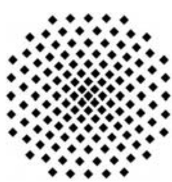
View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Online Publikationen der Universität Stuttgart



Universität Stuttgart

Fakultät Informatik, Elektrotechnik und Informationstechnik

A Language-Agnostic Framework for the Analysis of the Syntactic Structure of Process Fragments

Michele Mancioppi, Olha Danylevych,
Mike P. Papazoglou, Frank Leymann

Report 2010/07

Institut für Architektur von Anwendungssystemen

Universitätsstraße 38
70569 Stuttgart
Germany

CR: H.4.1

Abstract

Process fragments are a cornerstone of process modeling in both Service Oriented Architecture and Business Process Management. The state of the art lacks shared, language agnostic definitions of the basic concepts and properties of process fragments. This absence of a common foundation for the research on process fragments hinders the comparison and reuse of the results in the state of the art, and renders impossible the reaching of agreed, intensional definitions of the different typologies of process fragments. The present work aims at filling this gap by providing a framework of language agnostic definitions of properties of the syntactic structure of process fragments based on the mereotopology of discrete space. Alongside familiar mereologic concepts like inclusion, overlap and disjointness, we cover fundamental concepts for process fragments like (dis)connection, self-connectedness, borders, interiors and exteriors. Besides providing a foundation for further research on process fragments, we discuss the immediate application of the concepts defined in this work in the scope of the change management of process models.

The research leading to these results has received funding from the European Community's Seventh Framework Programme under the Network of Excellence S-Cube - Grant Agreement n° 215483.

This work is licensed under a:
Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License

A Language-Agnostic Framework for the Analysis of the Syntactic Structure of Process Fragments

Michele Mancioppi
European Research Institute in Services Science (ERISS)
Tilburg University, Tilburg, the Netherlands
m.mancioppi [at] uvt.nl

Olha Danylevych
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Stuttgart, Germany
olha.danylevych [at] iaas.uni-stuttgart.de

Mike P. Papazoglou
European Research Institute in Services Science (ERISS)
Tilburg University, Tilburg, the Netherlands
mikep [at] uvt.nl

Frank Leymann
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Stuttgart, Germany
leymann [at] iaas.uni-stuttgart.de

1 Introduction

In Service Oriented Architecture (SOA) and Business Process Management (BPM), process modeling supports the accomplishment of tasks such as modeling operational aspects of large scale applications, business process reengineering, human understanding of and communication involving processes, and the automation of software- and business processes [CKO92]. Standard process-modeling languages like Web Services Business Process Execution Language v2.0 (WS-BPEL), Business Process Modeling Notation v1.2 (BPMN v1.2) and the upcoming Business Process Model and Notation v2.0 (BPMN v2.0), widely accepted by industry and academy alike, allow for the specification of process models that represent (and, if executable, realize) software services and their compositions (possibly into other services), intra- and inter-organizational business processes, workflows, etc.

A *process fragment* is a part of an existing process model, i.e. a subset of the elements (control flows, activities, etc.) that are defined in that process model. Process fragments have a wide range of applications including reuse [WRR07, GKW08], autocompletion functionalities during process modeling [HKO07], distributed execution of process models [LD99, Kha08] possibly for optimizing Quality of Service (QoS) criteria [NCS04, DKL09], and adjusting to different categories of users the granularity of process models so to visualize only the most relevant information [PSW08].

Unfortunately, the research efforts devoted so far to process fragments are problem specific and rather ad-hoc. In the state of the art on process fragments there is no common foundation for basic concepts such as when process fragments overlap or what are their borders, and this prevents the generalization of the results that have been already achieved. This "fragmentation" of the state of the art is aggravated by the fact that the formal definitions of process fragments in the literature are deeply rooted in particular process-modeling languages such as WS-BPEL, classes of Petri-nets, or specific graph-based workflow languages. As a result, these definitions mix the *syntactic structure* of the process fragments, i.e. which process elements they contain and how they are related to each other, with their *operational semantics*, i.e. how the process fragments are executed at runtime or which piece of abstract business logic they represent. The state of the art of SOA and BPM encompasses a diversity of process-modeling languages that vary remarkably in both syntax and operational semantics. In this context, the tangle of syntax and operational semantics is a major setback that impedes the development of a general theory for process fragments. The aftermath is that we currently lack formal, shared, intensional and language-agnostic definitions of the typologies of process fragments.

This work is targeted at researchers that work on process fragments. To alleviate the lack of a formal, language agnostic foundation to process fragments, we present a framework of concepts and properties of process fragments that researchers and practitioners can apply with minimal effort to their process-modeling language of choice. In particular, in this work we focus on the syntactic structure of process fragments, which is their *sine qua non*. In fact, what process fragments *accomplish* (when executed) or *mean* (to users) depends on the interpretation of their syntactic structure according to some particular (operational) semantics. Therefore, the first step towards the introduction of a language agnostic theory of process fragments is the study and definition of their syntactic structure.

The work here presented has its theoretical roots in *mereotopology*. Mereotopology is a combination of *mereology*, i.e. the study of the relations between parts and wholes, and *topology*, i.e. the study of the spatial properties of objects [Var98]. In particular, we employ the mereotopology of discrete space introduced in [Gal99], which studies the properties of objects made of finitely many parts. By virtue of its mereologic aspects, the mereotopology of discrete space allows to formalize all the properties of process fragments that have been expressed in the literature using set theory, e.g. overlapping, disjointness and inclusion. On the other hand, the topologic aspects captures the connectivity among the process elements in process fragments, which is usually addressed in the literature by means of graph theory. Finally, the combination of the mereologic and topologic aspects enables the definition of the *anatomy* of process fragments, i.e. the subdivision of a process fragment and its surrounding environment. For example, in mereotopology we distinguish between the interior, i.e. the internal part of the process fragment that is not connected to its surrounding environment, the border, which is the part of the process fragment that is connected with the surrounding environment, and other fundamental parts of the process fragment as well as its surroundings. The results presented in this work are exemplified using the BPMN v2.0 Choreography process-modeling language.

This remainder is structured as follows. Section 2 establishes the necessary background on the syntax of process-modeling languages and the syntactic structure of their process models. Section 3 illustrates the application of the mereotopology of discrete space to process fragments. Finally, Section 4 discusses the contribution of this work in the scope of the research on process fragments.

