An Extensible Modeling Language for the Representation of Work Processes in the Chemical and Process Industries

Ri Hai, Manfred Theißen, and Wolfgang Marquardt

AVT – Process Systems Engineering, RWTH Aachen University,
Templergraben 55, 52056 Aachen, Germany

Abstract. Expressive models of the work processes performed in the chemical and process industries provide a basis for diverse applications like work process documentation, analysis, and enactment. In this contribution, we present a generic modeling language for different types of work processes to allow for their integrated representation in the life cycle of a chemical plant. Further, the generic language allows for extensions specific to certain types of work processes. For two important types – design and operational processes – such extensions have been elaborated. These extensions enable the adequate representation of the context of a work process that strongly depends on the process type: for instance, the specification of a chemical plant is a product of a design process, whereas the plant takes the role of a resource during an operational process. This contribution also briefly introduces a modeling tool developed by our group for applying the modeling language in industrial practice.

Keywords: work process, design process, operational process, process modeling language, ontology.

1 Introduction

A *work process* is a collection of interrelated actions in response to an event that achieves a specific result for the customer of the work process. This definition was originally proposed by Sharp and McDermott [1] for *business processes*. However, the term *business process* is ambiguous as there are different definitions in different communities such as in *business process engineering* (e.g., [2]) or in *workflow management* (e.g., [3]). Moreover, the term is typically applied exclusively for work processes which are completely determined and therefore can be planned in a detailed manner. Examples include the processing of a credit transfer in a bank or of a claim in an insurance company. In consequence, many work processes in chemical engineering, in particular design processes, are commonly not considered as business processes. To overcome the ambiguities, we prefer the term *work process* as defined above.

In the life cycle of a chemical plant, several types of work processes are performed, including design processes and operational processes. Whereas design processes aim at the specification of chemical products, production processes, or operating procedures for a plant, operational processes target at establishing, maintaining or preventing certain process conditions during production. Typical examples of operational processes are the start-up of a continuously operated plant or the operation of a multi-purpose batch plant. Our research group has investigated work processes in the chemical industry for more than a decade [4].

An important result of this work is the insight that there is a need for substantial support for engineers and technicians with a background in chemical engineering to create simple and informal models of work processes (e.g., operational procedures for a plant), to enrich these models with further details next and to finally increase their level of formality (e.g., in order to automate an operational procedure). We have performed several case studies focusing on the modeling of both design and operational processes [5,6]. The empirical results of these case studies have motivated the specification of a *modeling framework* comprising

– a modeling procedure as a guide for modelers that need a work process model for a certain application such as documentation, analysis, or (partial) automation of a work process [5];
– a modeling language, i.e., a meta-model providing the modeling elements to represent work processes;
– prototypical modeling tools as a prerequisite for the practical application and validation of our research results in an industrial environment.

The *modeling procedure* proposes several iterations with increasing levels of generality, detail, and formalization, as required by the application. The *level of generality* refers to the number of different work processes covered by a model, whereas the *level of detail* refers to the amount of information captured. Finally, the *level of formalization* refers to the representation of this information (e.g., by means of textual annotations or by using some model elements with well-defined semantics). A more elaborated description of the levels of generality, detail, and formalization has been published elsewhere [8].

The representation of work processes requires an adequate *modeling language*, which is the focus of this contribution. In Sec. 2, we discuss the requirements for such a modeling language. An overview of existing modeling languages as well as their evaluation with respect to industrial requirements is given in Sec. 3. We will see that existing languages partially meet the requirements, but none of them, to our knowledge, satisfactorily fulfills the requirements in a comprehensive manner. The following two sections introduce the *Work Process Modeling Language* (WPML), a language developed by our group based on existing approaches such as the C3 language [5,7,8] and the *activity diagram* of the *Unified Modeling Language* (UML, [9]). Sec. 4 deals with the behavioral aspect of work processes and the modeling elements offered by WPML that go beyond existing languages; these modeling elements are independent of the type of the work process under consideration. Sec. 5 is about WPML's support for the functional aspect of work processes, which is neglected by most existing languages. The function of a work process, and consequently the modeling elements required for representing functional aspects, strongly depend on the process type. For two important types of work processes in chemical engineering, namely design processes and operational processes, functional modeling elements have been elaborated. An overview on a prototypical *modeling tool* is given in Sec. 6.

2 Requirements for a Modeling Language

An adequate modeling language for work processes must fulfill several requirements which result not only from the characteristics of the work processes under consideration, but also from the modeling procedure sketched above. These requirements are discussed in the following.

2.1 Expressiveness

The expressiveness of a modeling language refers to the different aspects of or views on an object—a work process in our case—that can be represented by the language. The focus of this contribution is on behavioral and functional aspects of a work process. Further aspects, such as the actors performing a work process, the technical resources required for their execution (e.g., see [8]), or the complex decisions typically involved in engineering design processes [10,11], are not discussed in this paper.

Behavioral aspects. The focus of most work process models is the representation of process behavior, i.e., the dependencies between the elementary steps—called *actions* in the following—with respect to causal conditions (e.g., *if action A is executed, then action B must also be executed*) and temporal restrictions (e.g., *B is executed after A is finished*). Existing modeling languages often force the modeler to create a *complete* behavior specification from the beginning of the modeling process.

As an example, we consider Aspen Batch Process Developer [12], an engineering software tool for the *simulation* of operational processes in a batch plant. In industrial practice, this simulation tool is also often used as a *modeling* tool to create a first draft of a batch process. At this stage, many details of the batch process are still undefined. For instance, a chemical engineer may want to provide the information that a reactor should be charged with two reactants A and B in a yet undefined order. When creating a model in the tool, the engineer is *forced* to specify a certain order because the underlying modeling language does not allow to omit that information. In consequence, the work process model is likely to be enriched with uncertain or even accidental content, while other feasible alternatives are excluded.

During the design of an operational process, incomplete behavior specifications are useful intermediate steps on the way towards a complete specification, which is required to realize process automation system and ultimately to operate a chemical plant. In case of work process models in engineering design, complete behavior specifications are infeasible in general because design is an inherently creative process, which cannot be predetermined in principle.

In conclusion, a modeling language has to allow the representation of work process models with an incomplete behavior specification. In particular, there is a need for two kinds of abstraction [8]: *Structural abstraction* refers to the freedom to omit information about the conditions under which an action is performed, and *temporal abstraction* refers to the freedom to omit information about the temporal relations between actions *if* they are executed (e.g., by permitting a partial overlap of two subsequent actions).

Functional aspects. According to the definition of the term *work process* in the introduction, the *function* of a work process is to create a specific result. However, most modeling languages focus on the behavioral aspect of a work process and pay little attention to the representation of the function of an entire work process or of the actions within it. Usually, the functional aspect is exclusively specified by means of auxiliary constructs such as textual annotations.

Textual descriptions of functional aspects are advantageous in the early stages of a modeling process because a modeler can phrase a text more easily than choose a specific modeling concept and insert it into a model. There are also model applications in which a functional representation beyond textual descriptions does not provide any additional benefit. An example is the model of a simple operational process for a lab-scale plant, which only serves as a reference for the operator to execute the process.

However, explicit representations of functional aspects enable more advanced applications of work process models, including automatic model checks and model transformations for target applications that strongly depend on functional aspects. Automatic model checks rely on the level of formalization as discussed below. As for model transformations, we consider the example of an operational process that includes the heating of some process material. If the *heating* function is represented by explicit modeling elements rather than by ambiguous textual annotations, a transformation of the model into a simulation model for Aspen Batch Process Developer is possible.[1]

The function of a work process (or of an action) is typically associated with certain objects as prerequisites or outcomes; consequently, a complete modeling of the functional aspect should include these objects (see also [13,14]). To give an example, the specification of a *heating* function for an action is rather incomplete without the specification of the process material to be heated.

2.2 Formalization

A model is *formal* if it's semantics is defined by a mathematical formalism. *Formalization* refers to the transformation of an informal model into a formal one. A trivial prerequisite for a formal work process model is a formal modeling

[1] The tool does not support an abstract 'general' process step, but only concrete types such as *heating* and *cooling*.

language, where the semantics of the modeling elements are defined formally. At the same time, a modeling language should not compel the user to a certain level of formalization. Rather, a set of modeling elements should be provided at a level of formalization that is imposed by the intended application of the model. In particular, it should be possible to represent some aspects formally, but other aspects informally, e.g., by means of simple textual annotations.

Formal representations facilitate advanced applications of work process models like automatic model checks or the derivation of implicit knowledge from explicitly defined knowledge. As an example, we reconsider the modeling of a *heating* function, which could be defined as a modeling element for a *temperature change* resulting in an end temperature higher than the start temperature. Based on a formal representation of this definition, the automatic detection of a semantic error in a model with a *heating* step from 80°C to 60°C is possible. In a similar way, it could be derived that a *temperature change* step from 60°C to 80°C is actually a *heating* step; such derived knowledge would then be available for applications like model transformations.

