Proposals for a Virtual Machine for Business Processes

Thomas M. Prinz

Chair of Software Technology, Friedrich Schiller University Jena, Germany
Thomas.Prinz@uni-jena.de

Abstract Business process management (BPM) promises an efficient and simple application of business processes. Although there are many approaches belonging to BPM, BPM suffers from the wide heterogeneity of tools and approaches and their missing integration into a homogenous workflow.
In this paper, we argue for a consistent system, which supports each step of BPM. The system consists of a virtual machine, a process intermediate representation, a compiler, and analyses. Thereby, the major focus lies on the derivation of requirements on a virtual machine for business processes and its intermediate representation. In a first conceptual approach, we present our proposals for such a virtual machine.

Keywords: Virtual machine, processes, business process management

1 Introduction

The state of the art in research of business process management (BPM) conveys a simple and efficient application in practice. Popular business process modeling languages, e.g., the Business Process Model and Notation (BPMN) [9] and the Web Services Business Process Excution Language (BPEL) [7], as well as verifiable process properties, which promise the absence of deadlocks for example (cf. soundness [1,13]), strengthen this impression. In addition to the prevalence of service-oriented architectures (SOA) in companies, there is no reason why the safe and dynamic application of BPM should not be able to work successfully: (1) the process is analyzed, (2) implemented, and (3) applied. During the process's use, it is (4) evaluated (monitored) and (5) improved continuously — the life cycle of a business process (cf. Fig. 1) [2].

However, Koehler et al. already recognized the lacks in the life cycle [8] such that the tool landscape of BPM suffers from its wide heterogeneity and its missing integration into a homogenous workflow. Take the development of a business process as example: In most graphical process modeling languages exist elements, whose semantics have obvious dependencies to the process developer's intension and are therefore not universally and basically definable. In addition, most common used process modeling languages (except BPEL) neither have clear definitions for documents nor for instructions to modify them. As result,

T. S. Heinze, T. M. Prinz (Eds.): Services and their Composition, 7th Central European Workshop, ZEUS 2015, Jena, Germany, 19-20 February 2015, Proceedings – published at http://ceur-ws.org

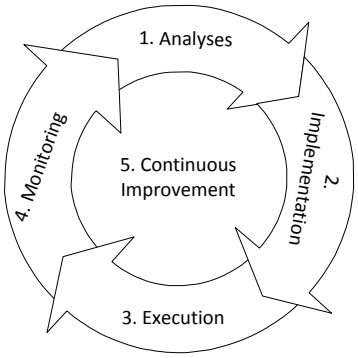

Figure 1. Process' life cycle

the majority of processes is performed manually and only an insignificant part of processes of an organization is executed by a process engine. In short: BPM would benefit from a system, which combines research approaches and which homogenizes the life cycle by new research results furthermore. In this paper, we focus on such a system and, especially, on requirements and on a conception of a homogenous virtual machine for the execution of business processes.

The remainder of this paper is as follows: At first, we explain our overall system's approach [11] (Section 2). In Section 3, we derive requirements of a machine, which are implemented in a first conception (Section 4). Eventually, we give an outlook on future work in Section 5.

2 Overall Approach

Our approach is a system, which is organized into a producer side (where the business process is developed) and a consumer side (where the process is then executed), inspired by Amme et al. [3] regarding mobile code. Our proposed system [11] consists of a compiler and a virtual machine (cf. Fig. 2), which support the development, analysis, storage, execution, maintenance, evaluation, and improvement of processes. We argue for a compiler since the development process of business processes has adjusted itself to the process of software development (cf. the life cycle of business processes to the spiral model [4]). Furthermore, the efficient development of software was improved through a well-defined computer architecture as well as the programming languages based on it. The consideration of state-of-the-art tools for business process development (for a list, visit `http://www.bpmn.org/`) shows that there is a wide heterogeneity of different execution techniques and that most tools are monolithically, i.e., it is hard to include additional analyses and to distribute processes between different execution engines.

