Business process models and entity life cycles

International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

Business process models and entity life cycles

Giorgio Bruno
Politecnico di Torino
Corso Duca degli Abruzzi 24, Torino 10129
Italy
giorgio.bruno@polito.it

Abstract:
Tasks and business entities are the major constituents of business processes but they are not always considered equally important. The activity-centric approach and the artifact-oriented one have radically different visions. The former focuses on the control flow, i.e., on the representation of the precedence constraints between tasks, and considers the dataflow an add-on. The latter emphasizes the states of the business entities and defines the transitions between states in a declarative way that makes it difficult to figure out what the control flow is. This paper presents the ELBA notation whose purpose is to integrate those different visions by leveraging the dataflow. The dataflow defines the input and output entities of the tasks in process models. Entities flowing through tasks change their states and then a process model results from the combination of the life cycles of the entities managed by the process. Process models are complemented by information models that show the attributes and relationships of the entity types handled by the processes. Life cycles are intertwined in process models but they can be separated by means of an extraction technique that is illustrated in this paper with the help of two examples.

Keywords:
business processes; control flow; dataflow; artifact; entity life cycle.

DOI: 10.12821/ijispm070304

Manuscript received: 28 March 2019
Manuscript accepted: 16 May 2019

1. Introduction

Business processes are made up of tasks that are usually meant to operate on persistent business entities, but, unfortunately, these two elements, tasks and entities, are not given the same level of importance in the current notations for business process modeling. In BPMN (Business Process Model and Notation) [1], business entities are denoted by variables in process instances and can optionally be shown in process models by means of graphical elements called data objects.

BPMN is an activity-centric notation: the control flow organizes the execution of tasks in a predefined manner. Tasks are divided into automatic tasks and human ones. Human tasks are associated with roles and are performed by persons playing the roles required; however, they have little influence on the evolution of the process and are mainly involved in tasks that are not easy to automate. BPMN fits repetitive situations, i.e., routines, and provides an efficient way to handle them.

While BPMN puts tasks before business entities, the opposite takes place with GSM (Guard-Stage-Milestone) [2]. GSM emphasizes that business entities evolve over time through life cycles made up of states (also called stages); tasks are included in stages. The overall process is a collection of interacting life cycles.

Knowledge-intensive processes [3] focus on individual cases and rely on the expertise of participants to find the best way to handle them. Flexibility is the major issue and, then, a declarative approach to business process modeling is preferred to the traditional procedural one. In GSM, stages are activated and closed by means of ECA (Event Condition Action) rules [4] and this confers great flexibility to the approach. CMMN (Case Management Model and Notation) [5] draws on GSM and provides process participants with the flexibility of deciding the activities to be carried out; moreover, participants can assign tasks to each other. BPMN provides some flexibility with the ad-hoc construct: a comparison with CMMN can be found in Zensen and Küster [6].

What is missing in the above-mentioned approaches is the explicit representation of the dataflow, which defines the input and output entities of the tasks in process models. Entities flowing through tasks change their states: states indirectly represent the processing performed by tasks on entities. A process then combines the life cycles of the entities involved.

This paper presents a notation, called ELBA, which addresses the above issues. Life cycles are intertwined in process models but they can be separated by means of an extraction technique. Since a life cycle emphasizes the evolution over time of an entity type its model may be important for a comparison with industry standards or with the expectations of stakeholders.

The extraction of life cycles is challenging for two reasons: one is their intertwining in process models, and the other is due to hidden states. Hidden states are states that do not explicitly appear in process models because the presence of entities in these states is not an event for the process. Such states might be ignored; however, if they are considered important, they can be made explicit so the extraction algorithm will show them in life cycle models.

This paper is organized as follows. Section 2 is about the related work. Section 3 introduces ELBA and shows how life cycles can be extracted from a simple process model. Section 4 concerns the handling of hidden states. Section 5 contains the conclusion.

2. Background

Most of the notations for business process modeling fall in three categories based on the predominant aspect: the control flow, the dataflow and the entity life cycles.

The activity-centric notations, such as UML (Unified Modeling Language) [7] activity diagrams and BPMN, focus on the control flow, whose purpose is to define precedence constraints between tasks by means of direct links or through the intermediation of control flow elements (called gateways in BPMN). This approach provides an efficient way to

deal with repetitive situations, i.e., routines. The dataflow may be added to the above mentioned notations but in a way that Sanz [8] judges as an "afterthought". The control flow and the dataflow are separate and this makes the notation more complicated.

Combi et al. [9] present an extension to BPMN whose purpose is to link tasks to data: the extension consists in textual annotations that describe the operations performed by tasks on data stored in a database. Moreover, the attributes and relationships of the data are modeled with a UML class diagram.