3 Modeling Languages for Work Processes

There is a plethora of modeling languages for work processes, and a complete review of the existing approaches is beyond the scope of this contribution. Instead, some representative languages are discussed and evaluated with respect to the requirements above. First, *generic* languages are addressed which have been developed for any type of work process (or at least a wide range of types), followed by *specific* languages for operational and design processes.

3.1 Generic Modeling Languages

The *activity diagram* of UML [9] is probably the best-known graphical modeling language for work processes. The language specification comprises typical modeling elements like actions and control flows as well as concepts to represent parallel or alternative branchings of the control flow (i.e., split and decision nodes).[2] The behavioral semantics of UML activities is defined by an informal textual description of the *token flow* in an activity diagram. Hence, the activity diagram can be interpreted as a kind of Petri net, but is not formally defined as such.

[2] In addition to the general action element, more specific subclasses are defined in UML such as *CreateObjectAction* or *WriteVariableAction*. Given that UML is rooted in software engineering, these specific action types are not suitable for the representation of work processes in chemical engineering. However, usage of the special action types in UML is not mandatory, and in practice it is restricted to rather advanced applications like *Model Driven Architecture* [15], which aims at the automated generation of code based on UML models. This justifies the classification of UML activity diagrams as a generic modeling language.

Petri nets are a widely-used family of graphical modeling languages [16,17]. Their strengths include the inherent support of concurrency and resource allocation concepts and the formal mathematical foundation, which eliminates any ambiguity and allows the application of Petri nets for the automated analysis and execution of work processes. The direct application of Petri nets to model complex work processes is often not practical, but they have been applied to specify the semantics of other modeling languages. Störrle and Hausmann [18], for instance, propose formal behavioral semantics for (part of) UML activities by means of a well-defined translation to Petri nets. Furthermore, the semantics of *Yet Another Workflow Language* (YAWL, [19]) is based on a direct mapping of its modeling elements to Petri nets.

The *Process Specification Language* (PSL, [20]) is an ontology-based language for the automatic exchange of information about work processes between different applications and to enhance their interoperability. The semantics of PSL is defined explicitly in first order logics. An outline of a formal specification of UML activities by means of PSL has been published [21] to illustrate its formalization capabilities.

The graphical representation and semantics of the *Business Process Modeling Notation* (BPMN, [22]) are similar to that of UML activity diagrams. Even though its name suggests a focus on business processes, the language is rather generic and allows to represent different types of work processes. In particular, BPMN addresses the creation of process models for workflow execution and automation; for this purpose mappings from BPMN to executable languages like the *Business Process Execution Language for Web Services* [23] or *XML Process Definition Language* [24] have been defined [22,25].

IDEF0 of the *IDEF* family (*Integrated Definition Methods*, [26]) is also widely used for work process modeling, especially in the manufacturing domain. IDEF0 contains a set of generic modeling concepts and allows a "*structured representation of the functions, activities or processes within the modeled system or subject area*" [27]. Temporal relations between actions are not included in IDEF0.

Batres and Naka [28] propose a *process plant model* which can be used to represent the structure of a plant and the processed material, including its physical behavior. Further, the process plant model contains a *management and operation ontology* which provides concepts such as *plan*, *activity*, *action* or *activity-performer* for describing work processes. Although the process plant model is mainly used for representing operational processes, it can be used to represent different kinds of work processes. The process plant model is implemented in a development environment called Ontolingua [29], which enables to define the semantics of modeling concepts explicitly and formally.

3.2 Specific Modeling Languages for Operational Processes

Currently, diverse modeling languages are used to represent operational processes (cf. [30]). These languages can be divided into two groups. The first group contains generic modeling languages such as Petri nets or UML. The focus of this subsection is on the second group, which includes languages developed exclusively or primarily for operational processes.

The IEC 61131-3 standard [31] defines several textual and graphical languages for the automation of operational processes. These languages are considered primarily as programming languages rather than modeling languages.

The VDI/VDE 3682 guideline [32] seeks to facilitate the formal representation of a production process throughout its life cycle. To keep the process representations simple and understandable for engineers from different disciplines as well as for plant operators, the guideline covers only a small set of modeling elements. Further, this guideline also supports the representation of the objects involved in a *process* and their relations by means of UML.

ANSI/ISA-S88 [33] addresses batch process control. The standard defines a layered structure for physical models of plants, process models, and procedural control models. Four types of recipes are defined, including general, site, master, and control recipes. Although the standard itself does not define a modeling language, it provides a useful frame for representing batch process operations.

Beyond these well established standards, current and recent research efforts also deal with the representation of operational processes. For example, on the basis of the ANSI/ISA-S88 standard, Viswanathan *et al.* [34] propose an approach for the synthesis of control recipes. The approach enables to capture the behavioral aspect of an operational process by using Grafchart, a variant of Petri nets. Moreover, domain specific knowledge about chemical plants, chemical processes, and operational processes is included.

Gabbar *et al.* [35] describe a *recipe formal definition language* (RFDL) to support the development of operating procedures of chemical batch plants. RFDL statements are composed of classified keywords, which are linked to domain knowledge, including knowledge about chemical plants, process materials, and operational procedures.

3.3 Specific Modeling Languages for Design Processes

The representation of design processes has been an active research area in the past. Some approaches aim at a framework for analyzing and understanding design processes (e.g., [36,37]). Other approaches focus on detailed models of design artifacts to facilitate design processes, but do not directly address the design process itself [38,39]. Further, there are some guidelines for design processes in the domain of chemical engineering (e.g., [40,41]). In the following, some representative modeling languages for design processes are introduced briefly.

Gorti *et al.* [42] have developed a generic model for representing domain independent design processes. On the one hand, artifacts produced during a design process can be represented under functional, structural, and behavioral aspects. On the other hand, *design processes* can be represented as *process objects* containing process-related components like *goal*, *plan*, or *context*.

The *Knowledge Based Design System* (KBDS, [43]) has been developed to support the design of chemical plants. Based on an exploration-based model of design and the hierarchical decision procedure of Douglas [41], design processes are represented as networks of design objectives, alternatives, and models. Design alternatives of chemical plants can be represented on different levels of detail. Further, design alternatives are related to both the design objectives and simulation models.

The C3 language (*cooperation*, *coordination*, and *communication*) is a simple graphical modeling language with a special focus on the requirements imposed by weakly-structured design processes [5,7,8]. In the interest of high usability, C3 provides a rather restricted number of modeling elements. However, the user is free to extend the language if needed. In addition to conventional modeling concepts for predetermined processes, C3 provides concepts for temporal and structural abstraction.

CLiP [13,44] is an information model providing a conceptualization of the domain of chemical process design. *Process Models* is a partial model of CLiP covering different types of work processes, such as the development of mathematical models or the design of chemical processes [45]. The model comprises a *meta class layer* which provides the concepts required for generalized processes, and a *simple class layer*, which enables the representation of more concrete design processes. Design processes are modeled as iterations of *synthesis*, *analysis*, and *decision* activities, linked by auxiliary activities [46].

3.4 Evaluation

Several of the modeling languages introduced above enable the representation of process behavior, but only a small subset provides a formal foundation (Petri nets, PSL, the languages of IEC 61131-3, and the approach by Viswanathan *et al.*). Only C3 supports temporal and structural abstraction, but lacks a formal definition of these concepts.

The functional aspect of a work process is covered by several approaches [13,34,28,35,43,44]. Interestingly, most of them also provide a detailed model of the objects involved in a work process. The approach proposed by Batres and Naka [28] is the only approach providing a formal specification of the functional aspect.

In short, the existing approaches are too restrictive with respect to process behavior, since they do not support behavioral abstraction. On the other hand, most approaches lack an explicit and formal representation of the functional aspect of work processes.

4 Behavioral Semantics of WPML

The Work Process Modeling Language WPML has been designed to address the weaknesses of the existing modeling languages discussed above. The core of the language provides modeling elements to represent the behavioral aspect of work processes. This part is independent from any application domain; similar as UML acitivy diagrams, it contains generic concepts to represent actions, decisions, concurrency, etc., which are required for any type of work process.

As argued above, functional modeling is required for diverse applications. Expressive functional models are domain-specific (e.g., the function to *heat* some material in a chemical plant). Thus, the domain-independent core of the language does not consider functions beyond the trivial aspect that an action in a work process serves to fulfil one or several functions.[3] Instead, WPML can be extended with domain-specific modules that enable functional models with the expressiveness required by an application. This section focuses on the domain-independant behavioral semantics of WPML. Two examplary domain-specific extensions are discussed in Sec. 5.

