Formalization of Workflows Using Fork-Join Automata

Tudor Jebelean[1] and Anna Medve[2]

[1] RISC, Johannes Kepler University, Linz, Austria
[2] Department of Electrical Engineering and Information Systems, University of Pannonia, Veszprem, Hungary

Abstract. We survey the current practice of using workflows for business process modeling and we present a simple formalism for reasoning about the structural correctness of fork–join workflow diagrams containing *and* and *or* nodes.

Introduction. Currently Business Process Management (BPM) is an important issue in the organization of enterprises. Various formalism and tools suport computer aided BPM, and there is a growing interest from academia and industry towards more sistematic approaches for the description and manipulation of business processes (BPs).

In particular, there is a growing research interest in the formalization of these activities, which include:

- creation of formal (mathematical) models of BPs and of formal languages for the specification of their properties;
- sistematic transformation of BPs, with various purposes (as verification of intermediate results, asurance of data security, etc.);
- verification of properties of BPs, as for instance: conformance of a BP to its specification, absence of deadlocks or infinite loops, conformance of a transformed BP with the original one, etc;
- (semi-)automatic synthesis of BPs from their specification.

The situation is similar to the formalization of the programming languages and of program specifications, which allows formal verification of their properties, as well as to the verification and synthesis of digital circuits. Moreover, a growing research community is now approaching the formalization and verification of web services, but many of these services are in fact based on business processes — see e. g. [18].

Our goal is to study the formalization of business processes and to transfer the results of the research into practice. The theoretical framework benefits from the experience of the first author in the *Theorema* research group (www.theorema.org) [16]. The second author has extensive experience in studying practical examples in industry [20, 19, 21], resulting in a methodology for business process specification and improvement. This methodology has been successfully used in a concrete project for improving the workflows in a large

company, which resulted in the design, analysis, and correction of more than 300 business processes [22, 23].

Our work is motivated on one hand, by the relative lack of rigorous mathematical models of workflows, and on the other hand, by our belief that the practical application of these formal techniques has the potential to increase the robustness and to allow safer improvement of business processes of companies who use business process management tools.

Current Trends in Computer Aided Business Process Management. Several standards for business processes (notations and/or methodology) have been established (e. g. ARIS, ISO 9000, IDEF, Sarbanes-Oxley, CMM/CMMI, BPEL, BPMN , UML, OMG Business Process Metamodel, OMG Business Rules Metamodel – see [4]). Organizations making an investment in workflow software want to be sure that their investment is going to be protected. With standards users can have confidence that essential criteria will be met, hence reducing the risk involved. The majority of organizations do not have a standard, enterprise-wide business process methodology, more have many different methodologies and only few have an enterprise standard process methodology with a dominant process methodology focusing on major process change.

Nowadays, it became quite usual for organizations to spent significantly on business process analysis, process management, monitoring, redesign and improvement in general by BPM management, Lean Six Sigma, process automation and overhead staff. Most large organizations have a group (or center of excellence) responsible for Business Process Management at strategy level or located within the strategy departments (IT, HR, Finance, Quality Control). They also must hire outside consultants to help their BPM strategy and planningby using some BPM products and services, from which the most important are: Graphics Modeling tools (Visio, PowerPoint), BPM Suite that can manage the runtime execution of business processes, Process Modeling tools (Casewise, MEGA, IBM Modeler, ProVision), Business Rules tools, BPMS Suite or execution environments (Workflow, EAI) (for details see [4]).

The Business Process Management Systems (BPMS) market is dominated by commercial vendors who supply their technologies to large accounts via lengthy, complex, and expensive business and Information System reengineering projects. The BPM Suite tools like Adobe Lifecycle Workflow, IBM WebSphere BPM (Filenet, ILOG), jBPM, Oracle BPEL/BEA/Fuego, SAP NetWeaver, Software AG (ARIS, WebMethod) — see [5], [8] — have imposed themselves to IT and business people as a better way to develop applications and to monitor existing processes, or even automate complete processes.

Open Source BPMS is the solution for businesses and organizations that want to avoid heavyweight proprietary software solutions and don't want to re-invent their IT systems. Most prominent open solutions are from Appian [2], Intalio [7], Bonita [3] — see also [9]. The former helps using visualization for decision making by business analysts.

