Modeling Actions in Dynamic Engineering Design Processes

Vadim Ermolayev[1], Natalya Keberle[1],
Eyck Jentzsch[2], Richard Sohnius[2], and Wolf-Ekkehard Matzke[2]

[1] Department of IT, Zaporozhye National University, Zhukovskogo 66, 69063, Zaporozhye, Ukraine
vadim@ermolayev.com, nkeberle@gmail.com
[2] Cadence Design Systems, GmbH, Mozartstr. 2 D-85622 Feldkirchen, Germany
{jentzsch,rsohnius,wolf}@cadence.com

Abstract. The paper presents the approach for modeling actions in the dynamic processes of engineering design in microelectronics and integrated circuits domain. It elaborates the formal framework for representing processes, the states of these processes and process environments, the actions being the constituents of the processes. Presented framework is implemented as the part of PSI suite of ontologies and is evaluated using three different methods: user evaluation, formal evaluation, and commonsense evaluation following PSI shaker modeling methodology. The suite of PSI ontologies is used for representing dynamic engineering design processes in Cadence Project Planning Expert System software prototype.

Keywords: PSI, action, task, activity, environment, design system, performance, framework, ontology.

1 Introduction

As many experts in microelectronic and integrated circuits design point out (e.g., [1]), one of the main industrial challenges is the gap between the capability of design technology and the productivity of design systems. For example, the capability of the design technology to accommodate digital gates on a chip is growing much faster than the capability of design teams using this technology and corresponding design environments to produce these gates in their designs. The consequence is that the effort required to be spent for designing a typical microelectronic device is growing substantially. Therefore, tools and methodologies for improving the performance of design systems are very highly demanded by industry.

PSI project[1] aims at developing models, methodologies, and software tools providing for rigorous engineering treatment of performance and performance management. PSI performance modeling and management approach focuses on performance as a

[1] Performance Simulation Initiative (PSI) is the R&D project of Cadence Design Systems GmbH.

J. Yang et al. (Eds.): UNISCON 2009, LNBIP 20, pp. 127–141, 2009.

pro-active action. A fine-grained dynamic model of an engineering design process, comprising a semantically rich action model, and a design system is therefore developed. PSI approach considers that performance is embodied in its environment and is controlled by the associated performance management process.

An engineering design process is a goal-directed process of transforming the representations of a design artifact in stateful nested environments. An environment comprises design artifact representations, resources, tools, and actors who perform actions to transform design artifacts using tools, and consume resources. Actions are admissible in particular environment states and may be atomic or compound, state-transitive or iterative, dependent or independent on other actions. The components of an environment may generate internal events or may be influenced by external events. Events may have causal dependencies. An engineering design process is a problem solving process which goals, partial goals, and environments may change dynamically. A decision taking procedure is associated with each state to allow environments to adjust the process taking these changes into account. Decisions are taken by actors modeled by software agents.

PSI software tools are developed [2] for assisting project managers to make robust planning, monitoring, and management of their design projects aiming at reaching best possible performance. Grounded decisions in planning are based on the knowledge base of project logs accomplished in the past. These logs provide vast and finely grained records of the performance of accomplished projects and may be used for simulating the behavior of the design system in response to different influences. At project execution phase PSI software may be used for predicting the behavior of the design system in the future based on the record of the partially accomplished dynamic engineering design process (DEDP), the knowledge about its environment(s), and performance simulations.

The focus of this paper is the framework for modeling actions. The rest of the paper is structured as follows. Section 2 analyses the related work in process modeling emphasizing the ways to model dynamic processes and pointing out the advancement of the presented modeling approach. Section 3 presents the action modeling framework of PSI. Section 4 reports how the framework has been implemented as the part of PSI suite of ontologies, evaluated, and used in PSI software prototype. Finally, concluding remarks are given and our plans for future work are outlined in Section 5.

2 Related Work

The framework presented in this paper is for modeling change and adequately accounting for dynamics in the processes of engineering design. Fundamentally, research in representing, reasoning, and capturing knowledge about change and dynamics produced the plethora of premium quality results which can't be even listed here due to space limit. Instead, we point to [3] as an excellent reference source. We also mention several related sources for analyzing our contribution.

McCarthy and Hayes [4] were the pioneers in introducing a logical formalism which became a mainstream for commonsense reasoning and reasoning about change in particular – the Situation Calculus (SC). Several authors have further developed

this approach resulting in several Event Calculi (EC) [5, 6]. Most of them use linear time instead of branching time characteristic to the SC. A topical representative of a branching time logic approach is [5]. Our approach is particularly close to DEC [6] because DEC uses discrete linear time representation. In difference to the mentioned EC our framework uses discrete linear time and time intervals with fuzzy beginnings and endings [7]. This enhancement makes our representation of events [8] and actions more flexible and expressive. For all other desired representational capabilities like causality, event triggering, context sensitivity, delays in effects, concurrency, release from the law of inertia [9] we rely on [6]. Some of these have already been accounted for: causality, triggering, delays. Elaboration of the rest is planned for the future work. The mainstream of formal business process modeling and engineering today is using PSL [10], PDL [11] or their extensions. Unfortunately, these formal process modeling frameworks do not fully allow breaking down the diversity of the processes encountered in real life. This diversity may be characterized for example by Sandewall's taxonomy [9] of the basic features of the processes. This classification embraces highly predictable, normal, manufacturing processes at one side and stochastic ("*surprising*"[2]), structurally ramified, time-bound processes characteristic for design domain, on the other side of the spectrum.