Both the WPML meta model (i.e., the domain-independent core and optional domain-specific extensions) and WPML instance models (i.e., the work process models created by a user) are represented in the Web Ontology Language OWL [47]. OWL is a language for knowledge representation with a formal foundation in the domain of description logics [48]. So-called *reasoners* (e.g., Pellet [49]) can be used to exploit the formal definition of the language for applications like consistency checks for work process models or to derive implicit knowledge (which, for instance, substantially reduces the effort to implement the transformation of models into other formats [50]). In the following, we will use a notation similar to that of UML class diagrams to represent parts of WPML although there are important differences between the semantics of OWL and that of UML.

4.1 Basic Elements of WPML

One goal of WPML is to give a formal definition of several behavior-related concepts of the C3 notation for cooperative work processes, which have proven to be valuable in several academic and industrial case studies [5,6], but that suffer from their ambiguous semantics. An informal description of WPML has been published elsewhere ([8], called *Process Ontology* there), including a discussion of its capabilities with respect to temporal and structural abstraction. The following discussion focuses on the formal specification of process behavior. It is restricted to a simplified subset of WPML, which nevertheless demonstrates the basic idea of the chosen approach. A more detailed introduction to WPML can be found in [50]. In addition, a complete specification of the language is in preparation; it will also cover the *roles* by which actions are executed, the *objects* produced or consumed in actions, and the *object flows* that link actions with objects.

In Fig. 1 the basic structure of a WorkProcessModel in WPML is shown. It is a graph that isComposedOf WorkProcessElements, i.e., WorkProcessNodes and WorkProcessArcs. Each arc is from exactly 1 WorkProcessNode to exactly 1 WorkProcessNode[4]. The only subclass of WorkProcessArc discussed here is the ControlFlow. Table 1 lists several concrete subclasses of WorkProcessNode and

[3] To a certain extent, functions could be represented on a domain-independent, but very abstract level (e.g., production, transformation, or consumption of material or information).

[4] The notation used in Fig. 1 and Table 1 (e.g., from exactly 1) is the Manchester OWL Syntax proposed in [51] as a human-readable alternative for OWL class expressions.

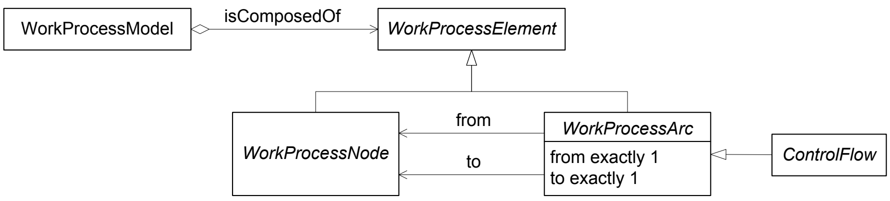

Fig. 1. The basic elements of a WorkProcessModel in WPML

ControlFlow. For the nodes, restrictions for their connection by ControlFlows are given; for instance, an Action must be the origin of exactly one ControlFlow (it must be the from node of exactly 1 ControlFlow), and a final node must not be the origin of a ControlFlow.

4.2 An Introductory Example

Figure 2 shows an example of a simple WorkProcessModel in a graphical notation similar to that of UML activities (the symbols for nodes and arcs are given in Table 1). The model describes the development process for chemical processes in an R&D department on a very coarse-grained level. The first Action a1 is to create a block flow diagram (*BFD*), a schematic representation of a chemical process. In a BFD, blocks or rectangles represent the unit operations or groups of unit operations in the chemical process. Next, more detailed process flow diagrams (*PFD*) must be made for the different parts of the BFD. This can either be done by a contractor or in the same R&D department. For the former case, only a single Action a2 (*Outsource PFD creation*) is given; further details on how the PFDs are created are left to the contractor. For the latter case, two Actions explicitly require that a PFD for the *reaction unit* (a3) and another PFD for the *separation unit* (a4) are created.

In WPML, like in other flow diagrams such as UML activity diagrams, a ControlFlow between two Actions expresses a causal dependency (*if* the first Action is executed, then also the second one must be executed). Conventional languages only include a single control flow, which, in addition to the causal dependency, expresses a temporal dependency (*when* the first Action is finished, the second starts[5]). This is often an unacceptable restriction for models of design or development processes. To this end, there are several types of ControlFlows in WPML, each resulting in different temporal relations between the Actions connected by a flow. In Fig. 2, two different types of control flows are used. The StrictControlFlow is the WPML equivalent of conventional control flows, which require the previous action to be completed before the subsequent action starts; it is drawn as a simple directed arrow. The OverlappingControlFlow allows

[5] It is often ill-defined whether the second Action must start immediately or whether there may be a temporal gap between the Actions.

Table 1. Concrete subclasses of WorkProcessNode (first section) and WorkProcessArc (second section)

Class	Symbol	Restrictions
Action	▭	inv(from) exactly 1 ControlFlow and inv(to) exactly 1 ControlFlow
InitialNode	◉	inv(from) exactly 1 ControlFlow and inv(to) exactly 0 ControlFlow
FinalNode	○	inv(from) exactly 0 ControlFlow and inv(to) exactly 1 ControlFlow
DecisionNode	◇	inv(from) min 2 ControlFlow and inv(to) exactly 1 ControlFlow
MergeNode	◇	inv(from) exactly 1 ControlFlow and inv(to) min 2 ControlFlow
ForkNode	▬	inv(from) min 2 ControlFlow and inv(to) exactly 1 ControlFlow
JoinNode	▬	inv(from) exactly 1 ControlFlow and and inv(to) min 2 ControlFlow
StrictControlFlow	$\downarrow$	
OverlappingControlFlow	$\overset{\sim}{\downarrow}$	

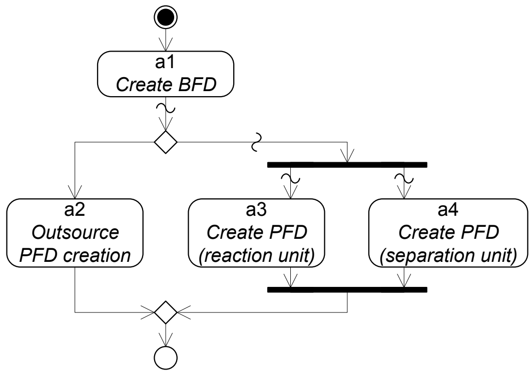

Fig. 2. A simple WorkProcessModel

a subsequent Action to begin while the preceding Action is still executed; it is drawn as a directed arrow decorated with a tilde symbol (˜).

This way, the semantics of WPML separates causality from temporal relations. Causality refers here to the necessary and sufficient conditions for executing an Action. *Causality* is treated similarly in WPML and in UML activity diagrams.

The causal relations between the Actions in the example model are equivalent to those in the UML activity diagram we would get if we replaced the special WPML control flows by the standard UML control flow: *If* the entire work process is executed, then a1 is executed. *If* a1 is executed, then either a2 or both a3 and a4 are executed.

The admissible *temporal relations* are indicated by the different types of control flows in WPML. Above, we have stated that a ControlFlow restricts the temporal relations between Actions connected by the flow. Note that the OverlappingControlFlows in the example do not directly connect Actions, but also control nodes such as the DecisionNode or the ForkNode. However, each OverlappingControlFlow is part of one or several *control flow paths* between Actions. For instance, the OverlappingControlFlow between a1 and the DecisionNode is the start of a path of two control flows from a1 via the DecisionNode to a2; the second control flow is a StrictControlFlow. The temporal restriction between a1 and a2 is defined as the more restrictive one of the two control flows in the path, i.e., the StrictControlFlow. Hence, the model contains the statement that *if* a1 and a2 are executed, i.e., if the PFDs are created by a contractor, then there must be no temporal overlap with the creation of the BFD. Given the obstacles of concurrent engineering across organizational boundaries, this is a reasonable restriction.

We get similar statements for the other control flow paths starting at a1: *If* a1 and a3 are executed, i.e., if the PFD for the reaction unit is created in-house, then there may be a temporal overlap—because the most restrictive control flow is an OverlappingControlFlow. Likewise, *if* a1 and a4 are executed, then there may also be a temporal overlap.

This example demonstrates the capibility of WPML for *temporal* abstraction: Whereas *causal* relations between the Actions are represented similar as in other languages, the *temporal* relations offered by WPML go farly beyond these languages. As for *structural* abstraction, WPML provides a *blob* element, which basically specifies a set of Actions and includes constraints for their occurrences (e.g, *at least 2 and at most 3 of the* Actions *must occur*). For further details on *blobs*, see [7,8].

