The Structured Phase of Concurrency

(Extended Abstract)

Artem Polyvyanyy[1] and Christoph Bussler[2]

[1] Queensland University of Technology, Brisbane, Australia
artem.polyvyanyy@qut.edu.au
[2] Analytica, Inc., Palo Alto, CA, USA
chris@real-programmer.com

Abstract. This extended abstract summarizes the state-of-the-art solution to the *structuring problem* for models that describe existing real world or envisioned processes. Special attention is devoted to models that allow for the *true concurrency* semantics. Given a model of a process, the structuring problem deals with answering the question of whether there exists another model that describes the process and is solely composed of *structured* patterns, such as sequence, selection, option for simultaneous execution, and iteration. Methods and techniques for structuring developed by academia as well as products and standards proposed by industry are discussed. Expectations and recommendations on the future advancements of the structuring problem are suggested.

Keywords: Structured, unstructured, process, modeling, process model, structuring, concurrency, true concurrency

1 Introduction to Concurrency

Processes are properties of dynamic systems that are usually defined as series of steps taken to achieve a goal, e.g., chemical or thermodynamic processes. Many scientific disciplines study processes by means of modeling. A model of processes, or a *process model*, is a particular representation of processes of the same nature. The study of processes is predominant to the computer science discipline. Computer programs and workflows are examples of process models.

Concurrency is a property of a process which indicates that steps of the process can be performed simultaneously by several autonomous "workers" that may coordinate their work by means of communication, e.g., instructions of a computer program carried out by several processing units within a single or across multiple computers. Usually, concurrent processes are more efficient than processes in which all work is performed by a single worker, where efficiency is the measure of the amount of work accomplished within a given time frame.

There exist several formalisms for describing concurrent processes, cf., [1] for examples and a classification of formalisms. Concurrency can be represented implicitly in a process model as a nondeterministic choice between all possible sequentializations of concurrent steps. Afterwards, an option for a simultaneous execution of steps can be deduced based on the interleaving of these steps. Alternatively, concurrency can be modeled based on partially-ordered sets of steps — an approach that is often referred to as the *true concurrency* semantics of processes. The true concurrency semantics specifies that unordered steps of a process can be enabled and performed simultaneously. Observe that process models which account for the true concurrency semantics are more informative. They explicitly represent steps that can be performed by different workers, as well as describe all possible sequentializations of steps.

Concurrent processes are *complex*. The complexity is primarily due to the fact that the number of all possible sequentializations of process steps is exponential to the number of concurrent steps in the process. Consequently, concurrent processes are complex to design and analyze because one needs to account for all possible interleavings of steps that can lead to potential flaws.

The next section defines the structured property of process models. Afterwards, Section 3 reviews progress achieved in academia and industry that relates to the structuring problem of process models. Finally, expectations and recommendations on the future advancements with respect to structuring are proposed.

2 The Structured Phase

A process can be formalized in many different ways following different modeling styles, i.e., it can take different forms. This reminds of various *states* that a physical matter can take on. For example, H_2O is a substance that its gas, liquid, and solid states are widely known, viz. steam, water, and ice, respectively. Often, there exist several phases of the same state of matter, where a *phase* is a region of space (a thermodynamic system), throughout which all physical properties of a material are essentially uniform [2]. For instance, diamond and graphite are both solid carbon with different physical characteristics, e.g., color, crystal shape, etc. Similarly, there can exist several different models of the same process that are captured in the same formal language.

Besides many existing modeling styles, processes are usually formalized by following on the principles of the *imperative* paradigm. Process models that follow the imperative style are described in terms of sequences of statements/commands that change a process state. Intuitively, an imperative process model can be perceived as a directed graph with nodes representing commands and pairs of subsequent commands representing edges.

Process graphs, i.e., graphs induced by imperative process models, can take different forms. However, it is often preferred that process graphs obey some structural rules, like the one that a process graph should be *structured*.

> *A process graph is* structured, *if for every node with multiple outgoing edges (a split) there exists a corresponding node with multiple incoming edges (a join), and vice versa, such that the subgraph between the split and the join forms a single-entry-single-exit fragment; otherwise the graph is* unstructured.