Presented modeling framework and the PSI suite of ontologies are the follow-up of our results published in [12]. The DEDP modeling framework in its part of process modeling bases its approach on [15-17]. The advancements of PSI approach are: (i) a rich typology of actions; (ii) environmentalistic approach to model processes, actions, their dependencies comprising concurrency; (iii) a state model refined using decision making mechanism and requirement sensitivity; (iv) an explicit difference between events and actions.

To the best of our knowledge, existing frameworks do not specify the difference between events and actions, except stating that actions are a kind of events: "the most important events are actions, and for a program to plan intelligently, it must be able to determine the effects of its own actions…" [cf. 18]. Such a view underestimates the role of events which occur without the involvement of an actor and the influence of those events on the environments of actions. Indeed, if we consider a person accidentally falling out from a window, this event can hardly be qualified as an action – the person had no purpose for or intention of falling out. The refinement proposed in PSI [19, 20] is that processes (compound actions) subsume to events, while atomic actions do that not. Atomic actions are a specific kind of an instrument for agents to proactively apply changes to their environment(s).

Our analysis of the variety of foundational ontologies [14] has revealed that the best matching ontological foundation for DEDP modeling is DOLCE [15] and the most appropriate referential commonsense theory is SUMO [16] extended by WordNet [17]. The semantics of our representation of actions, events, and environments is aligned with DOLCE through the PSI Upper-Level ontology [14]. The concepts of PSI Process ontology are mapped to SUMO through PSI Upper-Level ontology and WordNet using subsumptions.

---

[2] A process is considered "surprising" if it is allowed that a *surprising* or *exogenous* event may cause a change that is not anticipated by the process script [11].

3 Action Modeling Framework of PSI

A DEDP is the process of goal-directed (pro-active) transformation of a design artifact. A DEDP usually begins with collecting the initial available inputs (like the requirements, the high-level specification), continues in a sequence of stages normally defined by the design system, and ends up with the design artifact in a form which meets the goal of the design. These stages are the actions applied to a design artifact. Actions are distinct because they affect a design artifact differently by applying different changes (transformations) or causing no tangible change at all. Actions may be grouped in different combinations like sequences, branching structures, alternative or concurrent paths.

3.1 Preliminaries

A design artifact (DA) is a tangible product that is being designed in a DEDP. A DA may be a single indivisible design object or a hierarchical composition of design objects having the same or different types. A DA, and every design object in a DA, is incrementally elaborated as the emerging set of its representations. A DA representation (a representation further on in the paper) is the implementation of a DA in a particular form, format, or notation in which the DA is used for a distinctive purpose.

There exists only a partial order among representation types and respective representations. The semantics of this partial order is that a representation R_m which precedes another representation R_n is more abstract, while R_n is more elaborated. The distance between two representations R_m and R_n may also be of interest. Indeed, a question about how much is R_n more elaborated (or more abstract) than R_m is important because the answer characterizes the difficulty of the transformation of a DA between those representations. Difficulty is understood as the amount of abstraction crossed by a transformation. As transformations are applied by actions, difficulty (and distance) is somehow reflected by the effort to be spent for an action.

In any combination, actions lead processes to particular states. The simplest possible DEDP may be described by specifying its initial (triggering) influence, its initial state, its action, its target state, and the change in the design artifact caused by the specified action and reached in the target state. A more complex DEDP may comprise both atomic and compound actions. Hence, in a general case, a DEDP description should also contain the specification of its intermediate states. A DEDP state is a state S of DEDP environment which is characterized by the set of the pre-requisites for the associated actions. These pre-requisites are either the events [8] which, if perceived as happenings [8], trigger influences that change the course of action, the representations which are required for an action, or their combination. A DEDP state is the state of affairs in which a decision to perform one of the admissible actions (for example, to cease the process) is to be made:

$$S = < \mathbf{E}, \mathbf{R}, D_s, \mathbf{A}) > , \tag{1}$$

where: **E** is the set of associated events, **R** is the set of associated representations, D_s is the mechanism to make the decision to take a certain associated action, **A** is the set of admissible actions.

A representation R which is unconditionally available after a state S is reached is the characteristic representation of this state and belongs to the set $\Re$ of characteristic representations of this state.

Further on, to find out if a representation is really the thing we wanted to receive, the characteristics of the representation are verified against the requirements. For simplicity reasons only independent characteristics are taken into account. An independent characteristic c is the property of a representation which may be measured as recommended by the design system independently of the other characteristics. The set of independent characteristics of a representation is denoted as $\mathbf{C} = \{c_1,...c_n\}$. A requirement $\rho : \mathbf{C} \to \{true, false\}$ is the Boolean function of the independent characteristics of a representation. Provided that requirements are defined for a representation, the degree of the success of an action elaborating the representation could be measured. If an action was successful enough the corresponding state may be considered as achieved. Otherwise a corrective action should be taken to improve the result. Hence the difficulty of an action is the function of the requirements to its target representation.

Requirements to the same representation may differ in different phases of a DEDP. For example, the characteristic of the density of the elements on the integrated circuit becomes really important at a place and route phase, though accounted for more liberally at earlier phases. Hence, a requirement is the attribute of a representation in a certain DEDP State. Requirements may be changed when a DEDP is already being executed. Such (events of) late changes to the requirements may result in unpredictable changes in the DEDP. If $T = [0,T]$ is the life time of a DEDP then a dynamic requirement $\rho(t) : \mathbf{C} \to \{true, false\}, t \in [0,T]$ is a requirement which may be changed during the life time of a DEDP, otherwise it is a static requirement.

