow the SOA and SoC paradigms. First, the decisions embedded in the process flows need to be externalised into a separate DMN decision model. This leads to the introduction of additional complexity because of new types of models that are becoming part of the system, i.e., the decision models. Furthermore, the business processes need to undergo redesign [46] to eliminate the decision construct that were present in the legacy process models. This is done to avoid ambiguity and overlap between decision constructs in the process model and the decision specified in the decision model. As such, the SoC paradigm between processes and decisions is instated. Additionally, the redesigned process models need to be compatible with the newly established decision models and hence the decision services derived from them. Thus, decision service consistency constraints apply according to the service adherence criterion when designing processes that need to call upon the decision logic through the decision services. Hence, a more complicated data propagation management within processes is inherent to the DaaS design, as opposed to designs where the SoC paradigm is not respected.

11 Conclusion and Future Work

In this paper we have contributed a *SOA* design for process- and decision-aware information systems, enhancing the understanding of the interaction between decisions and processes according to the *SoC* paradigm. The provided framework consists of a process layer, interface, service layer, and decision layer, making possible the implementation of decisions as services, or **Decision as a Service (DaaS)**. The processes can access the decisions through this DaaS architecture on demand, which we named **Decision on Demand (DoD)**. Furthermore, this paper formally defines the key concepts of DaaS and DoD and the proposed design is evaluated against key SOA characteristics, elucidating the benefits in terms of service abstraction and usefulness of the DaaS/DoD mechanism. Furthermore, implications of the proposed framework regarding integrated process-decision modelling are elaborated upon as well, demonstrating that the DaaS design greatly benefits the SoC between processes and decisions, thus advancing scalability, maintainability, flexibility and understandability. Additionally, we illustrated that the proposed DaaS design exhibits itself in real-life event logs by applying automatic decision service discovery on an enriched event log on a real-life bank loan application and approval process. Finally, the DaaS design was assessed in terms of SOA maturity, illustrating that the design conforms to the highest maturity levels.

In future work, it will be investigated how the proposed architecture can aid in solidifying the SoC in the modelling and mining of decisions and processes. Moreover, the modelling complexity of integrated models will be investigated and evaluated based on empirical studies. Finally, decision services can be of particular interest for Internet-of-Things (IoT) application areas [19], where multiple actors need to access a shared decision logic.

References

[1] J. Gordijn, H. Akkermans, and H. Van Vliet, "Business modelling is not process modelling," in *Conceptual modeling for e-business and the web*. Springer, 2000.

[2] J. Vanthienen, F. Caron, and J. De Smedt, "Business rules, decisions and processes: five reflections upon living apart together," in *Proceedings SIGBPS Workshop on Business Processes and Services (BPS'13)*, 2013, pp. 76–81.

[3] T. Biard, A. Le Mauff, M. Bigand, and J.-P. Bourey, "Separation of decision modeling from business process modeling using new decision model and notation(dmn) for automating operational decision-making," in *Working Conference on Virtual Enterprises*. Springer, 2015, pp. 489–496.

[4] H. van der Aa, H. Leopold, K. Batoulis, M. Weske, and H. A. Reijers, "Integrated process and decision modeling for data-driven processes," in *Business Process Management Workshops*, ser. LNBIP, vol. 256. Springer, 2015, pp. 405–417.

[5] F. Hasić, L. Devadder, M. Dochez, J. Hanot, J. De Smedt, and J. Vanthienen, "Challenges in refactoring processes to include decision modelling," in *Business Process Management Workshops*, ser. LNBIP. Springer, 2017.

[6] F. Hasić, J. De Smedt, and J. Vanthienen, "A service-oriented architecture design of decision-aware information systems: Decision as a service," in *On the Move to Meaningful Internet Systems*, ser. Lecture Notes in Computer Science. Springer, 2017.

[7] F. Hasić, L. Vanwijck, and J. Vanthienen, "Integrating processes, cases, and decisions for knowledge-intensive process modelling," in *International Workshop on Practicing Open Enterprise Modeling*. CEUR, 2017.

[8] A. Zarghami, B. Sapkota, M. Z. Eslami, and M. van Sinderen, "Decision as a service: Separating decision-making from application process logic," in *EDOC*. IEEE, 2012, pp. 103–112.

[9] A. Bock, H. Kattenstroth, and S. Overbeek, "Towards a modeling method for supporting the management of organizational decision processes," in *Modellierung*, ser. LNI, vol. 225. GI, 2014, pp. 49–64.

