FUNSOFT Nets as Process Modeling Language

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

1 Introduction

Software process management [9, 5] and business process management [7] have been areas of research for some years now. While the question what a suitable process modeling language should look like has extensively been discussed in the software process community [1, 10], the workflow people pay less attention to the modeling languages used [2].

The management of business processes has gained some attention by the use of buzzwords like ***business process (re-)engineering, process innovation, lean management, and total quality management*** [11]. Enaction of business processes is often called *workflow management* [8].

2 FUNSOFT nets

In the FUNSOFT net approach, which, for example, is implemented in the LEU environment [4] and in the MELMAC environment [3], process models are created by integrating data models, activity models, and organization models. In the following we restrict ourselves to activity modeling.

Activity models describe the activities to be executed within processes and their order. They are represented by FUNSOFT nets [6]. 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). Edges from channels to agencies describe, that an object stored in the channel is used as input parameter for the agency, edges from agencies to channels describe where output parameters of agencies are stored.

In the following example, we consider a resource management process in which all resources are administrated. For this example, we focus on computer hardware and computer software. From time to time, resources are examined in order to decide whether resource replacements or purchases are necessary. If it is decided to replace a resource, either a new resource is designed, e.g. by writing a new program to replace an existing one, or a new resource is bought. Figures 2 to 5 describe this process. The structure of the example discussed is illustrated in figure 1. It shows that the process models fro *resource management* and *resource purchase* are related by interface channels. The process model *resource test* is bound to two agencies of process model *resource management.* Finally, process model *perform resource test* is a refinement of an agency of process model *resource test.*

In figure 2 to figure 5 we recognize different icons representing agencies:

- agencies representing manual activities (e.g. *select for change* in figure 2),
- agencies representing automatic activities (e.g. *print resource test* in figure 2),
- agencies with processes bound to them representing the call of processes from within other processes (e.g. *test buyable resource* in figure 2; described below),

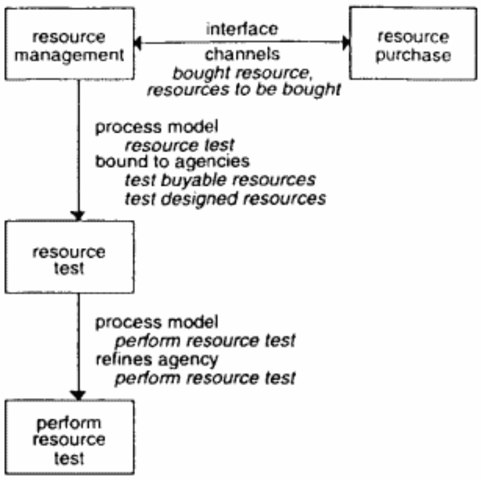

Figure 1: Structure of the software process example

- refined agencies which are used to hide details on a lower level (not occurring in the example),
- refined agencies marked as an agenda bypass. That means, all agencies in such a refinement represent closely related activities which have to be carried out by only one person (e.g. *perform resource test* in figure 4; described below).

The process creates and manipulates objects of several types. For reasons of space, the data model is not discussed here. The name of an object type assigned to a channel can be found above the channel. The meaning of the object types used in the example is as follows:

- An object of type *Resource Record* contains all information about a concrete resource.
- An object of type *Resource Distributor Record* contains all information about a distributor.
- An object of type *Test Record* contains all information about a resource test.
- An object of type *Resource Order* contains all information about an order being sent to a distributor.

All modeling concepts mentioned in the following example (and being underlined) are explained in the section 3. At the start of the process, the channel *existing resources* in figure 2 contains records of all available resources. A time-dependent predicate is assigned to the agency *select for change*. The predicate causes the agency to fire once a week.

We assume that the value of the laddering attribute for the agency *select for change* has been set to 5 when editing the process model. This means that five process participants can in parallel select resources to be changed.

When the agency *select for change* is executed, it moves the resource record selected into the channel *changing resources*. As soon as this channel contains at least one resource record, the agencies *arrange test* and *design resource* can be fired. For each member of the department the laddering attribute of the two agencies in the process model has to be increased in order to enable them to work in parallel. After testing the bought or the internally developed resource, a resource record is written into the channel *tested resources*.

In our example, the process model *resource test* of figure 4 is bound to the agencies *test designed resource* and *test buyable resource* of figure 2. This binding is based on the channel assignments as described in table 1 (i.e. the IN channel of process model *resource test* is mapped to channel *replacement test* arranged in the context of agency *test buyable resource* of figure 2).