As stated in [17] current process monitoring, and business activity systems do not provide a visual method for goal-oriented process monitoring. In addition,

existing quality methods and process improvement techniques are based on management and statistical tools and do not provide proper integration with information sources for process monitoring. Goal-orientation helps for better aligning business strategy with business process architectures. New BPMS-based environment includes many complex cross-organizational processes, involving different information systems during the process flow, and generating voluminous amounts of data.

The analysis and early verification of process specifications are probably the best way for simulation and business process automation as well as for coordinating enterprise process change efforts and for avoiding the business process rules (BPR) implementation failures [15].

BPM automation is the key for the success of redesigning enterprise wide processes because the majority of organizations have a cost-saving initiative focused on increasing productivity of specific processes or a set of new software technologies that makes it easier for IT to manage and measure the execution of process workflow and process software applications, rather than a systematic approach to analyzing, redesigning, improving and managing overall processes. Practically, because of human-interaction and social factors of dynamicity of processes, the support provided by automated applications is not consistent with the defined processes used by an organization [23].

Nathaniel Palmer's survey [24] on the characteristics which determine whether BPM initiatives succeed or fail was analyzing over 100 BPM deployment and business process initiatives. One of the most striking revelation of this survey and its subsequent analysis was the high correlation betwen BPM project success and the leverage of some type of formalized discipline or practice area. Overall major conclusion of this survey is that both the use of BPMS software and business process improvement initiatives in general are yielding favorable results for most users by providing a formalized way to have support from methodologies and automation.

Language and notations for business process computation are numerous, and they are supporting all phases of the lifecycle of business processes including modeling, verification, enactment, monitoring, mining and diagnosis. WS-BPEL (Web Services Business Process Execution Language) [10] , shortly BPEL, is an executable language for specifying actions within business processes with web services. Processes in BPEL export and import information by using web service interfaces exclusively. BPMN [12] creates a standardized bridge for the gap between the business process design and process implementation. BPMN provides a simple means of communicating process information to other business users, process implementers, customers, and suppliers. Another goal is that WS-BPEL can be visualized with a business-oriented notation. Other BPs supporting languages are UML Activity Diagram, UML EDOC Business Processes, IDEF, ebXML BPSS, Activity-Decision Flow (ADF) Diagram, RosettaNet, LOVeM, YAWL, URN and Event-Process Chains (EPCs) — see [12, 13, 6, 1, 26, 14, 11].

Fork-Join Automata. This structure is inspired by the fact that some BPM tools use fork–join diagrams in order to represent workflows. Most frequently

the fork and the join nodes in these diagrams are of type *and*, respectively *or*. The *and* fork nodes correspond to splits in the workflow which lead to parallel activities. The *and* join nodes correspond to aggregation of data and/or of product parts (also know as synchronization points). The *or* fork nodes correspond to decision points (similar to if-then-else) leading to alternative activities in the workflow. The *or* join nodes do not involve synchronization, but the first event which happens will trigger the continuation of the workflow on the join branch.

We define a fork–join automaton as being a directed acyclic graph. Each node is typed both as *fork* or *join*, as well as *and* or *or*. In contrast to the usual notion of graph, the arcs are represented as lists associated to each pair of nodes — thus it is possible to have several arcs between the same two nodes, and they are given in a certain order. (Conventionally we denote with $A_1, A_2, \ldots$ the arcs originating from a certain node A.)

Each *fork node* has a unique incoming arc and at least two outgoing arcs, except one fork node which has no incoming arcs and it is called the *start node*. Each *join node* has at least two incoming arcs and one unique outgoing arc, except one join node which has no outgoing arcs and it is called the *end node*. For brevity, we will frequently use "a fork" and "a join" instead of "a fork node" and "a join node". The number of incoming arcs of a join node, as well as the number of outgoing arcs of a fork node, is called its *arity*. The typing of the nodes (both the fork and the join ones) as *and* or *or* is motivated by the real–life interpretation of these nodes as presented above.

Additionally a fork–join automaton may be decorated with: conditions associated to the *or* nodes, activities associated to the arcs, data types and variables associated to the arcs, roles or departments associated to activites, etc. These allow a more precise characterization of the real–life workflow which is described by the respective fork–join automaton, however in this presentation we do not discuss these details, rather our focus is on the fork–join, *and–or* structure of the automaton.