Another kind of extension is aimed at associating resources (i.e., IoT devices) to BPMN tasks. For example, Martins and Domingos [10] show how to translate a BPMN model into a programming language for IoT devices.

The artifact-oriented approach shifts the focus from tasks to business entities. It does so by introducing the notion of artifact, which, in this context, represents an entity type and the life cycle of its entities. Artifacts designate concrete and self-describing chunks of information used to run a business [11]; moreover, they facilitate communication among stakeholders and help them focus on the primary purposes of the business [12].

The notation provided by GSM focuses on the life cycles of artifacts: they are separate and are made up of stages, which may include subordinate stages and tasks. Stages are provided with guards and milestones, which consist of ECA rules. When a guard becomes true, the stage is activated; the stage is closed, when a milestone becomes true. The event that causes the opening of a stage may come from a stage of another artifact. This mechanism confers great flexibility to the approach, but its major drawback is the difficulty of understanding the precedence between the stages. The control flow between stages is not shown; moreover, the dataflow is missing because the life cycles are separate.

In PHILharmonicFlows [13], the entity life cycles are represented with state-transition diagrams (called micro processes), and then the precedence constraints between stages are clearly shown. Since life cycles are separate the notation introduces a coordination mechanism (called macro process) to orchestrate their evolution: this entails the repetition of several stages in the macro process. However, not all the stages are repeated in the macro process; therefore, the observation of all the models is needed to fully understand the overall process.

Life cycles may be extracted from notations that provide the representation of the dataflow. Extraction approaches have been proposed for UML Activity Diagrams ([14], [15], [16]) as well as for BPMN models [17].

The recent case management standard CMMN stresses the abilities of case workers to decide the order of execution of tasks and to assign tasks to each other. It draws on GSM in that processes are made up of stages that are groups of tasks: the opening and closing of stages are based on events, such as the completion of a task, a time event or a human decision. However, the stages are not related to the life cycles of the business entities and, moreover, the dataflow is missing.

If the control flow and the dataflow are integrated, the tasks in the process turn out to be data-driven: they are performed when their inputs contain suitable entities.

In the case handling approach [18], tasks are data driven but the dataflow is not shown because the process data are kept in process variables; however, the process includes the links between the tasks and the variables that provide their inputs.

Two frameworks have been proposed to compare notations on the basis of their data modeling capabilities. The first framework [19] focuses on the representation of the dataflow and of the life cycles of the entities that make it up. As to BPMN and UML activity diagrams, the authors conclude that data modeling is optional and when the dataflow is shown it is subordinate to the control flow.

The second framework [20] is used to compare notations for business process modeling on the basis of 24 criteria subdivided in four groups: design, implementation and execution, diagnosis and optimization, tool implementation and practical cases. The authors compared three notations, i.e., case handling, GSM and PHILharmonicFlows. The conclusion is that most practitioners consider data-centric notations more complicated than activity-centric ones and therefore data-centric approaches need further research.

The ELBA notation provides a solution giving equal importance to tasks and business entities. Tasks are data driven and the dataflow is explicitly shown. The dataflow consists of entity states and the entity types along with their attributes and relationships are defined in an information model, similar to a UML class diagram. The entity life cycles are intertwined in the dataflow but they can be separated as illustrated in the next sections.

Data resides in the entities forming the dataflow: a process is implemented with a single instance and therefore it is easy to decompose and recompose entity flows. As a consequence, situations like the many-to-many mapping between requisition orders and procurement ones in build-to-order processes, which are difficult to handle with BPMN as pointed out in Meyer et al. [21], can be easily addressed with ELBA [22].

As to flexibility, ELBA [23] addresses human decisions that concern the selection of input entities when a task needs more than one, or the selection of the task when more than one are admissible to handle the input entities.

3. Introduction to ELBA

The main feature of ELBA is that a process model results from the combination of the life cycles of the entities managed by the process. The definition of the entities takes place through an information model that shows the entity types along with their attributes and relationships. In addition, ELBA is a dataflow language in that the activation of tasks is based on the input entities and not on the completion of previous tasks.

An ELBA process comes with a single instance: this implies that the entities handled by a process are not internal to the process but external to it. Entities are assumed to reside in an information system that guarantees their persistence.

A process model actually consists of two models: one is the information model that shows the domain related to the process and the other is the actual process from which the separate life cycles of the managed entities can be extracted.

This section presents a simple example to explain the basic features of the notation. The example concerns process CheckProposals, which operates in an organization that receives proposals from partners: the process checks the proposals with the help of a number of reviewers.