In the process *resource test*, first the resource to be tested has to be installed. A member of the test group will be in charge of the test installation. To notify him of the installation required, the agency *install resource* can be fired.

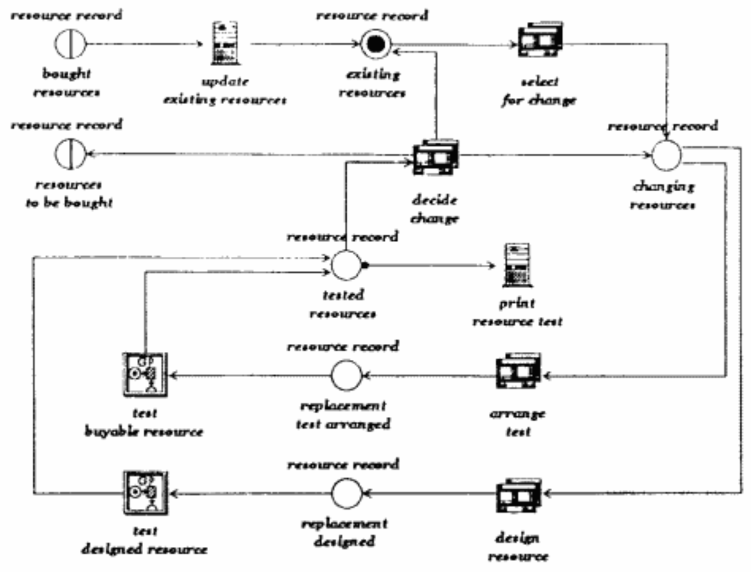

Figure 2: Resource Management

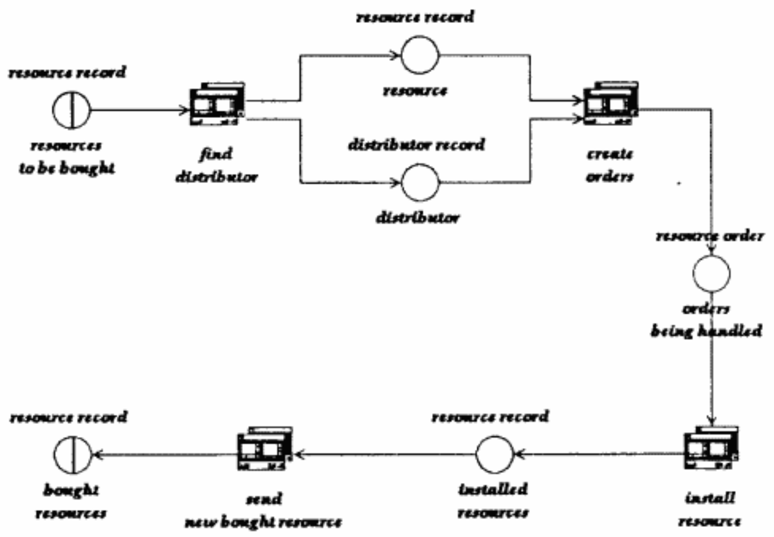

Figure 3: Resource Purchase

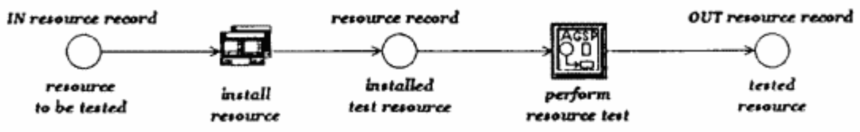

Figure 4: Resource Test

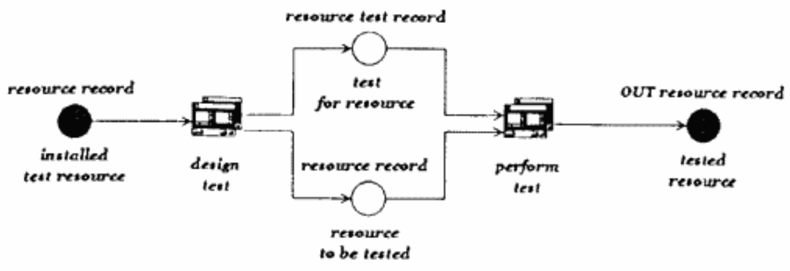

Figure 5: Perform Resource Test

	IN resource record	OUT resource record
agency *test buyable resource*	replacement test arranged	tested resources
agency *test designed resource*	replacement designed	tested resources

Table 1: Mapping of IN and OUT channels to pre- and postset of agencies to which process model *resource test* is bound

After the installation, a single tester can design and carry out the test. These two tasks are described in figure 5, which shows a refinement of the agency *perform resource test* of figure 4.

