

Why Users of the Internet of Things need Error-Avoiding Modeling Languages and Queries

Thomas M. Prinz¹

¹Course Evaluation Service, Friedrich Schiller University Jena, Am Steiger 3, Haus 1, 07743 Jena, Germany

Abstract

The Internet of Things (IoT) is used by non-technical users. In the future, they will be able to arrange their IoT devices and, therefore, create processes. Since the devices can send their information simultaneously, these processes must enable parallelism. This paper argues why modeling languages for non-technical users should only create structurally correct processes that avoid deadlocks and abundances during runtime. Furthermore, since many users will create their own processes, it should be possible to reuse and share process models. This makes a query language for process models necessary. Using loop decomposition, an approach to make cyclic processes acyclic, this paper explains why modeling languages can be simplified for users and why only acyclic processes should be stored in process collections. For all of these topics, IoT can learn from business process management and compiler theory.

Keywords

Internet of Things, processes, modeling language, structural correctness, process query

1. Introduction

The *Internet of Things* (IoT) enables the combination of different physical objects (the “things”) in a network (the “internet”) so that they can communicate with each other and, therefore, realize applications at a higher level than individual objects can [1]. As IoT devices enter houses and living rooms of non-technical users, the ability for non-technical users to combine their IoT devices for a larger goal becomes essential. IoT devices may receive some input, process information, and produce some outputs – they follow simple input-process-output principles. An arrangement of IoT devices to realize higher-level applications must, therefore, take into account the data between the devices and their temporal dependencies. In other words: IoT devices can be arranged as activities in processes (flowcharts). The more restrictive the modeling language is, the more restrictive the resulting applications will be. For example, if the modeling language only allows sequential activities (pipes of IoT devices where one device passes its output to the next in the pipe), decisions are not possible – a user could not create a process that only performs an action when the temperature exceeds a predefined threshold. On the contrary, if a user has a lot of freedom in modeling processes, there is a high probability that the resulting process will contain semantic errors. Especially since IoT devices like sensors can produce

EMPATHY: Empowering People in Dealing with Internet of Things Ecosystems. Workshop co-located with AVI 2022, June 6, 2022, Rome, Italy

✉ thomas.prinz@uni-jena.de (T. M. Prinz)

🌐 <https://www.ule.uni-jena.de/studiengangsevaluation/software> (T. M. Prinz)

🆔 0000-0001-9602-3482 (T. M. Prinz)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

output in parallel, the parallelism can lead easily to errors. In business process management (BPM), even experienced process modelers seem to make mistakes in their processes, as some studies show [2].

In this paper, we propose to learn from methods and mistakes from compiler theory and BPM when defining modeling languages for IoT processes used by non-technical users. Although IoT processes should not be as data-driven as source code and not as technical complex as business processes, we are convinced that a modeling language for IoT processes can benefit from their background. For this reason, we introduce a quality property — *structural correctness* — for IoT processes and argue for thinking about an IoT query language to find processes in a collection of IoT process models. Furthermore, with the help of *loop decomposition*, an approach to make cyclic process models acyclic, we argue why modeling languages should be simplified and why it is good to store only acyclic process models in process model collections.

The rest of this paper is structured as follows: Section 2 introduces the concept of workflow graphs for abstract representation of (IoT) processes. Subsequently, Section 3 argues for structural correctness and Section 4 for process query. Section 5 describes the advantages of the loop decomposition approach. Finally, this paper concludes in Section 6 with a short discussion.

2. Workflow Graphs

IoT processes can be modeled abstractly as *workflow graphs* [3], a variant of control-flow graphs that allow explicit parallelism. A workflow graph contains several *activities* and *gateways*. Activities are IoT devices or some other programs. They have up to one incoming and up to one outgoing edge. Activities without incoming edges are *start activities* (like sensors). Activities without outgoing edges are *end activities* (like screens). Gateways handle the control-flow, i. e., they split or join control. Some gateways make decisions (*XOR*), create parallelism (*AND*), or something in between (*OR*). *Splitting* gateways have exactly one incoming but multiple outgoing edges (*XOR*-, *AND*-, and *OR*-splits) and *joining* gateways have multiple incoming but exactly one outgoing edge (*XOR*-, *AND*-, and *OR*-joins). All nodes of the workflow graph are on a path from a start to an end activity.