[10] F. Boumahdi, R. Chalal, A. Guendouz, and K. Gasmia, "Soaˆ\mathrm {+ d}: a new way to design the decision in soabased on the new standard decision model and notation (dmn)," *Service Oriented Computing and Applications*, vol. 10, no. 1, pp. 35–53, 2016.

[11] M. Mircea, B. Ghilic-Micu, and M. Stoica, "An agile architecture framework that leverages the strengths of business intelligence, decision management and service orientation," *Business Intelligence-Solution for Business Development*, 2011.

[12] E. Kornyshova and R. Deneckère, "Decision-making ontology for information system engineering," in *ER*, ser. LNCS, vol. 6412. Springer, 2010, pp. 104–117.

[13] SAP, "Decision service management," https://www.sap.com/products/decision-service-management.html.
[14] F. Hasić, J. De Smedt, and J. Vanthienen, "An Illustration of Five Principles for Integrated Process and Decision Modelling (5PDM)," KU Leuven, Tech. Rep., 2017.
[15] J. Hu, G. Aghakhani, F. Hasić, and E. Serral, "An evaluation framework for design-time context-adaptation of process modelling languages," in *Practice of Enterprise Modelling (PoEM)*, ser. Lecture Notes in Computer Science. Springer, 2017.
[16] OMG, "Decision Model and Notation 1.1," 2016.
[17] J. M. Perez-Alvarez, M. T. Gomez-Lopez, L. Parody, and R. M. Gasca, "Process instance query language to include process performance indicators in dmn," in *20th International Enterprise Distributed Object Computing Workshop*. IEEE, 2016, pp. 1–8.
[18] L. Ochoa and O. González-Rojas, "Analysis and re-configuration of decision logic in adaptive and data-intensive processes (short paper)," in *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*. Springer, 2017, pp. 306–313.
[19] F. E. Horita, J. P. de Albuquerque, V. Marchezini, and E. M. Mendiondo, "Bridging the gap between decision-making and emerging big data sources: an application of a model-based framework to disaster management in brazil," *Decision Support Systems*, vol. 97, pp. 12–22, 2017.
[20] B. Nuseibeh, J. Kramer, and A. Finkelstein, "A framework for expressing the relationships between multiple views in requirements specification," *IEEE Transactions on Software Engineering*, vol. 20, no. 10, pp. 760–773, 1994.
[21] IBM, "Php object orientation: Separating concerns, building more modular php applications," https://www.ibm.com/developerworks/library/os-php-objectorient/os-php-objectorient-pdf.pdf, 2009.
[22] F. Hasić, J. De Smedt, and J. Vanthienen, "Augmenting processes with decision intelligence: Principles for integrated modelling," *Decision Support Systems*, vol. 107, pp. 1 – 12, 2018.
[23] W. Van Der Aalst, "Service mining: Using process mining to discover, check, and improve service behavior," *IEEE Transactions on Services Computing*, vol. 6, no. 4, pp. 525–535, 2013.
[24] W. van der Aalst and E. Damiani, "Processes meet big data: Connecting data science with process science," *IEEE Transactions on Services Computing*, vol. 8, no. 6, pp. 810–819, 2015.
[25] J. De Smedt, F. Hasić, and J. Vanthienen, "Towards a holistic discovery of decisions in process-aware information systems," in *Business Process Management*, ser. Lecture Notes in Business Information Processing. Springer, 2017.
[26] F. Hasić, J. De Smedt, and J. Vanthienen, "Developing a modelling and mining framework for integrated processes and decisions," in *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems". OTM Workshops*, ser. Lecture Notes in Computer Science, vol. 10697. Springer, 2017, pp. 259–269.
[27] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. M. van der Aalst, "Data-driven process discovery-revealing conditional infrequent behavior from event logs," in *International Conference on Advanced Information Systems Engineering*. Springer, 2017, pp. 545–560.
[28] E. Bazhenova, S. Buelow, and M. Weske, "Discovering decision models from event logs," in *International Conference on Business Information Systems*. Springer, 2016, pp. 237–251.
[29] J. Campos, P. Richetti, F. A. Baião, and F. M. Santoro, "Discovering business rules in knowledge-intensive processes through decision mining: an experimental study," in *International Conference on Business Process Management*. Springer, 2017, pp. 556–567.
[30] S. Goedertier and J. Vanthienen, "Compliant and flexible business processes with business rules," in *Proceedings of the CAISE Workshop on Business Process Modelling, Development, and Support BPMDS*, 2006.
[31] J. Bae, L. Liu, J. Caverlee, L.-J. Zhang, and H. Bae, "A similarity measure for process mining in service oriented architecture," *Web Services Research for Emerging Applications: Discoveries and Trends: Discoveries and Trends*, pp. 87–103, 2010.
[32] W. Wei, M. Indulska, and S. Sadiq, "Guidelines for business rule modeling decisions," *Journal of Computer Information Systems*, pp. 1–11, 2017.
[33] J. Bae, L. Liu, J. Caverlee, L.-J. Zhang, and H. Bae, "Development of distance measures for process mining, discovery, and integration," *International Journal of Web Services Research*, vol. 4, no. 4, p. 1, 2007.
[34] W. Wang, M. Indulska, and S. Sadiq, "Factors affecting business process and business rule integration," in *Australian Conference on Information Systems*, 2014.
[35] C. Nagl, F. Rosenberg, and S. Dustdar, "Vidre–a distributed service-oriented business rule engine based on ruleml," in *10th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2006, pp. 35–44.
[36] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004.
[37] G. Bieber and J. Carpenter, "Introduction to service-oriented programming (rev 2.1)," *OpenWings Whitepaper, April*, 2001.
[38] T. Erl, *Soa: principles of service design*. Prentice Hall Press, 2007.
[39] H. Chesbrough and J. Spohrer, "A research manifesto for services science," *Communications of the ACM*, vol. 49, no. 7, pp. 35–40, 2006.
[40] T. Erl, *SOA design patterns*. Pearson Education, 2008.
[41] IBM, "Soa fundamentals in a nutshell," https://www.ibm.com/developerworks/webservices/tutorials/ws-soa-ibmcertified/ws-soa-ibmcertified.html, IBM.
[42] M. H. Valipour, B. AmirZafari, K. N. Maleki, and N. Daneshpour, "A brief survey of software architecture concepts and service oriented architecture," in *International Conference on Computer Science and Information Technology*. IEEE, 2009, pp. 34–38.
[43] IBM, "New to soa and web services," https://www.ibm.com/developerworks/webservices/newto/index.html.
[44] C. W. Günther and W. M. Van Der Aalst, "Fuzzy mining–adaptive process simplification based on multi-perspective metrics," in *International conference on business process management*. Springer, 2007, pp. 328–343.
[45] C. Rathfelder and H. Groenda, "Isoamm: An independent soa maturity model," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2008, pp. 1–15.
[46] F. Hasić, J. De Smedt, and J. Vanthienen, "Redesigning processes for decision-awareness: Strategies for integrated modelling," in *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*. IEEE, 2018, pp. 247–250.