Starting from these coniderations, the first step should be the definition of a (virtual) machine, which realizes the execution of a (technical) process. In

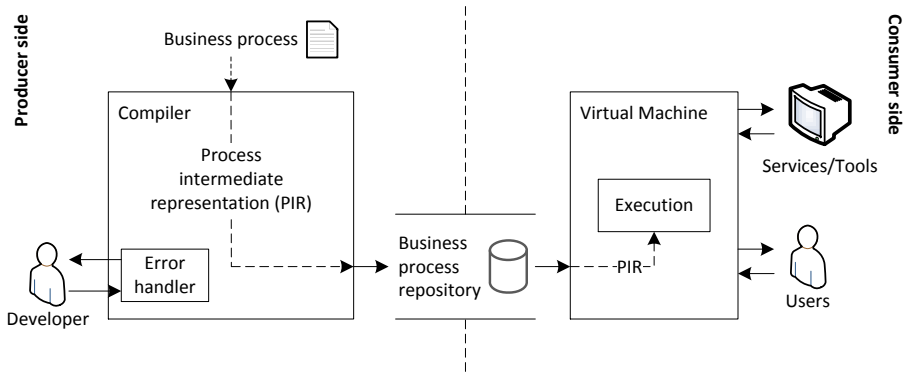

Figure 2. Overall system consisting of a compiler and a virtual machine

the second step, we should develop a technical and general process description language (process intermediate representation (PIR)). Afterwards, in the third step, existing process description languages should be modified in such that the automatic transformation into the PIR is possible. It is not the goal, in doing so, that the iterative derivation of a process from an abstract model to a technical process should be automated, since the inclusion of the process analyst is essential (cf. requirements analysis in software engineering). The goal is to define limitations for which such a transformation is possible at all and to allow a stepwise refinement of the business process into a technical one. In the last step, verifiable properties of processes based on the PIR should be found and some analyses should be developed which seriously support a solution-oriented failure diagnostic. Especially, in the context of business processes, such analyses must be applied to minimize the enormous costs in cases of malfunctions.

As mentioned before, in this paper, we focus on the first two steps: a conception of a virtual machine for the execution of business processes and the definition of a PIR. For this, we derive requirements for such a virtual machine in the next section.

3 Requirements on a Virtual Machine

One of the most famous virtual machines is the *Java Virtual Machine* (JVM). The JVM supports the execution of *Java Bytecode* (JBC), which is generated from *Java* programs by a compiler. The advantage is, that also other programming languages can be compiled to JBC and, therefore, programs based on multiple programming languages are possible. A process engine should follow that approach as it grows its flexibility. For this, *(R.1)* a virtual machine has to allow for the execution of a process code, i.e., the conception of such a machine affiliates with the language to be executed.

Popular process modeling languages have a quite large common subset of description elements on closer considerations. *(R.1.1)* A virtual machine should

allow the handling of basic elements to model the control flow, e.g., exclusive decisions, parallelisms, inclusive decisions, and simple tasks. *(R.1.2)* These elements should be extended by explicit exception and error handling, high-language loops, events and signals, as they are frequently used in processes. Furthermore, *(R.1.3)* the inclusion of roles also is essential to ensure organizational structures as well as access permissions and common security. Eventually, *(R.2)* a process modeling language should naturally enable the usage of instructions and data structures for the modification of data and branch conditions. Such instructions are currently rare in previous languages as mentioned before. *(R.2.1)* The reception and sending of messages besides the usage of documents and conditions naturally are very important. With traditional programming languages in mind, *(R.2.2)* arithmetic operations, comparisons, objects, and method invocations on objects should be part of a process modeling language to modify and compare data. For this, documents are special cases of objects. Method invocations encapsulate the access on object fields and have to be restricted on state changes of objects, in such that side effects are rare and the analysis of processes is more simple. The start of sub processes is incumbent upon the "super" process.