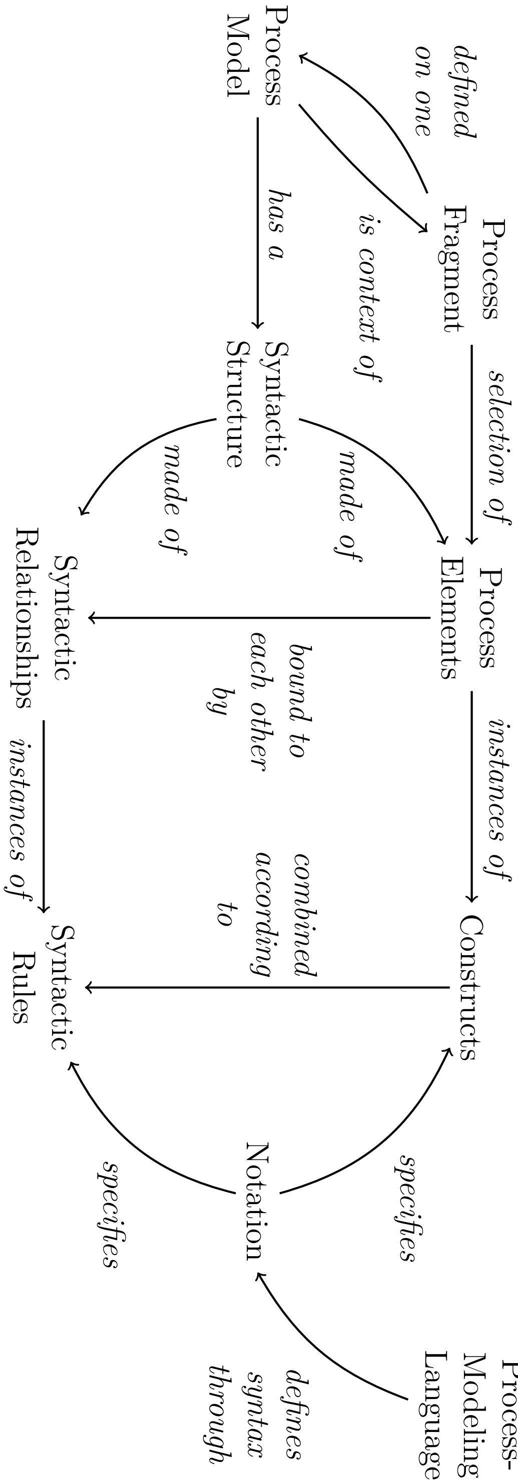


Figure 1: A concept map of the definitions provided in Section 2

2 On the Syntax of Process Models

The goal of the current section is to provide the necessary background on the syntax of process-modeling languages (Section 2.1) and the syntactic structure of their process models (Section 2.2), which build up to the definition of process fragments adopted in the scope of this work (Section 2.3). The concepts and their relations covered throughout this section are summarized in the concept map presented in Figure 1.

2.1 The Syntax of Process-Modeling Languages

A *process model* is a representation of the structure of a process in terms, for example, of the activities it comprises, their ordering, the events, faults, errors and exceptions that can occur, how these are handled, etc. *Process-modeling languages*, e.g. WS-BPEL and BPMN v1.2, provide syntaxes for specifying process models and, to different extents of formality, their semantics [CKO92, HR00]. In this work we are concerned exclusively with the syntax of process models, and therefore we will not discuss their semantics. The syntax of a process-modeling language is specified by its *notation* in terms of *constructs* and *syntactic rules* to combine them [HR00]. The constructs are the basic components of the notation such as activities, sequence flows and gateways in workflow-like notations. The *syntactic rules* (also called "composition mechanisms" in [HR00]) specify how the constructs can be combined in a process model whose syntax is correct. For example, a commonplace syntactic rule in workflow-based process-modeling languages is that a sequence flow must be connected with exactly two activities, one at each end. Meta-models, i.e. schematic representations of the constructs and syntactic rules of modeling languages [EW05] (N.B. not only process-modeling ones), have become in the past two decades the standard way for specifying notations by means, for example, of Unified Modeling Language (UML).

2.2 The Syntactic Structure of Process Models

The constructs provided by a process-modeling language are instantiated in its process models as *process elements*. For clarity, throughout this work we typeset the names of particular process elements with a different font, for example `MyInvokeActivity`.

The running example used throughout this work is the BPMN v2.0 choreography shown in Figure 2. In a nutshell, a service choreography – or choreography, in short – is a process model describing the ordering of message

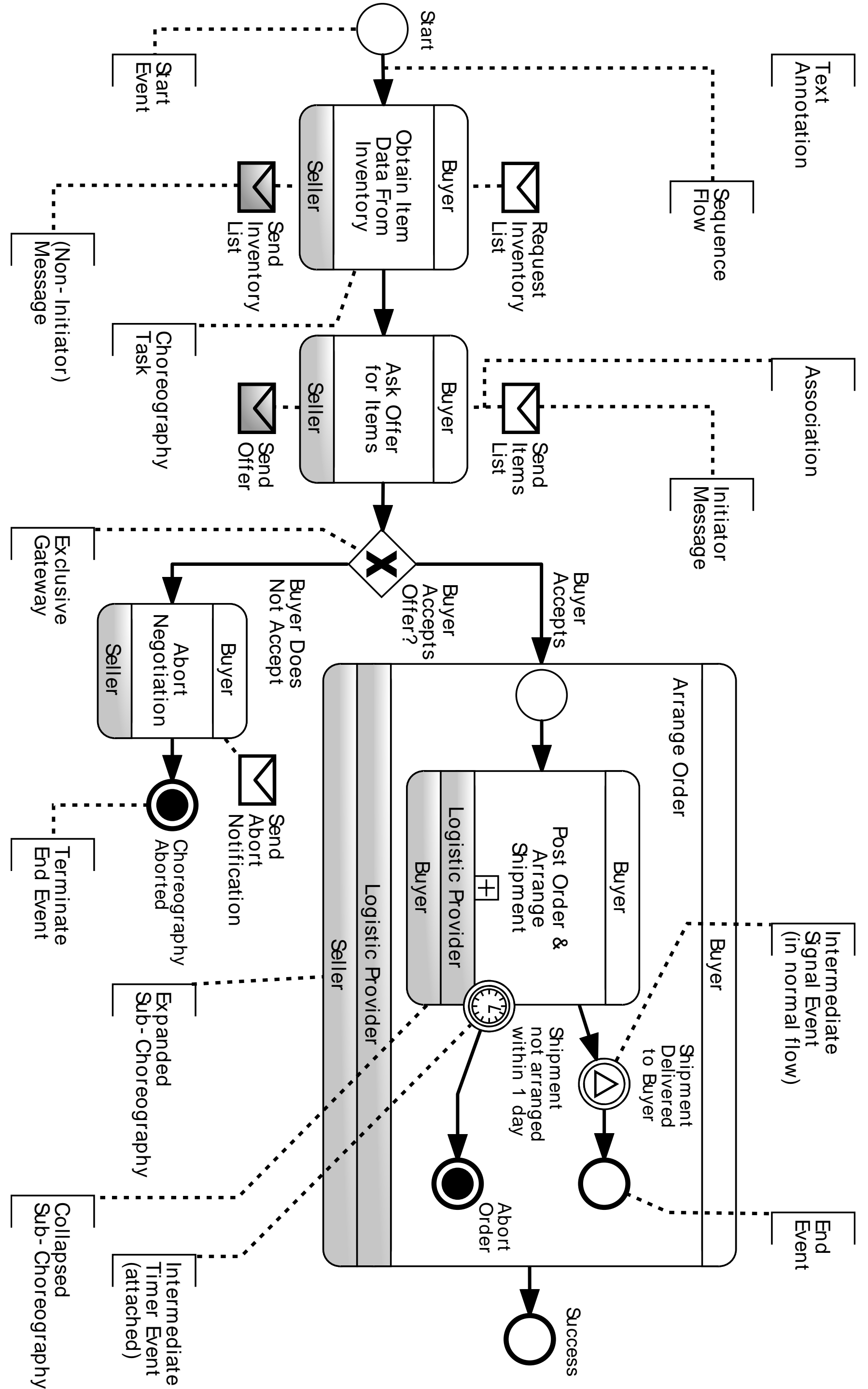


Figure 2: A choreography specified using BPMN v2.0

exchanges in message-based conversations that involve two or more participants [Pel03]. The conversations are modeled from a *global* perspective, i.e. without assuming the point of view of any particular participant. For reasons of space, in the following we will restrict the exposition of the BPMN v2.0 process-modeling language to the extent necessary to illustrate the example in Figure 2. The reader is referenced to the BPMN v2.0 specification [OMG09] for a complete overview of the BPMN v2.0 choreography constructs. For readability, the different types of process elements are highlighted in Figure 2 using text annotations.