The execution of a workflow graph can be modeled as a token game. A token game looks at different *states* of a workflow graph and how they can change. A *state* stores the number of tokens for each edge. In an *initial state*, the execution starts with tokens on outgoing edges of some start activities. Each ordinary activity takes a token from its incoming edge and puts it on its outgoing edge. This changes the state to another state. Each time a token moves, the state changes. When a token reaches a splitting gateway, how the state changes depends on the type of split. If it is an *XOR*-split, the token is placed on one of the outgoing edges depending on some conditions. If it is an *AND*-split, a token is placed on each outgoing edge. In case of an *OR*-split, a non-empty set of outgoing edges receives a token. For joining gateways, it is similar: An *XOR*-join takes a token from an incoming edge with a token and places it on its outgoing edge. An *AND*-join requires tokens on each incoming edge, otherwise, the tokens are retained. If all incoming edges have tokens, a token is taken from each incoming edge and a single token is placed on its outgoing edge. The semantics of an *OR*-join is more complex and non-trivial in workflow graphs with *cycles* [4, 5]. For acyclic workflow graphs, an *OR*-join takes a token

from each incoming edge with tokens if and only if there is no token that an incoming edge can receive in a subsequent state. Then, a single token is placed on the outgoing edge of the OR-join. For cyclic workflow graphs, the interested reader can find a comparison and a “most liberal” semantics in earlier work [5]. The execution and state changes of the workflow graph eventually stop when the end activities have only tokens.

3. Structural Correctness

We argue that IoT processes as workflow graphs should be *structurally correct*. Structural correctness is known in BPM as *soundness* [6]. It describes the absence of *deadlocks* and *abundances* [2], while data information is ignored [7]. In other words, structural correctness focuses only on the structure of the workflow graph and, therefore, decisions are assumed to be random.

A *deadlock* occurs during execution when there is an AND- or OR-join that attempts to synchronize parallelism, but there is no parallelism to join, i. e., it has a token on at least one incoming edge, but cannot execute in any subsequent state. An *abundance* occurs when there is an XOR-join that gets parallel control so that a subsequent activity can be executed twice (or more) in series, i. e., there is a state with an edge that carries at least two tokens [5].

Structural correctness is controversial in BPM. Some researchers argue that it leads to false negatives and false positives because data is ignored [8]. It is true that data can influence the flow of control and, therefore, mask errors or create the illusion of error [9]. For this reason, however, we prefer the term “structural correctness” instead of “soundness”. Structural correctness considers only the control-flow *without* data. It guarantees the fundamental correctness of parallel behavior, even if data information or code change. It provides basic assumptions for ongoing analyses and, therefore, simplifies them. In our view, the use of structural correctness is comparable to the use of reducible and irreducible control-flow graphs in compiler theory. Both — reducible and irreducible control-flow graphs — are executable, but reducible ones are preferred and used because they give faster algorithms or make them applicable at all [10, 11]. Since programs with irreducible control-flow graphs have *goto*-like instructions, they improve also readability and code quality [12]. Similar properties apply to structural correctness: for example, OR-join semantics is complete [5] and behaviors extraction is more efficient [13] in structural correct processes. From the user’s perspective, enforcing structural correctness should efficiently reduce errors and thus frustration during creation. For this reason, **we propose to use structural correctness as an indispensable quality criterion for IoT processes.**

4. IoT Process Querying

IoT processes created by one user may be of interest to other users — at least as a starting point for changes. If such processes are made available via a web application or repository, users should be able to search for matching processes. Since there can be many such processes in a repository, a trivial search of IoT devices does not seem to be an expedient solution.

BPM has introduced process queries through *behavioral relations*, i. e., the user can search for processes not only by describing the IoT devices they contain, but also by describing some

of their relationships during execution. Typical behavioral relations are, for example: *in total causal*, *in total parallel*, and *in conflict*. An IoT device is *in total causal* of a second IoT device in an IoT process if, in each execution of the process in which both devices are executed, the first is executed before the second. Two IoT devices are *in total parallel* if, in each execution in which both devices are executed, both devices are executed in parallel. Finally, an IoT device is *in conflict* with a second device if there is an execution that contains the first device but not the second. BPM research recognizes other behavioral relations between two devices that are grouped into different sets. Examples of such sets are the *behavioral profile* [14], the *causal behavioral profile* [15], and the *4C spectrum* [16]. **We propose to define a similar set of behavioral relations to enable queries for IoT processes in process repositories to empower non-technical users.**

5. Loop Decomposition

Checking structural correctness or calculating behavioral relations is difficult and time-consuming in some cases. If the process contains OR gateways, there are not even complete algorithms for both. For *acyclic* IoT processes as workflow graphs, the situation is different: There are efficient algorithms for checking structural correctness that provide detailed diagnostic information (e. g., Favre and Völzer [17] and Prinz and Amme [18]), and the computation of behavioral relations is efficiently [13]. Even processes that contain OR gateways can be handled. Of course, not every IoT process is acyclic. However, in our recent work [19], we have developed a *loop decomposition* algorithm that detects loops in structurally correct, cyclic workflow graphs and makes them acyclic: Loops are replaced by special *loop activities* (subroutines) and the loops themselves are truncated at allowed edges so that the loops become sequential. In this way, a cyclic workflow graph is transformed into a set of acyclic workflow graphs while retaining the semantics.

