A Design Methodology for Process-Programming *

Gerald Junkermann and Wilhelm Schäfer

University of Dortmund, Informatik X, D-44221 Dortmund, Germany

Abstract. An examination of existing process programming-languages has shown, that none of them explicitly supports the design of software processes. Most of the process programming-languages are on the same level as "traditional" programming-languages and, therefore, are difficult to understand and unsuitable for explaining the process defined. A promising starting point for an intuitive representation of the process is to apply the concepts of already well known and accepted graphical languages, which have been used successfully for the design of databases and complex reactive systems.

1 Process Design Issues

In [5], Lee Osterweil compares software processes with software and states, that the development of both requires similar disciplines and methods. This quotation let us focus our interest on the design of software and how we can profit from the knowledge available when developing a language for process design.

The result of a "traditional" software design process is a set of modules, each encapsulating an abstract data-type, related by uses-relationships. The encapsulated data-type is represented by the specification of the type and the specification of the methods. The methods are most important, because they specify the semantics of the data-type, and its dynamic behaviour together with the constraints which must hold. Consider for example a data-type person with an attribute street. If the attribute street is used within the methods as a placeholder for the family status, the semantic of this attribute is, to store the family status and not the street, even though the attribute name is street. A constraint on that attribute might be, that the value separated can only be assigned, if the value married has been assigned to this attribute in a former state. The instances to be managed by the encapsulated data-types only appear at runtime and can be ignored on the design level.

This tripartition into types, behavioural description together with constraints, and instances is also true for process programs. The counterpart for the data-types are the object types of a process program, e.g. document types, tool types, roles and relationship types. The behavioural part is represented by the control structures and statements of the process modeling language (e.g. rules in Merlin

* This work has been supported by the Provincial Ministry for Research (MWF) of the state of Northrhine Westphalia
or Marvel, PML-code in IPSE2.5) and instances are only created when executing the software process.

An important difference between software systems and process programs is the frequency of changes. Software systems are designed, implemented and changes may occur in the future. In contrast to this process programs have to be adapted to the needs of every company and can not be specified completely in advance. This results in many changes, e.g. a new document type has to be specified or an existing document type has to be deleted. A further requirement is, that these changes also may occur during the execution of the process (i.e. on the fly), because the work of many people is coordinated by the process program and therefore the system must not be stopped.

Rule-based systems are good candidates to fulfill this requirement. They offer a comprehensive technique for the description of complex process-programs and a high degree of flexibility by offering the possibility to performing changes on the fly, which is only possible by using an interpretive approach.

Process design in contrast to process enactment should however take into account a clear separation of concerns between the three levels of description as mentioned above, namely type definitions, behavioural and constraint definition and the instance level. This makes processes highly understandable and maintainable. If this separation of concerns is supported by using appropriate languages and a precise definition of how, those possibly somewhat different, languages are integrated, then a process design and its corresponding development process no longer follow the "traditional" programming-like approach.

As an example of such a process design approach we sketch the Merlin process design approach which is based on a combination of Extended Entity Relationship-diagrams [1] and statecharts [2].

2 The Merlin Example

In Merlin, the enactable process programming language is a Prolog-like rule-language [4]. An examination of existing Merlin process-programs has shown, that the process programs can be divided into independent parts according to the above mentioned tripartition. One part, describing the dynamic behaviour of the programs (represented by rules) and another two parts, used to handle the objects and their types which the process-program deals with (represented by facts). In more detail this structure is described as follows:

- **Process**: contains those facts used to specify the type of objects and relationships between types, the process-program deals with, and constraints on the dynamic behaviour of value assignment to attributes specifying the state of an object. These constraints are an essential part of the process-program, because they are used to specify the inter-object dependencies, i.e. how a change to other objects may influence the object of interest.
- **Project**: contains those facts which are instances of types introduced in the process-description. There may exist several different project-descriptions for one process, because they represent a concrete project under development.