The running example involves three participants, namely `Buyer`, `Seller` and `Logistic Provider`. The beginning of the choreography is denoted by the start event called `Start`. Because of the sequence flow connecting the start event with the choreography task named `Obtain Item Data From Inventory`, the latter is the first message exchange that takes place in the choreography, and it involves the participants `Buyer` and `Seller`. `Buyer` is the initiator of the message exchange because it is the sender of the initiator message, in this case `Request Inventory List`. The participant `Seller` concludes the message exchange by replying with the message called `Send Inventory List`. After the first message exchange is completed, `Buyer` and `Seller` engage in the choreography task named `Ask Offer for Items`, which involves the two messages `Send Items List` and `Send Offer`. After the completion of this second message exchange, `Buyer` decides whether to accept the offer of `Seller`. This decision is represented in the choreography by the decision gateway named `Buyer Accepts Offer?`. If `Buyer` does not accept the offer, the sequence flow named `Buyer Does Not Accept` is traversed, leading to the enactment of the choreography task named `Abort Negotiation`. This last message exchange is followed by the termination of the choreography enactment because of the reaching of the termination event named `Choreography Aborted`. On the contrary, if `Buyer` does accept the offer, then `Buyer Accepts` is traversed, and the expanded sub-choreography named `Arrange Order` is enacted. Beside the already mentioned `Buyer` and `Seller`, `Arrange Order` involves the participant `Logistic Provider`, that has not yet been involved in the enactment. `Buyer`, `Seller` and `Logistic Provider` enact the collapsed sub-choreography called `Post Order & Arrange Shipment`, which, if not completed within one day, causes the triggering of the `Shipment not arranged within 1 day` attached intermediate timer event. This event leads to the termination of the choreography enactment when `Abort Order` is reached. However, if `Post Order & Arrange Shipment` is completed within one day, the enactment eventually reaches the intermediate signal event named `Shipment Delivered to Buyer` that symbolizes the reception by `Buyer` of the goods it has ordered

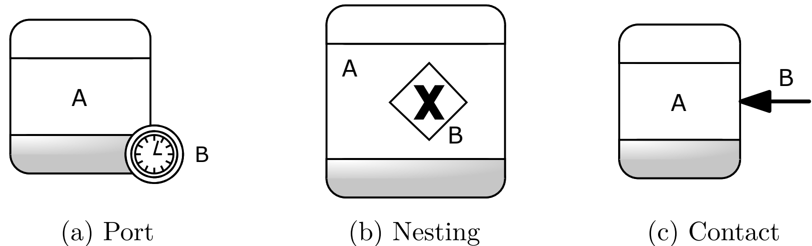
by `Seller`. This last event is followed by the completion of `Arrange Order`, followed by the completion of the entire choreography enactment when the end event called `Success` is reached.

In the above description of the running-example choreography, the process elements stand out. However, the choreography – and, more generally, any process model – is not specified exclusively in terms of its process elements, but also of how those process elements are combined. For example, the fact that the two ends of a sequence flow "touch" two certain process elements has a precise meaning in the choreography, i.e. the ordering of those two process elements. This ordering is conveyed by none of the process elements considered in isolation, but instead by the way they are combined in the process model, i.e. the *syntactic relationships*. In other words, syntactic relationships bind the process elements in order to convey information on the syntactic structure of the process model that single process elements cannot otherwise express. In the same way the process elements are instances of the constructs of the notation of the process-modeling language, the syntactic relations are instances of its syntactic rules. In the remainder of this work we assume syntactic relationships in process models to be *binary*, i.e. involving exactly two process elements. This assumption does not cause any loss of generality for the applicability of the framework proposed in this work. In fact, in the same way n-ary relations in Entity Relationships (ERs) diagrams can be "split" in multiple binary relations without losing information, n-ary syntactic relationships in process models can be replaced by multiple binary ones.

Each process-modeling language has its own types of syntactic relationships, and their classification falls outside the scope of this work. The reader will nevertheless benefit from some examples. In process-modeling languages that have notations based on eXtensible Markup Language (XML), e.g. WS-BPEL, the syntactic relationships are usually nesting (putting one XML element inside another), sequencing (positioning an XML element before/after another) and referencing (making references to XML elements from others by means of their name or another identifier, often specified as an XML attribute of the referenced XML element).

In the case of BPMN v2.0 choreographies, the types of syntactic relationships that occur are shown in Figure 3. The *port* syntactic relationship (see Figure 3a) occurs between choreography activities (i.e. choreography task, sub-choreography, etc.) and gateways, and the intermediate events attached to them. For example, `Arrange Shipment` and `Shipment not arranged within 1 day` in Figure 2 are bound by a port syntactic relationship.

The *nesting* syntactic relationship (Figure 3b) occurs between a sub-choreography and each process element immediately nested in it. That is, a



(a) Port (b) Nesting (c) Contact

Figure 3: Syntactic relationships in BPMN v2.0 choreographies

process element nested into a sub-choreography has a nesting relation only with the latter, regardless whether that sub-choreography is itself nested into another sub-choreography.

The *contact* syntactic relationship (Figure 3c) represents the fact that the graphic borders of two process elements touch. In BPMN v2.0 choreographies, the contact syntactic relationship occurs in the following four cases: (1) between a sequence flow and its source process element, (2) between a sequence flow and its target process element, (3) between an association and its source process element, and (4) between an association and its target process element. Given a sequence flow or an association, the connections with its source and target process element are two *distinct* instances of the syntactic relationship.

In the scope of this work, the *syntactic structure* of process model is defined as *the set of process elements contained in the process model and the syntactic relationships that bind them.* The syntactic structure of a process can be visualized as a graph in which the nodes are the process elements and the edges the syntactic relationships that bind them. It is interesting to notice that the syntactic structure of process models as defined above is conceptually similar to Abstract Syntax Trees (ASTs) in block-based programming languages (see e.g. [BYM+98]). However, ASTs are not applicable to process models for two reasons. Firstly, the syntactic relationships are not always directed. For example, when two process elements are connected is not always clear if either of them "comes first" or, using graph terminology, if there is a source and a target. Secondly, the syntactic relationships in a process model may form cycles, which are not allowed in tree-like data structures like ASTs. For example, consider the case of a BPMN v2.0 choreography task with a sequence flow that loops on it: in this case there would be two contact syntactic relationships, one per each end of the sequence flow.

2.3 Process Fragments

In the scope of this work a *process fragment* is defined as an *arbitrary subset of the process elements of a given process model.* In other words, a process fragment can be any selection of one or more process elements that are contained in one process model. Our definition of process fragment is deliberately generic in order to allow the application of the theories proposed in this work to all the different types of process fragments studied in the literature. No assumptions are taken on the process elements comprised in a process fragment other than they all belong to the same process model (otherwise the process fragment would not be an "integral part" of the process model). In particular, we assume no particular criterion for how the process fragments and their process elements are selected.

It should be noted that the definition of process fragment provided above does mention the syntactic relations between the process elements comprised in the process fragments. The reason is that the syntactic relationships are already part of the *context* of the process fragment, i.e. the encompassing process model. This approach is coherent with the notion of process fragment as integral part of the process model, and allows us to clearly distinguish between process model and process fragment: the process model is made of process elements *and* syntactic relationships, whereas the process fragment is made only of some process elements specified by one process model, and it requires the context of that process model to be studied.

3 Process Fragments & the Mereotopology of Discrete Space

This section incrementally presents the mereotopology of discrete space and how its concepts apply to process models and process fragments. The formulation of the mereotopology is based on first order logic (negation, conjunction, disjunction, implication, existential and universal quantification, etc.), in addition to some primitives reminiscent of set theory (e.g. intersection and difference). In each of the following subsections, the theory of mereotopology and its application are covered by "The Theory" and "The Practice" paragraphs, respectively. For reasons of space, we will not introduce all the mereotopologic concepts treated in [Gal99], but only the ones that are fundamental or that have an immediate application to the change management of process models and process fragments.

Compared to [Gal99], the mereotopology of discrete space presented in the remainder of the section has been reworked to drop one fundamental axiom, i.e.

