Towards Representing Processes and Reasoning with Process Descriptions on the Web

Andreas Harth
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
Fraunhofer Institute for Integrated Circuits IIS, Nürnberg, Germany

Tobias Käfer
Karlsruhe Institute of Technology (KIT), Germany

Anisa Rula
University of Brescia, Italy

Jean-Paul Calbimonte
University of Applied Sciences and Arts Western Switzerland HES-SO, Sierre, Switzerland
The Sense Innovation and Research Center, Lausanne, Switzerland

Eduard Kamburjan
University of Oslo, Norway

Martin Giese
University of Oslo, Norway

Abstract

We work towards a vocabulary to represent processes and temporal logic specifications as graph-structured data. Different fields use incompatible terminologies for describing essentially the same process-related concepts. In addition, processes can be represented from different perspectives and levels of abstraction: both state-centric and event-centric perspectives offer distinct insights into the underlying processes. In this work, we strive to unify the representation of processes and related concepts by leveraging the power of knowledge graphs. We survey approaches to representing processes and reasoning with process descriptions from different fields and provide a selection of scenarios to help inform the scope of a unified representation of processes. We focus on processes that can be executed and observed via web interfaces. We propose to provide a representation designed to combine state-centric and event-centric perspectives while incorporating temporal querying and reasoning capabilities on temporal logic specifications. A standardised vocabulary and representation for processes and temporal specifications would contribute towards bridging the gap between the terminologies from different fields and fostering the broader application of methods involving temporal logics, such as formal verification and program synthesis.

2012 ACM Subject Classification Information systems → Semantic web description languages; Theory of computation → Program semantics; Applied computing → Business process modeling; Applied computing → Event-driven architectures; Computing methodologies → Temporal reasoning; Computing methodologies → Ontology engineering

Keywords and phrases Process modelling, Process ontology, Temporal logic, Web services

Digital Object Identifier 10.4230/TGDK.2.1.1

Category Survey

Funding Harth acknowledges support from the German Federal Ministry of Education and Research via grant `MOSAIK` (01IS18070A), Käfer from the German Research Foundation (DFG) via grant `FOR 5339` (459291153), Rula from the M.U.R. Italian Minister of University within the PRIN 2022 Programme via grant `NEXTCART` (2022YXXZH5), Calbimonte from the Swiss National Science Foundation via the grant `StreamKG` (213369) and Kamburjan and Giese from the Research Council of Norway via grants `PeTWIN` (294600) and `SIRIUS` (237898). The article is based upon work conducted as part of the COST Action Distributed Knowledge Graphs (CA19134) supported by COST (European Cooperation in Science and Technology).

Acknowledgements We acknowledge the constructive comments of the anonymous reviewers. Harth acknowledges discussions over the years with Victor Charpenay, Stefan Decker, Dieter Fensel, Justus Fries, Stefan Jablonski and Daniel Schraudner regarding the formalisation of processes. Harth and Käfer acknowledge discussions with Brian Logan and the participants of the Dagstuhl Seminar 23081 Agents on the Web.

Received 2023-06-30 **Accepted** 2023-11-17 **Published** 2024-05-03

Editors Aidan Hogan, Ian Horrocks, Andreas Hotho, and Lalana Kagal

Special Issue Trends in Graph Data and Knowledge – Part 2

1 Introduction

Representing knowledge – describing what is true – is a mature field with a well-established terminology and existing standards. Knowledge representation with ontologies, using graph-structured representations, is standardised with W3C recommendations around the Resource Description Framework (RDF) [27], RDF Schema [17] and the Web Ontology Language OWL [11]. The languages specified by the W3C enjoy mature tool support, are widely applied in many different domains and have a clear connection with web architecture (Linked Data) to share and access data. The adoption of principles for sharing data in research, especially regarding findability, accessibility, interoperability and reusability (FAIR data [107]) provides organisational guidance. However, knowledge representation with ontologies has a focus on describing what is the case. Although OWL Time [26] provides primitives to represent time, support for representing changes and temporal aspects is often limited in the existing languages.

Representing processes – describing what is happening – is, in contrast, less standardised and there is less agreement on terminology and languages across fields. While many research fields are concerned with the temporal representation of processes and related concepts, each field has developed their own terminology and methods to address their requirements. Although many fields try to represent essentially the same concepts, terminology is often incompatible across approaches. Especially in the Semantic Web community, there is a lack of agreed-upon terminology and lack of actively-used vocabularies for representing processes and related concepts.

The literature on research around representing and reasoning with processes is vast and scattered across many fields. There is potential to bring hitherto unconnected languages, approaches and systems together and put sophisticated methods around model checking and planning into the hands of more users. Our aim is to try to capture the core of processes and process-related concepts, applicable to many different domains, by trying to distil essential aspects of events and processes.

We start with giving a brief historical overview of a selection of general works on events and processes from a selection of fields.

- Philosophy of Language and Linguistics: Publications from the field of philosophy of language and linguistics provide arguments for approaches to represent actions and events, but little in terms of mathematical formalisation. Linguistics and philosophy of language had notable publications starting in the 1950s and 1960s with a focus on understanding the nature of events. Vendler [105] investigates verb types and distinguishes temporal entities according to whether they occur gradually or instantaneously and whether they have an endpoint or not. States describe what is the case and last for a period of time (“A loved somebody from t_1 to t_2”), activities are ongoing and do not have an endpoint (“A was running at time t”), accomplishments are incremental and have an endpoint (“A was drawing a circle at t”), and achievements occur instantaneously and have an endpoint (“A won a race between t_1 and t_2”). Davidson investigates the structure of action sentences [28], assuming a universe that contains particular events. In a follow-up work, Davidson addresses a criticism regarding recurrence of events, *i.e.*, that certain events can occur multiple times [29] and discusses the possibility of events as universals. More recently, Galton argues that processes are "patterns of occurrences" [43] and thus treats processes as a kind of universal. The works in the field of philosophy of language and linguistics provide only rudimentary material that would lead to a mathematically rigorous formalisation that supports deductive reasoning.

- Temporal Logic and Formal Methods: Many of today's knowledge representation languages build on subsets or variants of first-order logic, often formalised using interpretations and models over a universe (or world). The introduction of modal logic brought ways to talk about necessity and possibility and truth relative to multiple worlds. The formalisation is based on Kripke structures (or transition systems). Prior developed Tense Logic [91] in the 1950s using modal operators to represent temporal aspects. Pnueli [90] introduced Linear-time Temporal Logic (LTL) in the 1970s and 1980s to represent and reason about the behaviour of computer programs. Computational Tree Logic (CTL) [40] is an alternative formalisation to represent and reason about the behaviour of (technical) systems. While LTL uses a linear time model, CTL uses a branching time model. Both temporal logics provide the basis for formally proving correctness of programs via model checking. While LTL and CTL typically operate over states expressed in propositional logic formulas, Abstract State Machines (ASMs) [53] (initially called Evolving Algebras) provide support for full first-order logic structures as state representation. ASMs have been used to formalise the semantics of programming languages, for instance. A related approach for formally treating the behaviour of systems is described by McDermott [76]. With the right descriptions of operations (with pre- and postconditions), a planner can synthesise a program (a sequence of operations) that reaches a given goal. Many of these approaches focus on representing the dynamics of systems as a progression of state. Other approaches focus on representing the behaviour and the interaction between processes. Communicating State Machines [16] formalise processes that can send and receive messages. Similarly, Communicating Sequential Processes [56] and Milner's work on a Calculus of Communicating Systems [77] and the π-calculus [78] have a focus on communication and interaction between processes.
- Business Processes and Workflows: While approaches from the philosophy of language and linguistics and from temporal logics and formal methods provide theory, the field of business processes and workflows apply process descriptions and executions in a business context. The focus in modelling business processes is on what is happening (events) and not what is true (states). That is, in the business process community [96], processes are primarily modelled as collections of activities (diverging from Vendler's definition of activity). Event-driven process chains [68] are an early approach to represent business processes. Often, Petri nets [88] are used to formalise the different business process and workflow languages. Process models (or workflows, *i.e.*, processes that have a unique source place and a unique end place [36]) use a "directly-follows" relation between activities to describe a process. MOBILE [58] provides a modular workflow model and architecture for implementing business processes in office environments and distinguishes between functional, behavioural, organisational and informational aspects. In the workflow field, researchers have identified patterns related to control flow and data (states) [96] to be able to catalogue the features of workflow languages and systems. More recently, approaches such as DECLARE [87] and Guard-Stage-Milestone (GSM) [57] break up the rather rigid description of traditional process models and encode relations and dependencies between temporal entities in a more flexible manner. Rather than just using the immediate-next relation to describe processes, these approaches allow for describing sets of "directly-follows" processes. Finally, in the last decades the field of process mining [103] has gained prominence, to re-discover the control flow from traces of process executions.

Next to the overview of general approaches from various fields, we survey the state of the art concerning representing processes and related concepts with ontological modelling in RDF in Section 2.

In this work we identify the main concepts that allow the formal representation of processes, which could later form the basis for a common vocabulary using Semantic Web standards. We try to be as general as possible, but focus our investigation on processes that can be executed and observed via a web interface. While our focus is on describing technical processes, we try to be generic enough to not explicitly exclude physical, chemical, biological or social processes. We believe that such a formalisation of processes can help going beyond description of physical and abstract entities, expanding towards the specification of changes, including temporal relationships and reasoning. To illustrate the different ways of how these process description concepts might be used, in Section 3 we provide a set of scenarios in which we highlight their respective main requirements, in terms of process modelling patterns as well as querying and reasoning tasks.

Our idea of process representation includes the definition of steps, events and other elements relative to process modelling, catering for both the state-centric and the event-centric perspective. We also identify the need to provide means for distinguishing between particulars (occurrences) and universals (events, states, processes) and keeping a trace of the events and states during the unfolding of a process. Moreover, we illustrate how the specification of temporal properties can be used to encode restrictions on the temporal order of process elements. We develop the core concepts required for a core process ontology in Section 4.

Our endeavour has the potential to address not only representation of events and processes but also operational and reasoning tasks, beyond previous attempts with mixed success, such as those related to Semantic Web Services [75, 94]. Thus, next to aligning terminology and vocabulary, the overall challenge is to bring the sophisticated methods from temporal logics and formal methods to the Semantic Web community. We could use process representations to query the graphs describing the processes (interface with systems around first-order logic, *e.g.*, a query processor or a reasoner), to query the temporal structure (interface with systems around temporal logic, *e.g.*, a model checker or a planner), or to generate and accept a sequence of events (interface with process engines or stream reasoners). Thus, we propose a research agenda focusing on the core properties needed to represent processes and related concepts and their semantics. We discuss the main challenges for representing processes on the web in Section 5.

To begin to structure the discussion of processes and related concepts, we have identified five dimensions, based on a combination of classifications found in the literature [96, 14]. While the work in [96] distinguishes control, data and resource, the approach in [14] distinguishes process, knowledge and trace. We believe that process/control and data/knowledge capture a similar notion, respectively. We also consider that [96] and [14] do not cover operational and architectural concerns. We want to distinguish between systems that operate offline (*e.g.*, on historical recordings of process executions) and those that operate online (*e.g.*, managing a set of currently running processes). See Table 1 for an overview.

The remainder of the paper is structured as follows. We start with a survey of the state of the art in Section 2. We continue with the presentation of scenarios in Section 3 and then introduce the assumptions and core concepts related to describing processes in Section 4. Next, we identify the main challenges for representing processes on the web in Section 5 before concluding in Section 6.

Table 1 Dimensions for representing processes and related concepts.

Dimension	**Description**
Control	The dimensions refers to the representation of the process control flow, *i.e.* the dependencies (temporal or otherwise) between different parts that make up a process. The most basic control flow construct is that of a sequence. Other constructs include branching conditions and repetitions. Note that control flow can apply to the level of particulars (*i.e.*, an occurrence of a process) as well as universals (*i.e.*, a process).
Data	The dimension refers to the representation data (*i.e.*, data objects, data items, knowledge) related to processes. The data related to the various steps could refer to the state or the lifecycle of the domain objects created or manipulated by the process at a given point in time. We assume the regular distinction between particulars (*i.e.*, instances) and universals (*i.e.*, classes) as in traditional knowledge representation languages, though not all approaches use an object-oriented paradigm for data.
Resource	The dimension refers to the resources that drive the progress of a process (*e.g.*, human workers, compute threads, organisation and roles). In the resource dimension, often the distinction between particulars and universals comes up again.
Trace	The dimension refers to the representation of "instances" of a process (*i.e.*, an occurrence of a process) and the associated history of evolution. A trace can contain both occurrences of events and states. A trace needs to connect the constituent elements, possibly via shared data objects with unique identifiers ('case ids').
Online	The dimension refers to the underlying system architecture. In particular we distinguish between online and offline processing. Information about processes and process executions can be available in batch for offline processing or can be available at runtime in a live system that makes control information available (*e.g.*, upcoming next, the process branches according to some conditions).