The refined agency *perform resource test* in figure 4 is marked as an agenda bypass. That means that all activities of the resource test have to be carried out by only one process participant. Thus, after carrying out the *design test* activity (which is the first activity of the refinement of agency *perform resource test* described in figure 5) all other activities of this refinement are immediately started as soon as they are activatable.

When the test of a resource is finished and the record arrives in the channel *tested resources* of figure 2, the agency *print resource test* is automatically executed by the process engine. The agency prints the description and the results of the latest test found in the record.

In order not to disturb the execution of the agency *decide change* the agency *print test* copies the resource record.

A predicate assigned to the agency *print test* ensures that the agency is executed only once for each record arriving in the channel. In parallel to the execution of *print resource test*, the agency *decide change* is put into the agenda of the head of the department of resource management. Depending on the decision taken, the resource record is written into one of three possible postset channels:

- if a replacement resource is to be bought, the record is written into the channel *resources to be bought*,
- if the resource is not to be replaced anymore, the record is written back into the channel *existing resources*,
- if an alternative replacement resource has to be considered, the record is written back into the channel *changing resources*.

The interface channel *resources to be bought* is used to exchange resource records between the resource management process (figure 2) and the purchase process (figure 3). In the purchase process, resources are bought after it has been decided to do this in the resource management process.

For each resource record arriving in the channel *resources to be bought* (figure 3), a distributor is looked for in the agency *find distributor*. The pair of resource and distributor is written into the postset channels which are read by the agency *create orders* (figure 3). Within this activity, the order is created and sent to the distributor. The object representing the order is written into the channel *orders being handled*.

For each ordered resource delivered, the agency *install resources* reads the respective object from channel *orders being handled*, creates a resource record, and writes it into the channel *installed resources* after the resource has been installed. The agency *send new bought resource* reads all the records and writes the resource records into the channel *bought resources.* This channel is accessed by the agency *update existing resources* of figure 2. Thus, the new resource record is put into the resource pool.

3 Glossary of modeling concepts used in FUNSOFT nets

Predicates: A predicate is a boolean function specifying conditions which have to be fulfilled before an agency can be fired. With the help of predicates, conditions depending on the values of objects can be specified. Furthermore, predicates can be used to specify time-dependent conditions.

Laddering attribute: Each agency has an attribute called *laddering attribute.* The value of this attribute is a positive integer with 1 as default. This value specifies how often an agency can be fired simultaneously. It eases modeling, because the degree of parallelism can easily be defined and modified.

Processes bound to agencies: Binding a process model to an agency supports the reuse of process models and is a mechanism to structure complex process models. In a process model, which is supposed to be bound to an agency, input and output channels have to be identified explicitly. This is necessary in order to specify how the process model is integrated into the overall process model. These channels have *IN* respectively *OUT* as prefix of their names. 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. Agencies with process models bound to them are represented by special icons (e.g. agency *test buyable resource* in figure 2). In this example, the mapping of IN and OUT channels of the process model *resource test* to the pre- and postset channels of the agencies *test buyable resources* and *test designed resource* is described in table 1.

Agenda bypasses Refined agencies have a boolean attribute called agenda bypass. Its default value is *FALSE.* If the value is set to *TRUE,* the agency is represented by a special icon (e.g. the agency *perform resource test* in figure 2). The 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 such a bypass son process differs from processes bound to agencies in the way activatable agencies are handled.

- First of all, only agencies belonging to the bypassed refinement may fire. Other agencies are suspended as long as the bypassed refinement is carried out.
- The second difference is the way objects are handled after the agency has fired. 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.
- The third difference is the way manual agencies are handled. 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).

Agenda bypasses ensure that certain parts of a software processes are carried out by only one process participant. The process participant is not known in advance, but the person who starts such a process part is also responsible for the rest of this process part.

Automation attribute: Each agency has a boolean attribute called *automation attribute.* The default value is *FALSE.* When editing a process model, this attribute may be set to *TRUE.* The attribute specifies whether the firing of the agency may be started without informing any process participant about it.