The simplified requirements of the process are as follows. Partners and reviewers are registered in the information system. The relations between the partners and the organization are managed by persons playing the account manager (shortly, accountMgr) role: each partner is associated with one account manager who can deal with a number of partners. After a proposal has been entered by a partner, the process hands it to the suitable account manager, who selects three reviewers and associates them with the proposal. Each reviewer is required to submit a review and, when all the three reviews are available, the account manager decides whether to accept or reject the proposal. Then, the partner is notified of the outcome. The attributes of the entities are kept to a minimum: a proposal has a description and an outcome (which initially is null and then becomes accepted or rejected), and a review has a comment. The process model and the information one are shown in Fig. 1.

The information model draws on the UML class diagram and adds a number of features such as required properties, rules, and indirect relationships.

Entity types may be divided into three groups: role types, managed types and background ones. A role type denotes the process participants playing the role that it represents. For example, role type Reviewer denotes all the persons acting as reviewers and an instance of it denotes a specific reviewer. Instances of role types are referred to as role entities.

Managed types, such as Proposal and Review, are the types of the entities forming the dataflow of the process. Background types denote entities that provide background information: the process does not generate background entities but may introduce associations between managed entities and background ones. In the example under consideration, there are no background entities.

Relationships show the constraints on the number of associations between entities and they do so by means of multiplicity indicators. Indicators include integer values as well as symbols “n” and “*” which mean one or more associations, and zero or more associations, respectively. The standard multiplicity is “*”.

Required properties encompass required attributes and required associations. The former are recognizable by the names underlined and the latter by the multiplicities underlined. Required properties must be set when new entities are generated. For example, a new proposal is linked to the partner who entered it; moreover, a new review is connected to the reviewer who provided it and to the proposal it is related to.

Indirect relationships are sequences of direct relationships. An example is given by the indirect relationship between Proposal and AccountMgr: a proposal is indirectly linked to an account manager through the association with a partner, and an account manager is indirectly linked to the proposals of the partners he or she is associated with.

Rules will be explained in the next section.

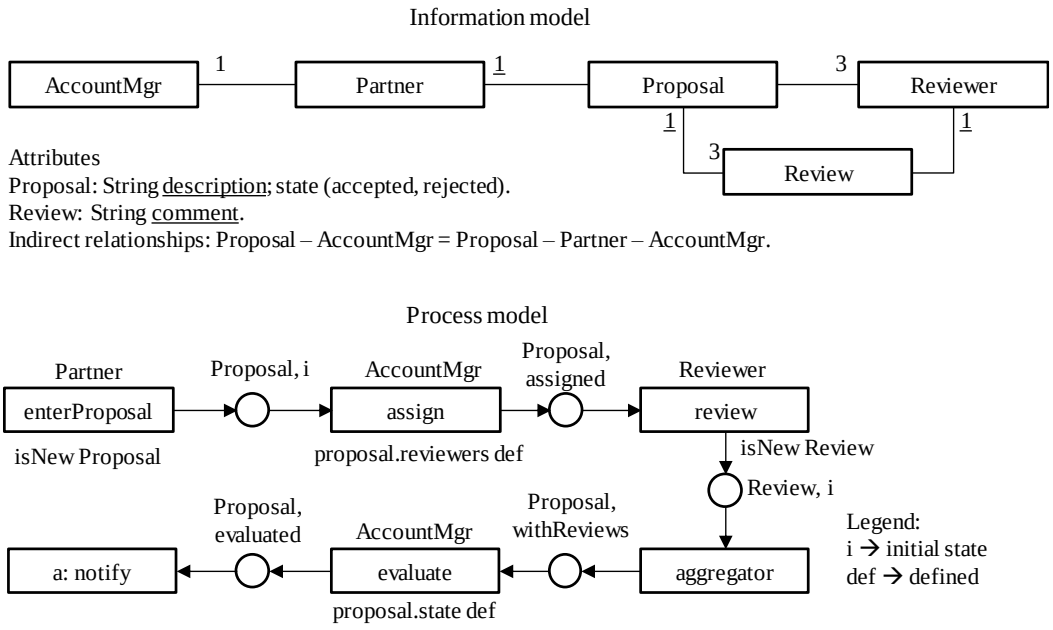


Fig. 1. The model of process CheckProposals and the related information model

A process model is a connected bipartite graph made up of tasks and entity states. The symbols of tasks and entity states are the rectangle and the circle, respectively. Tasks and entity states are connected by means of oriented arcs that establish input and output associations between them. The structure of ELBA processes has been inspired by Petri nets [24]: entity states correspond to places and tasks correspond to transitions.

Entity states have labels consisting of two parts: the entity type, which matches a managed type appearing in the information model, and the state name. The states indicate the progress of the entities in their life cycles.