(R.3) A virtual machine has to be able to load processes, which are ready to be started. As result, *(R.3.1)* the virtual machine should have a separate storage for processes. Besides the storage for processes, it needs a further data storage since a process being executed modifies objects. Furthermore, *(R.3.2)* the data storage should be separated (with reference to a single control flow) in a local and a global part as processes have commonly different parallel control flows with their own (private) data. *(R.3.3)* Each of those concurrent control flows should have its own local storage. It then has only the permission to access the local storages of those control flows leading to its execution, i.e., those local storages are global with regard to that single control flow. This is mandatory as some information is not safe on some points of the process. With this in mind, consider the process $P1$ of Fig. 3 in BPMN. This process defines a local object A in a task and starts two parallel sub processes $P2$ and $P3$. Both sub processes get the *global* object A and have access on it (like done in process $P2$). However, the process $P2$ has no permission to access the local object B of process $P3$ since $P3$ does not start $P2$ (neither direct nor indirect). Therefore, the instruction (3) *use* B is displaced. For this, *(R.3.4)* a virtual machine and the corresponding process intermediate representation should define the scopes of objects carefully.

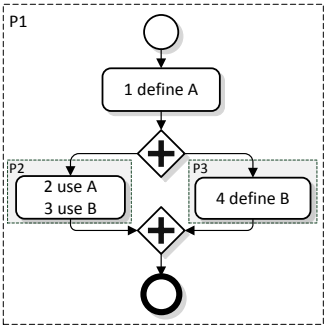

Figure 3. A process defines and uses data

The separation between local and global objects has less attention in previous research. The determination of local and global objects is simple in structured processes as it can be done like for traditional programming languages, e.g.,

by data flow analyses [5]. We propose the application of a process structure tree (PST) for unstructured or irreducible processes, who structures the process as a tree [14]. Although we worked on analyses for irreducible processes in previous work [10,12], we *(R.4)* plead in or for transformations to structured processes to simplify analyses or to make them possible at all. This step towards "structures" was already done for today's programming languages. However, the transformation of irreducible processes into structured ones is challenging since most algorithms (e.g., node splitting [6]) only work for sequential processes and have an exponential growth behavior of the entire graph.

The terms of local and global objects belong to a single control flow in processes and belong not to their spatial position. In general, *(R.5)* sub and partial processes of a process should be able to be executed on different physical machines. That belongs to the fact, that business processes are almost implemented by the use of SOA and that they work beyond company borders. As result, *(R.5.1)* a virtual machine has to abstract from the underlying physical network. For this reason, our considerations have a secure business network and standardized processes in mind as, otherwise, we have to know all details about processes which is commonly not the case. The difficulty is to identify communication partners in networks and to guarantee the correct usage of state-based services.

4 Conception of a Virtual Machine

The requirements mentioned before are implemented as a conception in the following. For this, Figure 4 shows the overall structure of our proposed virtual machine. Concluding from requirements *(R.1)* and *(R.3)*, the machine owns a *process loader* which loads a process and recognize the PIR part. We recommend to use a PIR in a tree representation as such a representation has many advantages [3] and leads to structured processes *(R.4)*. Furthermore, the representation should contain instructions *(R.1.1)* - *(R.2.2)*. Since the PIR is in a linearized file form, a *decoder* has to rebuild the PIR and, afterwards, a *verificator* checks the PIR for unallowed modifications with regard to security properties, e.g., soundness and reference safety. Then, the process is executed in the *process interpreter*.

As argued in previous work [11], a *dynamic semantic analyser* should monitor and analyze the process's execution to detect possible runtime process failures as early as possible. Those failures could be tried to be solved by a *runtime error handler* which interacts with the user or developer. So, a "rescue" of the process is still possible in failure situations and, afterwards, the process interpreter can continue its work.