Accounting for the requirements implies the changes in the model of a state. A ρ-sensitive state is a DEDP state in which representations are constrained by a non-empty set of requirements $\mathbf{P} = \{\rho_1,..., \rho_n\}$, which may be static or dynamic:

$$S =< \mathbf{E}, \mathbf{R}, \mathbf{P}, D_s, \mathbf{A}) > . \tag{2}$$

One important kind of a requirement is a quality requirement. Such requirements are based on the characteristics measured as prescribed by the used quality model.

3.2 Action Kinds

While modeling actions it is important to pay attention to the following characteristic features: (i) is an action simple or compound? (ii) does an action transit a DEDP to a different state? (iii) what are the changes applied to the DA? (iv) what are the dependencies among certain actions?

Compound and Atomic Actions. Actors may have different understanding of the actions in a DEDP (Fig. 1). Some of them, according to their role, prefer to operate high-level actions. For example, a project manager may find more appropriate to specify only high-level actions in the project plan, like *front-end design* and *back-end design*. The others may tend to go deeper in the details of the actions. A front-end designer will definitely notice that a high-level *front-end design* action comprises

several lower-level actions like *RTL development*, *testbench development*, etc. Therefore one may deduce the hierarchical structure of the actions in a DEDP[3] from the different facets of understanding a DEDP by different actors playing different roles. It is however important to find out if there are the basic building blocks for these actions – the ones which are understood as indivisible (atomic), by all the actors in a design system. It is rational to consider that such atomic actions exist and are defined by the design technology used within the design system. Such atomic actions are further on referred to as activities.

Definition 1: *an activity*. An activity is a basic indivisible (atomic) action which is allowed, supported, and provided by a design technology. An activity is the only action which is executed and applies the atomic chunk of the transformation to a design artifact.

Compound actions are the parts of a DEDP which are shrunk into one edge for convenience and the proper representation for different roles. These composite actions are denoted as tasks.

Definition 2: *a task*. A task is a compound action which may be represented as the composition of the other tasks and activities. Such representations are different in the knowledge of different actors.
As shown in Fig. 1, tasks may contain several transformation paths.

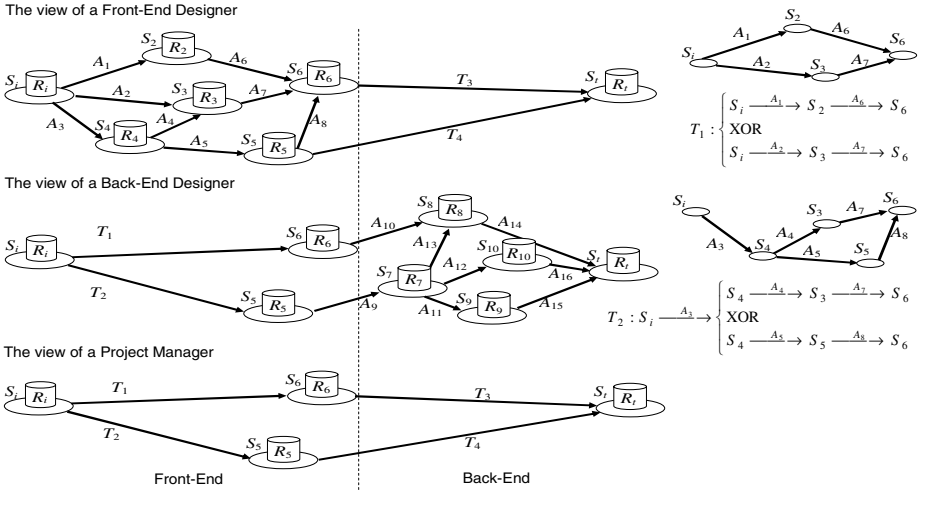

Fig. 1. Compound and atomic actions in subjective views of different actors

State-Transitive Actions. A DEDP transits to a new state when the pre-requisites for such a transition are met. These pre-requisites are: (i) events which indirectly trigger an action or (ii) the availability of characteristic representations for the target DEDP state. For clarity we shall consider the events occurring outside of DEDP environment

[3] Like in a project plan. Indeed, the Actions in a project plan are often presented in a hierarchy.

and the events generated within the environment of a DEDP as separate kinds of events – external and internal ones.

An action will transit a DEDP to a new state S if representations produced by this action "complete" the set $\Re$ of the characteristic representations of S. According to the classification of the results of actions, the following types of actions may result in DEDP state transition because they produce new representations: productive actions, decomposition and integration actions.

It can not be expected that external events occur in a controllable manner. Therefore unexpected happenings and appropriate reactions to them in the form of influences should be accounted for. External events may or may not be perceived by the actors in the DEDP environment. External events may cause environmental changes of different magnitude. We shall say that an environment is stable with respect to a particular external event if the change caused by this event is negligibly subtle. On the contrary the environment is not stable with respect to a particular external event if the magnitude of the incurred change is substantial. By saying "substantial" we mean that the magnitude of the change[4] requires that a corrective action is applied to the DEDP. In the latter case it is important to ensure that such an event is perceived and the influence is generated to execute required corrective actions. For the sake of uniformity and simplicity we shall consider a forced change of DEDP state as the only possible type of a corrective action.

