The FUNSOFT Net Interpretation Algorithm

Holger Burbach
Volker Gruhn
LION GmbH
Universitätsstraße 140
D-44799 Bochum
[burbach,gruhn]@lion.de

1 Introduction

The research in the area of software process management has yield basic progress in software process modeling and analysis. The benefits of defining a process in terms of a formal language have been discussed and a number of examples of process modeling languages have been proposed. These include rule-based [10, 11, 6], Petri-net-based [1, 3] and object-oriented languages [5, 9]. Despite some major process modeling and analysis projects, process enaction experience for large-scale software processes is hardly available. In this article we discuss the process engine for the enaction of process models described in terms of FUNSOFT nets. This process engine has been impacted by large-scale process modeling projects from different application domains.

2 The FUNSOFT net approach to process management

In the FUNSOFT net approach, process models consist of data models, activity models, and organization models. Data models describe types of objects to be manipulated and the relationships between object types. They are represented by extended entity-relationship models [8].

Activity models describe the activities to be executed within processes and their order. They are represented by FUNSOFT nets [4]. FUNSOFT nets are high-level Petri nets adapted to the requirements of process modeling. T-elements of FUNSOFT nets represent activities to be executed in a process. They are called agencies. An agency is activatable (i.e. it can be fired) as soon as all its input parameters are available. To fire an agency means to execute the activity represented by the agency. S-elements of FUNSOFT nets represent object stores. They are called channels. An object being stored in a channel is also called a token marking the channel (in Petri net terminology). The notion of objects and the use of objects in Petri nets is similar to the notion of objects in THOR nets [12]. Each channel is associated with an object type. A channel to which a certain type is assigned can only be marked with objects of that type. In FUNSOFT nets some modeling concepts are used which can not be found in ordinary petri nets. These concepts have a strong influence on the interpretation algorithm. The following list contains a short description of the concepts. For a more detailed discussion we refer to [2, 7].

Predicates:

A predicate is a boolean function associated to an agency. In addition to any other conditions the predicate has to be fulfilled to permit the firing of the agency. The result

of the predicate may depend on the values of objects in channels in the preset of the agency or may be time-dependent.

Laddering attribute:
The laddering attribute of each agency specifies, how often the agency may be fired simultaneously.

Processes bound to agencies:
In a process model, which is supposed to be bound to an agency, input and output channels have to be identified explicitly. In binding a process model to an agency, these channels are mapped to channels in the pre- and postset of the agency the model is bound to.

Agenda bypasses:
The boolean agenda bypass attribute determines how manual agencies of the refinement are treated. The first agency of such a refinement selected by a process participant initiates that a new son process is created. The treatment of this son process resembles the treatment of processes bound to agencies, however, with differences in the treatment of activatable agencies in the son process:

- Only agencies belonging to the bypassed refinement may fire.
- For channels into which objects have been written, it is checked whether they belong to the refinement or whether they are in the postset of the refined agency. If they are in the postset, the objects are immediately shifted back to the father process with all consequences for activation checks.
- If an agency of the bypassed refinement has already been selected by a process participant, then all activatable manual agencies are handled as if this process participant had selected them already. The bypass handling is terminated automatically when there is no activatable or active agency left in the bypassed refinement (i.e. the refining FUNSOFT net is dead).

Automation attribute:
The boolean automation attribute of each agency specifies whether the firing of the agency may be started without informing any process participant about it.

Copy access to channels:
The boolean copy flag of each edge linking a channel in the preset of an agency with this agency specifies whether the agency reads copies of the objects in the channel.

Interface channels:
If an object is written into an interface channel, all running processes in the models of which the interface channel occurs receive a copy of the object. If there is no running process with read access to the interface channel, the next process with read access receives the object.

3 The Process Engine Algorithm

In this section, we describe the process engine algorithm implemented in the FUNSOFT net process engine. Since the embedding of the process engine into other enaction components influences the algorithm we have a glance at these components. Each process engine communicates with its own agenda controller. The task of this controller is to manage the display of agenda entries. It checks permissions and communicates with the agendas of the process participants to fulfill its task. In order to perform firings the engine uses an activity handler which manages the function and dialog execution components. The work of the whole environment is managed by the environment controller. It allows to start and to stop process

engines and agenda controllers. The environment controller always knows, how many engines are running and what they are interpreting. Process participants who have the proper permission may use their agendas to send commands to the environment controller.

The purpose of the whole environment is the management of agenda entries and corresponding agency firings. This is realized by relating agenda entries and agency firings to a central entity of the process engine algorithm: the agency incarnation.

