# Modelling and Simulating Chip Design Processes

Amir Hassine
*Institute of Microelectronic Systems, Leibniz Universität Hannover*
*Germany*

## 1. Introduction

When the semiconductor industry emerged from anonymity in the 1960's, almost no one expected it to gain such a tremendous importance in human kind's life. 1981 Bill Gates, although he today denies to have said it, stated that "640K – program memory – ought to be enough for anybody". Indeed, no other industry than the semiconductor industry has experienced such a growth: The Semiconductor Industry Association[1] reports about an average annual growth rate of over 16% from 1975 to 2000. It is even said to be the most productive industry in the world today (Goodall et al., 2006). Continuous improvements in EDA[2] tools, reusability and technologies have made it possible to design complex functionalities and integrate hundreds of millions of transistors on a single chip.

None the less, figures reveal another - less pleasant – fact: managerial expertise in the chip design industry seems to lag behind technical expertise and progress. The ITRS[3] reported several times about a lingering and increasing design productivity gap: the number of available transistors growing faster than the ability to meaningfully design them.

Managers are nowadays faced with many challenges concerning design projects: They have to exchange resources, restructure the process, train designers etc. However, trial and error of several alternatives is not practicable due to narrow time to market windows and severe budget restrictions. In absence of simulative possibilities, decisions do not rely on evaluating the alternatives regarding their productivity and costs, but on estimations and gut feelings and are mostly biased. For planning decisions, it is therefore necessary to provide tools that support decision makers in evaluating different solutions in an objective and transparent manner.

Due to the rapid progress in the semiconductor industry, the wide range of available tools and resources and the immense competition as well as time and budget pressures, the

[1] Semiconductor Industry Association, *www.sia-online.org*
[2] Electronic Design Automation
[3] International Technology Roadmap for Semiconductors, *http://public.itrs.net*

necessity to investigate modelling and simulation design systems has emerged as a prevalent issue. The lack of tools to predict required resources and process runs is the principal reason for late project cancellations and delays. Just as worrying as the long-term growth slow down to 8-10%, so are the estimated $ 2 to $ 4 billion losses yearly due to project cancellations and aborts (Numetrics, 2006). Moreover, 85% of IC[4] projects miss their targeted schedules (Collett, 2004).

With simulative approaches, decision makers would be able to compare several process arrangements regarding their productivity and costs and choose the appropriate one instead of relying on estimations and gut feelings.

In this chapter a model and a simulator are presented to address the missing tools in the field of modelling and simulating chip design processes (CDPs). The model regards resources and design artefacts within a CDP in a generic manner. Thus, it is also applicable to any other engineering or production process.

## 2. State of the art

Triggered by the design gap, several contributions focussed on defining and measuring productivity (Hassine & Barke, 2005) (Numetrics, 2000) as well as establishing business and technical KPIs[5] (Leppelt et al., 2006). Infrastructures for data collection (Fenstermaker et al. 2000) and analysis methods (Kahng & Mantik, 2001) have therefore been elaborated and partially marketed. However, formalization and simulation of design systems have barely been addressed.

The prediction of the productivity of chip design processes still relies on simple estimations. It is in best cases based on the outcome of previous comparable projects. However, each chip design process is admittedly unique and thus the estimations often lead to grave deviations from plans and budgets.

Apart from (Matzke & Strube, 2006) (Ermolayevet al., 2006) and (Sohnius et al., 2007) few efforts have investigated modelling and simulating chip design processes in a granular way. The approach presented aims at modelling design processes in general and is based on multi-agent systems (MAS). The latter use the approach of collaborating agents (software programs) to solve a given problem. Thereby the agents stand for the elements of the original to be modelled and are endowed with behaviors, preferences, the ability to act with the environment, etc. The approach sets the emphasis on modelling human resources by means of agents. The latter are endowed with different abilities in carrying out activities, play different roles (manager, designer, etc.), negotiate with each other, build up teams and cooperate to execute tasks assigned to them.

The added value yielded by the agents is measured by so-called Units of Welfare (UoW) but are not specified deeply in detail. Furthermore, the simulation does not aim at optimizing the productivity of the whole process but at minimizing the total process duration. Above

[4] Integrated Circuit

[5] Key Performance Indicators

all and because of ethical, acceptance and feasibility reasons the approach is very likely to face strong resistance by industrials.

## 3. The RS Model

Within chip design processes - seen as a sequence of activities - design artefacts (DAs) are transformed into different states and/or are verified regarding their compliance to the constraints set. Each activity includes several resources and its duration as well as the quality of its outputs depend on the resources allocated, e.g. designer's experience, and on the DA properties, e.g. complexity. On this note, the Request Service (RS) model models resources as providers of services. The latter are requested by the activities to process DAs.

A series of attributes describes the resources, services and DAs in a quantitative and qualitative way. Calculation models specify activities' duration and outcome.

A petri-net-like notation is adopted to graphically represent the CDP elements in the RS model. Since the goal is a computer aided simulation, the scenarios to be modelled and simulated have to be expressed in a machine-readable manner. Ontologies are used to formally represent the CDP domain because they offer an easy way to separate the generic knowledge about CDPs from specific process instances to be simulated. Furthermore, it allows for splitting up the model from implemented simulators using it.

In order to better comprehend the RS model, the following section illustrates basic knowledge about Petri nets and ontologies.

### 3.1. Prerequisites

#### 3.1.1 Petri nets

Petri nets have been developed by Carl Adam Petri in the 60s and represent in their original form a time-independent and purely causal concept for modelling discrete systems. A Petri net is constituted of:

- Transitions (rectangles): active elements in a system, e.g. actions or events
- Places (outlined circles): passive elements, e.g. states or conditions
- Edges relating transitions and places.

Places contain tokens (black filled circles). The latter represent dynamic elements being transported and transformed, e.g. information. When a transition fires, tokens are removed from pre-located places (respectively to the activity) and transformed/transported to a post-located one. The number of transported tokens is specified by the edge weight (Fig. 1).