Internal events are generated by the components of the environment [8] of a DEDP – the design system. One possible kind of an internal event is that an actor executing an action becomes unavailable. Possible reactions are: (i) action suspension until the actor becomes available; (ii) actor substitution – no influence to DEDP; (ii) corrective action changing DEDP state and choosing a different transformation path executed by a different actor. Another possible internal event could be that a resource being consumed in the action becomes unavailable. Possible reactions are: (i) action suspension until the resource becomes available; (ii) resource substitution – no influence to DEDP; (iii) corrective action changing DEDP state and choosing a different transformation path where the resource is not consumed. Yet one more kind of an internal event is that the requirements to a representation have no chance to be met if the chosen iterative action is continued. Possible reaction is rolling back to the previous DEDP state and choosing a different transformation path.

If a DEDP has reached its target state S_t and all the requirements to the characteristic representations of S_t are met, a cessation action terminating the DEDP in success has to be applied. In all other cases either a change in the environment (like actor substitution or resource substitution) is sufficient or an action is decided to be suspended. Otherwise, a corrective action should be taken to choose a different transformation path in the process.

Corrective Actions. In a DEDP some transformation paths may be more risky than the others. Indeed, when for example a design system transits to a new design technology, the correlations among the requirements are not very well understood. The assessments of the quality provided by actions are not well grounded. If these settings are complicated by the dynamic factors, an action on a chosen transformation path

[4] Corresponding thresholds should be found out experimentally when calibrating the model of the design system.

may unexpectedly result in missing the requirements to a characteristic representation. Though iterative actions may help further elaborating or refining the representation, there might be a situation in which the refinement is not longer possible using available actions. A corrective action may improve such a situation by: (i) rolling-back the transformation path to the nearest successful state or (ii) choosing the next-most productive transformation path as the back-up plan. Corrective actions may also be used as a mechanism of collecting facts on bad experience to make risk assessments more grounded in the future.

Iterative Actions. Some types of actions will iterate a DEDP in a ρ-sensitive state S if characteristic representations of S do not meet the set of requirements $\mathbf{P} = \{\rho_1,...,\rho_n\}$ of state S, i.e. at least one of the requirement functions ρ is *false*. Moreover, only such types of actions may be triggered by D_S until the requirements are met. These types of actions are: refinement actions, elaboration actions, debugging actions, verification actions. A D_S may detect that several iterative actions should be performed at a certain phase of iterations at state *S*. By the analogy to productions-based inference engines these actions may be considered to be in a conflict set. The conflicts may be resolved by: (i) analyzing the dependencies among the actions in the conflict set; and (ii) assigning priorities to the (types of) independent actions.

Cessation Actions. The difference between a corrective action and a cessation action is that a corrective action transits a DEDP to its state, though different from the previous one, but a cessation action terminates a DEDP – i.e., moves it out of the state space. A cessation action may terminate a DEDP in success or in failure.

3.3 Dependencies and Concurrency

As emphasized by many authors, for example [18], best performance is achieved when the best achievable degree of coherence among the actions within a process is granted. Coherence among actions means several important things: (i) coherence in individual goals of different actors performing different actions in one process; (ii) proper distribution of the consumption of available resources in different actions; (iii) balance in the capacities of the tools used in different actions; (iv) appropriateness of the skills of the actors to the requirements of the actions assigned to these actors; (v) proper scheduling of the actions which use the results produced by other actions. All these aspects represent dependencies among actions. Hence, achieving coherence among these actions can be reached by coordination, which is the routine of "managing the interdependencies between activities" (cf. [19]). Therefore, a model of action dependencies should account not only for the direct dependencies of actions on the results of other actions (the latter case (v)), but for a broader variety of indirect dependencies (at least cases (i)–(iv)) revealed through the process environment. Generally speaking, action A_3 depends on another action A_1 when the post-effects of A_1 change the pre-requisites of A_3. This dependency may be denoted in the terms of a DEDP State, an event, a happening, and an influence.

Let S_1, S_2 be ρ-sensitive States (2).

Definition 3: *an action-related part of DEDP environment.* $\Sigma\big|_{A_1} = \{\mathbf{R}_1, \mathbf{AC}_1, \mathbf{RT}_1\}$ is the part of DEDP environment related to A_1 if:

- The execution of any action A associated with S_1 generates internal events changing the constituents of Σ_1, or
- The occurrence of any external event E changing $\Sigma\big|_{A_1}$, if percieved by a member of $\mathbf{AC}_1$, may change the course of actions associated with S_1 by influencing these actions at their execution time

The constituents of $\Sigma\big|_{A_1}$ are (Fig. 2):

$\mathbf{R}_1$=$\{R_1, R_2, \ldots, R_4\}$ – the set of representations available in S_1 and (if any) produced to reach S_2

$\mathbf{AC}_1$=$(Ac_1, Ac_2, \ldots, Ac_m)$ – the set of actors capable of executing A_1 associated with S_1 and available at the time when A_1 has to be commenced

$\mathbf{RT}_1$ – the pool of resources consumed by and tools used in A_1 associated with S_1