The loop decomposition approach is based on a property of loops in structurally correct workflow graphs: When a loop exit is reached (a gateway with an outgoing edge leaving the loop), no token can be inside the loop other than the one on the incoming edge of that loop exit. In such a state, there is, therefore, no parallelism in the loop. The loop can be cut at the outgoing edges of a loop exit and transferred to its own workflow graph. This step of cutting loops is repeated until every loop in the original workflow graph has been replaced and all workflow graphs representing the loops are also acyclic. The advantages of this approach are: (1) The resulting acyclic workflow graphs should be easier to understand as complexity is reduced, (2) loops are allowed but moved to subroutines, and (3) in structurally correct workflow graphs, it is unnecessary whether a joining gateway is an AND-, XOR-, or OR-join.

Loop decomposition has implications for an IoT process modeling language if that language forces structurally correct IoT processes to be created. Decomposition into acyclic graphs is even possible for structurally *incorrect* processes as it leads to the same decomposition — but the behavior of loops could not be maintained. In the resulting acyclic graphs, the modeling language is able to repair most structural errors by replacing the joining gateways with the correct ones. These replacements can also be applied in the original, undecomposed process. Eventually, however, this replacement of joining gateways is not necessary if the modeling

language does not distinguish between different joining gateways at all — in this case, all joining gateways have the semantics of OR-joins in acyclic graphs. This would probably reduce the number of errors in modeling. Therefore, **we propose that a modeling language for IoT process does not distinguish between different joining gateways in order to reduce complexity for non-technical users.**

In general, acyclic IoT processes should be easier to understand than cyclic ones, as mentioned earlier. Therefore, non-technical users should understand them better too. In addition, smaller processes should be more reusable than larger processes because they are not as specific. Consequently, acyclic workflow graphs resulting from loop decomposition seem to be more useful for storage in IoT process repositories than more complex cyclic processes. Another advantage is that the computation of behavioral relations for IoT process queries becomes more efficient in these repositories [13]. Therefore, **we propose to store the acyclic processes resulting from a loop decomposition in a process repository instead of the original user-created processes.**

6. Conclusion and Short Discussion

In this paper, we have explained why it is necessary to allow non-technical users to create their own processes of IoT devices. These processes should at least be structurally correct to avoid deadlocks and abundances during runtime. Since non-technical users may want to use existing processes to start their own ones, a query of processes in repositories is necessary. The approach of loop decomposition transforms cyclic processes into acyclic processes. Acyclic processes eliminate the need for the kind of joining gateways that simplifies the modeling of structurally correct processes. Loop decomposition, therefore, allows for non-differentiation even in cyclic processes. Furthermore, decomposition should make the finding and understanding of processes more efficient.

Direct process creation can be too complex for non-technical users to understand. For this reason, many IoT platforms use a simpler approach in form of *Trigger-Action Platforms* (TAP). TAPs allow users to define events and execute actions when these events are triggered. Examples of such platforms are IFTTT¹, SmartThings², and Power Automate³. On such platforms, it is possible to trigger new events as actions of other events. As a result, chains of events and actions can be created. These chains depend on conditional events and can trigger events in parallel, i. e., they are processes. Therefore, users *indirectly* create processes on TAPs. Although it is easier to create event-action rules, the underlying platforms have to maintain, verify, and execute the underlying processes. The suggestions made in this paper are thus also applicable to TAPs, and (indirect) modeling errors should be displayed to users or avoided with the right quality criteria such as structural correctness.

Future work should consider the proposals for an IoT process modeling language of this paper and evaluate them in a practical context. Furthermore, it is necessary to discuss the implications of the proposals for concrete modeling languages, for users, and for underlying

¹<https://ifttt.com/>, last visited on May 16th, 2022

²<https://www.smarthings.com/>, last visited on May 16th, 2022

³<https://powerautomate.microsoft.com/>, last visited on May 16th, 2022

IoT platforms. The brief discussion on indirect and direct process modeling shows that quality criteria are important even when a different representation of process creation is used as a solution for non-technical users. The community benefits from the proposals in this paper by avoiding the same mistakes as the BPM community, because the proposals are based on lessons learned in BPM and compiler theory and simplify process handling – even if they are only indirectly modeled by the users. Users benefit because the combination of IoT processes would be powerful, yet simplified.