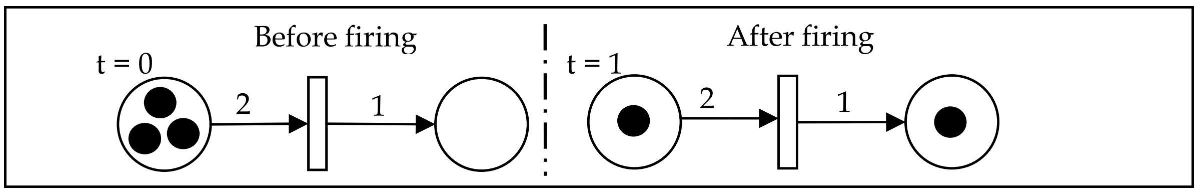


Fig. 1. Petri net: Firing of a transition

The so-called – low-level – Petri nets are highly formal and thus mathematically analysable. However marks contained in such nets are not distinguishable and transitions fire as soon as marks are available in pre-located places. Over time, several extensions have been added to the original Petri nets and new variations -- High-level and timed Petri nets – arose. The main new features are:

- Tokens are distinguishable through attributes/colours/values. Furthermore they may be tested and evaluated by guards in the transitions.
- Temporal behaviour can be assigned to any Petri net element. Transitions may for example fire after a defined or stochastic period, marks' transportation may be retarted by the edges, etc.

In spite of the various extensions, even simple processes are still sometimes hard to model. Particularly significant scenarios for this work, where marks should be shared by several transitions simultaneously (Conflict, Fig. 2) and scenarios where marks are transported or transformed in factions (Fig. 3) are not representable with Petri nets.

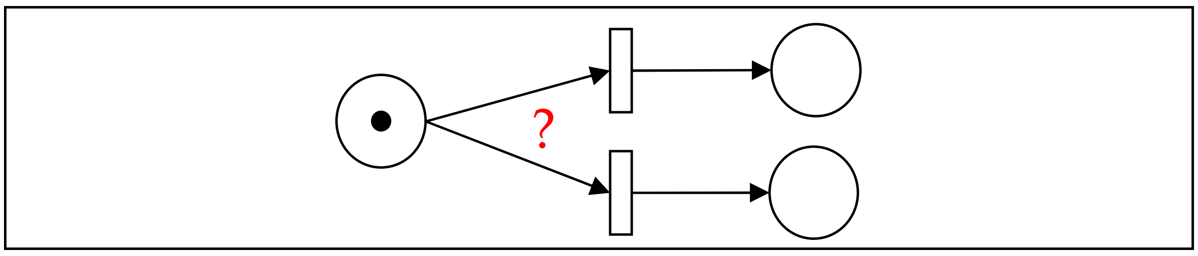


Fig. 2. Petri net: Conflict

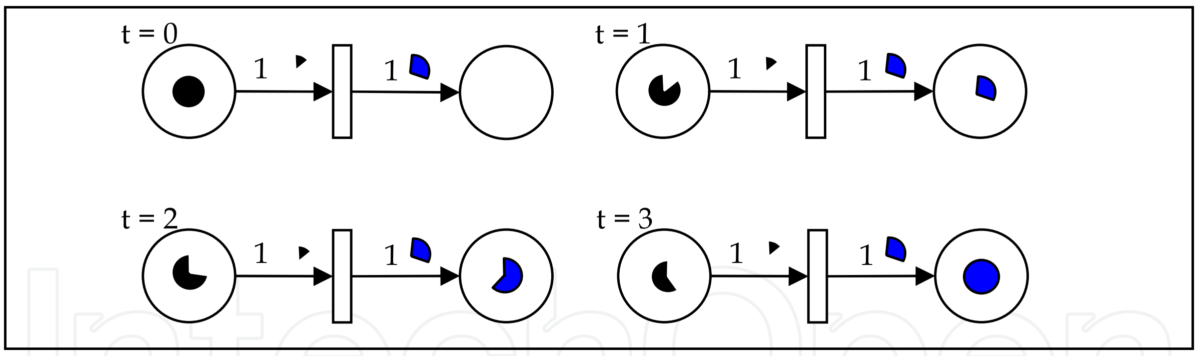


Fig. 3. Fractional Handling of marks

### 3.1.2 Ontologies

Ontology is a formal representation of knowledge within a specific domain. It not only specifies the syntax and the terminology used, but also and particularly it specifies the semantic binding of the concepts. Several languages are available to author ontologies. We opted for the OWL[6], a XML-derivative endorsed by the World Wide Web Consortium[7].