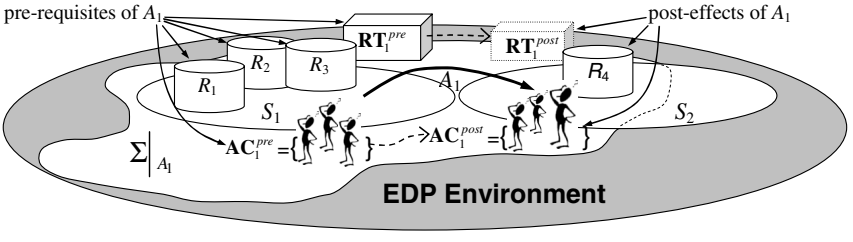

Fig. 2. A part of DEDP environment ($\Sigma\big|_{A_1}$) related to action A_1

It is important to note that, according to definition 3, if an action (A_1) causes state transition (S_1 to S_2), then $\Sigma\big|_{A_1} = \Sigma\big|_{A_1}^{pre} \cup \Sigma\big|_{A_1}^{post}$ comprises both the pre-requisites of A_1 and the post-effects of A_1. Some of these pre-requisites are not associated with DEDP states, like $\mathbf{AC}_1^{pre}$ and $\mathbf{RT}_1^{pre}$. Others belong to S_1, like $\mathbf{R}_1^{pre} = \{R_1, R_2, R_3\}$. Some of the post-effects are also state-independent: $\mathbf{AC}_1^{post}$ and $\mathbf{RT}_1^{post}$ containing those elements that have been changed by A_1. These changes may be in the availability of actors, resources and tools and the capability of actors. Other post-effects belong to the target DEDP state S_2: $\mathbf{R}_1^{post} = \{R_4\}$. These post-effects also contain changed representations only.

If A_1 does not cause the transition of the process to a different DEDP state, the configuration of $\Sigma\big|_{A_1} = \Sigma\big|_{A_1}^{pre} \cup \Sigma\big|_{A_1}^{post}$ still remain similar to the previous case. The only difference is that $\mathbf{R}_1^{post} = \{R_4\}$ belongs to the same DEDP state (S_1).

Let A_1 be an action causing the transition of the DEDP to DEDP state S_2, A_3 be an action associated with DEDP state S_3.

Definition 4: *a dependency*. A_3 depends on A_1, if $\Sigma\Big|_{A_3}^{pre} \cap \Sigma\Big|_{A_1}^{post} \neq \varnothing$, otherwise A_3 and A_1 are independent.

Following [20] we shall classify dependencies as *weak* and *strong*. Action A_3 is strongly dependent on action A_1 if the execution of A_3 could not be started until A_1 is accomplished and its goal is fully met. Having in mind that the goal of an action in a DEDP is reaching the DEDP state in which all the required representations are made available, we may denote strong dependency as follows.

Definition 5: *a strong dependency*. A_3 strongly depends on A_1 if $\Sigma\Big|_{A_3}^{pre} \cap \Sigma\Big|_{A_1}^{post} = \Sigma' \subseteq \mathbf{R}_1^{post}$

Please note that definition 5 holds true also if A_1 is not a state transitive action.

Definition 6: *a weak dependency*. A_3 weakly depends on A_1 if:

(i) $\Sigma\Big|_{A_3}^{pre} \cap \Sigma\Big|_{A_1}^{post} = \Sigma' \subseteq \Sigma\Big|_{A_1}^{post} \setminus \mathbf{R}_1^{post}$ – *environmental* dependency, or

(ii) $\Sigma\Big|_{A_3}^{post} \cap \Sigma\Big|_{A_1}^{post} = \Sigma' \subseteq \mathbf{R}_3^{post}$ – *facilitation* dependency

According to definition 6 environmental dependencies (i) are caused by sharing the pool of actors, the pool of tools, consuming the same resources, or by the combination of these reasons, normally indicating that the actions are competitive. On the contrary, facilitation dependencies (ii) indicate that actions are cooperative. Indeed, the interpretation of (ii) is as follows: A_1, facilitates A_3 in reaching its goal because it elaborates some part of the set of representations $\mathbf{R}_3^{post}$.

Accounting for action dependencies may help building better DEDP schedules thus improving their performance properties.

Concurrency among actions is one more aspect which may influence temporal properties of DEDP performance. It is evident that gaining more concurrency among the actions in a DEDP may result in shorter schedules and shorter execution times. Even partial overlaps in time intervals of action execution may optimize the overall performance. Unfortunately, it is not possible to execute all the actions in a DEDP in parallel or partly in parallel because of their dependencies.

Let, according to [7], I_{A_1} be the fuzzy time interval of the execution of A_1 and I_{A_3} be the fuzzy time interval of the execution of A_3. Then the following definitions of *full* and *partial* concurrency among two different actions hold true.

Definition 7: *full concurrency*. Action A_1 is fully concurrent to action A_3 if $Same(I_{A_1}, I_{A_3}) \vee Within(I_{A_1}, I_{A_3})$.

Definition 8: *partial concurrency*. Action A_1 is partially concurrent to action A_3 if $Overlaps(I_{A_1}, I_{A_3})$.

In terms of action dependencies we may rightfully state that if action A_1 is independent to action A_3 then we may schedule and execute them fully concurrently. Granting concurrency to dependent actions is not that straightforward. The case of a strong dependency is simpler.

Corollary 1: *concurrency of strongly dependent actions*. If action A_3 is strongly dependent on action A_1 then they can not be executed concurrently.

The case of a weak dependency is more complex.

Corollary 2: *concurrency of environmentally dependent actions*. If action A_3 is environmentally dependent of action A_1, then their concurrent execution, while being possible, may make the overall performance less optimal.

Corollary 3: *concurrency of actions with facilitation dependency*. If action A_1 facilitates the execution of action A_1, then their concurrent execution while result in better overall performance.