4.3 Basic Principles of the Formal Specification of WPML

The complete formal specification of the behavioral semantics of WPML is beyond the scope of this contribution. Instead, we describe the basic principles of the specification, which are based on a combination of Petri nets and axioms in first-order logics. As the semantics of the different variants of Petri nets are typically defined in a formal way, it should be possible to create a formal specification of WPML based exclusively on logical axioms[6]. However, this is not the

[6] A WPML semantics based solely on first-order logics would be similar to the approach of the Process Specification Language [20]. It may even be possible to define WPML using PSL, but several concepts of WPML—in particular the idea of overlapping control flows—have no equivalent in PSL and, in our opinion, this task would require substantial extensions or even modifications of several parts of PSL.

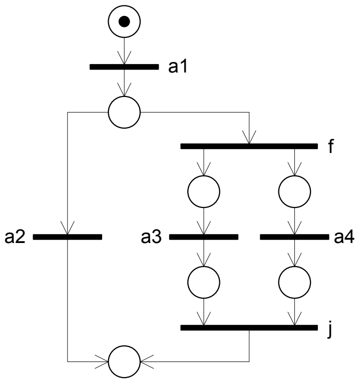

Fig. 3. State-transition net corresponding to the WPML model in Fig. 2

focus of our work; we are mainly interested in an application-oriented way to give a precise definition of WPML.

By an approach similar to that of Störrle and Hausmann [18] for UML activity diagrams, a mapping from WPML models to Petri nets is defined. The Petri net dialect chosen is the colored hierarchical Petri net as defined by Jensen [17]. To simplify the presentation in this overview, we use a simple state-transition net here. The state-transition net corresponding to the WPML model in Fig. 2 is shown in Fig. 3: Each Action is transformed in a transition (denoted by a1, . . . , a4). The DecisionNode has become a place with several outgoing arcs, the MergeNode node a place with several incoming arcs. Finally, the ForkNode and the JoinNode are transformed in auxiliary transitions (f, j). It can easily be verified that the net represents the *causal* dependencies of the Actions as discussed above: The topmost place, representing the StartNode of the WPML model, contains a single token. The only active transition is a1; the firing of the transition represents an execution or *occurrence* of Action a1. Then both transitions a2 and f are active. The case that a2 fires is interpreted as an occurrence of a2. Alternatively, f can fire, which will eventually lead to the firing of both a3 and a4 (in an undefined order).

The semantics of *temporal relations* is not involved in the Petri net translation and, in particular, the order in which transitions in the Petri net fire must not be interpreted as the order in which Actions occur. Instead, an approach based on axioms in first-order logics is used. These axioms impose restrictions on auxiliary objects called WorkProcessOccurrences, which can be interpreted as traces of the transitions fired in the Petri net. Like WorkProcesses, WorkProcessOccurrences are graphs composed of elements like ActionOccurrences and DecisionOccurrences[7]. An ActionOccurrence has two functional properties start

[7] It is essential to distinguish between the modeling elements in WPML, e.g., Action, and the auxiliary objects, e.g., ActionOccurrence. In more complex work processes than in the example discussed here, there may be several occurrences of the same element in the model, for instance several occurrences of a single Action in a loop.

and end, whose range is TimePoint. The concept of TimePoints has been adopted from PSL [20]; here, the existence of a strict total order before_TP on TimePoints is essential. We require that for any ActionOccurrence, its start must be before_TP its end.

In Fig. 4, several examples of WorkProcessOccurrences are given. wpo1 is composed of a StartOccurrence, two ActionOccurrences ao1 and ao2, an EndOccurrence, and three StrictControlFlowOccurrences. To simplify the presentation, the TimePoints are represented as integers, and we assume that the before_TP order is isomorphic to the *smaller-than* relation of their integer representations (i.e., 2 before_TP 4, . . .). Hence, the ActionOccurrences in the figure respect the condition introduced above (start before_TP end).

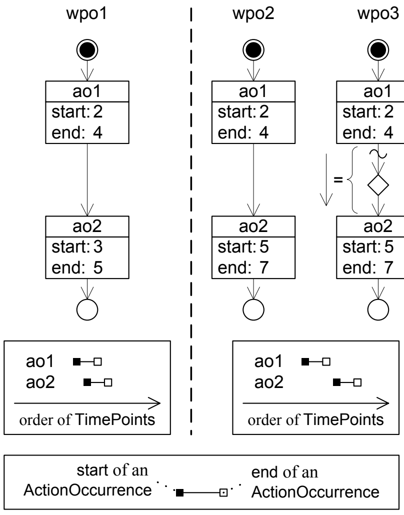

Fig. 4. Graphical depiction of three WorkProcessOccurrences. wpo1 is invalid because neither of the conditions ao1 before ao2 or ao1 meets ao2 holds. wpo2 and wpo3 are valid.

We can easily define thirteen relations between ActionOccurrences in an analogous manner to the time interval relations by Allen [52]. For instance, an ActionOccurrence ao1 meets an ActionOccurrence ao2 iff the end of ao1 and the start of ao2 are equal,

$$\begin{aligned}&\forall ao1, ao2 : \mathsf{ActionOccurrence}(ao1) \wedge \mathsf{ActionOccurrence}(ao2)\\ &\Rightarrow (\, \mathsf{meets}(ao1, ao2) \Leftrightarrow \mathsf{end}(ao1) = \mathsf{start}(ao2)\,) \quad ,\end{aligned}$$

and similarly

$$\begin{aligned}&\forall \mathit{ao1}, \mathit{ao2} : \mathsf{ActionOccurrence}(\mathit{ao1}) \land \mathsf{ActionOccurrence}(\mathit{ao2})\\ &\Rightarrow \; (\,\mathsf{before}(\mathit{ao1}, \mathit{ao2}) \Leftrightarrow \mathsf{before}(\mathsf{end}(\mathit{ao1}), \mathsf{start}(\mathit{ao2}))\,) \quad ,\end{aligned}$$

$$\begin{aligned}&\forall \mathit{ao1}, \mathit{ao2} : \mathsf{ActionOccurrence}(\mathit{ao1}) \land \mathsf{ActionOccurrence}(\mathit{ao2})\\ &\Rightarrow \; (\,\mathsf{overlaps}(\mathit{ao1}, \mathit{ao2}) \Leftrightarrow \;& \mathsf{before}(\mathsf{start}(\mathit{ao1}), \mathsf{start}(\mathit{ao2})) \land\\ && \mathsf{before}(\mathsf{start}(\mathit{ao2}), \mathsf{end}(\mathit{ao1})) \land\\ && \mathsf{before}(\mathsf{end}(\mathit{ao1}), \mathsf{start}(\mathit{ao2}))\,) \quad .\end{aligned}$$

The temporal relations imposed by the different ControlFlows are defined by axioms which restrict the valid time relations between the ActionOccurrences connected by their corresponding ControlFlowOccurrences. In case of a ControlFlowOccurrence of a StrictControlFlow between two ActionOccurrences ao1 and ao2, we require that ao1 must be before ao2 or that ao1 meets ao2, i.e.

$$\begin{aligned}&\forall \mathit{cfo}, \mathit{cf}, \mathit{ao1}, \mathit{ao2} :\\ &\mathsf{StrictControlFlow}(\mathit{cf}) \land \mathsf{ControlFlowOccurrence}(\mathit{cfo})\\ &\land \; \mathsf{occurrenceOf}(\mathit{cfo}, \mathit{cf})\\ &\land \; \mathsf{ActionOccurrence}(\mathit{ao1}) \;\land\; \mathsf{ActionOccurrence}(\mathit{ao2})\\ &\land \; \mathsf{from}(\mathit{cfo}, \mathit{ao1}) \;\land\; \mathsf{to}(\mathit{cfo}, \mathit{ao2})\\ &\Rightarrow \; \mathsf{before}(\mathit{ao1}, \mathit{ao2}) \lor \mathsf{meets}(\mathit{ao1}, \mathit{ao2}) \quad .\end{aligned}$$

In wpo1 (Fig. 4), there is a single ControlFlowOccurrence *cfo* for which the premises of this axiom can be applied, i.e., the ControlFlowOccurrence between ao1 and ao2. Thus, we require that ao1 is before ao2 or ao1 meets ao2. As neither relation holds, the axiom is violated and wpo1 is invalid. In contrast wpo2 is valid, because ao1 before ao2.

The semantics of an OverlappingControlFlow is defined by an axiom analog to that of the StrictControlFlow, but where the time relation overlaps is allowed in addition to before and meets. Hence, wpo4 (Fig. 5) is invalid, because ao3 starts before ao1. wpo5 is valid because all axioms are respected.