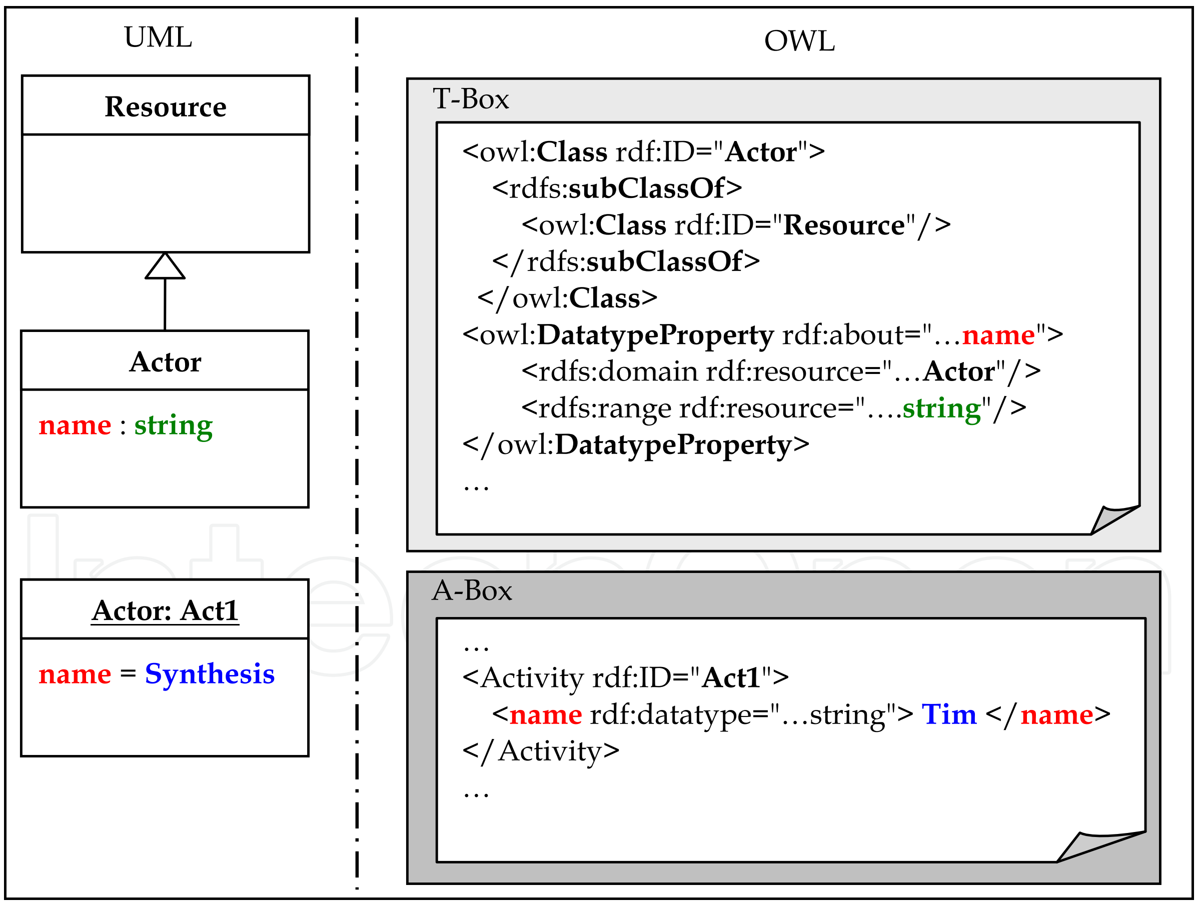
[6] Web Ontology Language, www.w3.org/TR/owl-features

[7]W3C, www.w3.org.

For illustration purposes, assume the following scenario to be modelled: A designer (actor) is a resource and may execute at most one activity at the same time. In contrast, an activity may be executed by several designers simultaneously, but at least by one. Finally, activities and actors are specified by names. In the following, a UML[8]-like notation is adopted to illustrate ontologies graphically.

The T-Box (Terminological Box) specifies the generic knowledge. It defines the concepts, e.g. Resource, Activity, DA, etc., their attributes and the relations and restrictions between the concepts. The A-Box (Assertions Box) contains concrete instances of the concepts, attributes and relations defined in the T-Box and hence, describe a concrete process to design a concrete DA using concrete resources/services. The semantic specified in the T-Box is assertive for the A-Box and defines how individuals are related to each other.

As shown in Fig. 4 and Fig. 5, the concepts (called Classes in OWL) to be modelled are Actor, Resource and Activity. To model a specific occurrence of the modelled scenario, e.g. Actors $A_1$ executing Activity $Act_1$, instances of the corresponding concept (Individuals) have to be created. The attributes (Datatype Properties) specify the concepts and accordingly their instances.



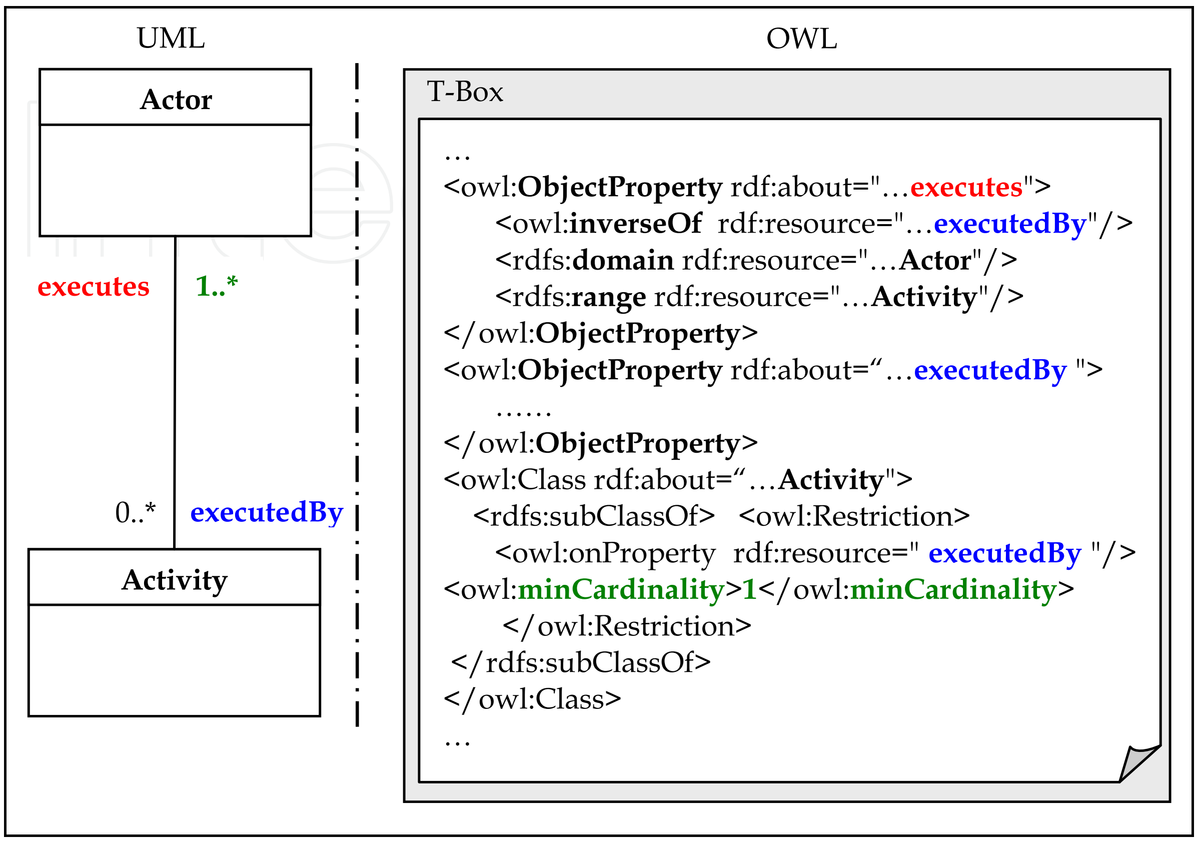
```
<owl:Class rdf:ID="Actor">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Resource"/>
  </rdfs:subClassOf>
 </owl:Class>
<owl:DatatypeProperty rdf:about="…name">
    <rdfs:domain rdf:resource="…Actor"/>
    <rdfs:range rdf:resource="….string"/>
</owl:DatatypeProperty>
…
```

```
…
<Activity rdf:ID="Act1">
  <name rdf:datatype="…string"> Tim </name>
</Activity>
…
```