References

- [1] F. Mattern, C. Floerkemeier, From the internet of computers to the internet of things, in: K. Sachs, I. Petrov, P. E. Guerrero (Eds.), *From Active Data Management to Event-Based Systems and More - Papers in Honor of Alejandro Buchmann on the Occasion of His 60th Birthday*, volume 6462 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 242–259. doi:10.1007/978-3-642-17226-7_15.
- [2] D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, K. Wolf, Analysis on demand: Instantaneous soundness checking of industrial business process models, *Data Knowl. Eng.* 70 (2011) 448–466. doi:10.1016/j.datak.2011.01.004.
- [3] M. Sadiq, W. Orłowska, Analyzing process models using graph reduction techniques, *Information Systems* 25 (2000) 117–134. doi:10.1016/S0306-4379(00)00012-0.
- [4] H. Völzer, A New Semantics for the Inclusive Converging Gateway in Safe Processes, in: R. Hull, J. Mendling, S. Tai (Eds.), *Business Process Management*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 294–309. doi:10.1007/978-3-642-15618-2_21.
- [5] T. Prinz, W. Amme, A Complete and the Most Liberal Semantics for Converging OR Gateways in Sound Processes, *Complex Systems Informatics and Modeling Quarterly* (2015) 32–49. doi:10.7250/csimq.2015-4.03.
- [6] W. M. P. van der Aalst, Verification of workflow nets, in: P. Azéma, G. Balbo (Eds.), *Application and Theory of Petri Nets 1997*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 407–426.
- [7] T. S. Heinze, W. Amme, S. Moser, Static analysis and process model transformation for an advanced business process to petri net mapping, *Softw. Pract. Exp.* 48 (2018) 161–195. doi:10.1002/spe.2523.
- [8] N. Sidorova, C. Stahl, N. Trcka, Workflow soundness revisited: Checking correctness in the presence of data while staying conceptual, in: B. Pernici (Ed.), *Advanced Information Systems Engineering, 22nd International Conference, CAiSE 2010, Hammamet, Tunisia, June 7-9, 2010. Proceedings*, volume 6051 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 530–544. doi:10.1007/978-3-642-13094-6_40.
- [9] T. M. Prinz, W. Amme, Why We Need Static Analyses of Service Compositions – Fault vs. Error Analysis of Soundness, *International Journal on Advances in Intelligent Systems* 10 (2017) 458–473. ISSN 1942-2679.
- [10] R. E. Tarjan, Testing flow graph reducibility, *J. Comput. Syst. Sci.* 9 (1974) 355–365. doi:10.1016/S0022-0000(74)80049-8.

- [11] L. Carter, J. Ferrante, C. D. Thomborson, Folklore confirmed: reducible flow graphs are exponentially larger, in: A. Aiken, G. Morrisett (Eds.), Conference Record of POPL 2003: The 30th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, New Orleans, Louisiana, USA, January 15-17, 2003, ACM, 2003, pp. 106–114. doi:10.1145/604131.604141.
- [12] E. W. Dijkstra, Letters to the editor: Go to statement considered harmful, *Commun. ACM* 11 (1968) 147–148. doi:10.1145/362929.362947.
- [13] N. L. Ha, T. M. Prinz, Partitioning behavioral retrieval: An efficient computational approach with transitive rules, *IEEE Access* 9 (2021) 112043–112056. doi:10.1109/ACCESS.2021.3102634.
- [14] M. Weidlich, J. Mendling, M. Weske, Efficient consistency measurement based on behavioral profiles of process models, *IEEE Transactions on Software Engineering* 37 (2011) 410–429. doi:10.1109/TSE.2010.96.
- [15] M. Weidlich, A. Polyvyanyy, J. Mendling, M. Weske, Causal behavioural profiles - Efficient computation, applications, and evaluation, in: *Fundamenta Informaticae*, volume 113, 2011, pp. 399–435. doi:10.3233/FI-2011-614.
- [16] A. Polyvyanyy, M. Weidlich, R. Conforti, M. La Rosa, A. H. M. ter Hofstede, The 4C Spectrum of Fundamental Behavioral Relations for Concurrent Systems, in: *Application and Theory of Petri Nets and Concurrency*, volume 8489 LNCS, 2014, pp. 210–232. doi:10.1007/978-3-319-07734-5_12.
- [17] C. Favre, H. Völzer, Symbolic execution of acyclic workflow graphs, in: *Business Process Management - 8th International Conference, BPM 2010, Hoboken, NJ, USA, September 13-16, 2010. Proceedings, 2010*, pp. 260–275. doi:10.1007/978-3-642-15618-2_19.
- [18] T. M. Prinz, W. Amme, Control-flow-based methods to support the development of sound workflows, *Complex Syst. Informatics Model. Q.* 27 (2021) 1–44. doi:10.7250/csimq.2021-27.01.
- [19] T. M. Prinz, Y. Choi, N. L. Ha, Understanding and decomposing control-flow loops in business process models, in: *Business Process Management - 20th International Conference, BPM 2022, Münster, Germany, September 11-16, 2022, Proceedings, Lecture Notes in Computer Science, Springer, 2022*, p. [To be published].