The analysis of the input and output states of tasks leads to the identification of various kinds of tasks, the most frequent of which are entry tasks, exit tasks, transitional tasks and mapping ones. An entry task has no inputs: its purpose is to introduce new entities in a process. An exit task has no outputs in that it removes the input entities from the process.

In a transitional task the input and output entities have the same type. When the task is completed the input entities are moved into the output states and, therefore, they undergo a change of state. A mapping task has one input state and one output state whose types are different, say, T1 and T2. The effect of the task is to map an input entity into a number of

output entities: the output entities are those associated with the input one on the basis of the relationship between types T1 and T2.

Tasks are divided into automatic tasks and human ones. The former are told apart because their names are preceded by "a:" and the latter are accompanied by the role of the performers (which is shown next to the task symbol). Tasks have a number of features, i.e., pre-conditions, post-conditions, and assignment rules. Pre-conditions and post-conditions are boolean expressions based on the input entities and draw on OCL (Object Constraint Language) [25]. In a declarative manner, the former express the conditions required for the execution of the tasks and the latter define the effects produced on the underlying information system.

A human task is carried out by a process participant playing the role required by the task. The determination of the actual performer can be established by means of rules. ELBA provides two standard rules. The first rule concerns tasks having no inputs: any person playing the role required by the task is entitled to perform it. In the other cases, the second rule is applied. It assumes that the input entities of a task are associated with one or more role entities whose types match the role required by the task. Since role entities denote process participants, the associations between input entities and role entities determine the process participants who can operate on the input entities by performing the task. The standard rules can be replaced by specific assignment rules; in the examples, there are no exceptions to the standard rules.

The process model shown in Fig. 1 is made up of six tasks and five entity states. The post-conditions are shown below the corresponding tasks.

The first task, enterProposal, is an entry task in that it has no inputs; any partner can perform it. The purpose of the task is to enter a new Proposal in the process, as shown by the post-condition "isNew Proposal". The effect of the execution of the task is the presence of a new proposal in the information system. From the dataflow point of view, the new proposal is put into the output state of the task because the type of the output state matches the type of the new entity. This state is the initial state of proposals. By convention, the name of any initial state is "i" (the first character of initial).

If the type, say, T1, of a new entity is subject to a required relationship with type, say, T2, then the new entity must be associated with a T2 entity. This association can be carried out automatically if a T2 entity is found in the contextual entities of the task that brought the new entity into existence. The contextual entities of a task consist of the performer (in case of a human task) and the input entities. Since there is a required relationship from type Proposal to type Partner, a new proposal must be associated with a partner. Task enterProposal has one contextual entity, the performer of the task, whose type is Partner; therefore the performer is automatically connected to the new proposal.

The second task, assign, enables an account manager to assign the proposal to three reviewers, as shown by post-condition "proposal.reviewers def". The number of reviewers is taken from the multiplicity of the relationship Proposal – Reviewer. Due to the indirect relationship Proposal – AccountMgr defined in the information model, the account manager who is entitled to operate on the proposal is the one associated with the partner that has entered the proposal.

The purpose of task assign is to associate the proposal with three reviewers selected by the performer. The choice can be based on various criteria such as the load balancing of the reviewers, but, from a declarative perspective, the effect is that the proposal turns out to be linked to three reviewers. The construct "proposal.reviewers" denotes the collection of the reviewers associated with the proposal. It is a navigational expression in which "proposal" represents the input entity: the input entity is referred to by means of its type name with the initial in lower case. The dot between proposal and reviewers shifts the focus from the proposal to the reviewers. Term "reviewers" is an associative attribute whose name is taken from type Reviewer: since the multiplicity of the associative attribute is greater than one, the plural of the type name with the initial in lower case is used. The effect of the post-condition "proposal.reviewers def", where def is the abbreviation of "defined", implies that the collection of reviewers, which initially is empty, will not be empty at the end of the task. On the basis of multiplicity 3 of the relationship from Proposal to Reviewer, the collection will refer to three reviewers. Task assign is a transitional task in that it has an input state and an output one of the same type: the input proposal is then moved into state assigned.

Task review enables the reviewers of a proposal to produce reviews for it as shown by the post-condition “isNew Review”. A proposal is linked to three reviewers as shown by relationship Proposal – Reviewer; then all the reviewers of a proposal get the same proposal but each of them provides a different review. Due to the required relationships Review – Proposal and Review – Reviewer, each review is automatically connected to the input proposal and to the task performer (a reviewer).