Fig. 4. Modelling with ontologies: Concepts, Individuals and Datatype Properties

[8] Unified Modelling Language, *www.uml.org*

Relations (Object Properties) describe the semantic between the concepts. By means of cardinalities, relations may be refined and constrained (Fig. 5).



```
…
<owl:ObjectProperty rdf:about="…executes">
     <owl:inverseOf  rdf:resource="…executedBy"/>
     <rdfs:domain rdf:resource="…Actor"/>
     <rdfs:range rdf:resource="…Activity"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="…executedBy ">
     ……
</owl:ObjectProperty>
<owl:Class rdf:about="…Activity">
   <rdfs:subClassOf>  <owl:Restriction>
     <owl:onProperty  rdf:resource=" executedBy "/>
<owl:minCardinality>1</owl:minCardinality>
      </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
…
```

Fig. 5. Modelling with ontologies: Object Properties and Cardinalities

### 3.2. Modelling resources

A resource is modelled as provider of services. A service is requested and used or consumed by an activity. Services have the following attributes:

- *workpower*: denotes the quality and/or quantity of the service offered. This may be a single value as it is the case of the RAM offered by a computer or a mapping of values to targets, e.g the ability of a designer to carry out activities.
- *availability*: of the service in h/day.
- *intensity*: to which the service is portioned. Services may be delivered at the amount requested (~), e.g. RAM, or at a fixed amount (=) independently from the request, e.g. the automation level to which a tool automates activities.
- *repartition*: describes how services are shared in case of concurrent requests. Services may be not shareable and delivered to the first requesting (FCFS, First Come First Served), e.g. RAM allocation, or may be split equally ( $\forall$ =) or weighted according to the requests (%).

Furthermore, a cost model is assigned to each service expressing the costs per *workpower* and time unit entailed through using the service.

In some cases, services are not – literally – consumed but affect the way how activities are executed. Thus, assigning costs for example to the automation level of a tool or setting limits to the availability of such services is a counterintuitive way to address the accessibility to the resource itself and the costs generated. To avoid doing so, every resource offers the special service "Access". This service can be restricted in its availability and shareability (*workpower*). Costs caused by non-consumable resources can then be modelled by assigning costs to the corresponding Access service. The services described above cling to each service of a resource and may have different values within the same resource.

Fig. 6 and Fig. 7 depict the T-Box of the resource ontology and a corresponding tool A-Box.