4 Implementation and Evaluation

The action modeling framework presented above has been implemented as several OWL-DL[5] modules of the PSI suite of ontologies. The initial implementation has been done in the suite of ontologies v.2.0. In this revision actions were modeled by the Task-Activity ontology and several relationships to the Actor ontology, Project ontology, Design Artifact ontology. In v.2.1 the ontological model of actions has been refined by introducing the associations to the Time ontology [7] and the Environment, Event, and Happening (E2H) ontology [8]. Finally, in v.2.2 the Upper-Level ontology [14] has been introduced and the modular structure has been refined by splitting the action model at the domain level into the Process Pattern ontology and the Process ontology. PSI shaker modeling methodology [14] has been used for refining the suite of ontologies and producing v.2.1 and v.2.2.

Fig. 3 pictures the UML diagram of PSI Process ontology v.2.2. Full details of this ontology are given in [21]. Uncolored packages in Fig. 3 represent different PSI core ontologies: Actor ontology; Project ontology; Design Artifact ontology; Environment, Event, and Happening ontology [8]; Time ontology [7]. The grey colored package represents the Resource extension ontology developed in the PRODUKTIV+ project[6].

PSI Process ontology has been evaluated as the part of the core of PSI suite of ontologies v.2.2 as suggested by the shaker modeling methodology. Three different evaluation activities have been performed: (i) user evaluation; (ii) formal evaluation; (iii) commonsense evaluation.

User evaluation has been performed as a goal-based evaluation of the adequacy of the knowledge model and its implementation in the ontology to the set of requirements by the group of subject experts in microelectronic engineering design. It found out that the ontology adequately answers the competency questions formulated using the requirements by subject experts. Several test cases have been developed for evaluating the suite of ontologies using simulation. A testcase is a real or a fictive project for which at least all the initial data instances required for design system modeling are available. Acquiring a testcase allows to verify that the ontology is capable of storing all initial data as its instances and to start simulation using the multi-agent system

[5] Web Ontology Language: http://www.w3.org/TR/owl-guide/.

[6] PRODUKTIV+ is the R&D project funded by the German Bundesministerium für Bildung und Forschung (BMBF).

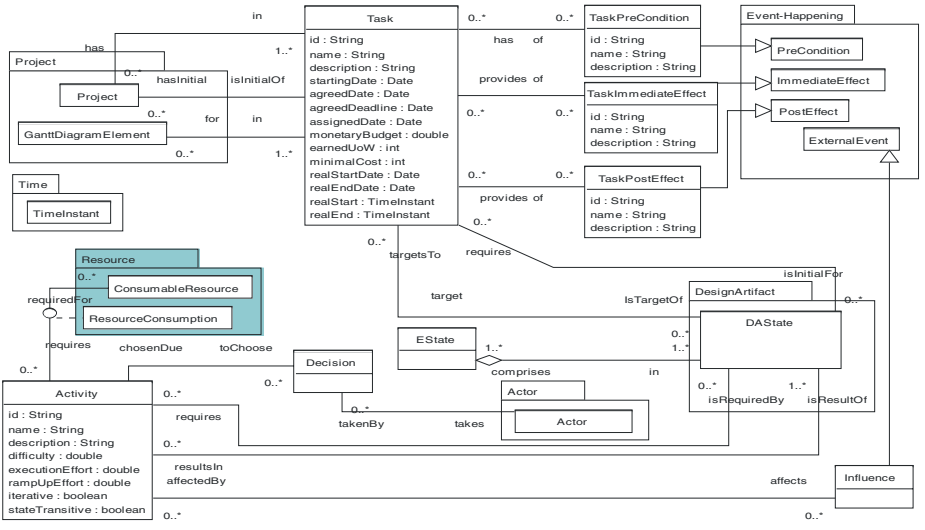

Fig. 3. UML diagram of the main concepts in the PSI Process ontology v.2.2

(MAS). Ideally, a testcase also provides a complete project execution record which is required for the calibration step and for the verification of the simulation results. Amongst the set of testcases are very simple and fictive ones describing tiny digital and tiny analog designs. These have been used to verify and refine the model and the ontologies. Others are based on real world projects of digital and analog chip design. These are fictive ones as well designed for demonstration purposes because real world project data usually may not be disclosed to public. For calibration the Project Planning Expert System MAS [2] was fed with the project definition and the knowledge base. Using these, it created a new work breakdown structure (Fig. 4). The result was then compared to the original structure, differences were analyzed, and corrections were made until the both roughly matched. At subsequent stages the MAS simulated project executions (Fig. 5) and again the results were compared to the original project course. Project log replay simulations and calibration experiments proved that the approach is effective and practical.

Formal evaluation has been performed using OntoClean methodology [22]. Its objective was to check the formal conformance of the taxonomy structure of the evaluated Suite of Ontologies v.2.2 to the meta-properties of rigidity, identity, and dependence [23]. Formal evaluation of PSI Process ontology together with PSI Upper-Level ontology [14] revealed that the taxonomy structure is conceptually correct.