Consequently, a process model is *structured* if and only if a process graph induced by the model is structured. The reader can refer to [3] for a detailed motivation of structured process modeling. In summary, most of the benefits of using structured process models boil down to the observation that it is often straightforward to *modularize* handling of structured process graphs at various stages of their life cycles. For example, it is evident that structured computer programs are easier to understand and maintain as opposed to unstructured ones, which contain arbitrary "jump" constructs leading to programs that induce "spaghetti" process graphs. Nevertheless, unstructured modeling is often praised for the *freedom* it offers when designing a process model. Indeed, there are no restrictions on which form a process graph can take. Considering modeling to be a highly creative activity, the freedom of expressing concepts and ideas in models is of significant importance.

When proposing a process modeling methodology, one faces a dilemma of whether to allow or forbid unstructuredness. Allowing for unstructuredness supports creativity of process designers, while forbidding it comes with a continuous and seamless ability to enjoy benefits of structured models. The problem of *structuring* a process model deals with the automated construction of a structured model that represents the process described in the given model, i.e., finding the *structured phase* of the process.

3 State of the Art

In this section, we review the progress achieved by the research community on a solution to the structuring problem for process models that account for the true concurrency semantics (Section 3.1), as well as analyze the support of structured and unstructured process modeling by the software industry sector (Section 3.2).

3.1 Research: Methods and Techniques

In this section, we discuss scientific results on methods for structuring process models. Special attention is paid to techniques that address structuring of models that account for the true concurrency semantics.

Results on structuring sequential process models, i.e., models that describe non-concurrent processes, are mainly due to results on elimination of `goto` constructs in computer programs. The *structured program theorem* [4] provides the theoretical foundation of structured programming. It states that every program can be expressed with three patterns: sequence, selection, and iteration. Moreover, it is well-known that a sequential process model, e.g., a program, can be formalized as a flowchart and structured, e.g., by using the techniques proposed in [5,6].

The concurrent world is a bit more complex. In this world, the results on structuring of sequential process models cease to hold. In [7], Kiepuszewski et al. showed that not all acyclic concurrent process models, i.e., models of concurrent processes that induce acyclic process graphs, can be structured. The above fact has been proven by means of a counter-example — a *Z-structure* pattern (the name is due to the constellation of causal relations between the concurrent steps). The authors demonstrated that under the true concurrency semantics there exists no structured model that captures the process described by the Z-structure pattern.

In [8,9], the authors proposed a solution to the structuring problem of acyclic concurrent process models. The technique is capable of recognizing inherently unstructured process models, i.e., models that have no equivalent structured representations. The theoretical basis of the approach builds on interplay of two parsing techniques: a technique for discovering the structure of process graphs [10] and a technique for decomposing causal, conflict, and concurrency relations [11,12] between process steps by means of the modular decomposition [13].

The work in [14] addresses structuring of acyclic concurrent process models that have no equivalent structured versions but which, nevertheless, can be partially structured into their *maximally-structured* representations. Intuitively, a process model M of a concurrent process P is maximally-structured if and only if there exists no process model that describes P and is composed of more structured modeling patterns (single-entry-single-exit fragments) than M. A maximally-structured process model is a mixture of structured and unstructured phases of the process, similar to an ice-water mixture which has ice cubes as one phase and water as a second phase of the same substance.

The cyclic case of the structuring problem is addressed in [3]. This work proposes a structuring technique and argues about the rationality of its individual stages (by means of proof-sketching). The technique can be seen as a two-stage approach. First, an input process model is transformed into an equivalent one in which all the concurrency is kept encapsulated in single-entry-single-exit fragments. Second, the obtained model is structured by iteratively applying the approach from [9] to its parts.

3.2 Industry: Products and Standards

The software industry has a long tradition in implementing workflow or process management systems that are based on explicit control flow. Early publications show quite a variety of systems that were available [15]. Based on the variety and sheer number of systems, standards were developed in context of process modeling and process execution: Web Services Business Process Execution Language (BPEL) [16] and Business Process Model and Notation (BPMN) [17].