An agency incarnation is created whenever the preset of an agency is marked and, thus, the agency could be fired. Roughly speaking, the only functionality of the process engine algorithm is to identify agency incarnations, to check if they are executable (with respect to predicates and conflicts) and to distribute them to the *right* process participants (i.e. to those process participants who have the permission to execute in their firing). Therefore, we describe structure of the algorithm by means of dataflow of agency incarnations.

The dataflow describing the process engine algorithm is depicted in figure 1. Sets of agency incarnations are denoted by capital letters. The flow between the sets is described with the help of edges which are denoted by capital letters with indices.

We refer to these capital letters in the following explanation of figure 1. The initial assignment of agency incarnations to sets is as follows: at the beginning of a process one agency incarnation (for each agency of the process model) is in set A, all further agency incarnations (the number of further agency incarnations is defined by the laddering attribute of each agency) are in set E. If, for example, a process model contains n agencies $t_1, \ldots, t_n$ with laddering attributes $lad(t_i)$, then A initially contains one agency incarnation for each $t_i, i = 1, \ldots, n$ and E initially contains $lad(t_i) - 1$ agency incarnations for each $t_i, i = 1, \ldots, n$.

A This set contains agency incarnations to be checked for activatability.

A_1 Agency incarnations can only be moved along this edge if the sets C, D, F, G, H are empty. This ensures that no agency incarnations are processed before the last ones are completely checked. Thus it is guaranted that in each process engine activation cycle all agency incarnation are considered. Only when this is done, the next cycle can be started.

B For all incarnations reaching this set it is ensured that multiple incarnations of the same agency are never processed at the same time. This would lead to syncronization problems with other enaction components.

B_1 All incarnations passing the check in set B are moved along this edge.

B_2 All incarnations failing to pass the check in set B are moved along this edge.

C For all incarnations reaching this set it is checked whether the preset of the agency incarnation is filled.

C_1 All incarnations passing the check in set C are moved along this edge.

C_2 All incarnations failing to pass the check in set C are moved along this edge.

D This set is used to manage predicates. If no predicate is assigned to the agency, then it is considered true for each agency incarnation as default. Agency incarnations in set D are moved into set F if the related predicate is true for at least one token combination and if conflicts have to be resolved. They are moved directly into set H if no conflict resolution is needed. If the predicate is not fulfilled by any token combination and if the predicate is not time-dependent(i.e. if it cannot be fulfilled by waiting) then the agency incarnation is moved back to set A (i.e. it is checked in the next activation check). If the predicate is time-dependent and if it not fulfilled by any token combination, then agency incarnations remain in set D.

E This set contains all agency incarnations which are not to be processed in the current state of process enaction.

E_1 An activation check may be started by moving to set A all agency incarnations, which have to be checked.

F This is the set of all agency incarnations which are waiting for other incarnations to start their firing, e.g. to wait for agency incarnations which have been moved from set D to set H directly. In order to enable a maximum number of firings these incarnations are started before any other incarnations are moved into D.

G This set is used to solve conflicts between agency incarnations which are expected to start their firing automatically. The agency incarnations reaching the set are divided into two subsets: the subset of the winners of conflicts and the subset of the others which are considered as conflict losers.

G_1 The losers of the conflict solution are moved along this edge.

G_2 The winners of the conflict solution are moved along this edge.

H All agency incarnations reaching set H fulfill the condition: if it is in conflict with another agency incarnations then at most one of both is to be started automatically. This set is used to distribute the agency incarnations between timer start, display in agenda and automatic start.

H_1 The agency incarnations for which the timer has to be started are moved along this edge.

H_2 The agency incarnations which have to be diplayed in the agenda are moved along this edge.

H_3 The agency incarnations which are to be started at once are moved along this edge.

I All agency incarnations reaching this set are displayed in the agenda.

I_1 All incarnations selected in the agenda are moved along this edge.

I_2 Agency incarnations which have been displayed in agenda and which lost their conflicts (because a process participant has selected a conflicting agency incarnation (compare I_1 and L) or because an automatic agency incarnation has been fired because a timer has elapsed (compare J_2 and L)) are moved from I to A.

J For all agency incarnations reaching this set a timer is started.

J_1 All agency incarnations for which a timer has elapsed are moved along this edge.

J_2 This edge is used by agency incarnations which have lost their conflict (compare edge I_2 and set L).

K For all agency incarnations reaching this set a firing is started.

K_1 Each incarnation which has ended its firing is moved along this edge.

L For all agency incarnations reaching this set conflicting incarnations in set I or J are moved along the edges I_2 or J_2 respectively.