The process interpreter gets the PIR and creates a new *process frame*. The structure of such a process frame illustrates Fig. 5. Each frame defines the state of each (sub) process and hence stores all necessary execution information *(R.3.1)*: A process object stack, a process extract, an in-/output, a process position pointer, and a process stack. The process stack (illustrated as arc in the figure) contains all process frames of started sub processes. Once a sub process starts,

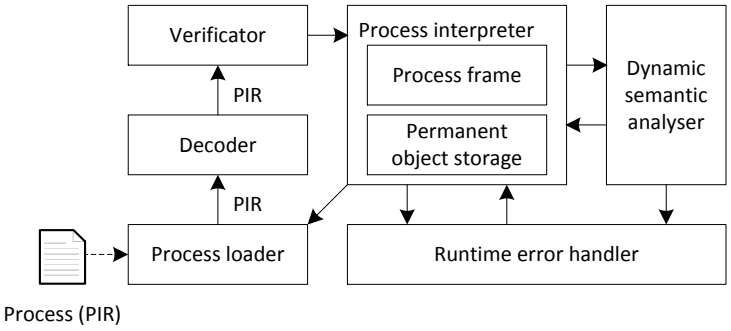

Figure 4. Proposals for a virtual machine for processes

the virtual machine creates a process frame for it. and pushs the frame on the process stack on the calling process. If the work of the sub process is done, its frame can be popped from the process stack. The "super" process can continue its work not until its process stack is empty (as, otherwise, it has to wait for the results of its sub processes). Although other implementations are possible, this stack-based approach reflects all hierarchic dependencies between different processes in a simple way. For example, Java uses a stack for its hierarchy of method invocations.

The process extract shows a subgraph of a tree representation (from the PIR) of the process that should be executed *(R.3.1)*. That tree representation is generated by the compiler with the help of the PST *(R.4)*. Each node of this tree has its own functionality (branch, parallelism, task, etc.) *(R.1.1)* - *(R.2.2)*. Through *link* edges in the tree, sequences are implicit and the tree can be traversed easily. Furthermore, that fact guarantees a simple reference safety, since global objects can be derived by a backward traverse from the current executed node to

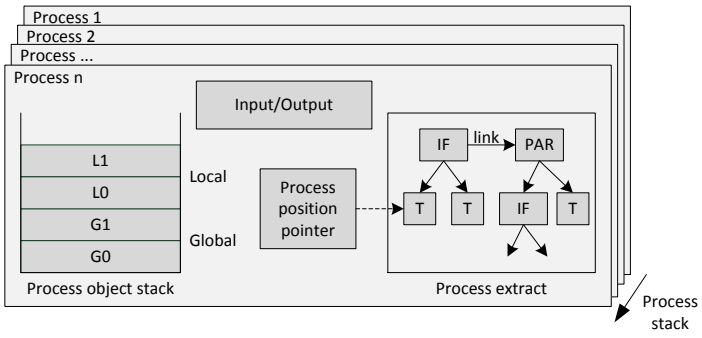

Figure 5. Structure of a process frame

the root of the tree *(R.3.4)*. The current executed node is marked by the process position pointer as the machine has to know which process's nodes are currently executed. The execution semantics of a node depends on its functionality. For example, if the current node produces parallelism, for each of its child nodes, the virtual machine creates a process frame, passes the process object stack, determines the subtree as process extract, and pushes the frame on the current process's process stack. After each sub process is finished, the process frames are popped from the process stack and the *link* node is executed afterwards. If an execution produces objects, they are pushed on the process object stack and they are locatable via this stack *(R.3.3)*. That stack contains all objects, which the (sub) process can access *(R.3.2),(R.3.4)*. Objects, which are stored over process borders, are stored within a permanent object storage (data base). Only processes, which have the permission, can access those objects.

If the execution arrives at an instruction for the reception or sending of a message and event, respectively, *(R.2.1)* the virtual machine uses the in-/output mechanism that handles the communication — regardless whether it talks locally or via a network *(R.5),(R.5.1)*. For this, each virtual machine can provide its processes by a RESTful web service, i.e., it automatically creates an unique uniform resource identifier (URI) for each process. Another process may then contact that URI via a network and gets an URI for the resource which points to the corresponding started instance of the process. Now, it is simply possible to communicate with a state-based process via a network. However, for this, it has to know its communication partners before.