So far, the semantics of ControlFlows has only been defined if their occurrences are from and to an ActionOccurrence. Other possibilities are reduced to this case. As an example, consider wpo3 in Fig. 4, which contains a path of two ControlFlowOccurrences from ao1 via a DecisionOccurrence to ao2. By definition, this path is equivalent to the most restrictive single ControlFlowOccurrence, i.e., an occurrence of a StrictControlFlowOccurrence in this example. Hence, wpo3 is equivalent to wpo2 and thus valid. In a similar way, wpo6 in Fig. 5 is equivalent to wpo5 and also valid. In general, the *most restrictive* one of a set of ControlFlows is the ControwFlow that allows *all* time relations allowed by each single ControwFlow, and *no further* time relations.

Note that wpo3 and wpo6 directly correspond to the example model in Fig. 2. wpo3 represents the case that a1 and a2 are executed, and it gives some valid (but still arbitrary) values for the start and end of the ActionOccurrences. wpo6

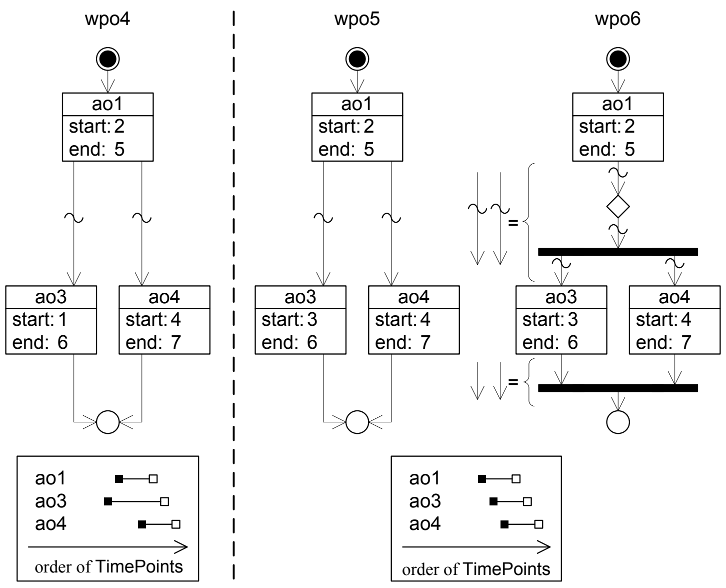

Fig. 5. Graphical depiction of three WorkProcessOccurrences. wpo4 is invalid because neither of the conditions ao1 overlaps ao3, ao1 before ao3, or ao1 meets ao3 holds. wpo5 and wpo6 are valid.

represents the case that a1, a3, and a4 are executed. The structure of both wpo3 and wpo6, excluding the time values, can be derived by means of the Petri net in Fig. 3[8]. Also, if we neglect the time values, wpo3 and wpo6 are the *only* WorkProcessOccurrences which can be generated by the Petri net.

The overall semantics of the example model is that it represents *all* concrete work processes described by wpo3 or wpo6 with arbitrary time values, provided the constraints induced by the axioms are respected.

5 Functional Modeling in WPML

In this section, we address the functional aspect of a work process. Each action in a work process typically serves a *function* which reflects the required input and the expected results. In a purely behavioral WPML model as discussed in the previous section, Actions produce or consume Objects[9] (see Fig. 6). To cover the functional aspect of a work process, Actions can be linked to the Functions

[8] The necessity to keep track of the transitions each token passes is one reason to use a more expressive Petri net variant than state-transition nets.

[9] The hasInputObject and hasOutputObject properties are simplifications to emphasize the analogy between the concepts on the behavioral side and those on the functional side. Actually, Actions and Objects are linked by ObjectFlows similar to the ControlFlows discussed in Sec. 4.

they fulfill, and Objects can be linked to States. Sections 5.1 and 5.2 deal with functional models of two different types work work processes. Finally, Sect. 5.3 elaborates the relation between behavioral and functional aspects in more detail.

In general, the functions that can appear in a work process model depend on the process type. For instance, within an operational process, the function of an action may be to change a reactor temperature to 80°C. To describe this function, concepts such as Reactor and SetPoint are required. Thus, subclasses of Function for different process types are introduced, such as the OperationalProcessStep for operational processes and the DesignStep for design processes. Fig. 6 shows typical function classes for both types of work processes.

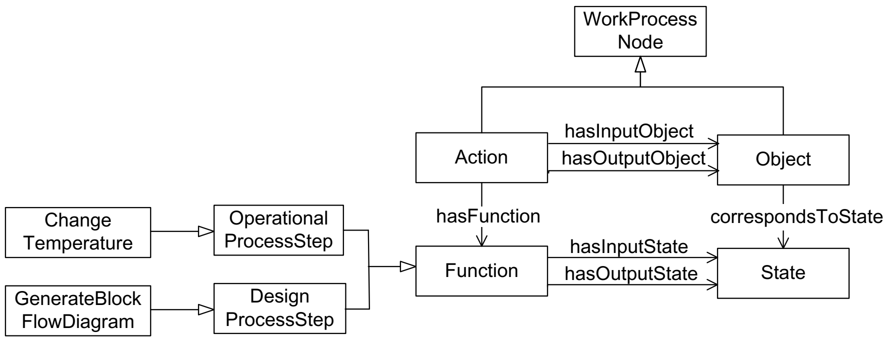

Fig. 6. Actions can be specified by Functions that serve to change an input State into an output State. The diagram shows exemplary function classes for design and operational processes.

The definition of adequate subclasses of Function allows to extend WPML with domain knowledge. As discussed in Sec. 2, such domain knowledge covers the function of work processes and the objects involved. This knowledge is represented as part of *OntoCAPE*, a comprehensive ontology for the CAPE (*computer-aided process engineering*) domain [53]. According to Gruber, an ontology is an explicit specification of a conceptualization, while a conceptualization can be seen as an abstract, simplified view of the world for a specific purpose [54]. By means of an ontology, the semantics of the modeling concepts is explicitly and formally specified.

Originally, OntoCAPE has been developed by our group to describe the artifacts created during the conceptual design of chemical processes. However, the extensible structure of the ontology allows to integrate concepts related to work processes seamlessly. That way, many concepts of OntoCAPE can be used to enrich the representation of the context of work processes. OntoCAPE is described comprehensively elsewhere [53,55,56]. In the following, we focus on the extensions developed for representing operational processes and design processes. These extensions are implemented as *partial models* of OntoCAPE.

5.1 Extensions for Operational Processes

During an operational process, a number of operational steps are conducted in different plant items. These operational steps support the chemical processes to achieve the desired chemical products. Accordingly, when modeling operational processes, modeling concepts must cover the operational steps (e.g. OpenValve or SetControllerParameter), the involved plant items (e.g. Instrumentation, Vessel or DistillationSystem), the production steps of chemical processes (e.g. Reaction or Distillation), as well as the materials processed or produced by the chemical process.

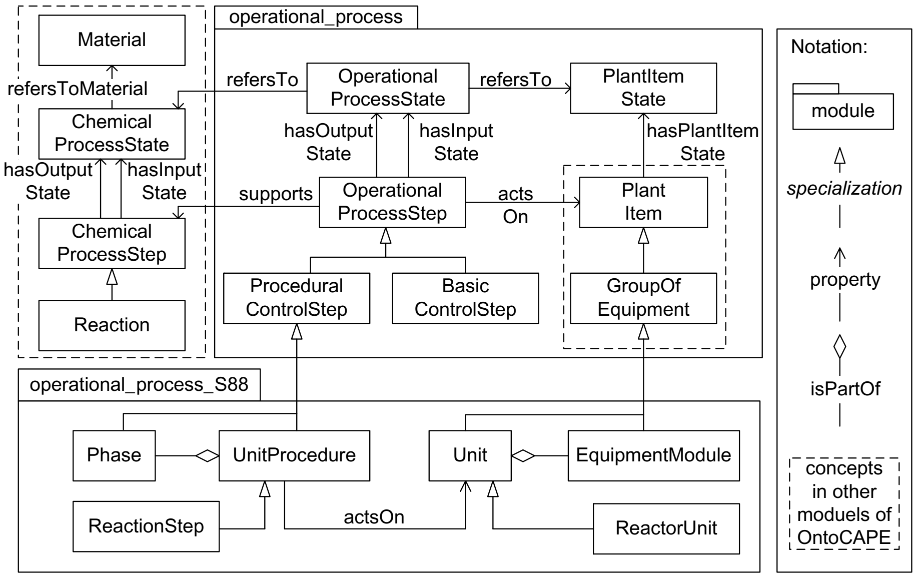

Fig. 7. The simplified partial model for operational processes

Moreover, concepts for a *structured* representation of chemical processes, operational processes, and plant items must be incorporated in order to permit descriptions on different levels of detail and granularity. In industrial practice, there are various ways to specify the hierarchies of the aforementioned entities. For instance, a very generic way to enable the structured representation of plant items is to specify a PlantItem as a PieceOfEquipment or a GroupOfEquipment. A more restricted alternative may involve concepts like ProcessCell, Unit, or SingleEquipment. Bearing in mind that various hierarchies may be needed when modeling operational processes, we propose an extensible modular structure for the partial models. The partial model for operational processes consists of the two modules shown in Fig. 7. While the module operational_process contains concepts for representing operational processes independent from any applications or standards, the module operational_process_S88 covers the concepts for describing operational process according to the well established ANSI/ISA-S88 standard [33]. In the following, the modules involved in this partial model are explained in detail.