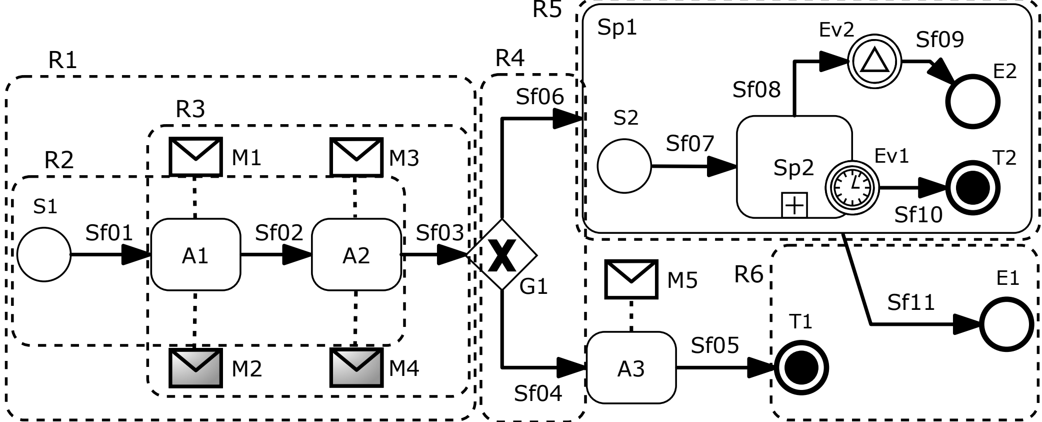


Figure 4: A simplified version of choreography in Figure 2, and process fragments selected on it

the reflexivity of the adjacency relation (see Section 3.2), that is problematic when applied to process models and process fragments. Additionally, we extend the theory by introducing two novel concepts, namely the border and the confine (see Section 3.5), which are of extreme usefulness when applied to process fragments.

The BPMN v2.0 choreography shown in Figure 4 is used as the running example throughout most of this section. It is a slightly modified version of the choreography shown in Figure 2 in which (1) the text annotations have been removed for reasons of space, (2) the graphical representation of choreography tasks and sub-choreographies is simplified by dropping the participant bands, (3) the descriptive names of the process elements have been replaced with short identifiers for brevity, and (4) the five process fragments $R1$, $R2$, $R3$, $R4$ and $R5$, delimited by dashed lines, have been identified.

3.1 Cells, Spaces and Regions

The Theory. Cells are the atomic constituents of a mereotopologic *space*. We denote with $c \in S$ the fact that the cell c is included in the space S. A *region* is a (possibly empty) subset of the cells of a space. Similarly to sets in set theory, regions are defined extensionally, i.e. as enumerations of the cells they contain. The symbol $\emptyset$ denotes the *empty region* of a certain space, i.e. the region in that space that contains no cells. We say that a region R belongs to a space S if and only if $R \subseteq S$. The region that contains all the cells in a space is called *universal region*. The inclusion of a region into another is defined as between sets in set theory, i.e. a region $R1$ is included in $R2$ if and

only if every cell of $R1$ is also contained in $R2$. Formally:

$$R \subseteq R' \iff \{\forall c \in R : c \in R'\} \qquad \text{(Inclusion of Regions)}$$

The *complement* $\overline{R}$ of a region R that belongs to the space S contains all and only the cells on S that are not in R:

$$\overline{R} := S \setminus R \qquad \text{(Complement of a Region)}$$

The symbol $\setminus$ denotes the difference operator as defined in classic set theory, namely:

$$R \setminus R' := \{c \in S : c \in R \wedge c \notin R'\} \qquad \text{(Difference of Regions)}$$

The intersection of two regions R and R' in a space S, denoted by $R \cap R'$ is defined as in set theory, i.e. as the region made of all and only the cells that are contained in both regions:

$$R \cap R' := \{c \in S : c \in R \wedge c \in R'\} \qquad \text{(Intersection of Regions)}$$

All the above operators are defined in the context of one space. The mereotopology of discrete space does not specify primitives for comparing regions defined in different spaces and does not treat regions whose cells belong to difference spaces (i.e. are allowed no "mixes" of different spaces). When clear from the context or not relevant, in the remainder we will omit to specify the space to which cells and regions belong.

The Practice. In order to apply the mereotopology of discrete space to a process model, it is necessary to map the latter syntactic structure of the latter (see Section 2.2) to a mereotopologic space. The mapping is divided in two steps, one concerning process elements and cells (described below) and one concerning syntactic relationships (illustrated in Section 3.2). The first step of the mapping is the following:

> *Each process element of a process model is mapped to a distinct cell in the mereotopologic space.*

Process fragments are straightforwardly mapped to regions. The complement of process fragments and their inclusion into each other is defined in the same way as the respective concepts for regions.

It is interesting to notice that mapping process fragments to regions implies the possibility of "empty" process fragments, i.e. process fragments that do not actually contain any process element. Intuitively, empty process fragments are not relevant in practice. For example, an empty process fragment has

no reuse value, as there is no logic in it to be reused. However, they are a useful abstract concept. In particular, empty process fragments come in handy in the definition of *change algebras*, i.e. sets of change operators for manipulating process models specified with some particular process-modeling language, see e.g. [WRRM08].

3.2 Adjacency of Cells

The Theory. Adjacency is a binary relation defined on the cells of a space. Intuitively, two cells c_1 and c_2 in a space S are *adjacent* – denoted by $\mathbf{A}(c_1, c_2)$ – if they are "next-door neighbors". Together with the mereologic inclusion of cells in regions and spaces, the adjacency relation is the fundamental relation of mereotopology. Given an application domain, the semantics of the adjacency relation, i.e. what does it mean that two cells are adjacent, determines the interpretation of all the other mereotopologic in that application domain. The semantics of the adjacency relation in this work is discussed in the following "The Practice" paragraph; here we discuss its general properties.

The adjacency relation is by definition symmetric and intransitive. Put formally:

$$\forall\, c, c' \in S \,:\, \mathbf{A}(c, c') \iff \mathbf{A}(c', c) \qquad \text{(Adjacency: Symmetry)}$$

$$\forall\, c, c', c'' \in S \,:\, \mathbf{A}(c, c') \wedge \mathbf{A}(c', c'') \nRightarrow \mathbf{A}(c, c'') \qquad \text{(Adjacency: Intransitivity)}$$

[Gal99] assumes a reflexive adjacency relation (i.e. each cell is adjacent to itself) for "technical reasons", i.e. because this assumption simplifies the presentation of some mereotopologic concepts that build on adjacency. However, the reflexivity of the adjacency relation does not suite well the application of the mereotopology of discrete space to process fragments (the explanation is provided in the "The Practice" part of this section), and it is dropped in favor of an irreflexive formulation of adjacency:

$$\forall\, c \in S \nRightarrow \mathbf{A}(c, c) \qquad \text{(Adjacency: Irreflexivity)}$$

In the remainder of the exposition of the mereotopology of discrete space we will adjust the definitions of the concepts affected by the irreflexivity of the adjacency relation to make them logically equivalent to the original formulation provided by [Gal99]. Those corrections will be made explicit. Overall, the mereotopology of discrete space and the concepts not directly impacted by the corrections are not affected. Therefore, all the lemmas and theorems presented in [Gal99] still apply.

Two adjacent, distinct cells c and c' are said to be *neighbors*. The *immediate neighborhood* of a cell c in the space S, denoted by N_c, is the region made of

c all and only the cells that are adjacent to it:

$$N_c := \{c' \in S \mid c = c' \vee \mathbf{A}(c, c')\} \qquad \text{(Immediate Neighborhood)}$$

This formulation differs from the one provided in [Gal99] to account for the irreflexivity of the adjacency relation. In particular, we must explicitly state that a cell is comprised in its immediate neighborhood, which in the original formulation is implied by the reflexivity of the adjacency relation.

The cells of a space and the adjacency relations between them can be graphically represented by means of an *adjacency graph.* The adjacency graph, e.g. the one shown in Figure 5, is an undirected graph in which the cells of a space are the nodes, and the adjacency relations between them are the edges.

The Practice. As discussed in Section 3.1, each process element of a process model is mapped to a distinct cell in the mereotopologic space that represents that process model's syntactic structure. The other constituent of the syntactic structure of process models, i.e. the syntactic relationships that bind process elements, are mapped to adjacency relations. In particular, the mapping from syntactic relationships to adjacency relations is the following:

> *Two cells are marked as adjacent if there is at least one syntactic relationship that binds the respective process elements.*

In other words, all the syntactic relationships between two given process elements are "collapsed" in one adjacency relationship binding the cells to which those two process elements are mapped. The directions (if any) of the syntactic relationships are discarded in the mapping.