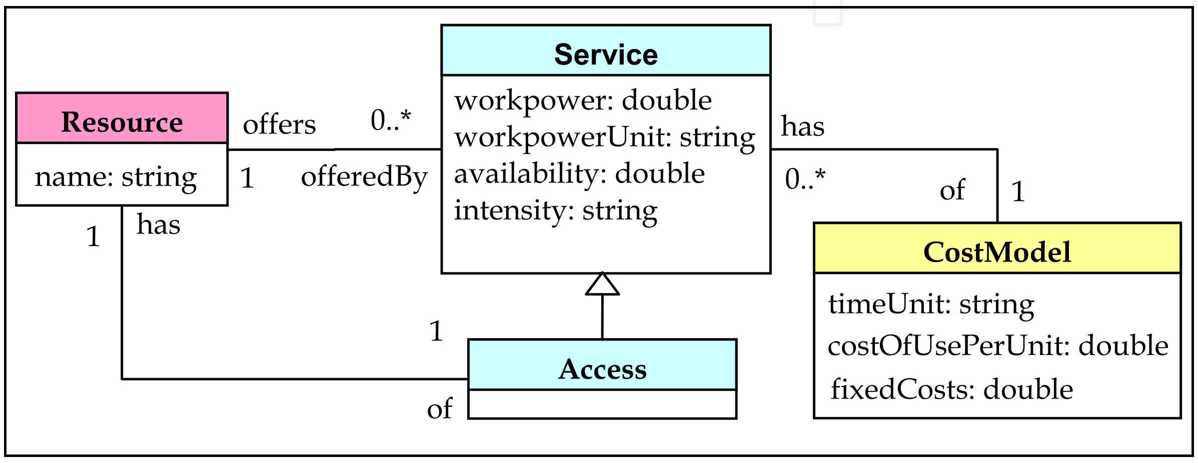


Fig. 6. Resource's T-Box

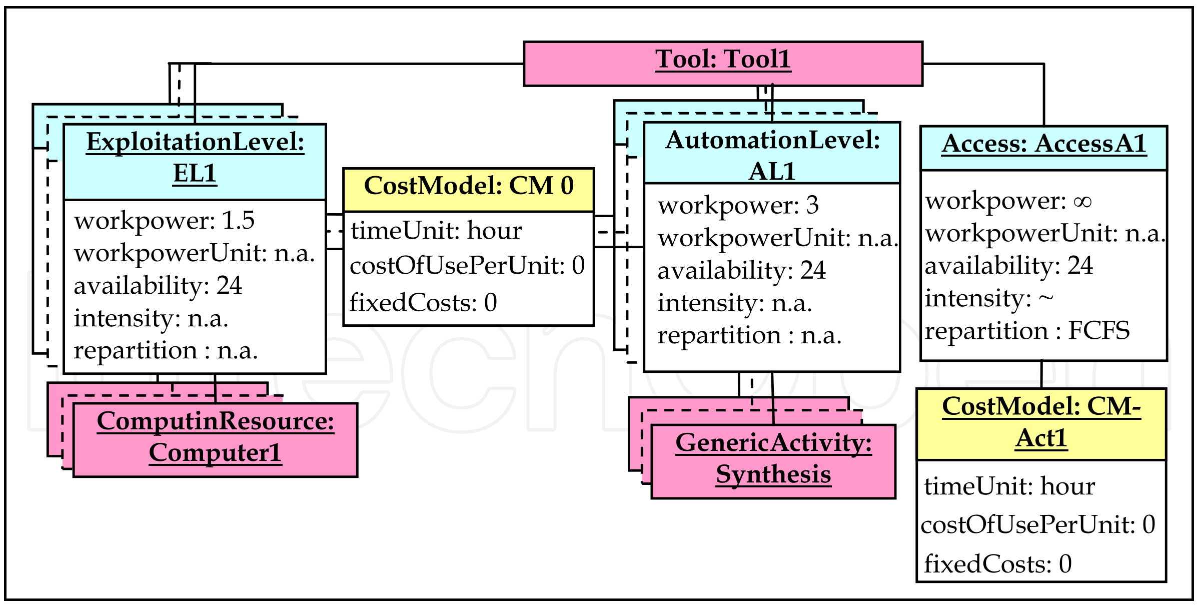


Fig. 7. A tool A-Box

A tool offers the service automating design activities to a certain extent (*AutomationLevel*) and the possibility to use a computing resource (*ExploitationLevel*). The latter allows for example for recognizing overdesigned computing resource as is the case for instance when a tool without multiprocessor capability runs on a multi-processor computer. Costs are generally not incurred by the tool itself but by the licenses. Similarly, simultaneous tool usages are also typically limited by the licenses.

For graphical representation purposes a Petri net-like notation is adopted (Fig. 8): Places represent resources and tokens residing in them stand for the different resource's services. The services' attributes are appended to the tokens (brackets). In case of several similar services with different values, services may be summarized in a table.

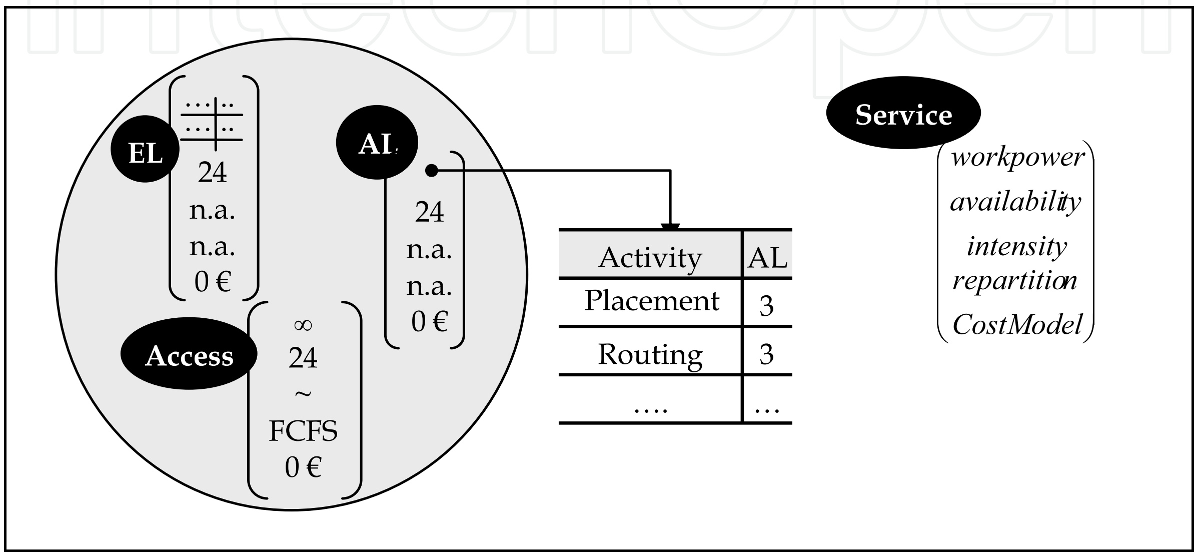


Fig. 8. Graphical representation of resources, exemplarily on the basis of a tool resource

### 3.3. Modelling design artefacts

In addition to the resources, the properties of the DAs and states being processed affect the duration of activities and the quality of their outcome to a decisive extend. DAs are described through two hierarchies of indicators and parameters: DAC (Design Artefact Complexity) addresses the technical characteristics of the DA whereas the DAQ (Design Artefact Quality) addresses its qualitative aspects. Both are applicable for the DA as a whole and are transitive for its states. They are typically approximated at the beginning of a design process and become more and more accurate as the process progresses. A Detailed description is given in (Leppelt et al., 2006).

The different states the DA undergoes within a CDP are specified through a format (e.g. text document, Verilog, DEF, GDSII, etc.) and a hierarchy of quality attributes (QA). In contrast to the DAC and DAQ, QA are state specific and not predefined. They may be adjusted to the user's needs to include parameters that are not defined in the DAC/DAQ.

Design artefacts are graphically represented similar to resources (Fig. 9, showing a cut-out of DAC and DAQ hierarchies). A place now stands for a design artefact and the tokens stand for the different DA states. The states' attributes (Format and the QA hierarchy) are appended to the tokens. Fig. 10 shows the design artefact's ontology.