2 State of the art

Given that process representations are investigated in many fields, in the following we can only provide an overview of a selection of publications. Our litmus test for inclusion of a work in this section is that the work considers both processes *and* ontological modelling or data represented in RDF. Such works contain ideas, concepts and use-cases from fields such as Upper Ontologies, (Scientific) Workflow Provenance, Semantic Web Services, Stream and Event Processing, Formal Methods and Business Process Management.

We group the presentation of publications according to fields. We start each group with a description of the aim of the field and then present related works and contrast them with our proposed approach. For comprehensive surveys, see [93], [31] and [9]. We give an overview in Table 2.

2.1 Upper ontologies

Upper ontologies aim to describe general concepts in modelling such that domain ontologies built using these upper ontologies follow a principled approach and do not conflate (metaphysical) categories. Often, upper ontologies distinguish between continuants (endurants) and occurrents (perdurants). Continuants are "three-dimensional", *i.e.*, exist in space only, while occurrents are "four-dimensional", *i.e.*, can exist in space and time.

The notion of a process appears in domain-independent and upper ontologies. While in the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [15] itself does not specifically consider processes, other works use DOLCE to talk about processes and plans, *e.g.*,

Table 2 Ontological and RDF-based works around processes with ideas, concepts and use-cases from different fields. We give whether (+) or not (-) a work allows for the representation of **C**ontrol, **D**ata, **R**esource, **T**race or **O**nline dimension of processes. Note that a + does not imply that semantics are defined for the representations. For the BPM works, we give their foundation (e.ġ. standard, process calculus), if applicable.

Field	**Work**	**C**	**D**	**R**	**T**	**O**
Upper Ontologies	DOLCE [15]	-	-	-	-	-
	BFO [84]	-	-	-	-	-
	OntoUML/UFO-B [4]	+	-	+	-	-
	Schema.org [52]	+	-	-	-	-
(Scientific) Workflow Provenance	PROV-O [59]	-	-	-	+	-
	OPMW [47]	+	-	-	+	-
	ProvONE[6]	+	+	-	+	-
	ProvONE+ [18]	+	+	-	+	-
	PWO [44]	+	-	-	+	-
	Taverna [101]	+	-	-	+	-
	Pegasus [50]	+	-	-	+	-
	Kepler [5]	+	-	-	+	-
Semantic Web Services	Agarwal et al. [1] (π-calculus)	+	+	+	-	-
	OWL-S/DAML-S [8]	+	+	+	-	-
	WSMO [55]	+	+	+	-	-
Stream and Event Processing	RDF Stream Processors [13, 89, 20]	-	+	-	+	+
	Stream Reasoners [12, 80, 106]	-	+	-	+	+
	Complex Event Processors [7, 6]	+	+	-	+	+
Formal Methods	Program Ontologies [10, 72, 30]	+	-	-	+	-
	Data Access Tools [73, 85, 98, 71]	-	+	-	-	+
	Transition Systems with KGs [66, 22, 38]	+	+	+	-	+
	Model Checkers [109]	+	+	+	+	-
	Runtime Enforcement [64, 22]	+	+	+	-	+
	Runtime Checking [66, 38]	+	+	+	-	+
Business Process Management	BPMN ontology [95] (BPMN)	+	+	+	-	-
	BPMO [19] (BPEL/BPMN)	+	+	-	-	-
	Bertoli et al. [14] (BPMN)	+	+	-	+	-
	Koschmider et al. [69] (Petri Nets)	+	-	-	-	-
	Gasevic et al. [48] (Petri Nets)	+	-	-	-	-
	Thomas et al. [99] (EPC)	+	-	+	+	-
	WiLD [61]	+	+	-	-	+
	GSM4LD [63] (GSM/CMMN)	+	+	-	-	+

[83]. The Basic Formal Ontology (BFO) [84] contains a term for Process[1], and [97] details how to represent continuants (covering space) and occurrents (covering space and time). In the Unified Foundational Ontology part B (UFO-B), or UFO's incarnation OntoUML, events are considered [4],

[1] `http://purl.obolibrary.org/obo/BFO_0000015`

where an event is something that has a beginning and an end in time and where other things (specifically endurants) may participate. One could compare an event in UFO-B to an activity or process in the terminology of the other surveyed fields. Then, the participants in an event could be regarded as the resource dimension. In addition, the mereological (*i.e.*, part-of) and historical relations between events could be regarded as the control dimension, albeit with only very few language constructs.

In Schema.org [52], there are no terms to model processes. However, special kinds of processes can be modelled, such as a how-to description or a (cooking) recipe[2], which, at least in the JSON representation, can consider sequential flow. However, in the translation to RDF, the ordering gets lost, but there is an ongoing discussion to change that[3].

2.2 (Scientific) workflow provenance

Works in Scientific Workflows and Provenance are mainly concerned with modelling entities, activities that transformed them and agents who carry out the activities. Those works were developed first for the recording of the activity of scientific labs that process samples, *e.g.*, in biology.

The Open Provenance Model for Workflows (OPMW) is an ontology to describe the workflow traces and their templates of scientific processes [45]. Part of the OPMW work is the Provenance and Plans ontology [46] (P-Plan[4]), which extends the Provenance Ontology [59] (PROV-O[5]) with terminology to describe the plans (that is, the control flow) that govern the execution of scientific processes. P-Plan can be used to describe plans and establish a relationship between such plans and the provenance records in PROV-O that provide details with a focus on the entities and steps of past executions of scientific workflows. However, P-Plan does not capture elaborate control-flow constructs that are used in other types of workflows. A more recent approach is *ProvONE*[6] for the publication of scientific workflows [47]. ProvONE provides constructs to model workflow specifications and workflow execution provenance. The OPMW and ProvONE capture traces of workflow executions. They aim to provide detailed records of how each step is performed, which aligns with the trace dimension. *ProvONE+* is an extension of ProvONE that specifically addresses the requirements for capturing the provenance of control-flow-driven workflows [18], including sequence, parallel split and synchronisation in the control category. ProONE and ProvONE+, address aspects of the data dimension in terms of how data is managed and utilised in control-flow-driven workflows. Outside the domain of scientific lab work, the Publishing Workflow Ontology (*PWO*) is an ontology for the description of workflows in scientific publishing [44]. They use ontology design patterns to address modelling requirements of workflows. PWO includes sequence, parallel split, synchronisation, exclusive choice and simple merge.

Various systems have been proposed to provide the environment for specifying and enacting workflows such as Taverna and Kepler but each of them partially provides all the patterns in the control-flow category. Taverna [101] uses SCUFL (the simple conceptual unified flow language, a data-flow language) to specify control constraints for execution ordering and conditional constructs, which can encompass various control flow patterns such as if/else or case structures. Kepler [5] provides support for branching and synchronisation. Wings for Pegasus [50] is a workflow system that uses semantic representations to describe the constraints of the data and computational

[2] `http://schema.org/Recipe`
[3] `https://github.com/schemaorg/schemaorg/issues/1910`
[4] `http://purl.org/net/p-plan`
[5] `https://www.w3.org/TR/prov-o/`
[6] `http://purl.org/provone`

steps in the workflow. These works aim to fulfil various requirements related to representing workflows, capturing provenance information, describing plans and steps, creating workflow templates, accommodating domain-specific extensions and managing control flow. In provenance execution, the focus is on capturing the detailed record of the actual execution steps. However, some of these models have limitations in accurately and fully specifying control-flow patterns for scientific workflows. This limitation primarily impacts the prospective provenance, which includes the structure and static context of a workflow. However, their primary focus is not on representing data or instances of processes but on control and provenance aspects. Moreover, they do not consider the representation of resources like human resources, computational resources, or roles in the workflow.

2.3 Semantic web services

Works in Semantic Web Services aim to annotate Web Services in order to facilitate automated discovery, composition (*e.g.*, into a process), invocation and re-use of such services. Their work builds on the works in Web Services that typically come without such annotations.

When business processes (*e.g.*, BPMN) are used to describe choreographies and orchestrations of electronic business activities, even beyond organisational boundaries [79], their execution requires interoperability between the electronic interfaces. Interoperability standards emerged from these needs, which in the context of process integration led to the creation of Web Services. Web Services[7] allow for invoking operations tunnelled through HTTP POST requests. There is an entire family of standards, called 'WS-*', in the area of Web Services, which allow for, *e.g.*, service description or service discovery. To allow for the composition of Web Services, compositions via an orchestrator can be achieved using process languages such as BPEL[8] (also known as BPEL4WS or WS-BPEL). [79] discusses that the life cycle of process instances and data items can be regarded as state machines.

Beyond these functionalities, Semantic Web Services aim at describing services through machine understandable representations, allowing the automated discovery, composition, invocation and reuse of services on the Web. Different alternatives have been proposed for representing Semantic Web services most prominently: OWL-S (Web Ontology Language for Web Services)[9] and WSMO (Web Service Modelling Ontology)[10]. OWL-S (previously called DAML-S [8] – DARPA (Defense Advanced Research Projects Agency) Agent Markup Language for Services – consist of three main parts: an ontology describing what a service does; an ontology to describe the service process and an ontology to specify how to interact with the service. OWL-S contains terminology to describe (hierarchical) process models with the following control flow elements: Sequence, Split, Split-Join, Any-Order, Choice. The 'service grounding' aspect of OWL-S could be argued to hint at the resource aspect of processes. The input, output, precondition, effect of OWL-S's 'service profiles' loosely relate to the data aspect. WSMO [94] proposes a conceptual model for describing Web services, separating the goals or expectations of service requesters from the actual descriptions of the service features, capabilities and interfaces. WSMO also introduces the concept of mediators allowing the conciliation of potential discrepancies between services. Execution in the context of WSMO, WSMX (Web Service Execution Environment) [55], is event-based. The 'interface' aspect of WSMO could be argued to hint at the resource aspect of processes. The input, output, precondition, effect of WSMO's 'capability' loosely relate to the data aspect. Another approach

[7] `https://www.w3.org/2002/ws/`
[8] `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`
[9] `http://www.w3.org/Submission/OWL-S/`
[10] `http://www.w3.org/Submission/WSMO/`

from Semantic Web Services [1] contains a process language with formal semantics based on the π calculus. This language can express sequential, parallel ('composition') and conditional ('summation') control flow. The approach can model agents and services (corresponding to the resource aspect) next to involved ontologies (corresponding to the data aspect).

2.4 Stream and event processing

Stream processing addresses the problem of efficient managing and querying rapidly changing flows of data and events, typically augmented with time annotations. Hereby, time is a fundamental element for representing concepts such as order, simultaneity or concurrence of a given process. Systems, processing frameworks and query languages have been developed over the years, targeting the increasing demands of high-velocity data consumers, in multiple domains like healthcare monitoring, security surveillance, environmental sensing and the industrial Internet of Things.

Nevertheless, traditional stream processing often faces the challenge of handling heterogeneous data sources and thus different models, which are hard to integrate and reason upon. RDF stream processing, stream reasoning and complex event processors have been presented in the literature, addressing this challenge form different perspectives [34].

RDF stream modelling has been presented as an approach to extend RDF with explicit temporal annotations, in order to enable continuous query processing over events or stream items [34]. Following a graph-based representation with explicit semantics provides a richer modelling approach for data streams, also offering ways of overcoming semantic heterogeneity through streaming data integration, *e.g.*, through methods like ontology-based data access [21]. Existing modelling approaches like RDF Stream Processing in SPARQL (RSP-QL) [33], use point-in-time or interval semantics to represent time, defined at either the triple or graph level. Compatible implementations of continuous query processors for RDF streams include Continuous SPARQL (C-SPARQL) [13], Continuous Query Evaluation over Linked Streams (CQELS) [89], or SPARQLStream [20]. Processes in these RDF stream systems are often represented as states that may change over time (*e.g.*, sensor observations denoting the state of a monitored subject) and the queries are used to formalise specific types of instances or patterns – thus reflecting the control flow. A first attempt to formalise the description of RDF stream resources and endpoints is the Vocabulary and Catalog of Linked Streams (VoCaLS) [100], although it does not explicitly include the specification of streaming processes.

Stream reasoning has taken a step beyond continuous queries, exploring the possibility of exploiting the semantic relationships among events through reasoning tasks [34]. This sub-field has studied different approaches, many of which implement variations of incremental reasoning. View maintenance-inspired approaches include systems like the system by Volz et al. [106], which is based on the delete and re-derive (DReD) algorithm and in which axioms are added and removed according to the entailment rules. In these stream reasoners, processes and activities are not explicitly modelled, but such reasoners allow for reasoning tasks that involve the temporal structure. Such tasks typically allow for deriving new knowledge (*e.g.*, new events) as a result stream, thus generating new occurrences. Benchmarking and evaluation of different stream reasoning approaches is still an area under active exploration [51], given the heterogeneity of existing reasoning approaches and underlying temporal logics.