M For all agency incarnations reaching this set an activation check for another incarnation of the same agency is started by moving this incarnation along edge E_1 into set A.

N For all agency incarnations reaching this set an activation check is started for one incarnation in set E of each agency whose preset has just been changed due to the termination of a firing.

The handling of agenda bypasses and processes bound to agencies in the process engine algorithm remains to be clarified: if a new son process is created, the respective agency incarnations are inserted into the sets A and E (following the rule for initial assigments of agency incarnations to sets mentioned above). If the graph is dead with respect to the agency incarnations of one process and if this process does not have son processes, then all agency incarnations of this process are removed from all sets.

4 Experiences and Conclusions

In using FUNSOFT nets for process modeling for about nine years, we identified several FUNSOFT net extensions which helped to make process modeling more comfortable. In fact, some extensions turned out to be absolutely inevitable in large-scale process modeling projects, even though smaller process modeling projects had not at all raised the need for these extensions.

These extensions meant to also extend the process engine algorithm. The initially simple FUNSOFT net process engine algorithm had to be extended to cope with these additional modeling concepts. The decision to integrate different process engines by message passing on the hand and by using a common database also meant to extend the process engine algorithm by communication facilities between process engines.

The extensions discussed were integrated into the process engine algorithm. Thereby, the process engine algorithm became more and more complex. Even though its internal structure remains clear it became less extensible. Based on this experience, it is rather difficult to decide between modeling comfort (usually demanding new modeling constructs) and ease of enaction (demanding that as few constructs as possible have to be taken into consideration).

References

[1] S. Bandinelli, A. Fuggetta, C. Ghezzi, and L. Lavazza. *SPADE: An Environment for Software Process Analysis, Design, and Enactment.* In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Technology - Proceedings of the 2nd European Software Process Modeling Workshop*, pages 223–244 Somerset, England, 1994. Research Studies Press Ltd.

[2] H. Burbach and V. Gruhn. *FUNSOFT Nets as Process Modeling Language.* In J. Desel, H. Fleischhack, A. Oberweis, and M. Sonnenschein, editors, *2. Workshop: Algorithmen und Werkzeuge für Petrinetze*, Oldenburg, October 1995. erschienen als Bericht Nr. 22 des Fachbereichs Informatik der Universität Oldenburg.

[3] W. Deiters and V. Gruhn. *The FUNSOFT Net Approach to Software Process Management. International Journal of Software Engineering and Knowledge Engineering*, 4(2):229–256, 1994.

[4] G. Dinkhoff, V. Gruhn, A. Saalmann, and M. Zielonka. *Business Process Modeling in the Workflow Management Environment LEU.* In P. Loucopoulos, editor, *Proceedings of the 13th International Conference on the Entity-Relationship Approach*, pages 46–63, Manchester, UK, December 1994. Springer. Appeared as Lecture Notes in Computer Science no. 881.

[5] G. Engels and L. Groenewegen. *SOCCA: Specifications of Coordinated and Cooperative Activities.* In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Technology - Proceedings of the 2nd European Software Process Modeling Workshop*, pages 71–100, Somerset, England, 1994. Research Studies Press Ltd.

[6] P. K. Garg and S. Bhansali. *Process Programming by Hindsight.* In *Proceedings of the 14th International Conference on Software Engineering*, Melbourne, Australia, May 1992.

[7] V. Gruhn. *Geschäftsprozeß-Management als Grundlage der Software-Entwicklung. Informatik Forschung und Entwicklung*, 11:94–101, Juli 1996.

[8] V. Gruhn, C. Pahl, and M. Wever. *Data Model Evolution as a Basis of Business Process Management.* In *OOER95: Object-Oriented and Entity-Relationship Modeling*, pages 270–281, Gold Coast, Australia, December 1995. Springer, Berlin. Appeared as Lecture Notes in Computer Science 1021.

[9] G. Junkermann, B. Peuschel, W. Schäfer, and S. Wolf. *MERLIN: Supporting Cooperation in Software Development Through a Knowledge-Based Environment.* In B. Nuseibeh A. Finkelstein, J. Kramer, editor, *Software Process Modelling and Technology*, pages 103–129, Somerset, England, 1994. John Wiley and Sons.

[10] G.E. Kaiser, S.S. Popovich, and I.Z. Ben-Shaul. *A Bi-Level Language for Software Process Modeling.* In *Proceedings of the* 15^{th} *International Conference on Software Engineering*, Baltimore, Maryland, US, May 1993.