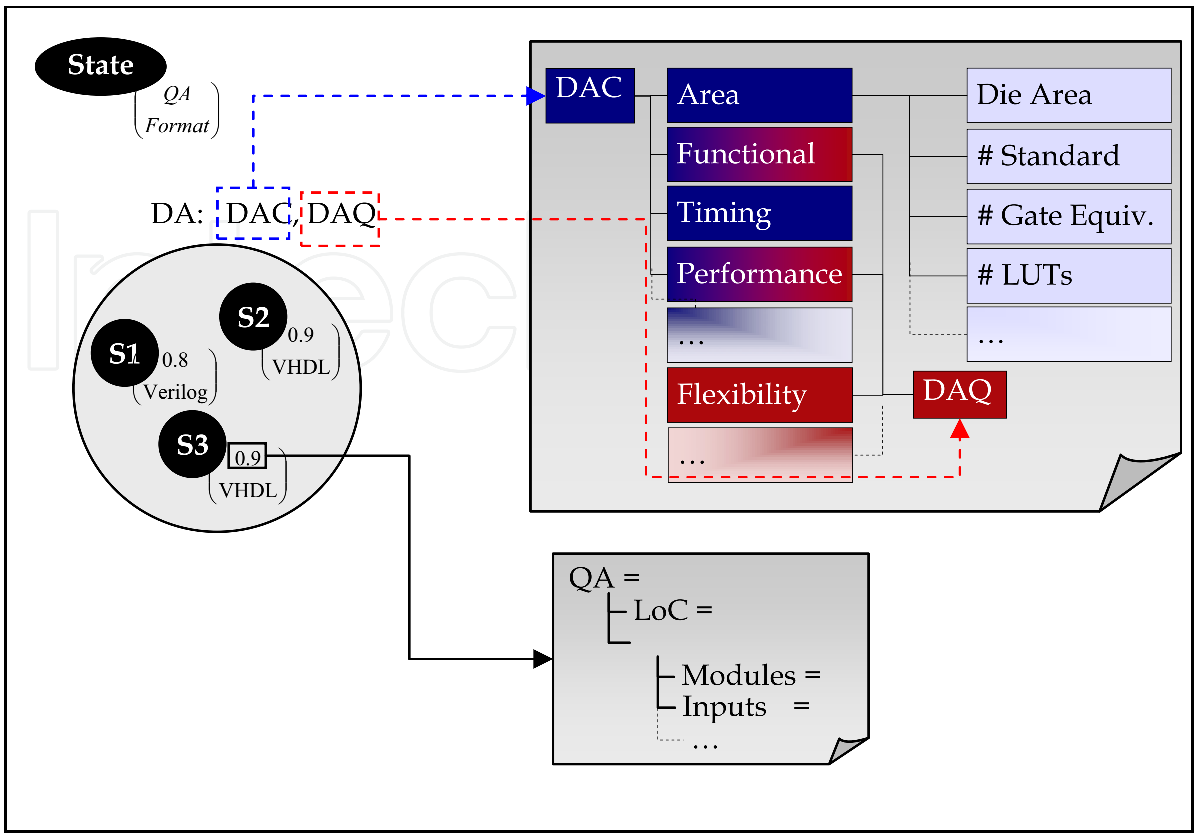


Fig. 9. A cut-out from the DAC and DAQ and graphical representation of design artefacts.

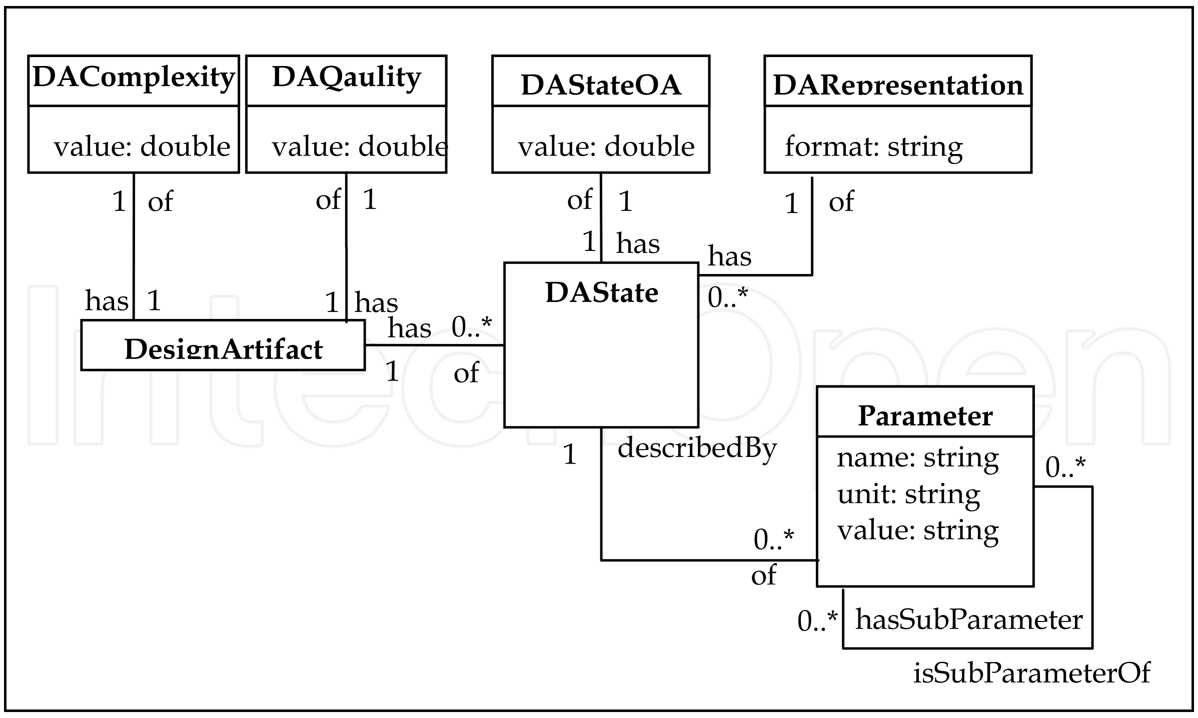


Fig. 10. Ontological representation of design artefacts

### 3.4. Modelling activities

Design artefacts are transformed within activities into different states. The services granted by resources and the properties of the DAs and states being transformed determine the duration of the activity and the quality of its output. The transformation behavior of an activity is composed of three parts:

- *Pre-Production (Pr-Pr)*: Before an activity starts, the input states have to be available (reqDA). Minimum amounts/qualities of services may be necessary to start (minReq) and dependent on the current case, more or less services are required (optReq). For example, in the case of a larger or a critical design more RAM or a more experienced designer may be required.
- *Production (Pr)*: Determines the time the activity takes, which states are produced and which quality results.
- *Post-Production (Ps-Pr)*: In case of insufficient outcome's quality, iterations may be initiated.

Activities are represented by transitions (rectangles) consisting of three blocks as described above (Fig. 11).

Beside the elements of a chip design process, i.e. resources, activities and DAs, transformation behaviours of activities – requests and calculation functions for activities' duration and quality dependent on resources/services and input DAs/states – must be formalized.