Semantic complex event processing (CEP) has studied the modelling and querying of streaming events, allowing the continuous querying of complex patterns. These patterns include, for example, finding sequences of events (*i.e.*, when an event e_1 is followed by another event e_2); or two evaluate if two events happen simultaneously [7]. CEP query languages like Event Processing SPARQL (EP-SPARQL) [6] allow using RDF triples to represent these events and add query language extensions that allow to express these temporal order and simultaneousness patterns, which represent the control flow among events. The language includes the SEQ operator to find sequences of triple patterns according to a time-based order. Similarly, the simultaneous presence of triple patterns can be detected using the EQUALS operator. Later on, these CEP operators were formalised and incorporated into the RSEP-QL model that combines both CEP and RSP query models [32]. In these CEP implementations, the activities and processes are reflected in the queries, where the expected order of sequences, patterns and other conditions are specified. While this is convenient for monitoring tasks and summarisation, the definition of explicitly defined processes could potentially facilitate and optimise the execution of CEP and reasoning query patterns in streaming environments.

2.5 Formal methods

Formal methods are formal techniques to model and analyse the behaviour of systems. Those techniques are concerned with modelling a general notion of process, *i.e.*, describing any sequence of state changes and the nature of these state changes. They can be applied to system designs or specifications as they can be found in engineered systems, *e.g.*, railway systems [42]. They can also be applied to more abstract systems, such as programs, where they form the basis to describe both program semantics [108] and program correctness [54], or indeed any kind of system that inhibits behaviour, including natural and sociological ones. As such, formal methods, in particular those targeting programs and program development, are closely related to process modelling, because both are concerned with describing possible and allowed sequences of states and events. Therefore, we believe that the rich toolkit of languages and tools from formal methods can successfully be applied in process modelling.

As we are eventually looking for a formal grounding for a core process ontology, which shall allow for reasoning about control flow and data as prescribed by a process, we also summarise related work that connects ontologies with formal methods to describe programs.

Programs are similarly concerned with describing control flow and data, but even though programs focus on *generating* traces, the ontologies describing programs and traces formalise similar ideas as process ontologies and can be a valuable source of inspiration, in particular for domain-aware simulation.

We subdivide our discussion of ontologies and programs into three groups, depending on their task: ontologies that describe programs and their output, ontologies that enable data access from a running program and approaches that tightly couple programs and ontologies that interact during program execution.

2.5.1 Describing programs and software systems

The following approaches are concerned with the syntax of a program, or the processes and the final traces generated by the program – during execution these systems are not applicable. On a purely syntactic level, Atzeni and Atzori [10] propose an ontology to describe the source code of Java programs, but are not concerned with the runtime semantics, *i.e.*, the generated traces.

On a higher level of abstraction, Diepenbrock et al. [35] focus on describing the architecture of a microservices system, while Oberle et al. [81, 82] give an ontology for general concepts in software systems, aiming for, among others, conceptual clarity for middleware. Lee et al. [72] elaborate an ontology of concepts in Java to help introduce them in courses to novice programmers, by connecting them to concepts from education. Similarly, de Aguiar [30] have produced an ontology for concepts in object-oriented programming. Such systems are not targeting to describe processes, but the concepts in the language and methodology in which the programs/processes are implemented.

As for the output of programs, the BOLD tool of Pattipati et al. [86] and Al Haider et al. [2] use ontologies to describe *execution traces*, *i.e.*, logs, of programs with the aim to simplify access to these quickly growing structures for static analysis [110]. In a more specific setting, Din et al. [37] describe a simulator for the geological process, where the final output trace is exported as a knowledge graph for easy query access by domain experts. If the program is the description of the process in question, or a part of the program, *e.g.*, a method, then the output is a description of the generated traces. The approach is, however, not suitable for the kind of modelling we envision, as it heavily tends towards low-level implementation details.

2.5.2 Enabling safe data access

To ensure safe interactions between ontologies and programs, one approach is to ensure that data loaded by some query language adheres to the type system of the program. These approaches focus exclusively on the relation of occurrence and data. This has been implemented to express loadable data by using SHACL (Shapes Constraint Language) shapes [73] or SPARQL query containment [65] as static approaches at compile time, or by generating classes in the programming language from OWL ontologies, see, *e.g.*, [85, 98, 71].

These approaches focus on RDF data, which may contain the description or parameters of a process serialised in it, but again focus on low-level implementation details, specifically the class system of, mostly, object-oriented languages.

2.5.3 Making knowledge available at runtime

The final task we examine is to make knowledge available at runtime. These works are, thus, concerned with processes (as procedures, etc.), occurrences (as runtime objects) data (as data types) and resources (as environment). While there are several languages that are concerned with connecting some kind of knowledge with programs, we focus on those that are designed to operate either on description logics models or on RDF graphs, or at least have an extension that does so. One such language is Golog [74] and its concurrent extension ConGolog [49], that uses first-order logic guards to examine and pick elements from its own state. Zarriess et al. [109] extend ConGolog to integrate description logic into a concurrent extension of Golog. They also investigate model checking against CTL (Computation Tree Logic). Model checking does not execute the program and, thus, has no occurrences. ConGolog is minimalistic and, thus, more suited for high-level modelling of workflows than low-level approaches.

A more direct connection between knowledge and transition is established by language where the knowledge, *e.g.*, as a knowledge graph, is external to the program state, but can be queried and manipulated using special operators. Fagin et al. [41] use epistemic operators to explicitly distinguish between what an agent believes and what is the case in the world, leading to the concept of *knowledge-based* programs. An extension, again using description logics to model knowledge, is presented by Calvanese et al. [22], which instead of learning focuses on revision [67]: actions change the (global) knowledge, which must then be revisited and repaired to remain consistent. The approach is interesting for modelling the behaviour of *'agents'* and is naturally concurrent, thus being another possible basis for modelling workflows in collaborative or competitive settings. This is akin to runtime enforcement, where an additional component fixes the trace during execution to ensure its some property, which Calvanese et al. understand as logical consistency.

The most direct connection is realised by semantically lifted programs [66], which interpret the runtime state as a knowledge graph using an external ontology and allow the program to access this knowledge graph at runtime. Semantically lifted programs support a modelling approach where knowledge and program data are unified, and a process is ontological and computational at the same time, depending on how it is accessed. Ontology-mediated programs [38] work similarly and interpret parts of the knowledge graph as program variables, but the user must provide the mapping from program variables themselves. Furthermore, their approach requires the number of instances to be known upfront. Both approaches allow for checking conditions at runtime to realise (internalised) runtime monitoring.

Finally, semantic lifting can also be used to enforce that a program adheres to domain knowledge by a universal guard to check that a transition will result in a consistent, lifted knowledge graph [64]. This is implementing runtime enforcement on a trace level.

2.6 Business process management

The field of Business Process Management is concerned with the modelling of processes in businesses with the aims of this modelling ranging from communicating how (*i.e.*, according to which process) things should get done in a business to automating the process using information systems.

Business Process Model and Notation (BPMN) is a visual language to represent processes[11]. The BPMN ontology provides a formal representation of the visual concepts defined in BPMN, such as steps, events, gateways and sequence flows [95]. The ontology in [95], as a representation of BPMN, covers the corresponding wide range of control flow patterns and allows for the specification of data requirements, such as input and output data and the assignment of roles or agents responsible for specific steps. The aim of the ontology was to faithfully represent the constraints of the visual modelling language, thus the necessary semantics for workflow execution is out of scope. The BPMN ontology has only limited support for modelling organisational or resource-related aspects of business processes.

The business process modelling ontology (BPMO) [19] has been developed to represent high-level business process workflow models, abstracting from existing notations and languages such as BPMN and BPEL (short for Business Process Execution Language[12]). BPMO focuses on the representation of process specification and includes various workflow elements such as tasks, events, block patterns (inspired by BPEL) and graph patterns (inspired by BPMN 1.0). These block and graph patterns serve as control flows, representing decision points within the workflow.

The work in [14] builds on [95] and also allows for modelling the data coming from process executions to enable querying and reasoning over the representations. The representations contain especially traces of past process executions next to the involved resources and data items. The authors use OWL reasoning (RDFS and OWL 2 RL) or SPARQL queries (with entailment regimes) to query processes, data and traces. The inferencing they consider is concerned with the resources that have been involved in specific traces.

Other works study the representation of Petri Nets using semantic technologies [69, 48]. Petri Nets are directed graphs consisting of *places* and *transitions* representing states and events/activities respectively. These works aim to enable the sharing of Petri nets in RDF or OWL. While the foundations of Petri Nets representations allow capturing different workflow patterns (*e.g.*, AND-Split, OR-Split, Loops, etc.), they are limited in their expressivity to the basic Petri Net language elements lacking consideration for extensions.

Another graphical modelling notation for business processes is the so-called event-driven process chain (EPC). EPC can be used to represent process models. Core language elements in EPC are functions (activities), events, control flow transitions, logical connectors (OR, AND, XOR) and resources. The authors of [99] investigate the semantic annotation of EPC process models.

[11] `http://www.omg.org/spec/BPMN/2.0/`
[12] `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`

Two approaches, WiLD (Workflows in Linked Data) and GSM4LD (Guard-Stage-Milestone for Linked Data), are designed to describe and execute workflows in Linked Data environments. WiLD [61] is a hierarchical process model [104], which allows for specifying sequential, parallel and conditional control flow. WiLD can also model process instances and together with its operational semantics in ASM4LD [60], workflows in WiLD can be executed. Correspondingly, the approach needs a notion of the data on which conditions are evaluated. A similar approach is taken in GSM4LD [63], which brings the Guard-Stage-Milestone (GSM) approach [57] (later standardised as CMMN – Case Management Model and Notation[13]) to Linked Data. Control in GSM is entirely determined data-driven, that is, via conditions evaluated on the environment and process-relevant milestones. Again, the operational semantics are given in ASM4LD. In contrast to GSM4LD, an execution of a WiLD workflow can give a light-weight process trace, by assigning finishing timestamps to finished activities, without additional modelling effort in the operational semantics. GSM4LD however supports cycles, thus to produce a complete trace would need extra data structures.

3 Scenarios

In this section, we provide an overview of various application areas that use processes and review the requirements outlined in these applications to capture various process aspects. These scenarios collectively cover a wide range of aspects, including control, data, resources, traces and architecture; the main task over these representations often relates to querying and reasoning over the representation of processes and related concepts. Additionally, we examine the features required of a process language, categorised based on *workflow patterns* [102] and we use WCPn to indicate patterns related to control and WDPm to indicate patterns related to data.

An aspect that was not explicitly mentioned in other works is that of hierarchies defined between processes. Concerning sub-process support the language should provide capabilities for defining a new relationship where the steps of a process include steps from other processes. We use H to indicate the hierarchical modelling pattern. These process hierarchies can be further divided into i) include, which indicates that a procedure is included in another procedure by specifying that the activities within a procedure are *inclusive* of the activities of another procedure. ii) extends, which identifies that a procedure presents a list of activities that are a variation of the activities of another procedure. iii) generalise, which is used in hierarchical process modelling to represent inheritance or specialisation relationships between processes.

3.1 Manufacturing process documentation

The *Manufacturing process documentation* use case involves the maintenance and adjustment of gear head lathe machinery[14] as described in a maintenance manual. The process consists of a sequence of steps that must be completed in a sequential order to address maintenance tasks and ensure optimal machine performance. It is crucial to record the order in which these steps are carried out. The initial step in the sequence involves a safety check of the machinery. Upon completion of the preceding step, the process enables the following steps, including instructions for adjusting the tension of motor belts, tailstock alignment, and spindle alignment. The spindle alignment step involves several sub-steps, including loosening the bearing lock nut, tightening the bearing adjustment nut, testing the spindle by rotating it, and finally re-tightening the bearing

[13] `https://www.omg.org/spec/CMMN/1.1/`
[14] This machinery features a gear-driven mechanism, crucial for various metalworking tasks commonly used in metalworking industries such as turning and threading.

locking nut, all contributing to the overall spindle alignment step. The process of maintaining and adjusting gear head lathe machinery involves a series of steps (such as safety checks, adjustments, and alignments) that transition the machinery between different states (such as operational, under maintenance, and calibrated).

Assumptions. For the use case we assume a graph-structured representation and basic control flow patterns.

Core concepts. In terms of **control** this applies since a sequential process with steps that must be completed in a specific order is described which may include patterns such as sequence (WCP1). **Data** could include information about the state of the machinery, measurements taken during maintenance, and any data related to the adjustments made thus including patterns such as case data (WDP5), and environment data (WDP8). In terms of **resources**, we may represent information about the individuals performing the maintenance and possibly the tools or equipment used as resources. In terms of hierarchies between processes, in this use case, we need to represent the include relationship between processes. Each time someone performs maintenance on the gear head lathe machinery, a system could generate a **trace** of the process, documenting how each step in the maintenance process was carried out, including the spindle alignment sub-steps. In the **online** dimension, should the trace be generated automatically, the use case would require some infrastructure to monitor the unfolding of the process.