The module operational_process. This module is simple, yet contains essential concepts for describing operational processes. The most important concepts within this module are OperationalProcessStep and OperationalProcessState. An OperationalProcessStep acts on PlantItems and supports a ChemicalProcessStep. Intentionally, only a very generic specification for OperationalProcessStep is included in this module: an OperationalProcessStep can be a BasicControlStep or a ProceduralControlStep. A BasicControlStep is a simple elementary step such as to open a valve, to set the parameters for a controller, or to switch on a pump. A ProceduralControlStep, on the contrary, consists of several operational steps. For instance, a charging step may contain a set of basic operational steps such as to open a block valve or to start a pump. An OperationalProcessStep changes the OperationalProcessState which captures all relevant states during an operational process, including

– PlantItemStates such as the position of a valve or the level of a tank,
– ChemicalProcessStates such as the temperature or pressure of the processed material in a reactor, and
– ExecutionStates of an operational process step like *running*, *aborted* or *complete* (not shown in Fig. 7).

The module operational_process_S88. Because of the generic character of the operational_process module, additional modules for the further specification of operational processes based on certain standards or guidelines can be included with minor effort. For illustration, we choose the ANSI/ISA-S88 standard as an example for further specification of the operational_process module. This standard provides a solid foundation for a hierarchical representation of plant operational processes and plant items. The structure defined in ANSI/ISA-S88 is partially adopted in this work (cf. Fig. 7). The module operational_process_S88 enables to represent plant items on four levels—ProcessCell, Unit, EquipmentModule, and ControlModule—to refine the modules incorporated in OntoCAPE. Furthermore, the concept ProceduralControlStep is specified into Procedure, UnitProcedure, Operation, and Phase. Relations between these concepts are also defined. For instance, a UnitProcedure can only take place on a Unit.

The modules operational_process and operational_process_S88 provide sufficient expressiveness to describe operational processes on different levels of granularity and detail. For instance, the module operational_process defines that an OperationalProcessStep acts on PlantItems and that an OperationalProcessStep supports a ChemicalProcessStep. This can be further specified as, for instance, a ReactionStep supports a chemical process step of type Reaction carried out in a ReactorUnit. By decomposing the ReactionStep into some Phases, more details of the ReactionStep can be captured. Another possibility for the structured representation of plant items and operational processes has been reported in [57].

When necessary, this can be defined in an additional module to extend the operational_process module. When modeling an operational process, one of the extensions can be applied.

An adequate representation of operational processes can be used not only to improve the quality of plant operations, but also to support the design of the operational specifications, as shown in the next section. For a more comprehensive description of the partial model for operational processes as well as its application, see [58].

5.2 Extensions for Design Processes

Currently, both conceptual design processes and the design of operational process specifications are considered the design_process module. For representing the conceptual design process, we have adopted and extended the Douglas methodology [40]. The Douglas methodology uses a hierarchical approach where design decisions are taken on a set of predefined levels of abstraction. We have applied the Douglas methodology to represent conceptual design processes hierarchically. In a similar way, modeling concepts for design processes of operational specifications have been developed based on the hierarchy defined in the ANSI/ISA-S88 standard.

As typical objects involved in a design process, documents contain information about a chemical process or parts of the process. Representing the *content* of the involved documents is crucial for describing design steps with sufficient detail, because only then it is possible to describe the required information for carrying out a design step and the outcome of that design step. Using concepts defined in OntoCAPE, a chemical process system can be described under different aspects from the abstract process level to the concrete plant item level. Such a detailed model of domain objects provides an essential base for representing design steps on different levels of abstraction.

As shown in Fig. 8, the partial model for representing design processes contains several modules. Whereas the module design_process contains concepts for modeling design steps, the module document_model can be used to describe the documents created or modified by a design step. A number of further modules may be involved to represent the content of the documents. Two of these modules, chemical_process_Douglas and operational_process_S88 are shown in Fig. 8. While the module operational_process_S88 has been already introduced in Sec. 5.1, we discuss the other three modules in Fig. 8 in the following.

The module document_model. This module plays a crucial role to relate the design steps to the domain knowledge about objects within the domain of chemical engineering. On the topmost level, a Document is required, produced or modified as an input or an output of a DesignStep. The content of a Document can be described by concepts representing domain objects. As a further specification, a set of documents in chemical engineering, represented by the concept ChemEngDoc and its subclasses, has been identified and related to the corresponding content. For instance, a BlockFlowDiagram hasContent about ReactionSystems, an

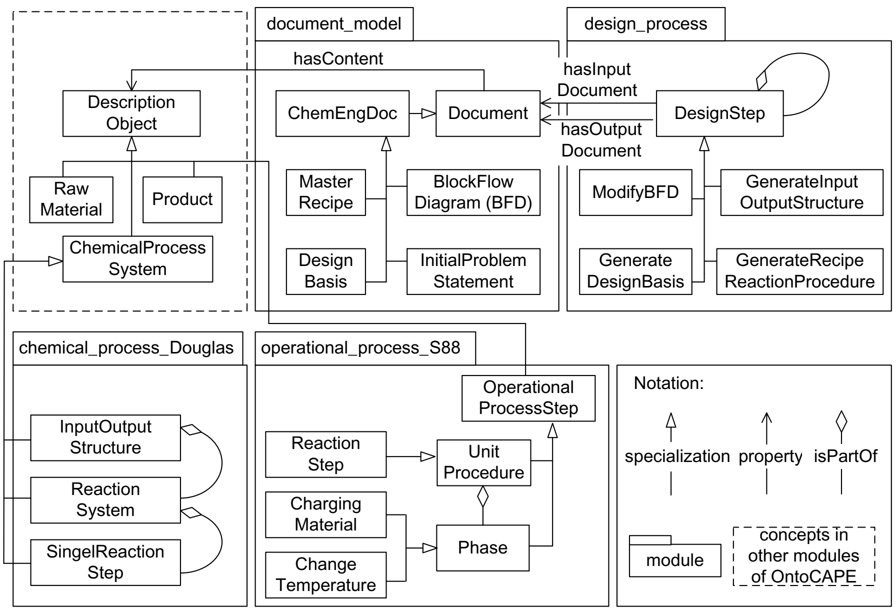

Fig. 8. The simplified partial model for representing design processes

InitialProblemStatement hasContent about RawMaterials and Products, or a MasterRecipe hasContent about a ReactionStep. The document_model module also captures a further aspect dealing with the management of document versions. More details about the document_model can be found in [59].

The module chemical_process_Douglas. This module includes a hierarchy based on the Douglas method to represent chemical process systems on different levels of abstraction. A chemical process system can be described on the following levels:

- The *process level* corresponds to the input-output structure in the Douglas hierarchy. On this level, only input and output materials are considered. Also recycle streams and purge streams can be investigated. Typical documents involved on this level are InitialProblemStatement and DesignProcessBasis.
- The *process system level* corresponds to the recycle structure of the Douglas hierarchy. On this level, reaction systems or separation systems are considered. Separation systems can be further classified as vapor separation systems, liquid separation systems, and flashes. Reaction system can be further specified into single reaction steps. Typical documents involved on this level are DesignProcessBasis and BlockFlowDiagram.
- The *standard unit level* is not included in the Douglas hierarchy, but is useful to describe standard unit operations and single reaction steps. Typical documents involved on this level are BlockFlowDiagram and ProcessFlowDiagram.

The module design_process. The topmost concept in this module is DesignStep which is further specified by typical design activities like *generate*, *modify* or *refine*, combined with the associated design product such as a block flow diagram or a master recipe. ModifyBlockFlowDiagram and GenerateMasterRecipe, for instance, are two specific DesignSteps. Representing design steps by combining design activities and the corresponding design products is a straightforward and intuitive way which helps to identify design steps in terms of the related design content. We intentionally avoid using typical terminology from the design research area like syntheses, analyze, induction, abduction etc. There are two main reasons for doing so. Firstly, design activities in industry are usually described by referring to the target product. For instance, *generate a flow sheet* is a typical design activity. Secondly, each design activity can be seen as an aggregate activity containing a synthesis step, an analysis step, and a decision step (cf. [46]). Hence, *synthesis*, *analysis*, and *decision* may be used as elementary building blocks for more expressive concepts, but are not sufficient for representing design processes.