It is important to notice that syntactic relations are generally not reflexive. For example, no process element in a BPMN v2.0 choreography can be in contact with or nested into itself. In point of fact, the mapping from syntactic relationships to adjacency relations motivates the reworking of the mereotopology of discrete space presented in this work and finalized to dropping the assumption on the reflexivity of the adjacency relation. Maintaining the assumption that every cell is adjacent with itself would lead to the inconsistent (and confusing) situation in which some adjacency relations result from the mapping of the syntactic relationships, and others from the axiomatization of the mereotopology of discrete space.

The general applicability of the mereotopology of discrete space to process models of any process-modeling language derives from the mappings from process elements to cells and from syntactic relationships to adjacency relations. The particular types of process elements vary from a process-modeling language to the other, and so do the syntactic relationships between them. However, each process element is mapped to a distinct cell regardless of its

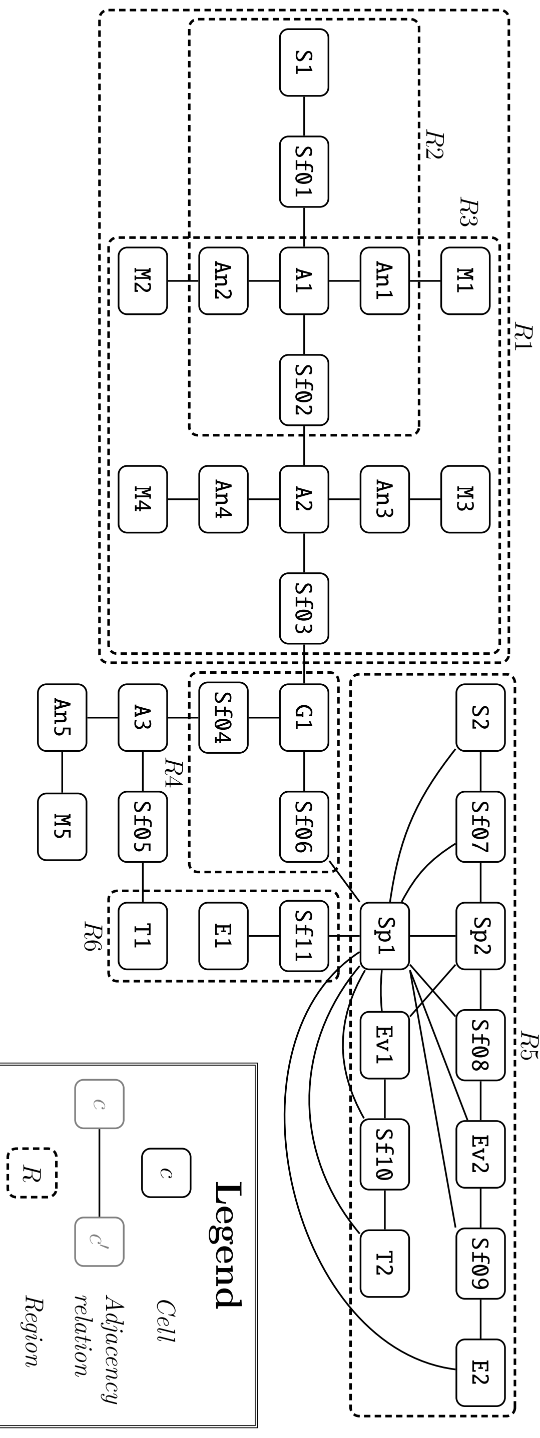


Figure 5: The adjacency graph and regions resulting from the choreography and process fragments of Figure 4

type (i.e. the particular construct it is instantiated from). Similarly, any syntactic relationships binding two process elements are mapped to one single adjacency relation between the relative cells.

Figure 5 shows the adjacency graph resulting from the BPMN v2.0 choreography presented in Figure 4. The process fragments identified on the choreography are mapped to the regions (delimited by dashed borders on the adjacency graph) that are identified by the same names. To ease the comparison between the choreography and the respective adjacency graph, the layout of the latter mirrors closely the one of the former. However, the actual positioning of the nodes in the adjacency graph bears no relevance for the mereotopologic properties that will be studied in the remainder of this section: what matter is only which cells are defined, which cells are adjacent to which others, and how the cells are grouped in regions.

3.3 Parthood and Connection of Regions

The Theory. The regions of a mereotopologic space can be bound by several different types of relations that are introduced in this section using as example the regions identified on the adjacency graph shown in Figure 5.

The non-empty region R is a *subregion* of R', denoted by $\mathbf{P}(R, R')$, if the first is a subset of the second, i.e.:

$$\mathbf{P}(R, R') \iff R \neq \emptyset \wedge R \subseteq R' \qquad \text{(Subregion)}$$

By definition, any non-empty region is a subregion of itself. A subregion R that contains strictly less cells than its enclosing region R' is called *proper subregion.* Formally:

$$\mathbf{PP}(R, R') \iff \mathbf{P}(R, R') \wedge R' \setminus R \neq \emptyset \qquad \text{(Proper Subregion)}$$

For example, the regions $R2$ and $R3$ are both proper subregions of $R1$.

Two regions R and R' *overlap* with each other, denoted by $\mathbf{O}(R, R')$, if their intersection is not empty (i.e. they share at least one cell):

$$\mathbf{O}(R, R') \iff R \cap R' \neq \emptyset \qquad \text{(Overlap)}$$

For example, the regions $R2$ and $R3$ overlap. The overlap is defined so that the equivalence of regions is a special case, i.e. every region overlaps with itself. Note also that a region overlaps with all its (proper) subregions.

The regions R and R' *partially overlap*, denoted by $\mathbf{PO}(R, R')$, if they overlap and neither one is a subregion of the other:

$$\mathbf{PO}(R, R') \iff \mathbf{O}(R, R') \neq \emptyset \wedge \neg\mathbf{P}(R, R') \wedge \neg\mathbf{P}(R', R) \quad \text{(Partial Overlap)}$$

For example, the regions $R2$ and $R3$ partially overlap.

Two regions R and R' that do not overlap, i.e. their intersection is empty, are said to be *disjoint*, which is denoted by $\mathbf{DJ}(R, R')$. Formally:

$$\mathbf{DJ}(R, R') \iff \neg\mathbf{O}(R, R') \qquad \text{(Disjointness)}$$

For example, the regions $R1$, $R4$, $R5$ and $R6$ are all pair-wise disjoint.

Two regions R and R' are *connected*, denoted by $\mathbf{C}(R, R')$, if either they overlap or they respectively contain two cells c_1 and c_2 that are adjacent. Put formally:

$$\mathbf{C}(R, R') \iff \mathbf{O}(R, R') \vee \exists\, c_1 \in R, c_2 \in R' \,:\, \mathbf{A}(c_1, c_2) \qquad \text{(Connection)}$$

For example, the regions $R1$ and $R4$ are connected because of the adjacency of the cells `Sf03` and `G1`, as well as $R3$ and $R4$ for the same reason. With respect to the formulation of connectedness provided in [Gal99], we explicitly add the clause that regions that overlap are connected, which was originally implied by the reflexivity of the adjacency relation.

Like adjacency, the connection relation is symmetric and intransitive. Two regions R and R' that are connected and disjoint are *externally connected*, denoted by $\mathbf{EC}(R, R')$. Formally:

$$\mathbf{EC}(R, R') \iff \mathbf{C}(R, R') \wedge \neg\, \mathbf{O}(R, R') \qquad \text{(External Connection)}$$

For example, of $R3$ and $R4$ are externally connected, as well as $R1$ and $R4$.

Disconnection is the opposite case to connection:

$$\mathbf{DC}(R, R') \iff \neg\, \mathbf{C}(R, R') \qquad \text{(Disconnection)}$$

In other words, the regions R and R' are disconnected if they do not overlap and none of their cells are adjacent. This is the case, for example, of $R1$ and $R6$.