Tasks. Queries can serve multiple purposes, including the retrieval of specific processes based on criteria like process name (*i.e.*, process retrieval), the extraction of specific steps or sub-steps from processes (*i.e.*, step extraction), and facilitating the comparison and analysis of multiple processes (*i.e.*, cross-process analysis). Users can query for common steps, shared sub-steps, or similarities/differences between processes, enabling the identification of reusable components, standardisation of processes, or the recognition of best practices across different processes. Reasoning plays a crucial role, contributing to tasks like process validation by employing logical reasoning rules to detect inconsistencies, contradictions, or missing steps within a process. Reasoning also supports process execution by determining the appropriate order of steps and handling dependencies (*i.e.*, process execution).

3.2 Manufacturing data spaces

The *Manufacturing data spaces* use case involves the cataloguing of data in manufacturing settings, where the goal is to catalogue the data sets collected during manufacturing, in order for data scientists to access the appropriate data for downstream data analytics. In this use case, the position in the manufacturing process must be part of the metadata of the data sets to consider this context during searches. The manufacturing process follows a stamp-forming flow, involving steps like heating, pre-forming, stamping, cooling, de-moulding, and reworking, with potential variations and additional pre- and post-processing steps. The **data** dimension encompasses the representation of specific data sets collected during manufacturing. As goods are manufactured, data is generated, and specific traces and data are recorded.

Assumptions. We assume a graph-based hierarchical process representation. The hierarchy allows for introducing variations on the lower levels of the process while keeping the high-level flow the same. Data analytics may involve the time spent on different event instances traces, hence time stamps are required.

Core concepts. In terms of **control**, various patterns are essential, including linear sequences (WCP1), case data (WDP5), environment data (WDP8), parallel split (WCP2), synchronisation (WCP3), and hierarchy/subprocesses (H). The aspect of **data** is relevant, as input and output materials and their occurrences are modelled. The aspect of **resource** is of interest, if the person who conducted the experiments is modelled. The aspect of **trace** is paramount in this scenario, as past experiments are recorded in traces for further querying. In this manufacturing data spaces scenario, the exclusion of the **online** dimension is justified by focusing on analysing past experiments rather than performing the analyses in real-time.

Tasks. The main task is querying traces. Queries focus on various aspects of the data and metadata generated during process instances, such as those related to products meeting key performance indicators (KPIs) or those executed by specific individuals or on particular days. Reasoning is limited in the use case, however a need can be foreseen to find the difference between two hierarchical process models.

3.3 Healthcare and tele-rehabilitation

The *Healthcare and tele-rehabilitation* use case involves modelling patient states following physical rehabilitation treatments, such as post-operative recovery, home-based reeducation or post-stroke physiotherapy. Traditional rehabilitation requires intensive intervention by healthcare professionals and may not adequately adapt to individual patient characteristics, context, or motivation. Digital rehabilitation addresses these challenges by offering personalised therapies – such as rehabilitation exercises – using sensors to monitor exercise performance and physiological markers. The patient's process is modelled as a set of states, with transitions triggered by their own decisions and influenced by a digital assistant's suggestions. Data gathered from wearable and environmental devices guide recommendations, such as resting, adjusting exercise intensity, or modifying the overall treatment based on the patient's characteristics, motivation, and emotional status.

Assumptions. The scenario assumes representation of events and processes as graphs, and temporal order among events related to the patients. For instance, exercises can be represented as events that follow a temporal order, *e.g.*, StartExercising SEQ ExerciseEasy would represent a sequence of two events: the patient starting to exercise, followed by an easy exercise.

Core concepts. The different steps in the use case can be represented as events (*e.g.*, EasyExercise), while states can also be used to represent activity stages of a patient (*e.g.*, resting). Thus, connections among states and events could be possible. For instance an expression as StopExercise $\rightarrow$ resting can represent temporal order among an event and a state. Moreover, occurrences are needed in order to characterise particular instances of events, *i.e.*, specific exercises performed by a specific patient.

Tasks. As part of the use case different tasks need to be performed. Among these, continuous queries have to be performed over the events captured by the motion and activity sensors. These queries may request sensor data related to exercise performance or the patient's physical status within specific time boundaries. These requests are expressed using SPARQL-based queries, for instance monitoring the number of exercise movements performed as well as the time spanned in and between exercises. Analysis of exercise progression and order can also be performed, *e.g.*, identifying the sequences of exercises over time. Furthermore, queries can also verify the sequence of correct and incorrect exercises performed during a session. These queries require the notion of *now*, *i.e.*, the moment used as a reference for computing time windows, and/or evaluating the occurrence of events over time. Query results can be continuously generated and used to create higher-level events through a stream reasoner, such as determining if a patient is fatigued based on movement quality, breathing and heartbeat. Additionally, the patient's treatment pathway can be modelled over time as a process, enabling the detection of non-compliance or continuous under-performance among patients, among other insights.

3.4 Domain-aware simulation

In the *Domain-aware simulation* use case, simulators must schedule and execute tasks in an initially unknown order to transform an initial state into a result state. To access this simulation result, they must also provide a means to query the result state and the sequence of executed tasks leading to it. A simulation task is modelling a state change according to some process in the domain of the simulation. Thus, they are inherently domain-specific and must have precisely described conditions for process initiation. These tasks can be modelled ontologically to allow the simulator to understand both their ontological description (conditions for process initiation) and computational description (changes needed to transform the current state).

While the domain modelling is often implicit in the simulation software, it can be made explicit. The simulator queries its state for possible steps at each simulation step, executes them in parallel for all parts, and analyses whether their changes trigger further tasks in the same simulation step. This can be done either by querying for newly enabled processes based on the state or by modelling dependencies between processes.

The geological simulator of Qu et al. [92], which simulates geological processes such as deposition of material and subsequent chemical process related to hydrocarbon generation, is an example where the domain modelling is made explicit: Such a domain-aware simulator relates its (internal) state to (external) geological descriptions, such as composition of geological layers, and its tasks to geological processes, such as deposition.

Assumptions. The use case assumes a graph representation of the domain, as well as a notion of sequence – while a transition is *realised* in the simulator, it must be *described and recorded* outside of it.

Core concepts. Concerning the representation of **control**, one must, beyond basic patterns like sequences (WCP1), express that all possible tasks must be checked for applicability in some order and may be executed concurrently (*e.g.*, Interleaved Parallel Routing WCP17, Interleaved-Routing WCP40) is necessary. For example, geological processes such as deposition can be concurrently active in different areas, such as erosion on two different mountains, or concurrently in the same area, such as heating of organic material and compaction, both due to increased pressure.

Representing **control** also affects data representation, as data may have cause-effect relationships between them, such as in Data-Based Routing (*e.g.*, Task Precondition Data - WDP35, Task Postcondition Data - WDP37, Data-based Task Trigger - WDP39), which should be able to interpret the simulated state as a knowledge graph.

The query tasks outlined below require an explicit representation of both the **trace**, especially for temporal properties of simulation runs and **data**, to examine the content of the simulation state at each point. In case of hyperproperties [24], *i.e.*, specification and comparison of multiple simulator runs, one also wants to represent, and query, multiple traces. It may, depending on the application, also be relevant to examine **resources**, *e.g.*, to examine the scheduling decisions of a concurrent simulator due to thread synchronisation or load balancing.

Tasks. The overall task involves executing the simulation and two additional reasoning and querying subprocesses after completion. First, it requires access to the final state and trace through queries to retrieve the simulation's results. Second, as intermediate steps, the simulator must execute membership queries to determine which tasks can run next. These queries may target either the current simulation state or the overall sequence of executed tasks, and they can express specifications over the simulation algorithm or domain-specific constraints on task sequences. In terms of the aforementioned geological simulator, such a query may retrieve all the parts of the state, where a certain geological process starts [92].

Beyond querying, one must be able to reason about the simulation runs and results with respect to the domain. This includes expressing temporal properties over single simulation runs to express temporal constraints. Potentially, one may need to investigate either *all* possible runs, *e.g.*, if certain parameters are not known, or compare multiple runs, to compare the effects of parameter changes. This is akin to model checking and expressing hyperproperties. Similarly, one may want to *query* multiple runs of the simulation.

3.5 Aircraft cockpit design

The *Aircraft cockpit design* use case focuses on ergonomic analyses of aircraft cockpits in virtual reality. Aircraft cockpits must be designed for the safe execution of pilot workflows. To support ergonomic analyses of aircraft cockpits during the early design phase when only CAD models are available, a workflow-aware analysis tool has been developed [62]. The tool enables the analysis and visualisation of pilot workflows in aircraft cockpits within a virtual reality environment. Hierarchical process models are employed to break down the processes pilots perform in aircraft cockpits. For example, take-off phases (*e.g.*, taxi, climb) are at a higher level, while physical pilot activities (*e.g.*, moving throttle, engaging autopilot) are at the leaf level. Data includes information related to aircraft cockpit design specifications, such as data structures in Virtual Reality derived from CAD models. During ergonomic analyses of aircraft cockpits, occurrences are generated to gain insights into the progress of pilot workflows and cockpit interactions.

Assumptions. We represent processes with hierarchies as graphs. The hierarchies allow us to cluster different activities according to domain practices, *e.g.*, to talk about certain phases of operation (taxi, climb) that involve different activities. Not all parts of the system are synchronised, but the process runtime aims to be fast to follow what happens in the different other subsystems. The process runtime does so by sending HTTP GET requests to the other subsystems in order to observe different states, which those subsystems are in. Thus, the state *now* can be determined by sending HTTP GET requests to all resources in all subsystems (or a subset thereof) and taking the union of the resource state representations in the responses.

Core concepts. In the process model, we tie together activities (*i.e.*, events). Conditions on system state allow to track the progress of occurrences and inform decisions which branch to follow in a conditional split. In terms of **control**, essential patterns include hierarchical modelling (H), sequential (WCP1), parallel (WCP2, WCP3), conditional (WCP4), any order (WCP40), and modelling conditions for splits and joins. Activity completion is determined using queries to state **data** (*e.g.*, SPARQL ASK queries) (WDP16, WDP35, WDP37, WDP40). Execution also requires modelling the tasks associated with activities (*e.g.*, unsafe HTTP requests) (WDP15). Similarly, conditions are evaluated on states. On top, execution requires support for **online** processing, *i.e.*, managing multiple process instances alongside models. **Resources** that drive the progress are not explicitly modelled, as there is only one agent (*i.e.*, the pilot) under consideration at a time. The aircraft simulation also in a way can drive the progress by corresponding state changes represented in the environment, but is also not explicitly modelled. The **trace** aspect has also only had a minor role, where timestamps have been attached to finished instances of activities.

Tasks. Querying is an integral part of the execution semantics, helping find tasks for activities to be executed or determining the next steps based on the control flow. RDFS Reasoning on the process model is required to ascertain when complex sub-processes have been completed or to identify the next task or all upcoming tasks within the hierarchical model's leaf level.

3.6 Summary of requirements for representation of processes

In addition to representing processes and related concepts to informally specify processes, as done for example in the various graphical notations, many scenarios require means to carry out tasks over the process descriptions. Thus, a formal representation of processes and related concepts is mandatory.

Table 3 gives an overview of the workflow patterns present in each scenario, providing a quick reference to understand how these patterns are distributed across the scenarios. Starting from the scenario *Manufacturing process documentation*, which covers only two workflow patterns, it is evident that the level of difficulty of each subsequent scenario increases. Therefore, as we progress to scenario *Domain-aware simulation*, a more complex use case, we can anticipate that it will have a broader coverage of workflow patterns. This incremental approach allows for a systematic exploration of workflow patterns, gradually building up complexity and understanding as we advance through the scenarios. Both control and data patterns are essential for designing comprehensive workflow models. They provide a structured way to represent the control logic and data interactions within a workflow.

3.7 Summary of requirements for querying and reasoning tasks

All scenarios assume that information about processes and related concepts are represented as graphs. The requirements of the previous scenarios regarding querying and reasoning can be grouped according to the five dimensions. Please note that the representation influences the kind of operations that can be applied on the representation (and vice versa). In the following, we thus outline and summarise typical operations that can be carried out over the state component, *i.e.*, mainly the data and resource dimensions in our classification, and typical operations that can be done on the temporal component, *i.e.*, mainly the control, trace and online dimensions.

If we only operate on the graph structure of the process descriptions, we can use standard graph-based data models (such as RDF) as basis for querying using SPARQL, possibly in conjunction with ontologies (RDFS or OWL). The requirements from the various scenarios regarding querying and reasoning are summarised and generalised in Table 4. In Table 4, we aim to simplify the presentation by concentrating on the requirements that directly align with the primary objectives addressed by the use case scenarios. Requirements can be further combined, particularly concerning the combination of process, data and online.

Next to performing reasoning on just the graph representation (*e.g.*, using RDF or RDFS semantics, using rules, using OWL DL (direct) semantics), some scenarios also require a way to perform reasoning on and with the temporal structure. Many graph query languages have means to deal with sequences, so basic retrieval and query tasks should be possible, including reachability. When applying reasoning, additional queries can be supported, for example for declaring a property transitive (*e.g.*, the next property) or declare two properties the inverse of each other (*e.g.*, next and previous).