At the end of each execution of task review, a new review enters the output state, which is the initial state of reviews. The process does not handle the reviews individually but deals with the proposal when all the reviews are available. In such cases, ELBA uses an automatic task called aggregator, whose requirements are as follows. The aggregator has one input state and one output state whose types (say, T1 and T2) are different but interrelated: the input type (T1) and the output one (T2) participate in a relationship with multiplicity many to one. When all the entities T1 related to the same entity T2 are present in the input state, the aggregator removes them and outputs the entity T2. Therefore, the aggregator shown in Fig. 1 puts a proposal into the output state when its reviews are available in the input state. Three reviews are needed on the basis of the multiplicity of type Review with respect to type Proposal in the information model.

Task evaluate is a transitional task because the input entities and the output ones have the same type. From the dataflow point of view, the effect is a change of state of the input entities. The effect on the input proposal is a modification of the outcome attribute as shown by the post-condition “proposal.outcome def”. The performer decides the value of the outcome, which initially is null, by selecting one of the two values, accepted or rejected, specified in the information model.

The last task, notify, is an automatic task which notifies the outcome of the proposal to the partner. It is an exit task in that it removes the input entity from the process. The post-condition is omitted because it has no impact on the information system.

The process model is a combination of different life cycles which are intertwined, but it can be useful to show them separately. The extraction of the life cycles is carried out with a simple algorithm as follows. The algorithm assumes that the resulting life cycles are sequential state models; therefore concurrent states are not handled.

The algorithm is informally explained with reference to the process model shown in Fig. 1. Firstly, it adds output states to exit tasks and aggregators. The output states added to an exit task have the same type as the input ones and their name is “final”. The algorithm may add a new output state to an aggregator: the new state has the same type as the input state of the aggregator and its name is “final”. The addition is made only if there are no other states of the same type in the outgoing paths of the aggregator. If there are other states, the life cycle of the input entities continues after the aggregator. The changes made to the process model by the algorithm are shown in Fig. 2.

For each managed type, say, T, the algorithm generates a life cycle graph whose nodes correspond to the states of type T in the process. The names of the nodes are the names of the states. For each node, say, N, of the graph, it determines the immediate successors and links node N to them: the successors are obtained by following the paths starting from the state corresponding to node N in the process model. For type Proposal, the graph is a linear sequence of nodes as shown in Fig. 2; likewise for type Review.

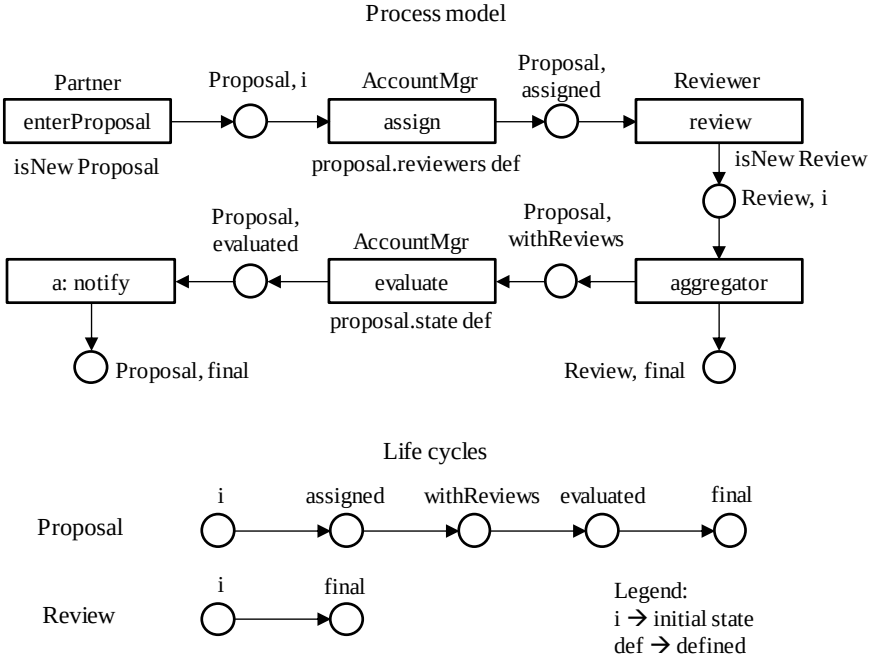


Fig. 2. The modified model of process CheckProposals and the life cycles extracted

4. Hidden states

The entity states that appear in a process model show the dataflow of the process; however, it may happen that certain states that are considered important to describe an entity life cycle do not explicitly appear in the process model because the presence of entities in these states does not bring about the activation of tasks. In such cases, these states, which are referred to as hidden states, may be made visible so that the algorithm that extracts the entity life cycles from the process model can include them. The example illustrated in this section explains how to make visible the hidden states.