The mappings of the concepts of the Process ontology to SUMO [16] through PSI Upper-Level ontology and WordNet [17] have been defined [13] for evaluating these ontologies with respect to the common sense [24]. This work revealed that the ontology adequately maps to SUMO – the chosen [14] commonsense reference ontology. This fact allows us believing that the process related part of the PSI suite of ontologies may be used not only internally in PSI and PRODUKTIV+ projects, but broader – as a descriptive theory of dynamically ramified processes in the domains which are

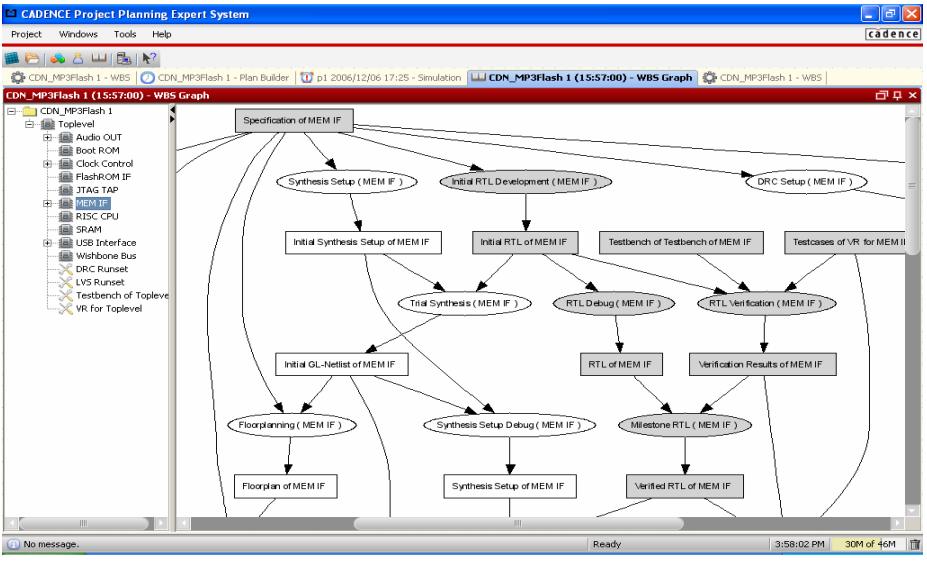

Fig. 4. The result of the planning phase of a design process simulation in Cadence Project Planning Expert System: generated Work Breakdown Structure graph. Ellipses stand for results, rectangles for tasks. Darker entries show the selected transformation path.

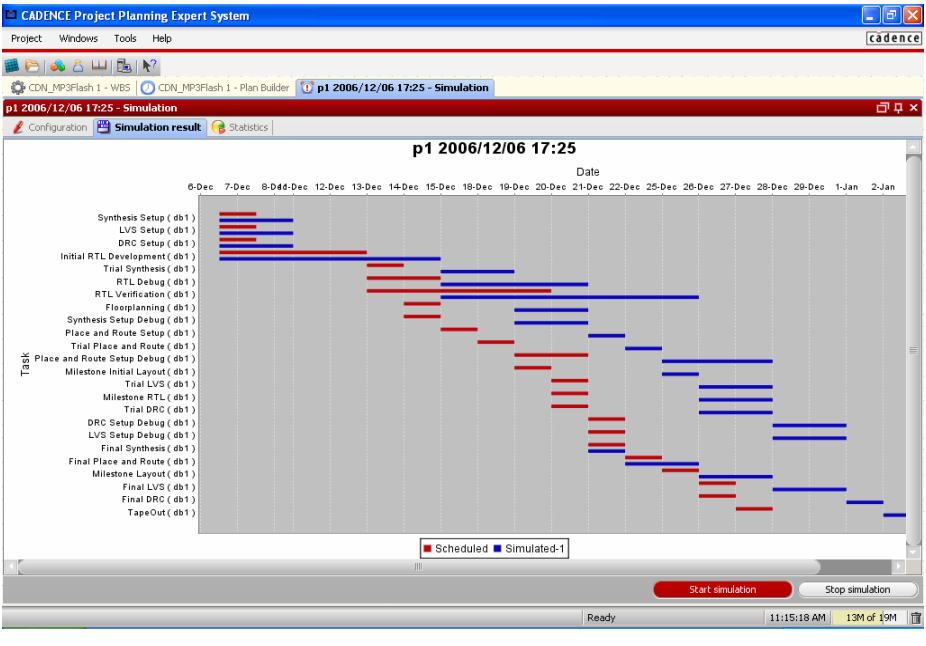

Fig. 5. The results of planning and execution phases of a design process simulation in Cadence Project Planning Expert System. Action durations are compared for planning (brighter bars) and execution (darker bars).

dynamic, complex, and non-deterministic similarly to engineering design. One good example is the domain of knowledge processes and knowledge workers investigated by the ACTIVE project[7].

5 Concluding Remarks

The paper presents the action-related part of the PSI theoretical framework and its implementation as the Process ontology of the PSI suite of ontologies v.2.2. The advantages of the presented approach to modeling actions and processes are: (i) a rich variety of action kinds; (ii) environmentalistic approach to model processes, actions, their dependencies comprising concurrency; (iii) a state model refined using decision making mechanism and requirement sensitivity; (iv) an explicit difference between events and actions. These allow making process and action models being flexible and adaptive to the extent required for modeling structurally and dynamically ramified, time-bound processes characteristic for engineering design domain.

The presented ontological model of actions has been iteratively refined starting from its initial revision in v.2.0 of the PSI suite of ontologies till its current revision in v.2.2 using PSI shaker modeling methodology. The methodology subsumes several kinds of evaluation activities which have been performed as reported in the paper. Evaluation proved that the presented approach to modeling actions and processes is practical and effective. The core part and the several extensions of the PSI suite of ontologies are used in the Cadence Project Planning Expert System software prototype.