For each DesignStep the required input and output documents are defined, both with the corresponding content. A DesignStep can be further decomposed into other DesignSteps. For the conceptual design process, design steps are introduced according to the levels defined in the module chemical_process_Douglas and the involved documents in chemical engineering. A similar approach is used for the design process of operation specifications. The modules operational_process_S88 is used to describe the content of the design steps in a plant operation design process. For illustration, two simple examples are given in the following.

Example 1: A design step during plant operation specification. A design step of type GenerateRecipeReactionStep requires a certain MasterRecipe (see Fig. 9). After this design step, the second version of this MasterRecipe is created. The second version contains information about a ReactionStep for the reactor R3. This ReactionStep contains two Phases: Charging reactant 1 of type ChargingMaterial and Heating of type ChangeTemperature.

Example 2: A design step in conceptual design. To develop a chemical process for producing benzene by hydrodealkylation (HDA) of toluene, a number of design steps are to be carried out (see Fig. 10). This example describes a design step of type GenerateInputOutputStructure (IOS) which requires a document of type InitialProblemStatement. This document contains a statement of the involved raw materials (i.e., Toluene and Hydrogen) and the desired products (i.e., Benzene and Methane). In addition, information about the ChemicalReaction of interest (i.e. the hydrodealkylation of Toluene) is also given in the InitialProblemStatement. The result of this design step is a report of type DesignBasis, which has content about the InputOutputStructure of this HDA process.

These two simple examples show the strength and necessity to incorporate detailed models of domains objects such as ChemicalReaction or RawMaterial when modeling design processes. Knowledge about both, the design processes

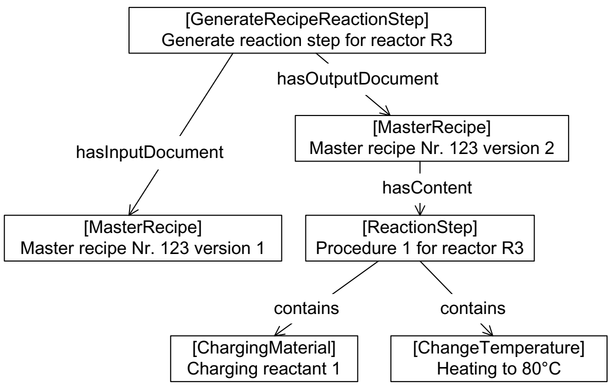

Fig. 9. Example of a design step for plant operation on the instance level

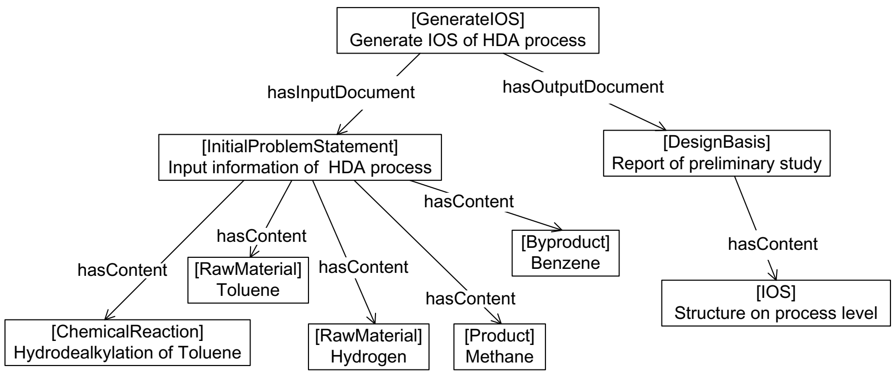

Fig. 10. Example of a conceptual design step on the instance level

and the involved objects, must be encompassed from the coarse-grained level to the fine-grained level to represent design steps with sufficient information.

5.3 Integrating Process Behavior and Function

So far, we have discussed the behavioral and the functional aspect of work processes independent from each other. This section is meant to illustrate the integration of the two aspects. As a simple (and incomplete) example, consider the work process model in Fig. 11. The left part of the figure shows part of a design process for HDA in the concrete syntax of WPML that has been adopted from UML activity diagrams. In the right part, the same design process is represented as an object diagram. The two Actions of the process are linked by a ControlFlow that captures the behavioral aspect of the process. Also, to each Action, an instance of a concrete Function class is assigned to represent the functional aspect.

Note that in the concrete syntax, it is sufficient to display the name of this function class. As for the functional aspect, it could be further specified as shown in the examples given in Sec. 5.2.

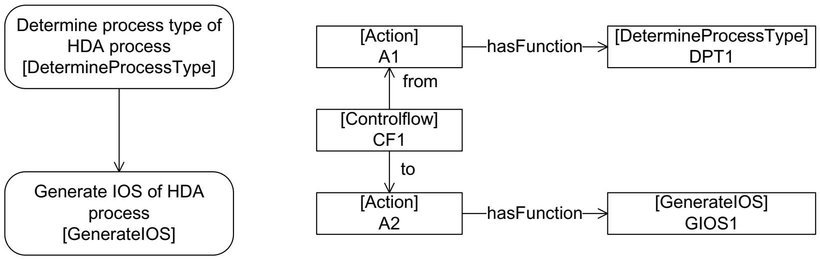
Determine process type of HDA process [DetermineProcessType]
Generate IOS of HDA process [GenerateIOS]
[Action] A1
hasFunction
[DetermineProcessType] DPT1
from
[Controlflow] CF1
to
[Action] A2
hasFunction
[GenerateIOS] GIOS1

Fig. 11. Example of a conceptual design process on the instance level. Both behavioral and functional aspects are captured.

6 Implementation and Tool Support

Both WPML and its extensions are implemented in OWL [47]. The formal specification of process behavior is additionally supported by Petri nets and axioms in first-order logics. WPML and the extensions can be seen as meta-models which provide the building blocks for work process models on the instance layer. We use the OWL editor Protégé [60] to implement the meta-models.

For the creation of work processes on the instance layer in a user-friendly manner, our research group has developed the Work Process Modeling System (WOMS+). WOMS+ provides a set of intuitive symbols for the WPML modeling concepts. In addition, meta-models for different types of work processes (e.g., the module for operational processes) can be imported into WOMS+, such that the classes and properties defined in these models are offered to the user, who can select among them for the formal specification of the elements in a WPML instance model. Moreover, further use of the models created by means of WOMS+ is also supported by an automated export into certain target formats of applications like the Aspen Batch Process Developer or a Petri net simulators.

7 Conclusions and Open Issues

This contribution motivates and presents the WPML language, a generic modeling language suitable for different types of work processes, which allows for further extensions specific to certain types of work processes. Extensions for design and operational processes have been discussed.

The WPML language supports the formal representation of both, the behavioral and functional aspects of a work process. Domain knowledge can explicitly be incorporated on different levels of abstraction. Not only the work process itself, but also the objects involved are covered with sufficient details. Models created with this language can easily be translated in different format such that *one* model can be used by different applications. This strategy supports the reuse of a work process model which captures the knowledge of an organization and thus increases engineering productivity.

Currently, cooperation with several industrial partners is in progress to achieve a practical evaluation of the modeling framework. These empirical studies aim at verifying the usability of the tool as well as the usability and expressiveness of WPML.

Acknowledgments. The authors acknowledge financial support from the German National Science Foundation (DFG) under grant MA 1188/29-1.

References

1. Sharp, A., McDermott, P.: Workflow Modeling: Tools for Process Improvement and Application. Artech House, Norwood (2001)
2. Davenport, T.H.: Process Innovation. Harvard Business School, Boston (1993)
3. Workflow Management Coalition (WfMC): Terminology & Glossary, Report No. WFMC-TC-1011 (1999), `http://www.wfmc.org`
4. Nagl, M., Marquardt, W.: Collaborative and Distributed Chemical Engineering: from Understanding to Substantial Design Process Support; Results of the IMPROVE Project. LNCS, vol. 4970. Springer, Heidelberg (2008)
5. Theißen, M., Hai, R., Marquardt, W.: Design Process Modeling in Chemical Engineering. J. Comput. Inf. Sci. Eng. 8(1), 011007 (9 pages) (2008)
6. Theißen, M., Hai, R., Morbach, J., Schneider, R., Marquardt, W.: Scenario-based Analysis of Industrial Work Processes. In: Nagl, M., Marquardt, W. (eds.) Collaborative and Distributed Chemical Engineering. LNCS, vol. 4970, pp. 433–450. Springer, Heidelberg (2008)
7. Killich, S., Luczak, H., Schlick, C., Weienbach, M., Wiedenmaier, S., Ziegler, J.: Task Modelling for Cooperative Work. BIT 18(5), 325–338 (1999)
8. Eggersmann, M., Kausch, B., Luczak, H., Marquardt, W., Schlick, C., Schneider, N., Schneider, R., Theißen, M.: Work Process Models. In: Nagl, M., Marquardt, W. (eds.) Collaborative and Distributed Chemical Engineering. LNCS, vol. 4970, pp. 126–152. Springer, Heidelberg (2008)
9. Object Management Group (OMG): Unified Modeling Language, Version 2.0 (2005), `http://www.omg.org/uml`
10. Theißen, M., Marquardt, W.: Decision Process Modeling in Chemical Engineering Design. In: 17th European Symposium on Computer Aided Process Engineering, pp. 383–388 (2007)
11. Marquardt, W., Theißen, M.: Integrated Modeling of Work Processes and Decisions in Chemical Engineering. In: Schlick, C.M. (ed.) Industrial Engineering and Ergonomics, pp. 265–279. Springer, Heidelberg (2009)
12. Aspen Technology, `http://www.aspentech.com/products/aspen-batch-plus.cfm`
13. Bayer, B., Marquardt, W.: Towards Integrated Information Models for Data and Documents. Comput. Chem. Eng. 28, 1249–1266 (2004)
14. Brandt, S., Morbach, J., Miatidis, M., Theißen, M., Jarke, M., Marquardt, W.: An Ontology-Based Approach to Knowledge Management in Design Processes. Comput. Chem. Eng. 32(1-2), 320–342 (2008)