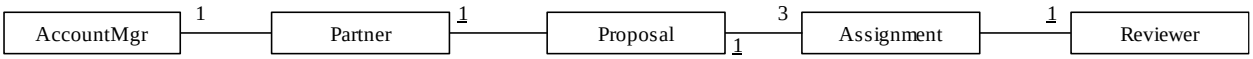
The example is a variant of process CheckProposals illustrated in the previous section; the variant is named CheckProposals2. The difference is that account managers do not directly associate reviewers with proposals but generate three assignments for each proposal; an assignment is connected exactly to one proposal and one reviewer. Reviewers fulfill their assignments and when all the expected assignments for a proposal have been fulfilled, the account manager can decide the outcome of the proposal. The information model of the process is shown in Fig. 3, while the process model and the entity life cycles are presented in Fig. 4.

The annotations of the information model include a rule prescribing that a proposal must be assigned to different reviewers. The rule is based on the navigational expression “proposal.assignments.reviewer”, where the first term, proposal, stands for any proposal. For any proposal, the collection of reviewers related to the assignments associated with the proposal must contain different elements.

Unlike the first process model, task assign is not a transitional task because the types of its input and output entities are different. It is instead a mapping task as it maps a proposal into three assignments.

The effect of task assign is the presence of three new assignments after its execution. The number of assignments is indicated between parentheses in the post-condition. The required relationships of type Assignment imply that a new assignment must be linked to a proposal and a reviewer. The contextual entities of task assign are the input proposal and the performer. The input proposal is automatically associated with the new assignment. Since no reviewer is found in

the contextual entities of the task, the reviewer is selected by the performer of the task during its execution. Rule 1 (included in the information model) guarantees that the new assignments are linked to distinct reviewers. The output state of the task is the initial state of the entities of type Assignment.



Attributes
Proposal: String description; outcome (accepted, rejected).
Assignment: String review.
Indirect relationships: Proposal – AccountMgr = Proposal – Partner – AccountMgr.
Rule 1: proposal.assignments.reviewer distinct

Fig. 3. The information model of process CheckProposals2

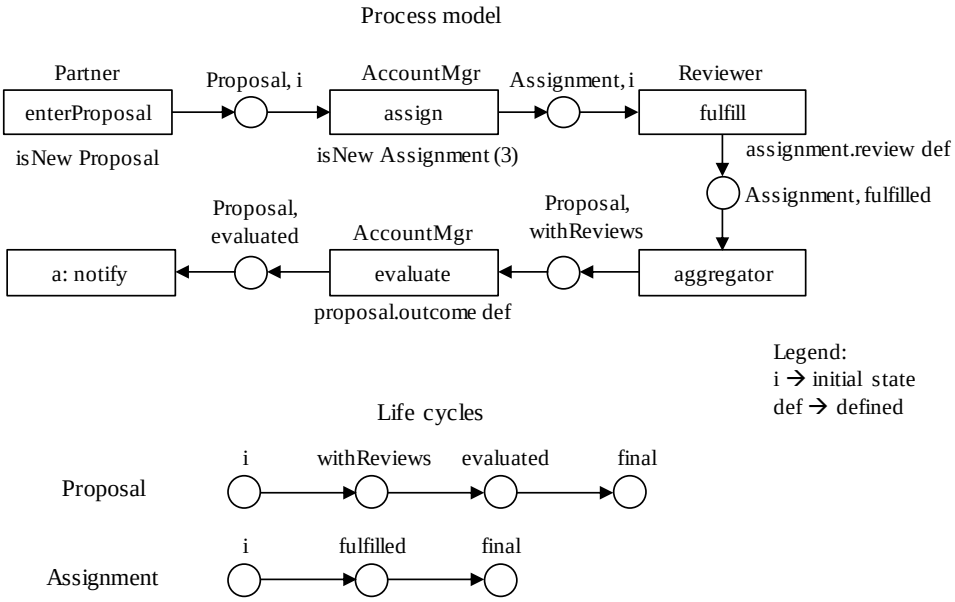


Fig. 4. The model of process CheckProposals2 and the entity life cycles extracted

The name (withReviews) of the output state of the aggregator indicates that the assignments associated with the proposal contain the reviews provided by the reviewers.

The Proposal life cycles extracted from processes CheckProposals and CheckProposals2 are different. The second life cycle has one state (assigned) less than the first one. The reason for this is that task assign is a transitional task in the first process and a mapping one in the second process. State assigned could be a significant state in the common vision of the life cycle of a proposal. In that case, state assigned can be considered a hidden state in the second process: it logically follows the initial state of a proposal as a consequence of task assign and precedes state “withReviews”. This hidden state can be made visible as follows. State “Proposal, assigned” is added to the process model and task assign is connected to it with a dashed arc so as to emphasize that it is a hidden state made visible. The new state has no outgoing links as it activates no tasks. The modified process along with the updated life cycles are shown in Fig. 5.