More elaborate operations on temporal structure are better supported with dedicated languages of temporal logics. However, only the *Healthcare and tele-rehabilitation* and the *Domain-aware simulation* scenarios require query and reasoning tasks beyond the standard tasks on graph-structure representations. The scenario *Healthcare and tele-rehabilitation* is special concerning the

Table 3 Dimensions of processes and scenarios that require them. We give specific features for the dimensions. If there exists a corresponding Workflow Pattern for the feature in [102], we give the name and its shorthand (WxPn). Features without a workflow pattern are given a single-letter shorthand.

	ID	**Name**	**Scenario**
Control	WCP1	Sequence: steps are executed in a linear order, one after another.	Manufacturing process documentation (Sec. 3.1) Manufacturing data spaces (Sec. 3.2) Healthcare and tele-rehabilitation (Sec. 3.3) Domain-aware simulation (Sec. 3.4) Aircraft cockpit design (Sec. 3.5)
	WCP2/3	Parallel Split: the divergence of a branch into two or more parallel branches each of which executes concurrently. Synchronisation: activities are executed in parallel, allowing multiple paths of execution to occur simultaneously.	Manufacturing data spaces (Sec. 3.2) Aircraft cockpit design (Sec. 3.5)
	WCP4	Exclusive Choice: the divergence of a branch into two or more branches such that when the incoming branch is enabled, the thread of control is immediately passed to precisely one of the outgoing branches based on a selection mechanism.	Aircraft cockpit design (Sec. 3.5)
	WCP17	Interleaved Parallel Routing: a set of tasks has a partial ordering that defines the requirements for their execution order.	Domain-aware simulation (Sec. 3.4)
	WCP23	Transient Trigger: ability for a task instance to be triggered by a signal from another part of the process or the environment.	Healthcare and tele-rehabilitation (Sec. 3.3)
	WCP40	Interleaved Routing: each member of a set of tasks must be executed once. They can be executed in any order but no two tasks can be executed at the same time.	Domain-aware simulation (Sec. 3.4), Aircraft cockpit design (Sec. 3.5)
	H	Hierarchical Modelling	Manufacturing process documentation (Sec. 3.1) Manufacturing data spaces (Sec. 3.2) Aircraft cockpit design (Sec. 3.5)
Online	O	Online processing	Manufacturing process documentation (Sec. 3.1) Aircraft cockpit design (Sec. 3.5) Domain-aware simulation (Sec. 3.4)
Trace	T	Trace	Manufacturing process documentation (Sec. 3.1) Manufacturing data spaces (Sec. 3.2) Domain-aware simulation (Sec. 3.4)
Data	WDP1/3	Task data: data elements can be defined by tasks accessible only within the context of individual execution instances of that task. Scope data: data elements can be defined which are accessible by a subset of the tasks in a case.	Healthcare and tele-rehabilitation (Sec. 3.3)
	WDP5	Case Data: where data elements which are specific to a process instance or case are supported.	Manufacturing process documentation (Sec. 3.1) Manufacturing data spaces (Sec. 3.2)
	WDP8	Environment Data: allows data elements from the external operating environment to be accessed by components of processes during execution.	Manufacturing process documentation (Sec. 3.1) Healthcare and tele-rehabilitation (Sec. 3.3), Manufacturing data spaces (Sec. 3.2)
	WDP15/16	Task to Environment Pull: the ability of a task to request data elements from resources or services in the operational environment. Task to Environment Push: the ability of a task to initiate the passing of data elements to a resource or service in the operating environment.	Aircraft cockpit design (Sec. 3.5)
	WDP35/37	Task precondition Data Value: can be specified for tasks based on the value of specific parameters at the time of execution. Task postcondition data value: can be specified for tasks based on the value of specific parameters at the time of execution.	Domain-aware simulation (Sec. 3.4), Aircraft cockpit design (Sec. 3.5)
	WDP39	Data-based Task Trigger: provide the ability to trigger a specific task when an expression based on data elements in the process instance evaluates to true.	Domain-aware simulation (Sec. 3.4)
	WDP40	Data-based Routing: provides the ability to alter the control flow within a case based on the evaluation of data-based expressions.	Aircraft cockpit design (Sec. 3.5)
Resource	R	Resource	Manufacturing process documentation (Sec. 3.1) Manufacturing data spaces (Sec. 3.2)

Table 4 List of requirements for querying and reasoning, grouped according to the five dimensions (Control, Trace, Data, Resource and Online), and pointing to the scenario where they appear.

Dimension	Requirement	Scenario
Control	Procedure Retrieval	All
	Steps Retrieval	All
	Sub-procedure Retrieval	Manufacturing documentation (Sec. 3.1)
		Manufacturing data spaces (Sec. 3.2)
		Aircraft cockpit design (Sec. 3.5)
Trace	Sequence Analysis	All
	Cross-procedure Analysis	All
Data	Retrieval of Domain Data	All
Resource	Retrieval of Personnel or Equipment	All
	Resource Allocation	Healthcare and tele-rehabilitation (Sec. 3.3)
		Domain-aware simulation (Sec. 3.4)
Online	Occurrence Retrieval	Healthcare and tele-rehabilitation (Sec. 3.3)
		Domain-aware simulation (Sec. 3.4)
		Aircraft cockpit design (Sec. 3.5)

representation of time. While the other scenarios only require a linear ordering, the querying and reasoning tasks in the *Healthcare and tele-rehabilitation* scenario require temporal queries with an operator defined over moving windows (*i.e.*, different time periods). The need for temporal logic specifications is the most clear in the scenario *Domain-aware simulation*. The query and reasoning tasks for one trace alone must be able to reason about safety and liveness properties, while comparing two traces is a natural setting for temporal logics.

4 Process description concepts

We start with a set of assumptions to scope our work on a possible core process ontology. The process ontology core should support a graph representation and temporal order and should operate in a web environment. For each assumption, we first give an overview of the concepts underlying a representation of processes and related concepts and then introduce a set of tasks that operate on the representations.

- Graph Representation: We assume a graph representation to describe the concepts of a possible core process ontology. In particular, for the description of processes and related concepts, we assume graphs according to the RDF recommendation of the W3C. Graphs provide flexibility in the descriptions of processes and related concepts, especially allowing for partial descriptions that just cover certain aspects and for incrementally adding descriptions. Depending on what is described in the graph, we can bring different methods to bear on the representations.
- Time and Temporal Properties: Next, we assume that we need to describe temporal properties of processes and related concepts. To support temporal reasoning, we need means to qualify the descriptions in terms of time. As a starting point, we assume discrete metric time, that is, we assign to each temporal entity a timestamp as integer. We could also assume a weaker notion of time, linear time, with just a partially ordered relation "happened-before" between temporal entities. Thus, with metric time, we can arrange temporal entities along a single global time line, leading to a sequence of temporal entities. With linear time, we do not necessarily have a single global time line, but may arrive at multiple time lines, leading to multiple sequences (connected via a "directly-follows" relation).

- The Web: The final assumption relates to infrastructure. The technologies around the web are widely deployed and can provide the infrastructure for exchanging representations and executing processes. Thus, we assume that the processes and related concepts we consider operate in a web environment. Combining graph representations with web architecture, we arrive at graph representations that are accessible via HTTP (Linked Data). In addition, on the web, multiple processes may run concurrently and communicate with each other. Thus, processes must be able to send and receive requests during execution. If we want to support runtime with some form of online processes, we need a dedicated 'now' point in the time line, representing the current point in time.

We now develop some of the core concepts for the representation of processes. Representing processes can be approached from different perspectives, considering the application domain. Given that the various fields have different central concerns and methods, the terminologies differ from field to field. Thus, in the following we attempt to identify concepts shared among different fields, used as a common foundation to represent processes.

We introduce these main concepts by means of an example, in the style of linguistics [43]. Let us consider a simple version of a purchasing process in an e-commerce setting.

John bought a pair of socks online last week. He first placed the order, then he paid and finally he received the socks from the retailer.

The example consists of three steps: John placing an order, John adding payment and John receiving the shipment. The different steps occur in a certain temporal order. Rather than using dates and wall clock time, or more abstractly integers to refer to different points in time, we can also build on a weaker notion of time. We have one relation "happened-before" [70] that we use also later in the temporal specifications (for example, in linear-time temporal logic). Linear (total) ordering of temporal entities is sufficient for most of the approaches we consider[15]. Figure 3 illustrates the temporal ordering of the various steps, *i.e.*, the sequence in which the steps are occurring.

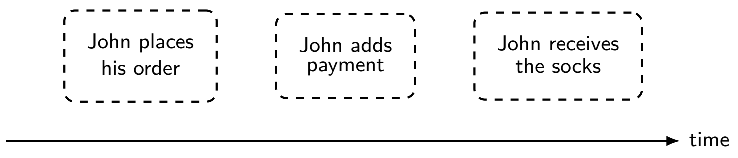

Figure 1 The different steps of the purchasing process happen in sequential order.

We assume temporal entities, *i.e.*, entities which have some attachment to time. Temporal entities could be distinguished into describing "what is being the case" (states) or "what is happening" (non-states). For now we assume that anything that can be put into a temporal order is a temporal entity.

Next to the different steps, in a state-centric perspective the purchasing process – or rather the data associated to the purchase – is going through various **states**. First, John's order is in the Placed state; next, John's order is in the Paid state; and finally, John's order is in the Received state.

The order object might also contain information about the product being purchased, in the example one pair of socks. Figure 2 illustrates the different states in a visual representation.

[15] Although stream reasoning requires a more expressive way to represent time.

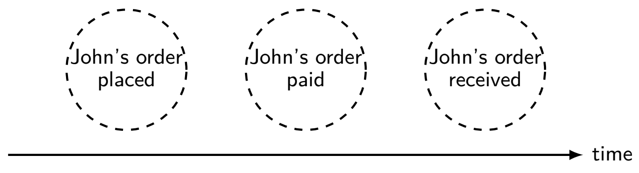

Figure 2 The object associated with the purchasing process goes through various states as the process unfolds.

We now discuss the classification of temporal entities as particulars or universals [93]. When representing state, an established way to data modelling is the distinction between classes and instances. Together with various logical modelling constructs, we can refer to ontology languages such as RDF Schema or OWL.

When it comes to non-states ("what is happening") we also want to make the distinction between particulars (the specific temporal entity where John ordered a pair of socks) and universals (the general temporal entity that describes all of the times when somebody is placing an order). Thus, we introduce the notion of **events**, which concern the level of universals, and **occurrences (of events)**, that concern the level of particulars. We denote as events the different steps (non-states, "what is happening") on a terminology or schema level. Figure 3 shows the three steps on the level of universals. In the example, an occurrence of the Place Order event would be John placing an order for a pair of socks. An occurrence of the Add Payment event would be John paying via bank transfer; and for the Receive Goods event, an occurrence would be John receiving the socks via mail.

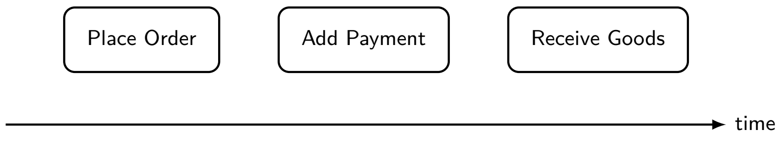

Figure 3 The different steps of the purchasing process happen in sequential order.

Similarly, we can consider the different states on the level of universals. First, an order object is in the Placed state; next, the object is in the Paid state; and finally, the object is in the Received state.

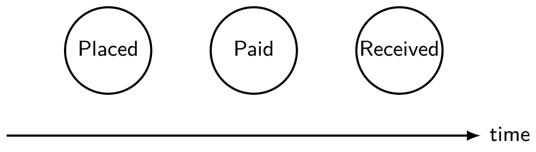

Figure 4 The object associated with the purchasing process goes through various states as the process unfolds.

To sum up, in event-centric approaches, the constituent elements are occurrences of events, while in state-centric approaches, the constituent elements are instances of objects. A trace needs to include information to connect the elements, *e.g.*, via a case identifier.

Each **process** is an event (and each occurrence of a process is an occurrence of an event). The distinction between a process and an event comes from whether the behaviour of the event is further described and separated into different (or events). Until now, we have only considered sequences within processes, but more elaborate control constructs are possible. For example, non-deterministic choice and parallelism (to cater for different agents that drive steps) are other control flow patterns that we can include into our process modelling approach.

So far, we have only assumed a temporal order in the events (or states) that make up a process. We now introduce the notion of a control flow that ties together the events. In the simplest case of sequences, we can tie together the events via a "next" relation (immediately-follows), indicating (in our example) that the next event after Place Order is Add Payment, and the next event after Add Payment is Receive Goods. Figure 5 illustrates in a graphical notation in the style of Petri nets. Circles represent states (related to places in Petri nets) and rectangular shapes represent activities (related to transitions in Petri nets). Other ways to string together events constituting a process are possible and are discussed later.