The Practice. Subregion, proper subregion, overlap, partial overlap and disjointness are standard mereologic relations between regions that apply verbatim to process fragments. The mapping from the syntactic structure of process models to mereotopologic spaces provides a straightforward, univocal way of defining the mereologic relations between process fragments. These mereologic relations, while seldom defined formally, are widely used in the literature of process fragments (see e.g. [VVL07, Kha08, VVK09, Van09]). Moreover, they have an important role in the intensional definitions of types of process fragments, i.e. definitions based on the properties of the process fragments. Consider for example process fragments defined as *partitions*, e.g. in [NCS04, Kha08]. Partitions are simply defined as process fragments that do

not overlap with each other and whose union (i.e. all the process fragments collectively) includes every process element defined in the process model.

Connection, external connection and disconnection are relations of a topologic nature that (mainly) build on the adjacency between the regions' cells[1]. Intuitively, the connection between two process fragments marks some sort of syntactic dependence occurring between them: either they overlap, or some of their process elements are adjacent with each other. Conversely, two disconnected process fragments are completely independent from each other in terms of syntactic structure.

The syntactic dependencies between process fragments that are outlined by connection, external connection and disconnection bear a big potential in the scope of change management of process models. Informally, a change is a modification of a process model's syntactic structure with the aim of achieving alterations in the process model's (operational) semantics and/or *secondary notation*. In [Pet06], the secondary notation of process models is defined as:

> "... things which are not formally part of a notation which are nevertheless used to interpret it, such as conventions (e.g., reading a circuit diagram left-to-right and top-to-bottom), use of locality (i.e., placing logically related items near each other), and labelling."

The part of the process model that is affected by the change is called *change region* (see e.g. [EKR95, RD98, MBRS06, RWR06, WRR07, WRRM08]), and it is a process fragment as we understand it in this work. Similarly to the case of software programs, it is well known that changes in process models have a way of "rippling through" and affect other parts of the process model in subtle, unforeseen and virtually always undesirable (and undesired) ways. In this respect, connection, disconnection and external connection between process fragments could be used to map how the ripples propagate or falter (i.e. stop) in process models, e.g. by adapting to process models the method proposed in [Raj97] for estimating ripple effects in software programs.

3.4 Self-Connected Regions

The combination of the concepts of subregions and connection allows to study the "extent of the connectedness" *within* a particular region. A region R

[1] Actually, the reason why the adjacency relation is defined as reflexive in [Gal99] is that this allows the connection between regions to be specified *only* in terms of the adjacency between cells, thus introducing a clear distinction between mereologic and topologic relations.

is *self-connected*, denoted by $\mathbf{SC}(Region R)$, if each of its non-empty proper subregions is connected to its complement in that region. Formally:

$$\mathbf{SC}(R) \iff \forall R' \subset R : R' \neq \emptyset \Rightarrow \mathbf{C}(R', R \setminus R') \qquad \text{(Self-Connection)}$$

The rationale of self-connected regions is that their cells are all "transitively" adjacent to each other. Recall that adjacency is intransitive (Section 3.2). By transitively adjacent we mean that, given an arbitrary couple of cells c and c' in a self-connected region, it is possible to find a path of adjacency, i.e. a sequence of adjacency relations "chained together" by shared cells, that leads from c to c'. Due to the symmetry of the adjacency relation, the same path traversed in the opposite direction leads necessarily from c' to c. An example of such path connecting `Sf04` and `Sf06` in $R4$ is the following:

$$\langle\, (\texttt{Sf04}, \texttt{G1}), (\texttt{G1}, \texttt{Sf06}) \,\rangle$$

In Figure 5, all the regions are self-connected, with the exception of $R6$. $R6$ is not self-connected because the cell `T1` is not adjacent with any other cell in that region. In other words, a region is self-connected if it has only one *connected component*, i.e. a maximal non-empty subregion in which all cells are transitively adjacent and that is not connected . Alternatively, a connected component $\mathbf{CC}(R)$ of R can be characterized as a proper subregion of R that is not connected with its complement in R. Formally, the set of connected components of a region R is defined as:

$$\{\mathbf{CC}(R) : \mathbf{PP}(R, \mathbf{CC}(R)) \wedge \neg\, \mathbf{C}(\mathbf{CC}(R), \overline{\mathbf{CC}(R)}) \wedge \mathbf{CC}(R) \neq \emptyset\} \qquad \text{(Connected Components)}$$

For example, in $R6$ there are two connected components. The first is made of the cell `T1` and the second comprises the cells `Sf11` and `E1`.

The Practice. A self-connected process fragment is fundamentally a "monolithic" part of a process model. The relevance to process fragments of the concept of self-connectedness is testified by the fact that it is integral to a number of definitions of process fragments, see e.g. [VVL07, ML08, PW09].

It is important to notice that the self-connectedness here defined is "undirected" in nature. That is, all the nodes (our case: the cells) in a self-connected region are mutually reachable through edges (adjacency relations) that do not have a direction, i.e. can be traversed both ways. This is different from the "directed" self-connectedness of subgraphs of workflow graphs, whose edges can usually be traversed in only one direction. The focus in mereotopology on the undirected aspect of self-connectedness is coherent with the syntactic approach to process fragments taken in this work. In fact, the directed aspects of connectivity in process models and process fragments, e.g. the execution order and mutual reachability of their process elements, is actually a matter of operational semantics instead of syntactic structure.

3.5 The Anatomy of a Region

The Theory. The concepts of connection, proper subregion and complement of a region allow to define notable regions – called interior, exterior, boundary, border and confine – with respect to one particular region R. These notable regions are collectively known as the *anatomy* of the region R. The concepts in this section are exemplified using the spaces and regions shown in Figure 6.

The *interior* R_{int} of a region R is made of all the cells of the latter whose immediate neighborhoods lay completely within R:

$$R_{int} := \{c \in R \mid N_c \subseteq R\} \qquad \text{(Region's Interior)}$$

In other words, the interior of a region is made of all the region's cells that are not adjacent to any cell outside that region. It is possible for regions to have empty interiors, for example the region depicted in Figure 6a.

Conversely, the *exterior* R_{ext} of the region R is made of all the cells whose neighborhood (which, we recall, includes the cells themselves) lays completely outside R:

$$R_{ext} := \{c \in R \mid N_c \cap R = \emptyset\} \qquad \text{(Region's Exterior)}$$

Likewise region interiors, regions exteriors may be empty, see e.g. Figure 6b. Straightforwardly, the exterior of a region if always a subregion (possibly proper) of the complement of that region.