Copy access to channels: Each edge linking an agency with one of its preset channels has a boolean attribute called *copy flag.* Its default value is *FALSE.* If the value of this attribute for an actual edge is *TRUE,* a small filled circle is displayed at the start of the edge (compare edge between channel *tested resources* and agency *print resource test* of figure 2). If the copy flag is set to *TRUE,* the agency accessing a channel using this edge reads a copy of the object. The object will not be removed from the channel. Conflicts between agencies with copy access are solved in a way allowing the maximum number of firings, i.e. if possible firings of agencies with copy access will be executed first. The copy flag enables several agencies to access the same object in an easy way.

Interface channels: Circles with a vertical line inside represent interface channels. Interface channels in different process models represent only one channel. That means, access to an interface channel has side-effects to all occurrences of that interface channel.

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. Interface channels allow easy data exchange between processes. A process using interface channels can be easily replaced by another process without having to change other processes, too.

4 Experiences and Conclusions

In using FUNSOFT nets for process modeling for about eight years, we came along different requests for more comfortable process modeling. Some of these requests resulted in modifications of the modeling tools, a very few had some influence on the modeling concepts.

In FUNSOFT nets we tried to keep the number of modeling concepts as small as possible as long as people were able to model things with the available concepts. Nontheless we had to integrate some extensions which were raised in a project, in which about 40 modelers described all business processes of several housing construction and administration companies. These extensions (agenda bypasses, processes bound to agencies) were not needed in the earlier software process modeling experiments, but they eased modeling of certain situations substantially.

Summing this up, we believe that the number of modeling concepts should be kept as small as possible. Comfortable modeling should not justify any extension. But, of course, the modeling language must allow to express business process situations as they are (without burdening them under too much syntactical details demanded by the modeling language). Only then, it is reasonable to let people, who know *their* processes, take part in process modeling.

References

[1] S. Bandinelli, A. Fugetta, and S. Grigolli. *Process Modelling In-the-Large with SLANG.* In *Proceedings of the 2nd International Conference on the Software Process - Continuous Software Process Improvement*, pages 75–83, Berlin, Germany, February 1993.

[2] G. Chroust. *Interpretable Process Models for Software Development and Workflow.* In W. Schäfer, editor, *Software Process Technology - Proceedings of the 4th European Software Process Modeling Workshop*, pages 144–153, Noordwijkerhout, Netherlands, April 1995. Springer. Appeared as Lecture Notes in Computer Science 913.

[3] W. Deiters and V. Gruhn. *Managing Software Processes in MELMAC.* In *Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments*, pages 193–205, Irvine, California, USA, December 1990.

[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] C. Ghezzi, editor. *Proceedings of the 9th International Software Process Workshop*, Airlie, VA, US, October 1994.

[6] V. Gruhn. *Validation and Verification of Software Process Models.* In A. Endres and H. Weber, editors, *Proceedings of the European Symposium on Software Development Environments and CASE Technology, Königswinter, FRG*, pages 271–286, Berlin, FRG, June 1991. Springer. Appeared as Lecture Notes in Computer Science 509.

[7] M. Hammer and J. Champy. *Reengineering the Corporation.* Harper Business, New York, US, 1993.

[8] S. Jablonski. *Functional and Behavioral Aspects of Process Modeling in Workflow Management Systems.* In *Connectivity '94 - Workflow Management - Challenges, Paradigms and Products*, pages 113–133, Linz, Austria, October 1994. R. Oldenbourg, Vienna, Munich.

[9] W. Schäfer, editor. *Software Process Technology - Proceedings of the 4th European Workshop on Software Process Modelling*, Noordwijkerhout, The Netherlands, April 1995. Springer. Appeared as Lecture Notes in Computer Science 913.

[10] S.M. Sutton, D. Heimbigner, and L. Osterweil. *Language Constructs for Managing Change in Process-Centered Environments.* In *Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments*, pages 193–205, Irvine, California, USA, December 1990. Appeared as Software Engineering Notes, 15(6), December 1990.

[11] B. Warboys. *Reflections on the Relationship Between BPR and Software Process Modelling.* In P. Loucopoulos, editor, *Proceedings of the 13th International Conference on the Entity-Relationship Approach*, pages 1–9, Manchester, UK, December 1994. Springer. Appeared as Lecture Notes in Computer Science no. 881.