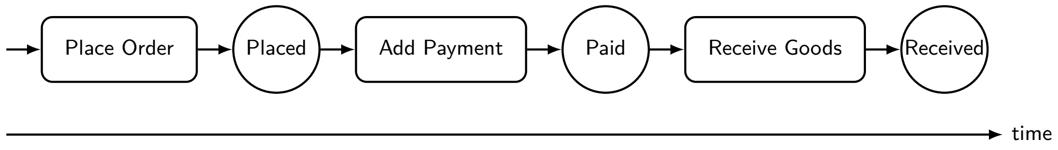

Figure 5 The temporal entities of the purchasing process in temporal order with connections indicating the control flow. Boxes denote events and circles denote states (in the style of Petri nets).

Business processes and workflows use the notion of an event log, *i.e.*, time-stamped occurrences of events together with an associated "case identifier". Instead of a process "instance", which is commonly used in the class/instance distinction, we introduce the concept of an **occurrence of a process**. To be able to capture the history of processes, we introduce the notion of **traces**. Traces record the specific occurrences of an event (and possibly of states) together with the temporal order in which they occurred.

Figure 6 shows a trace involving both the occurrences of events and the respective states.

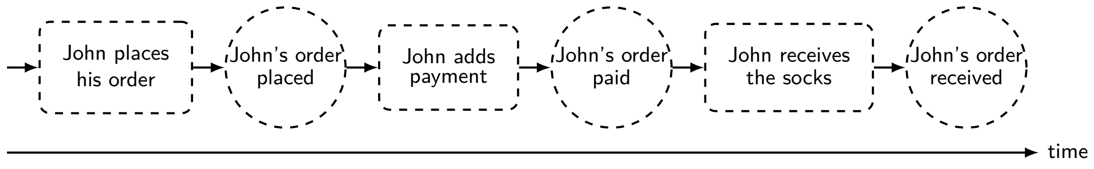

Figure 6 The temporal entities (occurrences of events and states) of the purchasing process occurrence represented as trace. Dashed boxes denote event occurrences and dashed circles denote state occurrences (in the style of Petri nets). Lines with arrows denote a temporal dependency.

Events might happen without an agent responsible for the event. In the business process community the non-state ("what is happening") entities are known as activities on a terminology or schema level; but the notion of activity implies some sort of agent that carries the activity out. Although in the example we can identify the different people (or roles) that carry out a step, in our conceptualisation we aim to be as general as possible and do not assume agents that are causing a step to happen.

Nevertheless, many scenarios require the identification of the agents (or tools) involved in a process. Thus, we introduce the notion of **resources**, that is, the agents being responsible for carrying out an event (or being patient/participant in an event). We can identify resources both at the level of particulars or universals (*e.g.*, John vs. customer role).

Next to sequences of "directly-follows" temporal entities, we can also introduce the notion of choice. Extending our example, we could say that the next event after Place Order is Add Payment via Bank Transfer or Add Payment via Credit Card.

For now, we do not further specify the conditions attached to the choice and assume random (*i.e.*, non-deterministic) choice. In addition, for a more complete description of a process, we could assume multiple processes that interact with each other. In the purchasing process example, we would have John's view on the process and the retailer's view. After John pays, he awaits delivery of the socks. After the retailer receives John's payment, the retailer ships John's socks. Thus, we need a way to represent multiple (possibly concurrent) processes.

A final construct that is commonly found in process representations is that of hierarchical decomposition, which implies a hierarchical relationship between a main process and other processes. In our example, we could decompose the Add Payment event into a choice between Add Payment via Bank Account and Add Payment via Credit Card. Furthermore, there can be different relationships in hierarchical modelling. The relationship may indicate either i) a dependency between a main process and other processes, *i.e.*, when the main process is executed, simultaneous execution of other processes is required in the meantime, or ii) a generalisation between a main process and other processes, *i.e.*, when there are variations in terms of the activities and the number of activities between the main process and the other processes. Thus, hierarchical modelling may facilitate the organisation of complex processes.

5 Challenges for a core process vocabulary

Based on the scenarios outlined above, we can identify a number of challenges for representing processes and related concepts and perform certain reasoning tasks over the representation. We structure our discussion into challenges related to representing processes as graphs, representing queries and formulas and reasoning with process descriptions and formulas.

5.1 Representing processes as graphs

A core process vocabulary should be broadly applicable to describe processes and represent a consensus from different fields on the abstractions and concepts as well as the terminology. All our use cases take a discrete-time perspective, which we can take as a fundamental assumption going forward. We have already illustrated why we need support for both state-centric and activity-centric perspectives. Similarly, all of our scenarios benefit from a representation based on graphs. We have started to work out required concepts from first principles and distil commonalities between different fields based on the five dimensions, which we will discuss further in the remainder of the section. Still, to arrive at a fixed set of vocabulary terms requires a deeper alignment of concepts and more study of the representations of processes in related fields.

In a process core vocabulary, we have to decide which terms to include and which terms to possibly relegate to extensions. That is, a process core vocabulary should be layered, where additional tasks can be supported if the underlying representation is gradually extended. In the control dimension, the question is what patterns to support next to the sequence pattern (WCP1) expressing temporal order. More patterns might be useful for a core vocabulary. Especially the any order pattern (WCP40) could be rather directly supported, as well as the exclusive choice pattern (WCP4), given that both patterns relate to choice and do not require parallelism. Parallelism (WCP2 and WCP3) might be a worthwhile pattern to support, especially when the core vocabulary should include support for multiple roles (see the resources dimension). A process core could support hierarchical modelling (the "H" pattern in Table 3) to be able to modularise the process representations and thus help to keep an overview in large and complex processes and provide a mechanism for iterative modelling. How to provide useful primitives for specialisation or generalisation is an open question.

Representing occurrences and traces of processes is also a rather fundamental pattern, so fundamental in fact that the business process community just silently assumes such a primitive without explicitly identifying the pattern. We think that the particular/universal distinction in the context of processes is important, mirroring the type/instance distinction in ontology languages. Thus, a core vocabulary should make the particular/universal distinction for events and processes as well. What effects the particular/universal distinction on the side of events and processes should have on the modelling of data items (and vice versa) is another open question.

Regarding the data dimension, the representation with graphs as syntax allows for more flexibility compared to a dedicated language with a grammar of its own. In particular, a graph representation allows for freely extending the graphs with additional data, such as annotations on the level of the process descriptions, and enables the possibility of querying and deductive reasoning. Thus, a core vocabulary should include support for data, in particular environment data (WDP8), and integration with graph-structured data to be able to support data integration. For example, mappings to other vocabularies (*e.g.*, PROV-O) could be provided.

The representation of agents or roles involved in the process is required. Whether it is enough to have the flexibility of graphs to allow for such descriptions, or whether we need dedicated terms as part of a process vocabulary remains to be seen.

5.2 Representing queries and formulas

Representing processes as graphs using a shared vocabulary provides already some benefits, such as performing query evaluation and certain reasoning tasks on the graph representation. The scenarios identified various possible querying and reasoning tasks. Overall, the challenge is that the different querying and reasoning tasks should be possible to perform on a unified representation. In particular, a major challenge is to combine the purely graph-structured process representations with specifications of temporal properties for querying and reasoning. To support querying and reasoning tasks that go beyond checking satisfiability on the graph representation, we require means to encode queries and logical formulas pertaining to temporal properties.

If we assume a framework of time based on temporal ordering ("happened-before"), we could use temporal logics that operate over the linear order for specification of temporal properties to encode restrictions on temporal order. In particular, Linear-time Temporal Logic (LTL) or Computational Tree Logic (CTL) are temporal logics that can operate over linear time (LTL) or branching time (CTL). A challenge is that formulas written in temporal logics directly can be difficult to understand. Thus, a core process vocabulary could use higher-level abstractions for temporal specification patterns, such as precedence and cause and effect [39]. Using such high-level specifications presents an opportunity to increase the flexibility when modelling processes or temporal properties of traces. For example, instead of directly representing processes (*e.g.*, first A, then directly followed by B), users could use (weaker) temporal properties to give a higher-level representation of temporal structure (*e.g.*, A precedes B). More research is required to investigate the trade-offs regarding modelling flexibility and computational complexity. An additional challenge using the formalisms of temporal logics is how to bring together the state perspective and event perspective, as temporal logic formalisms often prefer to specify properties on sequences of states (and not events) [23].

Another question for a core process vocabulary is whether we should go beyond the simple "happened-before" relation and assume a richer framework for time. For example, Allen's interval algebra [3] supports time points but also intervals and defines relations between time points and intervals. Similarly, stream reasoning assumes time stamps that can be used within window operators akin to intervals. An underlying temporal logic to capture such constructs could be Metric Temporal Logic (MTL). Again, the challenge is to decide on whether the increased expressive power is required and worth the computational cost.

In general, representation and reasoning will involve trade-offs depending on the underlying logics used: while some constructs could be written down using a process core vocabulary, performing operations (with a desired semantics) over such constructs might be computationally expensive or even infeasible. Ideally, we would like to find a representation that can in principle support a broad variety of tasks, with fragments or profiles for specific tasks, such as temporal querying, planning and synthesis, formal verification or process mining.

5.3 Reasoning with process descriptions and formulas

The range of operations for reasoning with process descriptions and formulas span from relatively basic queries on the graph structure to more intricate queries concerning the temporal structure of process representations. These advanced queries essentially encompass operations akin to model checking and formal verification.

Regarding operations on the graph structure, there are two overarching objectives: query answering, that is, acquiring solutions to queries, and reasoning, involving the inference of new formulas when given a set of formulas or a graph. RDF, RDFS, and various OWL profiles in conjunction with SPARQL support querying and lightweight reasoning on the graph representation of objects and their associated properties, in particular those in the data and resource dimension. These operations rely on standard semantics all underpinned by the formal notion of satisfiability. A possible avenue for further research is to identify whether and to which extent reasoning on the graph structure might yield simple inferences on the temporal structure, although the specifics of this interaction remain a topic of exploration.

Of particular interest is the provision of support for operations on the temporal structure. A fundamental operation relates to checking the satisfiability of temporal specifications, which can be performed on process descriptions or traces. In addition, other operations are conceivable, such as finding processes that satisfy temporal specifications, akin to planning or synthesis. Conversely, another operation would be process mining, *i.e.*, identifying a process that can generate a given set of occurrences.

As part of a specification of the semantics of a core process vocabulary, generic logic formulas could be introduced that operate on the temporal structure independently of the domain-specific elements. These formulas can capture intricate concepts like "happened-before" or the temporal ordering of events and occurrences. Furthermore, domain-specific formulas can encode restrictions specific to a given domain, such as the requirement that an order can only be shipped once the payment has been completed.

Another open challenge involves the interplay between the semantics of objects and data items and the semantics of the temporal structure. Related is the connection between particulars, representing instances and occurrences, and universals, embodying classes and events. Combining state and event perspectives, both on the levels of particulars and universals, presents a challenge for the development of a semantics, which requires the integration of state descriptions, represented as objects and their properties in graph form, with event descriptions that influence these states. Given an occurrence of a process in a specific state and a sequence of events, one should be able to derive the states of the related objects and data items. Conversely, it should be possible to work backward: starting with a sequence of states for the data items, one should be able to deduce the sequence of events that led the data items to those states. Moreover, the possible triggering of events needs to be evaluated over the state, with the introduction of a condition language to express "guards".

A graph-structured representation is advantageous for its flexibility in describing process-related knowledge. However, different use cases may require different semantic approaches. For instance, a data integration scenario could benefit from an open-world semantics, allowing partial process descriptions from multiple sources to be seamlessly combined. In contrast, a process execution use case may require a closed-world semantics, particularly because it necessitates making assumptions about the initiation and completion of processes and the comprehensiveness of each step within a process. Decisions surrounding whether to open or close the world depend on the specific tasks at hand.

A final open question relates to specialisation of processes involving hierarchy. In such cases, a closed-world assumption could become useful. A closed-world assumption would be equally applicable when working with prototypes [25], as opposed to the classic class-instance distinction based on sets. Prototypes could also support partial process descriptions that can later be amalgamated or refined. Issues may arise when one considers formalisms with open world and closed world side by side and interacting, both on the graph representation and with the temporal component.

A process core vocabulary which supports all conceivable tasks is likely to be too expressive and thus prohibitively expensive in terms of computational complexity. Thus, we probably want to end up with a semantics for the entire vocabulary and define profiles of the vocabulary of different expressive power suitable for the intended task. One challenge then relates to the identification and investigation of profiles, *i.e.*, subsets of the vocabulary, that support relevant reasoning tasks, while avoiding that parts of the vocabulary that have different meaning depending on the context.

6 Conclusion

We have introduced core concepts related to representing processes from different perspectives (state-centric and event-centric) that can be integrated into a combined perspective. Our aspiration is that a process core vocabulary should be more broadly applicable than to only business processes, especially given that such a vocabulary would use a graph representation that can be flexibly extended. We believe that the development of a process core vocabulary (the syntax for processes and related concepts as well as temporal properties) in a graph structure and the development of an associated formal semantics have to go hand in hand and influence each other.