By its nature, BPEL is a structured language. It does not propose native support for unstructured control flow patterns. Whenever an inherently unstructured process needs to be specified in BPEL, one has to rely on the means in BPEL for implicit control flow definition, e.g., the *event handler* construct [18] or a combination of *flow* and *link* constructs [19]. For example, an event handler can be used to orchestrate a subset of concurrent process steps via *event-action* rules, which essentially are preconditions for execution of concurrent steps. In contrast, BPMN allows unstructured process modeling. This means that BPMN as the process definition language supports unstructured process specifications. Systems that implement their execution model and semantics based on BPMN can support unstructured process execution.

In the following, we give some examples of process management systems proposed by the software industry (the list is by no means exhaustive). Examples of systems that support BPEL are: Oracle BPEL Process Manager [20] and IBM Business Process Manager Advanced [21]. Those supporting BPMN are: TIBCO ActiveMatrix BPM [22], IBM Business Process Manager [21], Appian

BPM Suite [23] and Pegasystems [24]. The system from IBM appears to be a hybrid system that supports both, BPMN and BPEL. Yet, there are process management systems that follow neither BPEL nor BPMN, and those might very well support unstructured processes. For instance, a system not based on standards is Microsoft Workflow Manager [25,26]; note that the documentation of this product suggests that the control flow model is supporting only structured modeling. Finally, the web site `www.workflowpatterns.com` [27] discusses some details of unstructured support and compares a few industry products.

Constructs for concurrency are natively supported in several programming languages, e.g., `Java` and `C#`. The Windows Workflow Foundation (WWF) has been introduced as a part of the .NET Framework and is a means of implementing long-running processes. It is a common requirement for programming languages that fragments of control flow that include concurrency must be structured. For instance, the WWF does not support arbitrary cycles with parallel branching [28].

4 Expectations and Recommendations

Concurrent process modeling/programming has been practiced for years. However, recently, one can observe a remarkable growth of interest in concurrent processes; mainly, for the purpose of *automation*. One of the reasons for this is physical constraints that forbid frequency scaling in modern computer processors. Nowadays, parallel computing is the dominant paradigm in computer architectures. It gets harder to rely on hardware when getting performance improvements out of good old sequential processes [29]. Rather, sequential processes should be redesigned to allow for simultaneous execution of their parts. We expect that this trend will remain for years to come leading to new requirements on the way processes are modeled.

A formal language that includes constructs for representing process steps and arbitrary jumps between steps can be employed to describe any process (with respect to control flow). Nevertheless, for considerations like clarity, quality, maintainability, modularity, etc., of process models, formal process languages often include high level constructs. Advantages of the structured process modeling style over the unstructured one (and vice versa) have been a topic of active debates for decades. Structured process modeling confines itself to constructs that map to single-entry-single-exit patterns. If the structured process modeling methodology is enforced, one has to accept that certain concurrent processes cannot be modeled.

Results reported in [3,7,8,9,14] provide a basis for structured modeling of concurrent processes, most of which are implemented in a tool called `bpstruct`; the tool is publicly available at `http://code.google.com/p/bpstruct/`. Evaluations conducted in [9,14] report on small average times required to structure real world process models taken from industry. However, in theory, the structuring techniques are inherently complex. For instance, for certain inputs, these techniques subsume a problem for which finding a solution is NP-complete. Future studies must show if the theoretic complexity of structuring algorithms can be improved. Other directions for future work on structuring are outlined in [3].

In terms of products and standards, our expectation is that over time more and more products will support structured as well as unstructured modeling in order to cater for more and wider use cases. Our recommendation is that standards and companies explicitly discuss their support for unstructured processes and extend their modeling tools to incorporate automated transformation from unstructured to structured processes where possible.

Acknowledgments. The first author is supported by the ARC Linkage Project LP110100252 "Facilitating Business Process Standardisation and Reuse".