The algorithm for extracting the entity life cycles includes the new state in the outgoing paths of task assign: as a result, state assigned is the successor of the initial state, and state withReviews is the successor of state assigned. The algorithm adds final states to task notify and to the aggregator task, but these states are not shown in Fig. 5.

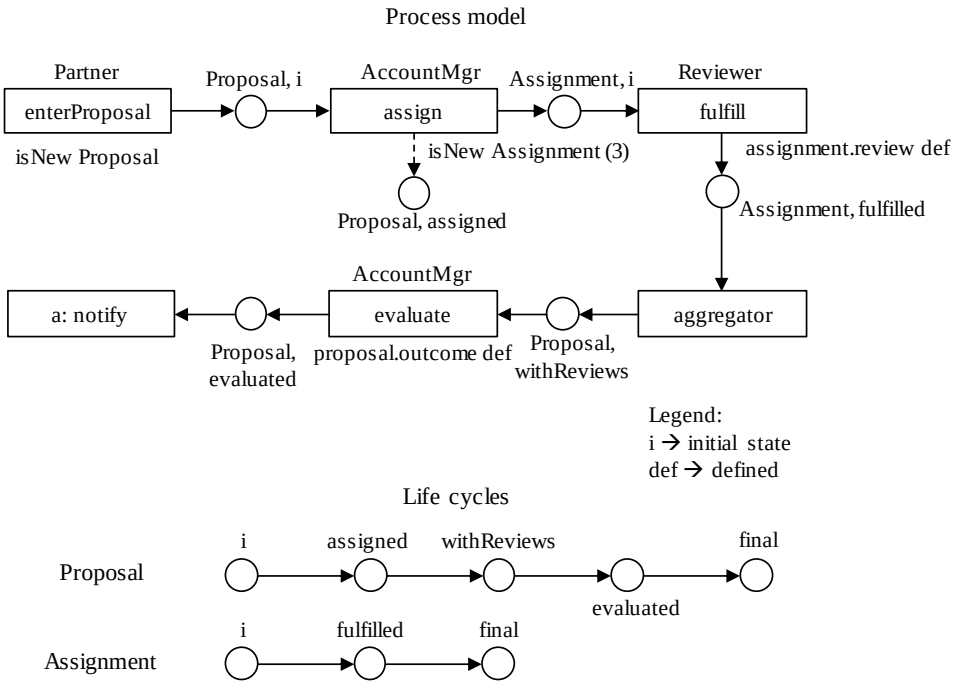


Fig. 5. The model of process CheckProposals2 with hidden states and the life cycles extracted

5. Conclusion

This paper has presented the ELBA notation which is aimed at integrating the notions of dataflow and entity life cycle. The dataflow shows the input and output entities of tasks in terms of types and states. The states of the entities of a given type are the constituents of the life cycle of that entity type.

The main difference between ELBA and GSM, which is the major representative of the artifact-oriented approach, is that process models in ELBA result from the intertwining of entity life cycles, while in GSM they are made up of separate entity life cycles. This paper has described an algorithm that is able to extract the entity life cycles from process models.

The changes of state in two or more life cycles may be interrelated: in GSM the interrelationships are orchestrated by means of ECA rules and are not shown graphically. On the other hand, ELBA considers such interrelationships as the effect of spanning tasks, i.e., tasks having inputs of different types. Such tasks can be used to synchronize input flows of different types.

Further work on ELBA is devoted to the introduction of collaborative features so as to enable processes to interact on the basis of agreed protocols. Such extensions are meant to be applied to the realm of Cyber Physical Systems.

References

[1] Object Management Group (2013, 12, 09). Business Process Model and Notation (2.0.2) [Online]. Available: https://www.omg.org/ spec/BPMN/2.0.2/

[2] R. Hull et al., "Introducing the Guard-Stage-Milestone approach for specifying business entity lifecycles," in Proc. 7th International Workshop on Web Services and Formal Methods, Hoboken, NJ, USA, 2010, pp. 1-24.

[3] C. Di Ciccio, A. Marrella, A. Russo, "Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches," Journal on Data Semantics, vol. 4, no.1, pp. 29-57, March, 2015.

[4] N. Gehani, H.V. Jagadish and O. Smueli, "Event specification in an active object-oriented database," in Proc. of the ACM-SIGMOD International Conference on Management of Data, San Diego, USA, 1992, pp. 81-90.