We plan to start with a (minimal) core vocabulary for representing processes, which later can be extended to cater for more elaborate use cases. While currently only a subset of the scenarios we have described require querying and reasoning on the temporal structure, the core vocabulary should support the specification of temporal properties, to later enable querying and reasoning using such temporal property specifications. A formal semantics for the temporal structure would enable temporal querying and reasoning functionality on the graph-structured representations.

To connect with the established temporal logics formalisation, we require a combination of temporal specification patterns with the graph representation of processes, for example via extraction of temporal logic models from graphs. Such an approach would provide a separation of concerns and would reuse temporal logics languages and reasoning through a clear interface to the knowledge representation languages and reasoning systems. However, many questions regarding the specifics of such an integration remain open.

While the finite-state machine and traces formalisation as well as various temporal logics are well established, for the (non-technical) terms related to the level of knowledge representation (*e.g.*, for the resource dimension) there is currently no consensus across communities. Our paper is to be seen as a first suggestion of which concepts would go into a broadly applicable vocabulary for processes and related concepts as well as temporal properties. We plan to continue experimentation with process descriptions in a variety of use cases, both for representing processes and reasoning with process descriptions in first implementations based on existing systems (*i.e.*, model checkers, automated planners or process mining algorithms) to validate the representation in the vocabulary. We also encourage the community to experiment with process descriptions and report on their experiences.

References

1 Sudhir Agarwal, Sebastian Rudolph, and Andreas Abecker. Semantic description of distributed business processes. In *Proceedings of AI Meets Business Rules and Process Management*, pages 1–11. AAAI, 2008. URL: `http://www.aaai.org/Library/Symposia/Spring/2008/ss08-01-001.php`.

2 Newres Al Haider, Benoit Gaudin, and John Murphy. Execution trace exploration and analysis using ontologie. In *Proceedings of the Second International Conference on Runtime Verification (RV)*, volume 7186 of *LNCS*, pages 412–426. Springer, 2011. `doi:10.1007/978-3-642-29860-8_33`.

3 James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. `doi:10.1145/182.358434`.

4 João Paulo A. Almeida, Ricardo de Almeida Falbo, and Giancarlo Guizzardi. Events as entities in ontology-driven conceptual modeling. In *Proceedings of the 38th International Conference on Conceptual Modeling (ER)*, volume 11788 of *LNCS*, pages 469–483. Springer, 2019. `doi:10.1007/978-3-030-33223-5_39`.

5 Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludäscher, and Steve Mock. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, pages 423–424, 2004. `doi:10.1109/SSDM.2004.1311241`.

6 Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *Proceedings of the 20th International Conference on World Wide Web (WWW)*, pages 635–644. ACM, 2011. `doi:10.1145/1963405.1963495`.

7 Darko Anicic, Sebastian Rudolph, Paul Fodor, and Nenad Stojanovic. Stream reasoning and complex event processing in ETALIS. *Semantic Web Journal*, 3(4):397–407, 2012. `doi:10.3233/sw-2011-0053`.

8 Anupriya Ankolekar, Mark H. Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, Katia P. Sycara, and Honglei Zeng. DAML-S: semantic markup for web services. In *Proceedings of the First Semantic Web Working Symposium (SWWS)*, pages 411–430, 2001.

9 Amina Annane, Mouna Kamel, and Nathalie Aussenac-Gilles. Comparing business process ontologies for task monitoring. In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART)*, pages 634–643. SCITEPRESS, 2020. `doi:10.5220/0008978706340643`.

10 Mattia Atzeni and Maurizio Atzori. CodeOntology: RDF-ization of source code. In *Proceedings of the 16th International Semantic Web Conference (ISWC)*, volume 10588 of *LNCS*, pages 20–28. Springer, 2017. `doi:10.1007/978-3-319-68204-4_2`.

11 Jie Bao, Elisa Kendall, Deborah McGuinness, and Peter Patel-Schneider. OWL 2 Web Ontology Language Quick Reference Guide. Recommendation, W3C, 2009. Latest version available at `https://www.w3.org/TR/owl2-quick-reference/`. URL: `https://www.w3.org/TR/2009/REC-owl2-quick-reference-20091027/`.

12 Davide Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, Yi Huang, Volker Tresp, Achim Rettinger, and Hendrik Wermser. Deductive and inductive stream reasoning for semantic social media analytics. *IEEE Intelligent Systems*, 25(6):32–41, 2010. `doi:10.1109/MIS.2010.142`.

13 Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-SPARQL. In *Proceedings of the 18th International Conference on World Wide Web (WWW)*. ACM Press, 2009. `doi:10.1145/1526709.1526856`.

14 Piergiorgio Bertoli, Francesco Corcoglioniti, Chiara Di Francescomarino, Mauro Dragoni, Chiara Ghidini, and Marco Pistore. Semantic modeling and analysis of complex data-aware processes and their executions. *Expert Systems with Applications*, 198:116702, 2022. `doi:10.1016/j.eswa.2022.116702`.

15 Stefano Borgo, Roberta Ferrario, Aldo Gangemi, Nicola Guarino, Claudio Masolo, Daniele Porello, Emilio M. Sanfilippo, and Laure Vieu. DOLCE: A descriptive ontology for linguistic and cognitive engineering. *Applied Ontology*, 17(1):45–69, 2022. `doi:10.3233/AO-210259`.

16 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983. `doi:10.1145/322374.322380`.

17 Dan Brickley and Ramanathan Guha. RDF Schema 1.1. Recommendation, W3C, 2014. Latest version available at `https://www.w3.org/TR/rdf-schema/`. URL: `https://www.w3.org/TR/2014/REC-rdf-schema-20140225/`.

18 Anila Sahar Butt and Peter Fitch. ProvONE+: A provenance model for scientific workflows. In *Proceedings of the 21st International Conference onf Web Information Systems Engineering (WISE)*, volume 12343 of *LNCS*, pages 431–444. Springer, 2020. `doi:10.1007/978-3-030-62008-0_30`.

19 Liliana Cabral, Barry Norton, and John Domingue. The business process modelling ontology. In *Proceedings of the 4th International Workshop on Semantic Business Process Management (SBPM)*, pages 9–16. ACM, 2009. `doi:10.1145/1944968.1944971`.

20 Jean-Paul Calbimonte, Oscar Corcho, and Alasdair J. G. Gray. Enabling ontology-based access to streaming data sources. In *Proceedings of the 9th International Semantic Web Conference (ISWC)*, volume 6496 of *LNCS*, pages 96–111. Springer, 2010. `doi:10.1007/978-3-642-17746-0_7`.

21 Jean-Paul Calbimonte, Jose Mora, and Oscar Corcho. Query rewriting in RDF stream processing. In *Proceedings of the 13th European Semantic Web Conference (ESWC)*, volume 9678 of *LNCS*, pages 486–502. Springer, 2016. `doi:10.1007/978-3-319-34129-3_30`.

22 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Actions and pro-

grams over description logic knowledge bases: A functional approach. In *Knowing, Reasoning, and Acting: Essays in Honour of Hector J. Levesque*. College Press, 2011.

23 Marsha Chechik and Dimitrie O. Păun. Events in property patterns. In *Proceedings of 5th and 6th International SPIN Workshops*, volume 1680 of *LNCS*, pages 154–167. Springer, 1999. doi:10.1007/3-540-48234-2_13.

24 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *Proceedings of the Third International Conference on Principles of Security and Trust POST*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.

25 Michael Cochez, Stefan Decker, and Eric Prud'Hommeaux. Knowledge representation on the web revisited: the case for prototypes. In *Proceedings of the 15th International Semantic Web Conference (ISWC)*, volume 9981 of *LNCS*, pages 151–166. Springer, 2016. doi:10.1007/978-3-319-46523-4_10.

26 Simon Cox and Chris Little. Time Ontology in OWL. Recommendation, W3C, 2017. Latest version available at https://www.w3.org/TR/owl-time/. URL: https://www.w3.org/TR/2017/REC-owl-time-20171019/.

27 Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. Recommendation, W3C, 2014. Latest version available at https://www.w3.org/TR/rdf11-concepts/. URL: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

28 Donald Davidson. The logical form of action sentences. In *The Logic of Decision and Action*, pages 81–95. University of Pittsburgh Press, 1967. doi:10.1093/oso/9780195136975.003.0036.

29 Donald Davidson. Events and particulars. *Noûs*, pages 25–32, 1970. doi:10.2307/2214289.

30 Camila Zacché de Aguiar, Ricardo de Almeida Falbo, and Vítor E. Silva Souza. OOC-O: A reference ontology on object-oriented code. In *Proceedings of the 38th Conference on Conceptual Modeling*, volume 11788 of *LNCS*, pages 13–27. Springer, 2019. doi:10.1007/978-3-030-33223-5_3.

31 Wellington Moreira de Oliveira, Daniel de Oliveira, and Vanessa Braganholo. Provenance analytics for workflow-based computational experiments: A survey. *ACM Computing Surveys*, 51(3):53:1–53:25, 2018. doi:10.1145/3184900.

32 Daniele Dell'Aglio, Minh Dao-Tran, Jean-Paul Calbimonte, Danh Le Phuoc, and Emanuele Della Valle. A query model to capture event pattern matching in RDF stream processing query languages. In *Proceedings of the 20th European Knowledge Acquisition Workshop (EKAW)*, volume 10024 of *LNCS*, pages 145–162. Springer, 2016. doi:10.1007/978-3-319-49004-5_10.

33 Daniele Dell'Aglio, Emanuele Della Valle, Jean-Paul Calbimonte, and Oscar Corcho. RSP-QL semantics. *International Journal on Semantic Web and Information Systems*, 10(4):17–44, 2014. doi:10.4018/ijswis.2014100102.

34 Daniele Dell'Aglio, Emanuele Della Valle, Frank van Harmelen, and Abraham Bernstein. Stream reasoning: A survey and outlook. *Data Science*, 1:59–83, 2017. doi:10.3233/DS-170006.

35 Andreas Diepenbrock, Florian Rademacher, and Sabine Sachweh. An ontology-based approach for domain-driven design of microservice architectures. In *INFORMATIK*, volume P-275 of *LNI*, pages 1777–1791. Gesellschaft für Informatik, 2017. doi:10.18420/in2017_177.

36 Remco M Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software technology*, 50(12):1281–1294, 2008. doi:10.1016/j.infsof.2008.02.006.

37 Crystal Chang Din, Leif Harald Karlsen, Irina Pene, Oliver Stahl, Ingrid Chieh Yu, and Thomas Østerlie. Geological multi-scenario reasoning. In *Proceedings of the 12th Norwegian Information Security Conference*. NIK: Norsk Informatikkonferanse, 2019. URL: http://hdl.handle.net/11250/2633598.

38 Clemens Dubslaff, Patrick Koopmann, and Anni-Yasmin Turhan. Ontology-mediated probabilistic model checking. In *Proceedings of the 15th International Conference on Integrated Formal Methods (IFM)*, volume 11918 of *LNCS*, 2019. doi:10.1007/978-3-030-34968-4_11.

39 Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st International Conference on Software Engineering (ICSE)*, pages 411–420, 1999. doi:10.1145/302405.302672.

40 E Allen Emerson and Edmund M Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982. doi:10.1016/0167-6423(83)90017-5.

41 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. Knowledge-based programs. *Distributed Computing*, 10(4):199–225, 1997. doi:10.1007/s004460050038.

42 Alessio Ferrari and Maurice H. ter Beek. Formal methods in railways: A systematic mapping study. *ACM Computing Surveys*, 55(4):69:1–69:37, 2023. doi:10.1145/3520480.

43 Antony Galton. Processes as patterns of occurrence. In *Process, Action, and Experience*, pages 41–57. Oxford University Press, 2018. doi:10.1093/oso/9780198777991.003.0003.

44 Aldo Gangemi, Silvio Peroni, David M. Shotton, and Fabio Vitali. The publishing workflow ontology (PWO). *Semantic Web*, 8(5):703–718, 2017. doi:10.3233/sw-160230.

45 Daniel Garijo and Yolanda Gil. A new approach for publishing workflows: abstractions, standards, and linked data. In *Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science (WORKS)*, pages 47–56, 2011. doi:10.1145/2110497.2110504.

46 Daniel Garijo and Yolanda Gil. Augmenting PROV with plans in P-Plan: scientific processes as linked data. In *Proceedings of the Second International Workshop on Linked Science 2012 -*

Tackling Big Data, volume 951 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012. URL: `https://ceur-ws.org/Vol-951/paper6.pdf`.

47 Daniel Garijo, Yolanda Gil, and Óscar Corcho. Abstract, link, publish, exploit: An end to end framework for workflow sharing. *Future Generation Computer Systems*, 75:271–283, 2017. `doi:10.1016/j.future.2017.01.008`.

48 Dragan Gašević and Vladan Devedžić. Petri Net ontology. *Knowledge-Based Systems*, 19(4):220–234, 2006. `doi:10.1016/j.knosys.2005.12.003`.

49 Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000. `doi:10.1016/s0004-3702(00)00031-x`.

50 Yolanda Gil, Varun Ratnakar, Ewa Deelman, Gaurang Mehta, and Jihie Kim. Wings for Pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 1767–1774. AAAI Press, 2007. `doi:10.5555/1620113.1620127`.