Faruk Hasić is a PhD candidate in the Department of Information Management, Modelling and Simulation at KU Leuven. His interests lie in information systems engineering with an emphasis on the integrated process and decision engineering, with applications in the IoT domain. His research has been published in leading journals and conferences such as Decision Support Systems, International Conference on Business Process Management, and the International Conference on Cooperative Information Systems.

Johannes De Smedt obtained his PhD at KU Leuven and is currently a lecturer at the University of Edinburgh. His main interests include flexible business process modelling and mining and temporal item set mining. His research was published in leading journals such as Decision Support Systems, Expert Systems with Applications, and conferences such as Conference on Advanced Information Systems Engineering, and International Conference on Cooperative Information Systems.

Seppe vanden Broucke is an Assistant Professor at the Leuven Institute for Research on Information Systems (LIRIS) in the Department of Decision Sciences and Information Management at KU Leuven. Seppes research interests include business data mining and analytics, machine learning, process management, and process mining. His work has been published in well-known international journals and presented at top-tier computer science conferences.

Estefanía Serral is an Assistant Professor at KU Leuven. She has a highly international and interdisciplinary profile, with research in topics such as Internet of Things, ubiquitous business processes, and context-adaptive systems. In 2018, she was an Assistant Professor at TU/e (Netherlands). From 2012 to 2014, she was at the TU of Vienna (Austria). Until 2012, she worked at the TU of Valencia (Spain). Prof. Serral has many publications in high-ranking conferences and journals, e.g., CAiSE, ER, UIC, ESWA, SOSYM.