[5] Object Management Group. (2016, 12, 01). Case Management Model and Notation (1.1) [Online]. Available: https://www.omg.org/spec/ CMMN/1.1/

[6] A. Zensen and J. Küster, "A comparison of flexible BPMN and CMMN in practice: a case study on component release processes," in Proc. 22nd IEEE International Enterprise Distributed Object Computing Conference, Stockholm, Sweden, 2018, pp. 105-114.

[7] Object Management Group. (2017, 12, 05). Unified Modeling Language (2.5.1) [Online]. Available: http://www.omg.org/spec/UML/

[8] J.L.C. Sanz, "Entity-centric operations modeling for business process management - A multidisciplinary review of the state-of-the-art," in Proc. 6th IEEE International Symposium on Service Oriented System Engineering, New York, USA, 2011, pp. 152-163.

[9] C. Combi, B. Oliboni, M. Weske and F. Zerbato, "Conceptual modeling of inter-dependencies between processes and data," in Proc. 33rd ACM Symposium on Applied Computing, Pau, France, 2018, pp. 110-119.

[10] D. Domingos and F. Martins, "Using BPMN to model Internet of Things behavior within business process," International Journal of Information Systems and Project Management, vol. 5, no. 4, pp. 39-51, 2017.

[11] A. Nigam and N. S. Caswell, "Business artifacts: an approach to operational specification," IBM Systems Journal, vol. 42, no. 3, pp. 428-445, 2003.

[12] T. Chao et al., "Artifact-based transformation of IBM Global Financing," in Proc. 7th International Conference on Business Process Management, Ulm, Germany, 2009, pp. 261-277.

[13] V. Künzle and M. Reichert, "PHILharmonicFlows: towards a framework for object-aware process management," Journal of Software Maintenance and Evolution: Research and Practice, vol. 23, no. 4, pp. 205-244, June, 2011.

[14] R. Eshuis and P. van Gorp, "Synthesizing object life cycles from business process models," in Proc. 31st International Conference on Conceptual Modeling, Florence, Italy, 2012, pp. 307-320.

[15] S. Kumaran, R. Liu and F.Y. Wu, "On the duality of information-centric and activity-centric models of business processes," in Proc. 20th International Conference on Advanced Information Systems Engineering, Montpellier, France, 2008, pp. 32-47.

[16] K. Ryndina, J.M. Küster and H. Gall, "Consistency of business process models and object life cycles," in Proc. Workshops and Symposia at MoDELS 2006, Genoa, Italy, 2006, pp. 80-90.

[17] C. Cabanillas, M. Resinas, A. Ruiz-Cortés and A. Awad, "Automatic generation of a data-centered view of business processes," in Proc. 23rd International Conference on Advanced Information Systems Engineering, London, UK, 2011, pp. 352-366.

[18] W.M.P. van der Aalst, M. Weske and D. Grünbauer, "Case handling: a new paradigm for business process support," Data & Knowledge Engineering, vol. 53, no. 2, pp. 129-162, May, 2005.

[19] A. Meyer, S. Smirnov and M. Weske, Data in business processes, Universitätsverlag Potsdam, 2011.

[20] S. Steinau, A. Marrella, K. Andrews, F. Leotta, M. Mecella and M. Reichert, "DALEC: a framework for the systematic evaluation of data-centric approaches to process management software," Software & Systems Modeling, September, 2018, pp. 1-38.

[21] A. Meyer, L. Pufahl, D. Fahland and M. Weske, "Modeling and enacting complex data dependencies in business processes," in Proc. 11th International Conference on Business Process Management, Beijing, China, 2013, pp. 171-186.

[22] G. Bruno, "Business process modeling based on entity life cycles," Procedia Computer Science, vol. 138, pp. 462-469, 2018.

[23] G. Bruno, "Data flow and human tasks in business process models," Procedia Computer Science, vol. 64, pp. 379-386, 2015.

[24] T. Murata, "Petri nets: properties, analysis and applications," Proceedings of the IEEE, vol. 77, no. 4, pp. 541-580, April, 1989.

[25] Object Management Group. (2014, 02, 03). Object Constraint Language (2.4) [Online]. Available: https://www.omg.org/spec/OCL/2.4/.

Biographical notes

Giorgio Bruno
Giorgio Bruno is an associate professor of software engineering with the Department of Control and Computer Engineering, Politecnico di Torino (Polytechnic University of Turin), Italy. His teaching activities are concerned with software engineering and object-oriented programming. In the past he has dealt with languages for robots, real time systems, production control and scheduling, and plant simulation. His current research interests are in the development of notations for the operational modeling of processes in the domains of business applications and Cyber Physical Systems. He has published two books and over 180 refereed papers on the above-mentioned subjects in journals, edited books and conference proceedings. He has served in several program committees of international conferences.