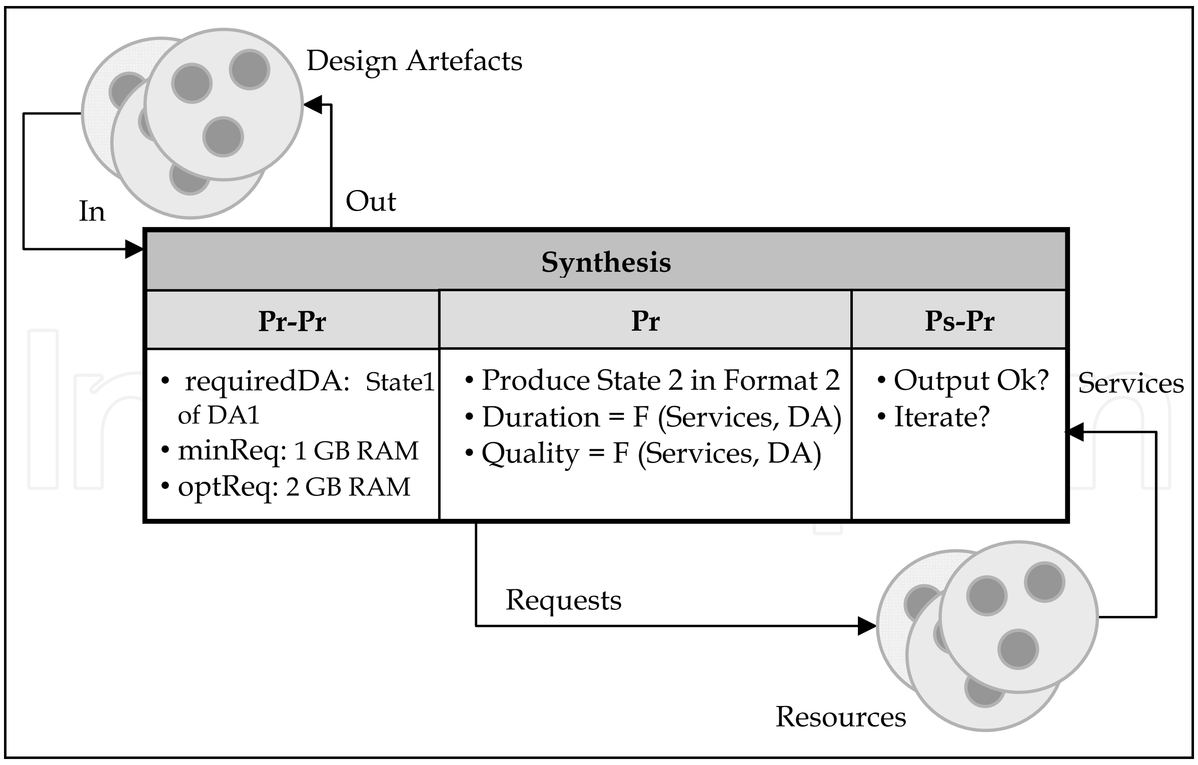


Fig. 11. Graphical representation of activities

It is very likely that such calculation models will often have to be revised, improved and adjusted. Therefore, they have to be expressed in a flexible way allowing for later changes without having to change the code of an implemented simulator. The simulator offers an interface to read in calculation models expressed in a specific developed notation – Prefix-Selector-Point (PSP) – from XML files. The developed notation generated from a context-free grammar $PSP_G$.

$PSP_G$ is a 4-tuple (T, N, S, P) where

- Terminals T = {. , ( , ) , [ , ] , ` , ' , value, instance, indicator, parameter} ∪ O ∪ R ∪ S ∪ A where
  O = {+, -, x , &, | , , /, >=, <=, =, . . . }: Operators
  R = {Quality, QA, Complexity, . . . }: Reserved words
  S = {Access, RAM, AbilitWrtTool, . . . }: Services
  A = {workpower, availability, . . . }: Attributes
- Non-terminals N = {OPERATOR, SERVICE, ATTRIBUTE, RESOURCE, COMPUTER, ACTOR, LICENSE, TOOL, SUPPORT, DALIBRARY, GA, DA, DASTATE, INDICATOR, PARAMETER, DAC, DAQ, STATEQA, OPTCRIT, FORMULA, OPERAND}
- Start symbol S = Formula
- Productions P={
  OPERATOR ::= + | - | x | & | / | …,
  SERVICE ::= Access | RAM | AutomationLevel | . . . ,
  ATTRIBUTE ::= workpower | availability | . . . ,
  RESOURCE ::= COMPUTER | ACTOR | LICENSE | TOOL | SUPPORT DALIBRARY,
  COMPUTER ::= instance,
  ACTOR ::= instance,
  LICENSE ::= instance,
  TOOL ::= instance,
  SUPPORT ::= instance,
  DALIBRARY ::= instance,
  GA ::= instance,
  DA ::= instance,
  DASTATE ::= instance,
  INDICATOR ::= indicator,
  PARAMETER ::= Parameter.Parameter | parameter,
  DAC ::= DA.Quality | DA.Quality.INDICATOR | DA.Quality.INDICATOR.PARAMETER,
  DAQ ::= DA.Complexity | DA.Complexity.INDICATOR | DA.Complexity.INDICATOR.PARAMETER,
  STATEQA ::= DA.DAState.QA | DA.DAState.QA.PARAMETER,
  OPTCRIT ::= Quality | Speed | Cost,
  FORMULA ::= (OPERATOR, OPERAND, OPERAND) | REQDA,
  OPERAND ::= value | FORMULA | OPERAND,OPERAND | RESOURCE.SERVICE.ATTRIBUTE |

```
                    RESOURCE.SERVICE[instance] | DA.DAC | DA.DAQ |
                    DA.STATEQA | Optimization.OPTCRIT,
        REQDA ::= DA.DAState|DA.DAState,DA.DAState,
    }
```

The terminals *indicator* and *parameter* are to be substituted through indicator and parameter names defined in the DAC, DAQ and QA. *instance* is to be substituted through names of concrete concept instances and *value* $\in \Re$. Calculation models and request rules are generated by applying the production rules of the $PSG_G$ and substituting the corresponding variables. The sentences derived from the $PSP_G$ typically have the form (operator, operand, operand) where operand may represent a point separated address.

