Process Inheritance

Christoph Bussler

Oracle Corporation
Redwood Shores, CA 94065, U. S. A.
Chris.Bussler@Oracle.com

Abstract. In large process deployments where an enterprise has to model and maintain a large number of processes that are specializations of each other it is advantageous to provide formal support for process specialization called process inheritance. A process inheritance model definition is introduced as well as a process definition language construct for defining process specialization.

1 Introduction

A very naïve but very pragmatic way to achieve the specialization of processes is to copy the most relevant process definition and modify the copy until it implements the desired behavior. Maintenance of processes becomes very difficult this way since changes that have to be applied to the original and all of its copies have to be updated manually (let alone the copies of the copy, etc.). Furthermore, the differences between the modified copy and the original are not explicitly stated. Anyone who would like to understand the difference has to compare the definitions.

A much more elegant and useful approach is to achieve public process specialization through process inheritance. In the following first a definition of "process" is given followed by the definition of "process inheritance".

2 Process Definition

A process definition consists of several different process aspects [2] [3] [4]:

- **Functional aspect**. The functional aspect provides constructs to define process types and process steps. A process type is the topmost entity that is decomposed into several process steps. Process steps are the unit of work a human or an executable program has to fulfill. Subprocesses are process steps that are themselves further decomposed. Through this mechanism an abstraction hierarchy of arbitrary depth can be defined.

  Process types as well as process steps have input and output parameters. Parameters are used to communicate data values to process steps or obtain results from process step execution.

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 701-705, 2002.

- **Control flow aspect**. The control flow aspect provides constructs to define the sequencing between process steps. The most important constructs are sequence, parallel branching, conditional branching as well as looping.
- **Data and data flow aspect**. Process types as well as process steps have to access data in order to execute the business logic. Process data are stored locally as local variables in the scope of process types. Data flow constructs connect process data with the input parameters and output parameters of process steps. Data flow constructs define which process data are connected to which process steps in order to define which process step can read or write which process data. Data flow makes explicit which process step reads or writes to which process data.
- **Organizational aspect**. The organizational aspect defines which human user is responsible for executing a process step. If a process step is to be executed automatically, it is an automatic process step without any human user assignment.
- **Operational aspect**. The operational aspect defines the binding of process steps to executable programs. These programs can be productivity tools for human users or executables for automatic process steps.

3 Process Inheritance Definition

A process that inherits from another process is called a subprocess class. The process it inherits from is called the superprocess class. A subprocess class has to have a unique name amongst all processes. If no changes are applied to the subprocess class its instances at execution time behave exactly the same way than instances of the superprocess class. If changes are applied, the changes dictate the new behavior. Changes to subprocess classes can be of two fundamental types, process element overwrite and process element addition.

- **Process element overwrite**. A process element overwrite replaces a process element as defined in the superprocess class with another element. The overwrite takes place in the subprocess class. For example, if two process steps have a sequence control flow construct between them in the superprocess class, the control flow construct can be overwritten by a parallel branching construct in the subprocess class. This would allow the two process steps to be executed in parallel in any instance of the subprocess class. All process aspects as introduced earlier can be subject to overwrite.
- **Process element addition**. A subprocess class does not necessarily have to only overwrite process elements of its superprocess class. It is possible that in the subprocess class new process elements are added. An example for this type of change is the addition of a logging process step that explicitly logs data from the data flow. All process aspects as introduced above can be added.

These two fundamental types of changes that can be applied to subprocess classes at the same time define the concept of public process inheritance.

Related work in process inheritance that requires explicit discussion at this point can be found in [1]. While the above definition targets at a pragmatic and complete

approach addressing all process aspects, [1] targets an abstract and theoretical definition of process inheritance. [1] abstracts in its discussion from the data flow aspect, from subworkflows in the functional aspect, from the organizational aspect and from the operational aspect. Since these process aspects are not addressed, no definition of their inheritance semantics is provided. Furthermore, it reduces the process definition to a specific class of Petri-Nets.