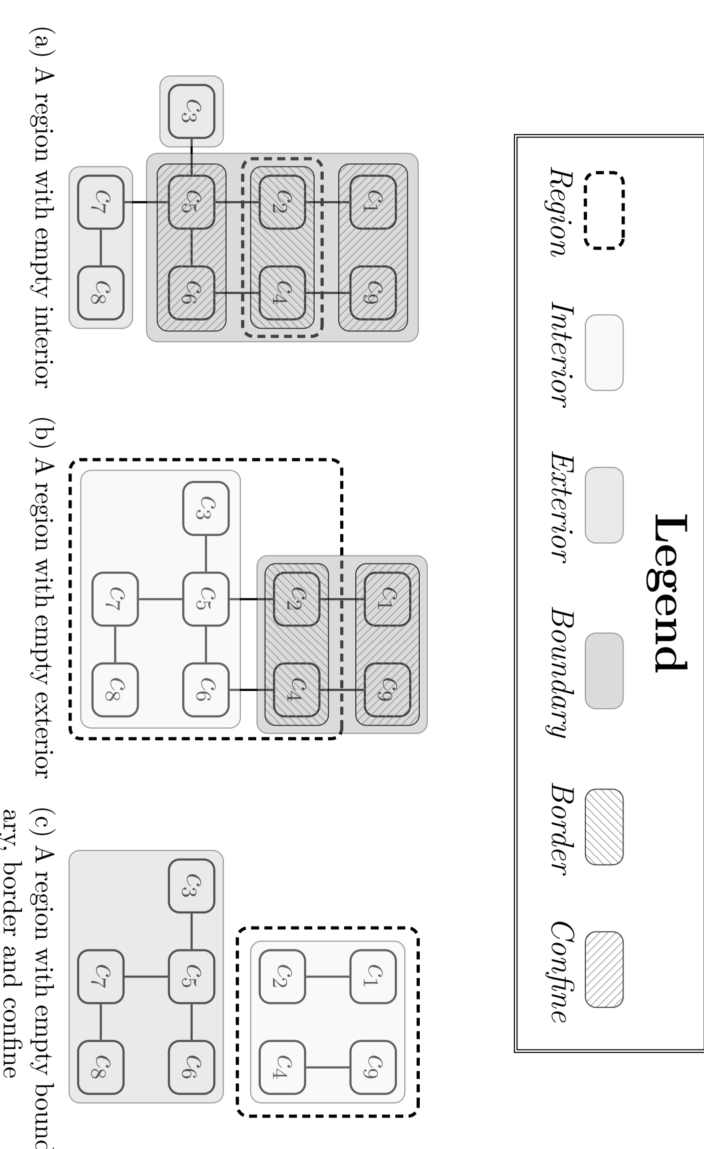
The interior and the exterior of a region are not necessarily complement of each other. A cell whose immediate neighborhood overlaps with both a region and its complement is included in neither of them. This is the case, for example, of the cells c_2, c_3 and c_4 in Figure 6b. The *boundary* of a region R, denoted by R_{bnd}, comprises all and only the cells that are in neither the interior nor the exterior of R. Formally:

$$R_{bnd} := (S \setminus R_{int}) \setminus R_{ext} \qquad \text{(Region's Boundary [Alt. 1])}$$

Alternatively, the boundary of a region can be characterized as the set of the cells whose immediate neighborhood overlaps with both the region and its complement:

$$R_{bnd} := \{c \mid \mathbf{O}(N_c, R) \wedge \mathbf{O}(N_c, \overline{R})\} \qquad \text{(Region's Boundary [Alt. 2])}$$

That is, as long as any of their neighbors is in the region, the cells in the boundary of that region may or may not be comprised in the region itself. If a region is a connected component (see Section 3.4), its boundary is empty. This is the case, for example, of the region depicted in Figure 6c.



(a) A region with empty interior

(b) A region with empty exterior

(c) A region with empty boundary, border and confine

Figure 6: Examples of regions' anatomies

In the original formulation of the mereotopology of discrete space proposed in [Gal99] there is no concept that clearly represents the "physical inner limit" and "physical outer limits" of a region. The boundary, in fact, does not differentiate between cells that are inside and outside the region. To fill this gap, we introduce the concepts of *border* and *confine* of a region to represent that region's inner- and outer limits, respectively. The border R_{brd} of the region R is made of the cells that are in R and that have at least one neighbor cell outside R:

$$R_{brd} := \{c \mid c \in R \wedge \exists\, c' \in \overline{R} \,:\, \mathbf{A}(c, c')\} \qquad \text{(Region's Border [Alt. 1])}$$

Put in terms of neighborhood, the border of a region is made of the cells in that region whose neighborhood overlaps with the complement of the region:

$$R_{brd} := \{c \mid c \in R \wedge \mathbf{O}(N_c, \overline{R})\} \qquad \text{(Region's Border [Alt. 2])}$$

However, the most intuitive characterization of the border of a region is the following: the border of a region R is the proper subregion of R containing all and only the cells that are not included in the internal of R:

$$R_{brd} := R \setminus R_{int} \qquad \text{(Region's Border [Alt. 3])}$$

All the above characterizations of the border of a region are easily proven equivalent using the theorems and lemmas demonstrated in [Gal99].

The confine of a region R, denoted by R_{cnf}, is the region made of the cells that are not in R, but whose neighborhood overlaps with it. Formally:

$$R_{cnf} := \{c \mid c \notin R \wedge \mathbf{O}(N_c, R)\} \qquad \text{(Region's Confine [Alt. 1])}$$

In other words, the confine of the region R is the (possibly empty) subregion of R's boundary that does not overlap with R's border:

$$R_{cnf} := R_{bnd} \setminus R_{brd} \qquad \text{(Region's Confine [Alt. 2])}$$

Additionally, the confine of a region can be characterized as the difference between the complement of that region and its exterior:

$$R_{cnf} := \overline{R} \setminus R_{ext} \qquad \text{(Region's Confine [Alt. 3])}$$

The Practice. Interior, exterior, boundary, border and confine are extremely useful concepts when applied to process fragments. Fundamentally, the border of a process fragment is its "interface" with the rest of the process model. Conversely, the confine is the interface of the rest of the process model to the process fragment. From the standpoint of the change management of

process models, provided the absence of "ripple effects", the exterior of the change area can be considered unaffected by the change. And when ripple effects do occur, assuming that they propagate over syntactic dependencies, the confine can be used to iteratively calculate the ripples, i.e. the parts of the process model that at each consecutive step become affected. Straightforwardly, the ripple at the n-th step of propagation is the confine of the ripple at the step $n - 1$.

Another immediate application of the anatomy of process fragment is in the scope of distributed execution of process models. For example, the decomposition technique proposed in [Kha08] for executing distributedly WS-BPEL process models introduces message exchanges between the resulting process fragments in place of the original sequence flows that are "broken" during the decomposition. In other words, the control control flows that are replaced with message exchanges are those laying in the border of the process fragments. Therefore, calculating the border of a process fragment can simplify the detection of those process elements that must be adjusted to enable the distributed execution of the overall process, while the process elements in the interior should not require adjustments.

4 Discussion

This section discusses our contribution in the scope of the research on process fragments (Section 4.1), and the future work regarding the application of spatial logics to process fragments (Section 4.2).

4.1 Our Contribution to Process Modeling

The research on process fragments would greatly benefit from increasingly formal, language agnostic approaches grounded, for example, in spatial logics (e.g. the mereotopology of discrete space here used) and language theory, i.e. the synthetic representation of the traces that models (e.g. process models and process fragments) can execute.

The independence from particular process-modeling languages is a fundamental aspect that has not been tackled so far. We have been experiencing the introduction of numerous new process-modeling languages, as well as new versions of existing ones, with remarkable diversity and variety in the syntax and operational semantics so that extensive classifications of their constructs was deemed to be necessary, see e.g. [AHKB03]. Furthermore, the growing popularity of Domain Specific Languages (DSLs) and the interest they receive suggests that the future will bring even more domain-specific

process-modeling languages focused on modeling business processes, scientific workflows, service compositions, mash-ups, and so on. As process-modeling becomes integral to the practice of more and more domains, the need for a language agnostic theory of process fragments becomes pressing.

To the best of our knowledge this work is the first to propose the application of spatial logics, such as the mereotopology of discrete space, to process fragments. However, it is interesting to notice that the term “region”, which has strong topologic connotations, is often connected to process fragments in the literature. In the context of change and evolution of process models, a change region is a set containing the process elements that are affected by any particular change, see e.g. [EKR95, RD98, MBRS06, RWR06, WRR07, WRRM08]. Another attempt for interpreting process fragments as regions can be found in [HFKV06]. In this reference, the term region is used to denote an arbitrary subset of the process elements of a process model, i.e. the region is defined as an integral part of the process model.

The mereotopology of discrete space is a simple and yet powerful framework to study relations between and among process fragments that are based on their syntactic structure. This versatile approach helps developing language agnostic theories for process fragments. In particular, it provides general, formal definitions of familiar concepts like process fragments’ overlapping, disjointness and self-connectedness. Moreover, it enriches the vocabulary of process fragments with primitives to describe topologic relations (i.e. based on connectedness) and the anatomy of process fragments. The “entry barrier” to applying the concepts presented in this work to process fragments specified with any given process-modeling language simply amount to identifying (1) what the process elements are and (2) what syntactic relationships bind them.