51 Nathan Gruber and Birte Glimm. A comparative study of stream reasoning engines. In *Proceedings of the 20th European Semantic Web Conference (ESWC)*, volume 13870 of *LNCS*, pages 21–37. Springer, 2023. `doi:10.1007/978-3-031-33455-9_2`.

52 Ramanathan V. Guha, Dan Brickley, and Steve Macbeth. Schema.org: evolution of structured data on the web. *Communications of the ACM*, 59(2):44–51, 2016. `doi:10.1145/2844544`.

53 Yuri Gurevich. *Evolving algebras 1993: Lipari guide*, pages 9–36. Oxford University Press, 1995.

54 Reiner Hähnle and Marieke Huisman. Deductive software verification: From pen-and-paper proofs to industrial tools. In *Computing and Software Science*, volume 10000 of *LNCS*, pages 345–373. Springer, 2019. `doi:10.1007/978-3-319-91908-9_18`.

55 Armin Haller, Emilia Cimpian, Adrian Mocan, Eyal Oren, and Christoph Bussler. WSMX - A semantic service-oriented architecture. In *Proceedings of the 2005 IEEE International Conference on Web Services (ICWS)*, pages 321–328. IEEE Computer Society, 2005. `doi:10.1109/ICWS.2005.139`.

56 Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978. `doi:10.1145/359576.359585`.

57 Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno F. Terry Heath III, Stacy Hobson, Mark H. Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya, and Roman Vaculín. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proceedings of the 7th International Workshop on Web Services and Formal Methods (WS-FM)*, volume 6551 of *LNCS*, pages 1–24. Springer, 2010. `doi:10.1007/978-3-642-19589-1_1`.

58 Stefan Jablonski. Mobile: A modular workflow model and architecture. In *Working Conference on Dynamic Modelling and Information Systems*, 1994. URL: `https://www.researchgate.net/publication/2720558_MOBILE_A_modular_workflow_model_and_architecture`.

59 Ni Jing. A PROV-O based approach to web content provenance. In *Proceedings of the 2015 International Conference on Logistics, Informatics and Service Sciences (LISS)*, pages 1–6. IEEE, 2015. `doi:10.1109/LISS.2015.7369688`.

60 Tobias Käfer and Andreas Harth. Rule-based programming of user agents for linked data. In *Workshop on Linked Data on the Web (LDOW)*, volume 2073 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018. URL: `https://ceur-ws.org/Vol-2073/article-05.pdf`.

61 Tobias Käfer and Andreas Harth. Specifying, monitoring, and executing workflows in linked data environments. In *Proceedings of the 17th International Semantic Web Conference (ISWC)*, volume 11136 of *LNCS*, pages 424–440. Springer, 2018. `doi:10.1007/978-3-030-00671-6_25`.

62 Tobias Käfer, Andreas Harth, and Sebastien Mamessier. Towards declarative programming and querying in a distributed cyber-physical system: The i-VISION case. In *Proceedings of the Second International Workshop on Modelling, Analysis, and Control of Complex CPS (CPSData)*, pages 1–6. IEEE, 2016. `doi:10.1109/CPSData.2016.7496418`.

63 Tobias Käfer, Benjamin Jochum, Nico Aßfalg, and Leonard Nürnberg. Specifying and executing user agents in an environment of reasoning and RESTful systems using the guard-stage-milestone approach. *Journal on Data Semantics*, 10(1-2):57–75, 2021. `doi:10.1007/s13740-021-00123-0`.

64 Eduard Kamburjan and Crystal Chang Din. Runtime enforcement using knowledge bases. In *Proceedings of the International Conference on Fundamental Approaches to Software Engineering FASE*, volume 13991 of *LNCS*, pages 220–240. Springer, 2023. `doi:10.1007/978-3-031-30826-0_12`.

65 Eduard Kamburjan, Vidar Norstein Klungre, and Martin Giese. Never mind the semantic gap: Modular, lazy and safe loading of RDF data. In *Proceedings of the 19th European Semantic Web Conference ESWC*, volume 13261 of *LNCS*, pages 200–216. Springer, 2022. `doi:10.1007/978-3-031-06981-9_12`.

66 Eduard Kamburjan, Vidar Norstein Klungre, Rudolf Schlatte, Einar Broch Johnsen, and Martin Giese. Programming and debugging with semantically lifted states. In *Proceedings of the 18th European Semantic Web Conference ESWC*, volume 12731 of *LNCS*, pages 126–142. Springer, 2021. `doi:10.1007/978-3-030-77385-4_8`.

67 Hirofumi Katsuno and Alberto O. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 387–394. Morgan Kaufmann, 1991. `doi:10.1017/cbo9780511526664.007`.

68 Gerhard Keller, Markus Nüttgens, and August-Wilhelm Scheer. *Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)"*. Veröffentlichungen des Instituts für Wirtschaftsinformatik. - Saarbrücken : IWI, ISSN 1438-5678. - Vol. 89, 1992.

69 Agnes Koschmider and Andreas Oberweis. Ontology based business process description. In *Proceedings of the Open Interop Workshop on Enterprise Modelling and Ontologies for Interoperability*, volume 160 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005. URL: `https://ceur-ws.org/Vol-160/paper12.pdf`.

70 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 1978. `doi:10.1145/359545.359563`.

71 Jean-Baptiste Lamy. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in Medicine*, 80:11–28, 2017. `doi:10.1016/j.artmed.2017.07.002`.

72 Ming-Che Lee, Ding Yen Ye, and Tzone I. Wang. Java learning object ontology. In *Proceedings of the 5th International Conference onf Advanced Learning Technologies ICALT*, pages 538–542. IEEE, 2005. `doi:10.1109/icalt.2005.185`.

73 Martin Leinberger. *Type-safe Programming for the Semantic Web*. PhD thesis, University of Koblenz and Landau, 2021. `doi:10.3233/ssw52`.

74 Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *The Journal of Logical Programming*, 31(1-3):59–83, 1997. `doi:10.1016/s0743-1066(96)00121-5`.

75 David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katja Sycara. Bringing semantics to web services: The OWL-S approach. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, volume 3387 of *LNCS*, pages 26–42. Springer, 2005. `doi:10.1007/978-3-540-30581-1_4`.

76 Drew McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6(2):101–155, 1982. `doi:10.1207/s15516709cog0602_1`.

77 Robin Milner. *A calculus of communicating systems*. Springer, 1980. `doi:10.1007/3-540-10235-3`.

78 Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge University Press, 1999.

79 Michael zur Muehlen, Jeffrey V. Nickerson, and Keith D. Swenson. Developing web services choreography standards - the case of REST vs. SOAP. *Decision Support Systems*, 40(1):9–29, 2005. `doi:10.1016/j.dss.2004.04.008`.

80 Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. RDFox: A highly-scalable RDF store. In *Proceedings of the 14th International Semantic Web Conference (ISWC)*, volume 9367 of *LNCS*, pages 3–20. Springer, 2015. `doi:10.1007/978-3-319-25010-6_1`.

81 Daniel Oberle. How ontologies benefit enterprise applications. *Semantic Web*, 5(6):473–491, 2014. `doi:10.3233/sw-130114`.

82 Daniel Oberle, Stephan Grimm, and Steffen Staab. An ontology for software. In *Handbook on Ontologies*, pages 383–402. Springer, 2009. `doi:10.1007/978-3-540-92673-3_17`.

83 Daniel Oberle, Steffen Lamparter, Stephan Grimm, Denny Vrandecic, Steffen Staab, and Aldo Gangemi. Towards ontologies for formalizing modularization and communication in large software systems. *Applied Ontology*, 1(2):163–202, 2006. URL: `http://content.iospress.com/articles/applied-ontology/ao016`.

84 J. Neil Otte, John Beverley, and Alan Ruttenberg. BFO: basic formal ontology. *Applied Ontology*, 17(1):17–43, 2022. `doi:10.3233/AO-220262`.

85 Alexander Paar. *Zhi# - programming language inherent support for ontologies*. PhD thesis, Karlsruhe Institute of Technology, 2009. `doi:10.5445/IR/1000019039`.

86 Dileep Kumar Pattipati, Rupesh Nasre, and Sreenivasa Kumar Puligundla. BOLD: an ontology-based log debugger for C programs. *Automated Software Engineering*, 29(1):2, 2022. `doi:10.1007/s10515-021-00308-8`.

87 Maja Pesic, Helen Schonenberg, and Wil MP Van der Aalst. DECLARE: Full support for loosely-structured processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 287–300. IEEE, 2007. `doi:10.1109/EDOC.2007.14`.

88 Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, 1962.

89 Danh Le Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *Proceedings of the 10th International Semantic Web Conference (ISWC)*, volume 7031 of *LNCS*, pages 370–388. Springer, 2011. `doi:10.1007/978-3-642-25073-6_24`.

90 Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS)*, pages 46–57. IEEE, 1977. `doi:10.1109/SFCS.1977.32`.

91 Arthur N Prior. *Time and modality*. Oxford University Press, 1957. `doi:10.2307/2216989`.

92 Yuanwei Qu, Eduard Kamburjan, Anita Torabi, and Martin Giese. Semantically triggered qualitative simulation of a geological process. *Applied Computing and Geosciences*, 21:100152, 2024. `doi:10.1016/j.acags.2023.100152`.

93 Fabrício Henrique Rodrigues and Mara Abel. What to consider about events: A survey on the ontology of occurrents. *Applied Ontology*, 14(4):343–378, 2019. `doi:10.3233/ao-190217`.

94 Dumitru Roman, Uwe Keller, Holger Lausen, Jos De Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005. `doi:10.1007/978-3-642-19193-0_7`.

95 Marco Rospocher, Chiara Ghidini, and Luciano Serafini. An ontology for the business process modelling notation. In *Proceedings of the 8th International Conference on Formal Ontology in Information Systems (FOIS)*, volume 267 of *Frontiers in Artificial Intelligence and Applications*,

pages 133–146. IOS Press, 2014. doi:10.3233/978-1-61499-438-1-133.

96 Nick Russell, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. *Workflow Patterns: The Definitive Guide*. MIT Press, 2016. doi:10.7551/mitpress/8085.001.0001.

97 Barry Smith. Classifying processes: an essay in applied ontology. *Classifying Reality*, pages 101–126, 2013.

98 Graeme Stevenson and Simon Dobson. Sapphire: Generating Java runtime artefacts from OWL ontologies. In *Proceedings of the CAiSE Workshops*, volume 83 of *Lecture Notes in Business Information Processing*, pages 425–436. Springer, 2011. doi:10.1007/978-3-642-22056-2_46.

99 Oliver Thomas and Michael Fellmann. Semantic EPC: enhancing process modeling using ontology languages. In *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM)*, volume 251 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007. URL: https://ceur-ws.org/Vol-251/paper9.pdf.

100 Riccardo Tommasini, Yehia Abo Sedira, Daniele Dell'Aglio, Marco Balduini, Muhammad Intizar Ali, Danh Le Phuoc, Emanuele Della Valle, and Jean-Paul Calbimonte. VoCaLS: Vocabulary and catalog of linked streams. In *Proceedings of the 17th International Semantic Web Conference (ISWC)*, volume 11137 of *LNCS*, pages 256–272. Springer, 2018. doi:10.1007/978-3-030-00668-6_16.

101 Daniele Turi, Paolo Missier, Carole A. Goble, David De Roure, and Tom Oinn. Taverna workflows: Syntax and semantics. In *Proceedings of the Third International Conference on e-Science and Grid Computing*, pages 441–448. IEEE, 2007. doi:10.1109/E-SCIENCE.2007.71.

102 Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003. doi:10.1023/A:1022883727209.

103 Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004. doi:10.1109/TKDE.2004.47.

104 Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The refined process structure tree. In *Proceedings of the 6th International Conference on Business Process Management (BPM)*, volume 5240 of *LNCS*, pages 100–115. Springer, 2008. doi:10.1007/978-3-540-85758-7_10.

105 Zeno Vendler. Verbs and times. *The Philosophical Review*, 66(2):143–160, 1957. doi:10.2307/2182371.

106 Raphael Volz, Steffen Staab, and Boris Motik. Incrementally maintaining materializations of ontologies stored in logic databases. *Journal on Data Semantics II*, pages 1–34, 2005. doi:10.1007/978-3-540-30567-5_1.

107 Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, et al. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3(1):1–9, 2016. doi:10.1038/sdata.2016.18.

108 Glynn Winskel. *The formal semantics of programming languages - An introduction*. MIT Press, 1993. doi:10.7551/mitpress/3054.001.0001.

109 Benjamin Zarrieß and Jens Claßen. Verification of knowledge-based programs over description logic actions. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 2015. doi:10.25368/2022.216.

110 Yue Zhao, Guoyang Chen, Chunhua Liao, and Xipeng Shen. Towards ontology-based program analysis. In *Proceedings of the 30th European Conference on Object-Oriented Programming (ECOOP)*, pages 26:1–26:25, 2016. doi:10.4230/LIPIcs.ECOOP.2016.26.