Structural Correctness. Obviously not any fork–join structure corresponds to a practically feasible workflow. For instance, a structure consisting only of an *or* fork followed by an *and* join will never correspond to a real workflow. In complex workflow structures, which sometimes occur in practice, such inconsistencies may reveal errors in the design of the workflow. The purpose of this presentation is to summarize two equivalent characterizations of the structural correctness of fork–join automata.

A *trace* of the automaton is a fork–join automaton containing only *and* nodes, obtained from the original one by disambiguating (choosing one alternative of) each *or* fork. This contrasts the usual notion of a trace of a finite automaton (which is a list of states), because in our case we have to represent parallel activities created by *and*–forks. Traces correspond practically to possible developments of workflows. In order for a fork–join automaton to be structurally correct, all its traces must be valid fork–join automata, and all nodes must have the same arity as the original ones. The rigurous construction of traces requires however some care, thus we suggest the following procedure: First construct a set of all

candidate traces by marking for removal, at each *or*–fork, all arcs except one; then apply the reduction process described below for each candidate trace.

- Repeat as long as possible: mark for removal any node whose all incoming arcs are already marked, as well as the node itself and its outgoing arcs.
- When there are no new markings possible, check whether each remaining *or* node has only one input and one output unmarked. (If the check does not pass, then the corresponding trace is not correct, and this indicates a combination of alternatives which is problematic in the corresponding real–life workflow.)
- Remove all marked nodes and arcs, and transform each pair of arcs separated by and *or* node into a single arc.
- Check whether all *and* nodes have the original arity (otherwise the trace is again not correct).

Structural Execution. While the technique presented above for checking the structural correctness may be easily implemented in practice, it is quite inefficient because it will perform significant redundant work on similar alternatives. In order to make the check more efficient we suggest a verification method which proceeds by a kind of *structural execution* of the automaton on all arcs, by keeping track of the *and* and *or* splits.

The execution associates a certain *set–value* to each arc, starting from the *start node* with the empty set. A correct fork–join automaton will lead to the final value of empty set at the *end* node. The elements of each set value are pairs of a node name and a set of arcs.

The evolution rule in the case of a fork node A having the outgoing arcs $A_1, A_2, \ldots, A_n$ consists in propagating the current set value to each outgoing arc A_k and adding to it the pair $\langle A,\ \{k\}\rangle$.

In the case of a join node, first all the set values corresponding to the incoming arcs are joined (the union of all the sets), then some of the elements of the resulting set are merged according to the following rules:

- Only those elements are merged whose named nodes are of the same type (that is *and* or *or*) with the fork node.
- If $\langle A, S_1\rangle, \langle A, S_2\rangle, \ldots, \langle A, S_k\rangle$, are all the elements having the node name A, then $S_1, S_2, \ldots, S_k$ must be disjoint (otherwise the automaton is not correct), and these elements are merged into the pair consisting of A and the union of $S_1, S_2, \ldots, S_k$.
- The merger above takes place for all instances of names having several elements in the set value.
- After each merge operation, if the set of a pair equals the original set of arcs of the corresponding node, then this pair is removed.

This procedure avoids the combinatorial explosion of the first definition of correctness given in the previous paragraph, however it checks the structural correctness of the fork–join automaton in an effective way. Moreover, the set value collected on the arcs gives, for practical situations, useful information about the corresponding alternatives and parallel flow of the work.

Conclusion Our formalism based on fork–join automata appears to be significantly simpler than other approaches, like e. g. Petri Nets [25] or CSP [27]. It is true, however, that our formalism does not handle cycles, but we believe that – in contrast to algorithms and programs – in business processes the notion of a cycle with an unbunded number of iterations is rather unnatural. Future work could possibly investigate the treatment of cycles as abbreviations of loops with a predefined (small) number of iterations.

Nevertheless, a number of important issues remain to be studied, like verification of conformity in case of workflow transformations, detection of unfeasible branches (based on the conditions associated to *or* forks), type checking, etc. We believe that this kind of problems can be approached in an efficient and natural way using the formalism proposed in the present work.