References

1. Van Staa, P., Sebeke, C.: Can Multi-Agents Wake Us from IC Design Productivity Nightmare? In: Mařík, V., Vyatkin, V., Colombo, A.W. (eds.) HoloMAS 2007. LNCS (LNAI), vol. 4659, pp. 15–16. Springer, Heidelberg (2007)
2. Sohnius, R., Jentzsch, E., Matzke, W.-E.: Holonic Simulation of a Design System for Performance Analysis. In: Mařík, V., Vyatkin, V., Colombo, A.W. (eds.) HoloMAS 2007. LNCS (LNAI), vol. 4659, pp. 447–454. Springer, Heidelberg (2007)
3. Mueller, E.T.: Commonsense Reasoning. Morgan Kaufmann Publishers, San Francisco (2006)
4. McCarthy, J., Hayes, P.J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence. Machine Intelligence 4, 463–502 (1969)
5. Shafer, G., Gillett, P.R., Scherl, R.B.: The logic of events. Ann. Math. Artif. Intelligence 28(1-4), 315–389 (2000)
6. Mueller, E.T.: Event Calculus Reasoning through Satisfiability. J. Logic and Computation 14(5), 703–730 (2004)
7. Ermolayev, V., Keberle, N., Matzke, W.-E., Sohnius, R.: Fuzzy Time Intervals for Simulating Actions. In: Kaschek, R., Kop, C., Steinberger, C., Fliedl, G. (eds.) UNISCON 2008. LNBIP, vol. 5, pp. 429–444. Springer, Heidelberg (2008)

[7] ACTIVE: Knowledge Powered Enterprise (http://www.active-project.eu/) is an Integrating Project funded by Framework Program 7 of the European Union.

8. Ermolayev, V., Keberle, N., Matzke, W.-E.: An Ontology of Environments, Events, and Happenings. In: Proc. 31st IEEE Annual International Computer Software and Applications Conference (COMPSAC 2008), pp. 539–546. IEEE Computer Society, Los Alamitos (2008)
9. Sandewall, E.: Features and Fluents. The Representation of Knowledge about Dynamical Systems, vol. 1. Oxford University Press, Oxford (1994)
10. Bock, C., Gruninger, M.: PSL: A semantic domain for flow models. Software and Systems Modeling Journal 4, 209–231 (2005)
11. Workflow Management Coalition. Workflow Standard. Process Definition Interface – XML Process Definition Language. V. 2.00, Doc. No. WFMC-TC-1025 (Final), October 3 (2005)
12. Ermolayev, V., Jentzsch, E., Karsayev, O., Keberle, N., Matzke, W.-E., Samoylov, V., Sohnius, R.: An Agent-Oriented Model of a Dynamic Engineering Design Process. In: Kolp, M., Bresciani, P., Henderson-Sellers, B., Winikoff, M. (eds.) AOIS 2005. LNCS, vol. 3529, pp. 168–183. Springer, Heidelberg (2006)
13. Ermolayev, V., Jentzsch, E., Keberle, N., Sohnius, R.: Performance Simulation Initiative. Meta-Ontology v.2.2. Reference Specification, tech. report PSI-ONTO-TR-2007-4, VCAD EMEA Cadence Design Systems GmbH (2008)
14. Ermolayev, V., Keberle, N., Matzke, W.-E.: An Upper-Level Ontological Model for Engineering Design Performance Domain. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 98–113. Springer, Heidelberg (2008)
15. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: WonderWeb Deliverable D18 Ontology Library (final). In: ICT Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web (2003)
16. Niles, I., Pease, A.: Towards a Standard Upper Ontology. In: Guarino, N., Smith, B., Welty, C. (eds.) Int. Conf. on Formal Ontologies in Inf. Systems, vol. 2001, pp. 2–9. ACM Press, New York (2001)
17. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
18. O'Donnel, F.J., Duffy, A.H.B.: Design Performance. Springer, London (2005)
19. Malone, T., Crowston, K.: Toward an interdisciplinary theory of coordination. Center for Coordination Science, Technical Report 120, MIT Sloan School of Management (1991)
20. Nagendra Prasad, M.V., Lesser, V.R.: Learning situation-specific coordination in cooperative multi-agent systems. Autonomous Agents and Multi-Agent Systems 2(2), 173–207 (1999)
21. Ermolayev, V., Jentzsch, E., Keberle, N., Sohnius, R.: Performance Simulation Initiative. The Suite of Ontologies v.2.2. Reference Specification. Technical report PSI-ONTO-TR-2007-5, VCAD EMEA Cadence Design Systems, GmbH (2007)
22. Guarino, N., Welty, C.: Supporting Ontological Analysis of Taxonomic Relationships. Data and Knowledge Engineering 39(1), 51–74 (2001)
23. Guarino, N., Welty, C.A.: A Formal Ontology of Properties. In: Dieng, R., Corby, O. (eds.) EKAW 2000. LNCS, vol. 1937, pp. 97–112. Springer, Heidelberg (2000)
24. Keberle, N., Ermolayev, V., Matzke, W.-E.: Evaluating PSI Ontologies by Mapping to the Common Sense. In: Mayr, H.C., Karagiannis, D. (eds.) Proc. 6th Int'l Conf. Information Systems Technology and its Applications (ISTA 2007). GI LNI, vol. 107, pp. 91–104. GI Bonn (2007)