4 Process Language

In the following the process definition language (similar to the one defined in [3]) is taken as the basis for adding constructs for specialization. Due to the space limitations, only a short and abstract example is given.

```
WORKFLOW_CLASS S /* SEND */
   (IN message: send_message) /* step sending a message */
END_WORKFLOW_CLASS

WORKFLOW_CLASS R /* RECEIVE */
   (OUT message: received_message) /* step receiving a message */

END_WORKFLOW_CLASS
```

Based on these two steps, a process called R/R (for request/reply) is defined. After declaring two steps the control flow definition "cf_1" between the two steps follows. They are executed in strict sequence without any branching or parallelism. Then the data flow definitions "df_1" and "df_2" follow. Two local variables are defined that are going to store the two messages that are sent or received. Two types of message are necessary: “out_message” and “in_message”. The data flow section defines which steps set or read the local variables. The organizational aspect definitions "o_1" and "o_2" define that the process steps are executed automatically.

```
WORKFLOW_CLASS R/R
   SUBWORKFLOWS
   S: send_step;
   R: receive_step;
   END_SUBWORKFLOWS
   CONTROL_FLOW
   cf_1: sequence(send_step, receive_step);
   END_CONTROL_FLOW
   WORKFLOW_DATA
   Message: out_message;
   Message: in_message;
   END_WORKFLOW_DATA
   DATA_FLOW
   df_1: out_message -> send_step.send_message;
   df_2: receive_step.received_message -> in_message;
   END_DATA_FLOW
   ORGANIZATION
    o_1: send_step: automatic;
    o_2: receive_step: automatic;
   END_ORGANIZATION
END_WORKFLOW_CLASS
```

5 "subclass_of" Construct

In order to denote process inheritance the "subclass_of" construct is introduced to the process language. Local variables, steps, subworkflows, control flow, data flow, the organizational aspect as well as the operational aspect can be extended and/or overwritten in the subclass by introducing new ones or referring to the name of these constructs in the superclass.

In the following the concept of process inheritance is applied to the R/R example. In this example acknowledgements are added to confirm the receipt of messages or to receive the receipt of a sent message.

```
WORKFLOW_CLASS R/R_ACK
   SUBCLASS_OF R/R
   SUBWORKFLOWS
    S: send_acknowledgment;
    R: receive_acknowledgment;
   END_SUBWORKFLOWS
   CONTROL_FLOW
    cf_1: sequence(send_step, receive_acknowledgment);
    cf_2: sequence(receive_acknowledgment, receive_step);
    cf_3: sequence(receive_step, send_acknowledgment);
   END_CONTROL_FLOW
   WORKFLOW_DATA
   Message: in_acknowledgment;
   Message: out_acknowledgment;
   END_WORKFLOW_DATA
   DATA_FLOW
    df_3: receive_acknowledgment.received_message ->
   in_acknowledgment;
    df_4: out_acknowledgment -> send_acknowledgment.send_message;
   END_DATA_FLOW
   ORGANIZATION
    o_3: send_acknowledgment: automatic;
    o_4: receive_acknowledgment: automatic;
   END_ORGANIZATION
END_WORKFLOW_CLASS
```

Two steps are added to send and to receive the acknowledgements. "cf_1" is overwritten in order to add the "receive_acknowledgment" step after the "send_step". Data and data flow is added to store the acknowledgements and to pass them to the correct steps. Organization definitions are added for the two additional steps indicating their automatic execution.

References

1. Aalst, W. M. P. van der; Basten, T.: *Inheritance of Workflows - An approach to tackling problems related to change*. Computing Science Reports 99/06, Eindhoven University of Technology, Eindhoven, 1999
2. Aalst, W. M. P. van der; Hee, Kees van: *Workflow Management. Models, Methods, and Systems*. The MIT Press, 2002

3. Jablonski, S.; Bussler, C.: *Workflow Management. Concepts, Architecture and Implementation*. International Thomson Publisher, 1995
4. Leymann, F.; Roller, D.: *Production Workflow. Concepts and Techniques*. Prentice Hall PTR, 2000