References

1. YAWL: Yet Another Workflow Language. `http://www.yawlfoundation.org/` .
2. Appian Corp. BPM KIT. `http://www.appian.com/promo/kit.jsp`.
3. BonitaSoft BPM Solutions. `www.bonitasoft.com`.
4. BPM EA, Process Modeling and Simulation Tools Report. `http://www.bptrends.com/reports_landing.cfm`.
5. BPM Suite Reports. `http://www.bptrends.com/reports_landing.cfm`.
6. Business Process Management Center, a collaborative virtial research center. `www.bpmcenter.org`.
7. Intalio Inc. `www.intalio.com`.
8. Object Management Group/Business Process Management Initiative - Business Center Exellence. `www.bpmn.org`.
9. User Requirements Notation tools - jUCMNav 3.1. `http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome`.
10. Web Services Business Process Execution Language (WSBPEL) 2.0, OASIS Standard, April 2007. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.
11. Z. 150. User Requirements Notation (URN) - Language Requirements and Framework, ITU-T Standard, Feb 2003. `http://www.itu.int/rec/T-REC-Z.150-200302-I/en`.
12. Business Process Management Notation 2.0, BPMN Standard, January 2011. `http://www.omg.org/spec/BPMN/Current`.
13. Unified Modeling Language Specification V2.1.2: Superstructure, OMG, Nov. 2007. `http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF1`.
14. Z. 151. User Requirements Notation - Language definition, ITU-T Standard, Nov 2008. `http://www.itu.int/rec/T-REC-Z.151-200811-I/en`.
15. M. Al-Mashari and M. Zairi. BPR Implementation Process: an Analysis of Key Success and Failure Factors. *Journal of Business Process Management*, 5(1):87–112, 1999.
16. B. Buchberger et al. Theorema: Towards Computer-Aided Mathematical Theory Exploration. *Journal of Applied Logic*, 4(4):470–504, 2006.
17. D. Amyot et al. Towards an Integrated User Requirements Notation Framework and Tool for Business Process Management. In *3rd Int. MCeTech Conference on eTechnologies*, pages 3–15, Les Diablerets, Switzerland, 23–26 January, 2011.

18. K.-D. Schewe et al., editor. *Correct Software in Web Applications (ESF Strategic Workshop)*, Hagenberg, Austria, 25–28 Sep., 2011.
19. A. Medve. Advanced Steps with Standardized Languages in the Re-engineering Process. *Jurnal of Computer Standards & Interfaces*, 30(5):315–322, 2006.
20. A. Medve. Enterprise Modelling with the Joint Use of User Requirements Notation (URN) and UML (Book chapter). In P. Rittgen, editor, *Enterprise Modeling and Computation with UML*, pages 46–69. IDEA Group, 2007.
21. A. Medve. Workflow–Based Re-engineering Process of Socio–technical Systems. In Novadat Kiado Bt., editor, *International Symposium on Business Information Systems (ISBIS'2007)*, pages 36–38, 2007.
22. A. Medve. Integrated Engineering of Information Systems' Security from Generic Models Based on Standards (in Hungarian). In Novadat Kiado Bt., editor, *International Symposium on Business Information Systems - OGIK Hungarian Track (ISBIS'2011)*, 2011. to appear.
23. A. Medve. Standards-based Framework for Functionally Integrated Engineering of Information Systems. In T. Szmuc Ed. M. Biro, editor, *5th IFIP TC2 Central and Eastern European Conference on Software Engineering Techniques (CEE-SET'2011)*, pages 52–58, Debrecen, Hungary, 24–27 Aug., 2011.
24. N. Palmer. A Survey of Business Process Initiatives. Technical Report 1, 2007. `http://www.bptrends.com/reports_landing.cfm`.
25. W. M. P. van der Aalst. Pi calculus versus petri nets: Let us eat humble pie rather than further inflaate the pi hype. *BPTrends*, 3(5):1–11, 2005.
26. A. H. M. ter Hofstede W. M. P. van der Aalst. YAWL: Yet Another Workflow Language. Technical report, 2002.
27. P. Y. H. Wong and J. Gibbons. A process–algebraic approach to workflow specification and refinement. In *Lecture Notes in Computer Science 4829:51*, 2007.