References

1. Sassone, V., Nielsen, M., Winskel, G.: Models for concurrency: Towards a classification. Theoretical Computer Science (TCS) **170**(1–2) (1996) 297–348
2. Modell, M., Reid, R.: Thermodynamics and Its Applications. International Series in the Physical and Chemical Engineering Sciences. Prentice-Hall (1974)
3. Polyvyanyy, A.: Structuring Process Models. PhD thesis, University of Potsdam (2012)
4. Böhm, C., Jacopini, G.: Flow diagrams, Turing machines and languages with only two formation rules. Communications of the ACM (CACM) **9**(5) (1966) 366–371
5. Williams, M.H., Ossher, H.L.: Conversion of unstructured flow diagrams to structured form. The Computer Journal (CJ) **21**(2) (1978) 161–167
6. Oulsnam, G.: Unravelling unstructured programs. The Computer Journal (CJ) **25**(3) (1982) 379–387
7. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.: On structured workflow modelling. In: Conference on Advanced Information Systems Engineering (CAiSE). Volume 1789 of Lecture Notes in Computer Science., Springer (2000) 431–445
8. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. In: Business Process Management (BPM). Volume 6336 of Lecture Notes in Computer Science., Springer (2010) 276–293
9. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. Information Systems (IS) **37**(6) (2012) 518–538
10. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: Web Services and Formal Methods (WS-FM). Volume 6551 of Lecture Notes in Computer Science., Springer (2010) 25–41
11. McMillan, K.L.: A technique of state space search based on unfolding. Formal Methods in System Design (FMSD) **6**(1) (1995) 45–65
12. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. Formal Methods in System Design (FMSD) **20**(3) (2002) 285–310
13. McConnell, R.M., de Montgolfier, F.: Linear-time modular decomposition of directed graphs. Discrete Applied Mathematics (DAM) **145**(2) (2005) 198–209
14. Polyvyanyy, A., García-Bañuelos, L., Fahland, D., Weske, M.: Maximal structuring of acyclic process models. The Computer Journal (CJ). (first published online September 19, 2012) doi:10.1093/comjnl/bxs126.
15. Jablonski, S., Bussler, C.: Workflow Management — Modeling Concepts, Architecture and Implementation. International Thomson (1996)
16. OASIS: Web Services Business Process Execution Language Version 2.0. OASIS Standard. (April 2007) http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.
17. Object Management Group (OMG): Business Process Model and Notation (BPMN) Version 2.0. OMG Standard. (January 2011) http://www.omg.org/spec/BPMN/2.0.

18. Ouyang, C., Dumas, M., ter Hofstede, A.H.M., van der Aalst, W.M.P., Mendling, J.: From business process models to process-oriented software systems. ACM Transactions on Software Engineering and Methodology (TOSEM) **19**(1) (2009)
19. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In: Modellierung. Volume 127 of Lecture Notes in Informatics., GI (2008) 57–72
20. Oracle BPEL Process Manager: http://www.oracle.com/technetwork/middleware/bpel/overview/index.html.
21. IBM Business Process Manager and IBM Business Process Manager Advanced: ftp://ftp.software.ibm.com/software/integration/business-process-manager/library/pdf801/ibpm_overview_pdf_en.pdf.
22. TIBCO ActiveMatrix BPM: http://www.tibco.com/multimedia/ds-amx-bpm_tcm8-11546.pdf.
23. Appian BPM Suite: http://www.appian.com/bpm-software/bpm-for-designers/process-management.jsp.
24. Craggs, S.: Comparing BPM from Pegasystems, IBM and TIBCO. (August 2011) http://soapower.com/IBMBPM/Whitepapers/IBM-BPM-Analyst-Report-on-IBM-vs-Pega.pdf.
25. Microsoft Workflow Manager: http://msdn.microsoft.com/en-us/library/windowsazure/jj193528%28v=azure.10%29.aspx.
26. Control Flow Activity Designers: http://msdn.microsoft.com/en-us/library/ee829560.aspx.
27. www.workflowpatterns.com: Pattern 10 (Arbitrary Cycles). http://www.workflowpatterns.com/patterns/control/structural/wcp10.php.
28. Zapletal, M., van der Aalst, W.M.P., Russell, N., Liegl, P., Werthner, H.: An analysis of Windows workflow's control-flow expressiveness. In: European Conference on Web Services (ECOWS), IEEE Computer Society (2009) 200–209
29. Sutter, H.: The free lunch is over: A fundamental turn toward concurrency in software. Dr. Dobb's Journal **30**(3) (2005) 202–210