5 Conclusion and Outlook

In this paper, we have extracted the need for a complete and consistent system for the development, analysis, and execution of processes in business process management. For this, we have focused on the conception of a virtual machine for the execution of processes. Besides the conception, we derived requirements for such a virtual machine.

In future work, these proposals have to be implemented and have to be compared to and extended for its application on business process management. Additionally, some parts of the machine (e.g., an efficient intermediate representation of a process) are in our focus of research.

References

1. van der Aalst, W.M.P., Hirnschall, A., Verbeek, H.M.W.E.: An alternative way to analyze workflow graphs. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Özsu, M.T. (eds.) Advanced Information Systems Engineering, 14th International Conference, CAiSE 2002, Toronto, Canada, May 27-31, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2348, pp. 535–552. Springer (2002), `http://link.springer.de/link/service/series/0558/bibs/2348/23480535.htm`

2. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business process management: A survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2678, pp. 1–12. Springer (2003), http://dx.doi.org/10.1007/3-540-44895-0_1
3. Amme, W., Heinze, T.S., von Ronne, J.: Intermediate representations of mobile code. Informatica (Slovenia) 32(1), 1–25 (2008), http://www.informatica.si/PDF/32-1/11_Amme-Intermediate%20Representations%20of%20Mobile%20Code.pdf
4. Boehm, B.W.: A spiral model of software development and enhancement. IEEE Computer 21(5), 61–72 (1988), http://doi.ieeecomputersociety.org/10.1109/2.59
5. Hecht, M.S.: Flow Analysis of Computer Programs. Elsevier Science Inc., New York, NY, USA (1977)
6. Hecht, M.S., Ullman, J.D.: Flow graph reducibility. SIAM J. Comput. 1(2), 188–202 (1972), http://dx.doi.org/10.1137/0201014
7. Juric, M.B.: Business Process Execution Language for Web Services BPEL and BPEL4WS 2Nd Edition. Packt Publishing (2006)
8. Koehler, J., Gschwind, T., Küster, J.M., Völzer, H., Zimmermann, O.: Towards a compiler for business-it systems - A vision statement complemented with a research agenda. In: Huzar, Z., Kocí, R., Meyer, B., Walter, B., Zendulka, J. (eds.) Software Engineering Techniques - Third IFIP TC 2 Central and East European Conference, CEE-SET 2008, Brno, Czech Republic, October 13-15, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 4980, pp. 1–19. Springer (2008), http://dx.doi.org/10.1007/978-3-642-22386-0_1
9. OMG: Business Process Model and Notation 2.0. formal/2011-01-03 (2011), http://www.omg.org/spec/BPMN/2.0
10. Prinz, T.M., Amme, W.: Practical compiler-based user support during the development of business processes. In: Lomuscio, A., Nepal, S., Patrizi, F., Benatallah, B., Brandic, I. (eds.) Service-Oriented Computing - ICSOC 2013 Workshops - CCSA, CSB, PASCEB, SWESE, WESOA, and PhD Symposium, Berlin, Germany, December 2-5, 2013. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8377, pp. 40–53. Springer (2013), http://dx.doi.org/10.1007/978-3-319-06859-6_5
11. Prinz, T.M., Heinze, T.S., Amme, W., Kretzschmar, J., Beckstein, C.: Towards a compiler for business processes - a research agenda. In: de Barros, M., Rückemann, C.P. (eds.) SERVICE COMPUTATION 2015, The Seventh International Conferences on Advanced Service Computing. pp. 49–55. IARIA, IARIA XPS Press, Nice, France (March 22 - 27 2015)
12. Prinz, T.M., Spieß, N., Amme, W.: A first step towards a compiler for business processes. In: Cohen, A. (ed.) Compiler Construction - 23rd International Conference, CC 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8409, pp. 238–243. Springer (2014), http://dx.doi.org/10.1007/978-3-642-54807-9_14
13. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Inf. Syst. 25(2), 117–134 (2000)
14. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data Knowl. Eng. 68(9), 793–818 (2009), http://dx.doi.org/10.1016/j.datak.2009.02.015