[11] C. Montangero and V. Ambriola. *OIKOS: Constructing Process-Centred SDEs.* In B. Nuseibeh A. Finkelstein, J. Kramer, editor, *Software Process Modelling and Technology*, pages 131–151, Somerset, England, 1994. John Wiley and Sons.

[12] S. Schöf, M. Sonnenschein, and R. Wieting. *Efficient Simulation of THOR Nets.* In G. De Michelis and M. Diaz, editors, *Proceedings of the* 16^{th} *European Workshop on Application and Theory of Petri Nets*, pages 412–431, Torino, Italy, 1995. Springer. Appeared as Lecture Notes in Computer Science no. 935.

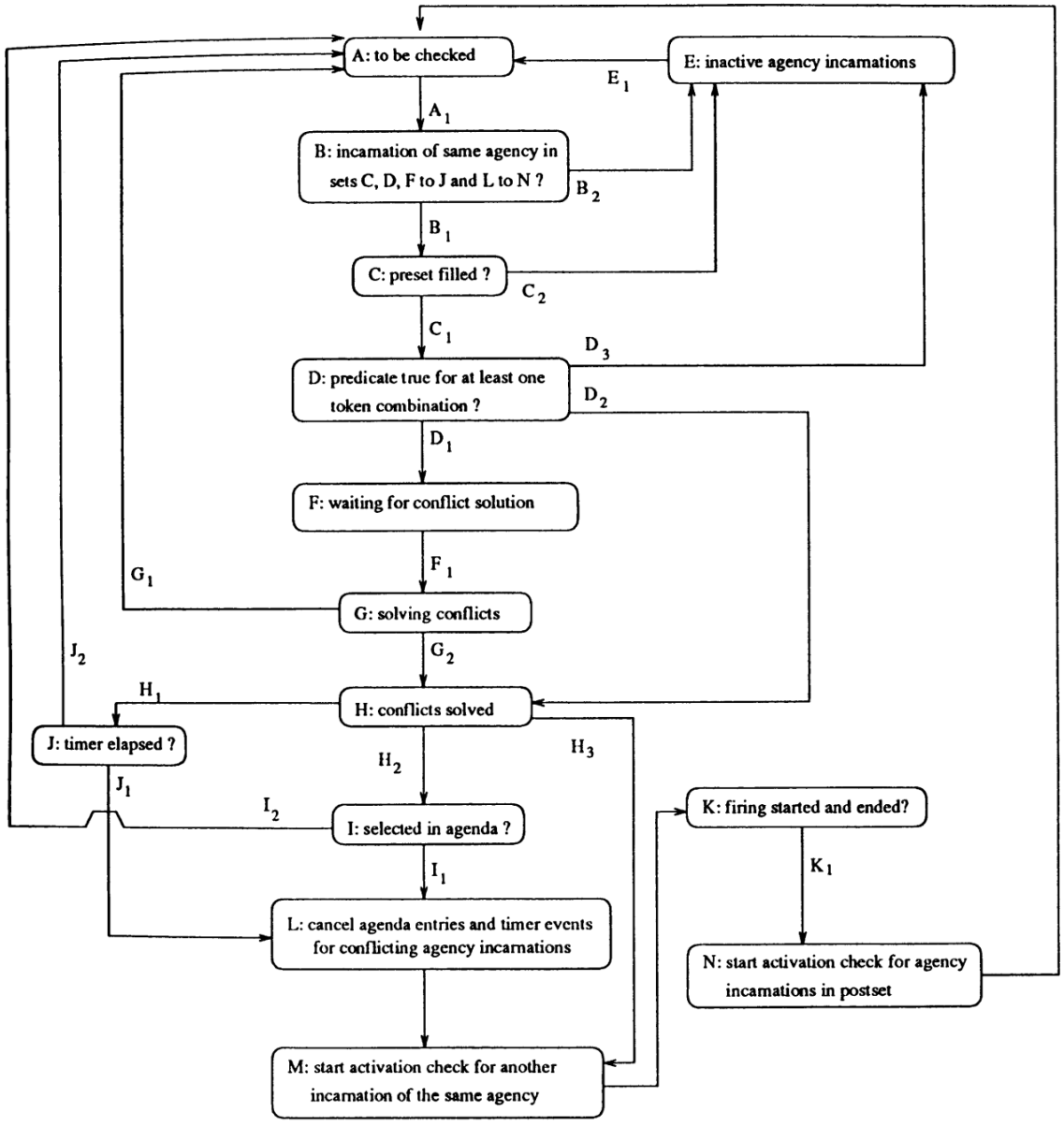

Figure 1: The Agency Incarnation Dataflow