15. Object Management Group (OMG): Model Driven Architecture, http://www.omg.org/mda
16. Petri, C.A.: Kommunikation mit Automaten. Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik (1962)
17. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Basic Concepts, vol. 1, 2nd corrected printing. Springer, Berlin (1997)
18. Störrle, H., Hausmann, J.H.: Towards a Formal Semantics of UML 2.0 Activities. In: Liggesmeyer, P., Pohl, K., Goedicke, M. (eds.) Software Engineering. LNI, vol. 64, pp. 117–128. Gesellschaft für Informatik (2005)
19. van der Alst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Inform. Syst. 30(4), 245–275 (2005)
20. National Institute of Standards and Technology (NIST): The Process Specification Language (PSL): Overview and Version 1.0 Specification, Report No. NISTIR 6459. Gaithersburg, MD (2000)
21. Bock, C., Gruninger, M.: PSL: A Semantic Domain for Flow Models. SoSyM. 4, 209–231 (2005)
22. Object Management Group (OMG): Business Process Modeling Notation (BPMN) (2006), http://www.omg.org
23. BEA Systems, IBM, Microsoft, SAP AG, Siebel Systems: Business Process Execution Language for Web Services, version 1.1 (2003), http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
24. Workflow Management Coalition (WfMC): XPDL 2.0 Specification (2005), http://www.wfmc.org
25. White, S.A.: XPDL and BPMN. In: Workflow Handbook 2003, pp. 221–238. Future Strategies Inc., Lighthouse Point (2003)
26. Mayer, R.J., Painter, C.M.K., de Witte, P.S.: IDEF Family of Methods for Concurrent Engineering and Business Re-Engineering Applications, http://www.idef.com/
27. National Institute of Standards and Technology (NIST): Integrated Definition for Function Modeling (IDEF0), Report No. NISTIR 183. Gaithersburg, MD (1993)
28. Batres, R., Naka, Y.: Process Plant Ontologies Based on a Multi-dimensional Framework. In: Malone, M.F., Trainham, J.A., Carnahan, B. (eds.) Proceedings of the Fifth International Conference on Foundations of Computer-Aided Process Design, pp. 433–437 (2000)
29. Knowledge Systems Laboratory: Ontolingua, http://www.ksl.stanford.edu/software/ontolingua
30. VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik: Classification and Evaluation of Description Methods in Automation and Control Technology, VDI/VDE 3681 (2005)
31. International Electrotechnical Commission (IEC): Programmable controllers – Part 3: Programming languages, IEC 61131-3 Ed. 2.0 (2003)
32. VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik: Formalised Process Descriptions, VDI/VDE 3682 (2005)
33. International Society of Automation (ISA): Batch control Part 1: Models and terminology, ANSI/ISA-S88.01-1995 (1995)
34. Viswanathan, S., Johnsson, C., Srinivasan, R., Venkatasubramanian, V., Ärzen, K.E.: Automating Operating Procedure Synthesis for Batch Processes: Part I. Knowledge Representation and Planning Framework. Comput. Chem. Eng. 22(11), 1673–1685 (1998)
35. Gabbar, H.A., Aoyama, A., Naka, Y.: Recipe Formal Definition Language for Operating Procedures Synthesis. Comput. Chem. Eng. 28(9), 1809–1822 (2004)

36. Chandrasekaran, B.: Design Problem Solving: A Task Analysis. AI Mag. 11(4), 59–71 (1990)
37. Gero, J.S., Kannengiesser, U.: A Function-Behavior-Structure Ontology of Processes. AI EDAM 21(4), 379–391 (2007)
38. Stone, R.B., Wood, K.L.: Development of a Functional Basis for Design. J. Mech. Des. 122(4), 359–370 (2000)
39. Kitamura, Y., Riichiro, M.: Ontology-Based Systematization of Functional Knowledge. J. Eng. Des. 15(4), 327–351 (2004)
40. Douglas, J.M.: A Hierarchical Decision Procedure for Process Synthesis. AICHE J. 31, 353–362 (1985)
41. Douglas, J.M.: Conceptual Design of Chemical Processes. McGraw-Hill, New York (1988)
42. Gorti, S.R., Gupta, A., Kim, G.J., Sriram, R.D., Wong, A.: An Object-Oriented Representation for Product and Design Processes. Comput.-Aided Des. 30(7), 489–501 (1998)
43. Bañares-Alcántara, R.: Design Support Systems for Process Engineering – I. Requirements and Proposed Solutions for a Design Process Representation. Comput. Chem. Eng. 19(3), 267–277 (1995)
44. Bayer, B.: Conceptual Information Modeling for Computer Aided Support of Chemical Process Design. Fortschritt-Berichte VDI: Reihe, vol. 3(787). VDI-Verlag, Düsseldorf (2003)
45. Eggersmann, M.: Analysis and Support of Work Processes within Chemical Engineering Design Processes. Fortschritt-Berichte VDI: Reihe, vol. 3(840). VDI-Verlag, Düsseldorf (2004)
46. Eggersmann, M., Gonnet, S., Henning, G., Krobb, C., Leone, H., Marquardt, W.: Modeling and Understanding Different Types of Process Design Activities. Lat. Am. Appl. Res. 33, 167–175 (2003)
47. World Wide Web Consortium: OWL Web Ontology Language. Reference. Recommendation (2004), `http://www.w3.org/TR/owl-ref/`
48. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
49. Clark & Parsia.: Pellet OWL Reasoner (2010), `http://clarkparsia.com/pellet/`
50. Theißen, M., Hai, R., Marquardt, W.: A Framework for Work Process Modeling in the Chemical Industries. Submitted to Comput. Chem. Eng. (2009)
51. Horridge, M., Patel-Schneider, P.F.: Manchester Syntax for OWL 1.1. In: Clark, K., Patel-Schneider, P.F. (eds.) OWL: Experiences and Directions 2008 DC, Fourth International Workshop, Washington, DC (2008)
52. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. Comm. ACM 26(11), 832–843 (2009)
53. Marquardt, W., Morbach, J., Wiesner, A., Yang, A.D.: OntoCAPE - A Re-Usable Ontology for Chemical Process Engineering. Springer, Berlin (2010)
54. Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowl. Acquis. 5(2), 199–220 (1993)
55. Morbach, J., Yang, A.D., Marquardt, W.: OntoCAPE - A Large Scale Ontology for Chemical Process Engineering. Eng. Appl. Artif. Intel. 20(2), 147–161 (2007)
56. Morbach, J., Wiesner, A., Marquardt, W.: OntoCAPE 2.0 - A (Re)usable Ontology for Computer-Aided Process Engineering. Comput. Chem. Eng. 33, 1546–1556 (2009)
57. Aoyama, A., Yamadai, I., Batres, R., Naka, Y.: Multi-Dimensional Object Oriented Approach for Automatic Generation of Control Recipes. Comput. Chem. Eng. 24(2-7), 519–524 (2000)

58. Hai, R., Theißen, M., Marquardt, W.: An Ontology Based Approach for Operational Process Modeling. Submitted to Advanced Engineering Informatics (2010)
59. Morbach, J., Hai, R., Bayer, B., Marquardt, W.: Document models. In: Nagl, M., Marquardt, W. (eds.) Collaborative and Distributed Chemical Engineering. LNCS, vol. 4970, pp. 111–125. Springer, Heidelberg (2008)
60. Stanford Center for Biomedical Informatics Research: The Protégé ontology editor and knowledge acquisition system, `http://protege.stanford.edu`