The framework presented in this work has a number of potential applications to the research on and the practice of process fragments. First of all, it can be used as foundation for formal, language agnostic categorizations of the different typologies of process fragments. Moreover, the language agnostic definitions of properties of process fragments included in the framework appear often, albeit in a language dependent formulation, in approaches to the analysis and management of process models and process fragments. For example, throughout this work we have discussed the immediately applicability of our framework to the change management of process models. Leveraging the framework here presented in new analysis methods (or reformulations of the already available ones) would increase their portability and applicability across the various process-modeling languages in the state of the art. Additionally, the concepts here presented are of easily implementation in software and integration in process-modeling tools. To assist the adoption of our framework in the scope of process modeling, we have realized the M3DUS4

Java library[2] that allows to model and manage mereotopologic spaces.

4.2 Beyond the Mereotopology of Discrete Space

The adjacency relation in the mereotopology of discrete space is *qualitative*: two given cells are either adjacent, or they are not. One direct effect of this is that the mereotopologic properties of a space and the regions identified on it are independent from the actual spatial distribution of its cells. That is, as long as the adjacency relations between the cells in a space are not altered, those cells can be arranged spatially according to any layout without modifying the mereotopologic properties of the space and its regions [Var98]. This is advantageous in the quest for shared, intensional definitions of the typologies of process fragments. In fact, this means that the definitions of the structural properties of process fragments depend only on their syntactic structure, and not on the layout of the process models and the spatial aspects of the secondary notation they embed.

If, however, our work were to be extended to cover the human dimension of process fragments in process modeling, e.g. the secondary notation, would require, for example, (1) a *quantitative* notion of adjacency, e.g. distance between process elements in terms of units of length, and (2) a notion of *orientation* of process elements (at least in non-textual process-modeling languages). To this end, we think that the application of *digital topology*, defined in [KR89] “the study of the topological properties of image arrays”, to process models and process fragments is a topic that deserves research scrutiny.

Acknowledgments

The authors thank Manuel Carro Liñares and Vasilios Andrikopoulos for the feedback and the fruitful discussions from which this work has profited.

References

[AHKB03] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[2] M3DUS4 homepage: `http://code.google.com/p/m3dus4`

[BYM+98] I. D. Baxter, A. Yahin, L. M. de Moura, M. Sant'Anna, L. Bier. Clone Detection Using Abstract Syntax Trees. In *ICSM*, pp. 368–377. 1998.

[CKO92] B. Curtis, M. I. Kellner, J. Over. Process Modeling. *Commun. ACM*, 35(9):75–90, 1992.

[DKL09] O. Danylevych, D. Karastoyanova, F. Leymann. Optimal Stratification of Transactions. In *ICIW*, pp. 493–498. IEEE Computer Society, 2009.

[EKR95] C. A. Ellis, K. Keddara, G. Rozenberg. Dynamic change within workflow systems. In *COOCS*, pp. 10–21. ACM, 1995.

[EW05] J. Evermann, Y. Wand. Toward Formalizing Domain Modeling Semantics in Language Syntax. *IEEE Trans. Software Eng.*, 31(1):21–37, 2005.

[Gal99] A. Galton. The Mereotopology of Discrete Space. In *COSIT*, volume 1661 of *Lecture Notes in Computer Science*, pp. 251–266. Springer, 1999.

[GKW08] T. Gschwind, J. Koehler, J. Wong. Applying Patterns during Business Process Modeling. In *BPM*, volume 5240 of *Lecture Notes in Computer Science*, pp. 4–19. Springer, 2008.

[HFKV06] R. Hauser, M. Friess, J. M. Küster, J. Vanhatalo. Combining Analysis of Unstructured Workflows with Transformation to Structured Workflows. In *EDOC*, pp. 129–140. IEEE Computer Society, 2006.

[HKO07] T. Hornung, A. Koschmider, A. Oberweis. Rule-based Autocompletion of Business Process Models. In *CAiSE Forum*, volume 247 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[HR00] D. Harel, B. Rumpe. Modeling Languages: Syntax, Semantics and All That Stuff - Part I: The Basic Stuff. Technical Report MCS00-16, The Weizmann Institute of Science, 2000.

[Kha08] R. Khalaf. *Supporting business process fragmentation while maintaining operational semantics: a BPEL perspective.* Ph.D. thesis, University of Stuttgart, 2008.

[KR89] T. Y. Kong, A. Rosenfeld. Digital topology: Introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48(3):357–393, 1989.

[LD99] F. Lindert, W. Deiters. Modelling Inter-Organizational Processes with Process Model Fragments. In *Enterprise-wide and Cross-enterprise Workflow Management*, volume 24 of *CEUR Workshop Proceedings*, pp. 33–41. CEUR-WS.org, 1999.

[MBRS06] J. A. Mülle, K. Böhm, N. Roper, T. Sunder. Building Conference Proceedings Requires Adaptable Workflow and Content Management. In *VLDB*, pp. 1129–1139. ACM, 2006.

[ML08] Z. Ma, F. Leymann. A Lifecycle Model for Using Process Fragment in Business Process Modeling. In *Proceedings of the 9th Workshop on Business Process Modeling, Development, and Support (BPDMS 2008)*. 2008.

[NCS04] M. G. Nanda, S. Chandra, V. Sarkar. Decentralizing execution of composite web services. In *OOPSLA*, pp. 170–187. ACM, 2004.

[OMG09] OMG. Business Process Model and Notation Version 2.0. OMG Specification, Beta 1, Object Management Group, 2009.

[Pel03] C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, 2003.

[Pet06] M. Petre. Cognitive dimensions 'beyond the notation'. *J. Vis. Lang. Comput.*, 17(4):292–301, 2006.

[PSW08] A. Polyvyanyy, S. Smirnov, M. Weske. Process Model Abstraction: A Slider Approach. In *EDOC*, pp. 325–331. IEEE Computer Society, 2008.

[PW09] A. Polyvyanyy, S. S. 0002, M. Weske. The Triconnected Abstraction of Process Models. In *BPM*, volume 5701 of *Lecture Notes in Computer Science*, pp. 229–244. Springer, 2009.

[Raj97] V. Rajlich. A Model for Change Propagation Based on Graph Rewriting. In *ICSM*, pp. 84–91. IEEE Computer Society, 1997.

[RD98] M. Reichert, P. Dadam. ADEPT$_{flex}$ - Supporting Dynamic Changes of Workflows Without Losing Control. *J. Intell. Inf. Syst.*, 10(2):93–129, 1998.

[RWR06] S. Rinderle, A. Wombacher, M. Reichert. Evolution of Process Choreographies in DYCHOR. In *OTM Conferences (1)*, volume 4275 of *Lecture Notes in Computer Science*, pp. 273–290. Springer, 2006.

[Van09] J. Vanhatalo. *Process Structure Trees: Decomposing a Business Process Model into a Hierarchy of Single-Entry-Single-Exit Fragments.* Ph.D. thesis, Institut für Architektur von Anwendungssystemen der Universität Stuttgart, 2009.

[Var98] A. C. Varzi. Basic Problems of Mereotopology. In *Formal Ontology in Information Systems, IOS*, pp. 29–38. IOS Press, 1998.

[VVK09] J. Vanhatalo, H. Völzer, J. Koehler. The refined process structure tree. *Data Knowl. Eng.*, 68(9):793–818, 2009.

[VVL07] J. Vanhatalo, H. Völzer, F. Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In *ICSOC*, volume 4749 of *Lecture Notes in Computer Science*, pp. 43–55. Springer, 2007.

[WRR07] B. Weber, S. Rinderle, M. Reichert. Change Patterns and Change Support Features in Process-Aware Information Systems. In *CAiSE*, volume 4495 of *Lecture Notes in Computer Science*, pp. 574–588. Springer, 2007.

[WRRM08] B. Weber, M. Reichert, S. Rinderle-Ma. Change patterns and change support features - Enhancing flexibility in process-aware information systems. *Data Knowl. Eng.*, 66(3):438–466, 2008.