To illustrate, assume for example that in order to start, an activity requires that the assigned computer $Comp_C$ offers at least 1 GB free RAM and that the assigned License $Lic_L$ is adequate for using tool $Tool_T$ (*minReq*). In the following, the derivation of the corresponding phrase in the PSP notation is described. The highlighted words are non-terminals to be substituted in the next step by applying a $PSP_G$ production rule:

minReq: Formula ⇒ (**Operator**, Operand, Operand)
⇒ (&, **Operand**, Operand)
⇒ (&, **Formula**, Operand)
⇒ (&, (**Operator**, Operand, Operand), Operand)
⇒ (&, (>=, **Operand**, Operand), Operand)
⇒ (&, (>=, **Resource.Service.Attribute**, Operand), Operand)
⇒ (&, (>=, instance.RAM.workpower, **Operand**), Operand)
⇒ …
⇒ (&, (>=, instance.RAM.workpower, value), **Operand**)
⇒ (&, (>=, instance.RAM.workpower, value), **Formula**)
⇒ …
⇒ (&, (>=, instance.RAM.Workpower, value),
(=, instance.AllowsUsing[instance], value)
)

Replacing the terminals *instance* and *value* through instances' names and figures results in the concrete condition to start the activity in question:
(&, (>=, $Comp_C$.RAM.Workpower, 1), (=, $Lic_L$.AllowsUsing[$Tool_T$], 1))

To allow generic formulas, the PSP notation is extended to permit the use of generic entries like [Tool] instead of the instance name $Tool_T$. The assignment of a concrete tool is thus done by the simulator itself based on the process entered.

## 4. The RS Simulator

Adrenalin is a simulator implemented in Java based on the RS model. It offers different views for data entry, a simulation and a charts view for simulation results. The data entry is guided through the semantic specified in the ontologies and thus guarantees a semantically correct entry. Activities' behavioural models are read in from an auxiliary XML file.

Simulation results are also logged allowing for a transparent backtracking of simulation runs. Fig. 12 illustrates the setup of the simulator.

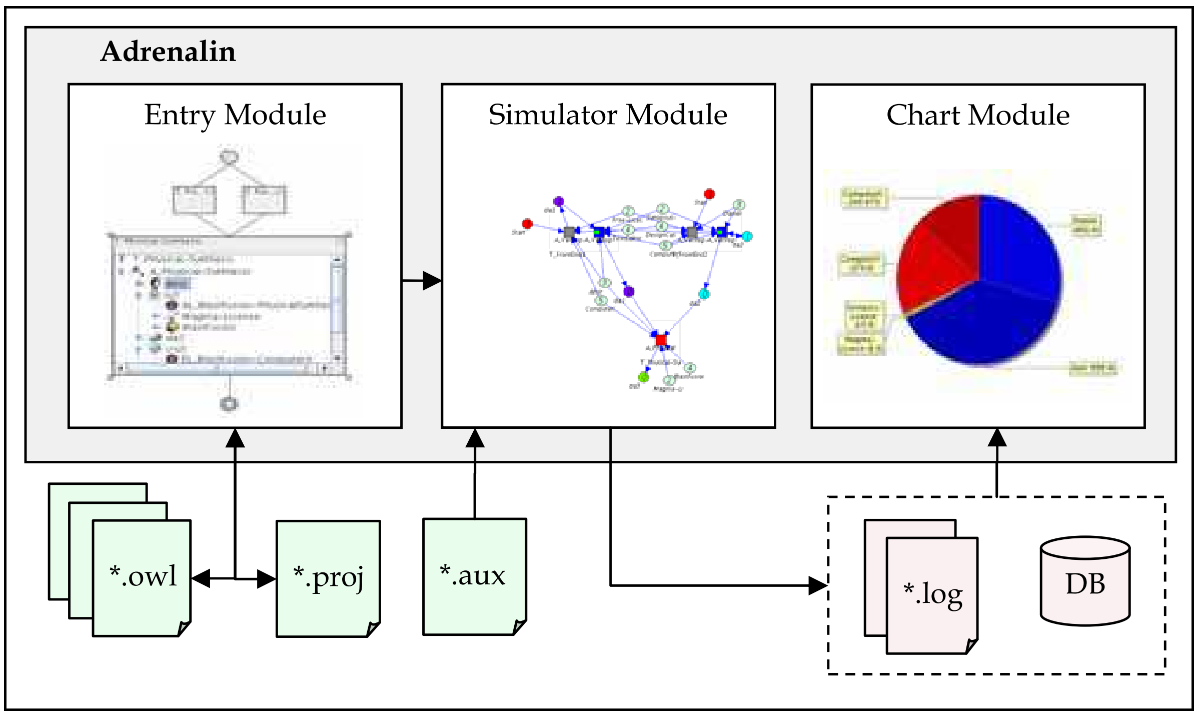


Fig. 12. Setup of the Adrenalin simulator

The simulator allows users to answer the most urgent questions about process duration and costs and to compare alternatives to recognize potential bottlenecks and identify critical factors such as deadline over-run, resource overload or wastage:

1. How much time would the process take?
2. How much would it cost?
3. Which quality would be reached?
4. Which resources arrangement is appropriate to handle a given design complexity?
5. Which design complexity is manageable with a given resources arrangement?

In the present implementation, the resources allocation to activities is considered as given. The simulation can then deliver answers to the first three questions. The other questions can be addressed by manually adjusting the process, re-allocating resource and re-simulation.

Fig. 13 depicts a basic simulation step. Each step is made up of four phases:

- Check phase: Activities check availability of needed input DA states (*Pr-Pr*).
- Request phase: Activities send out their requests.
- Grant phase: If *minReqs* are satisfiable, resources grant activities as much as possible from the requested services (*optReqs*) taking into account their services properties, e.g. shareability.

- Produce phase: Activities produce a portion of their output (*Pr*). After the whole production finishes (*Ps-Pr*) and dependent on the achieved quality, activities decide about initiating iterations or writing out corresponding DA states.

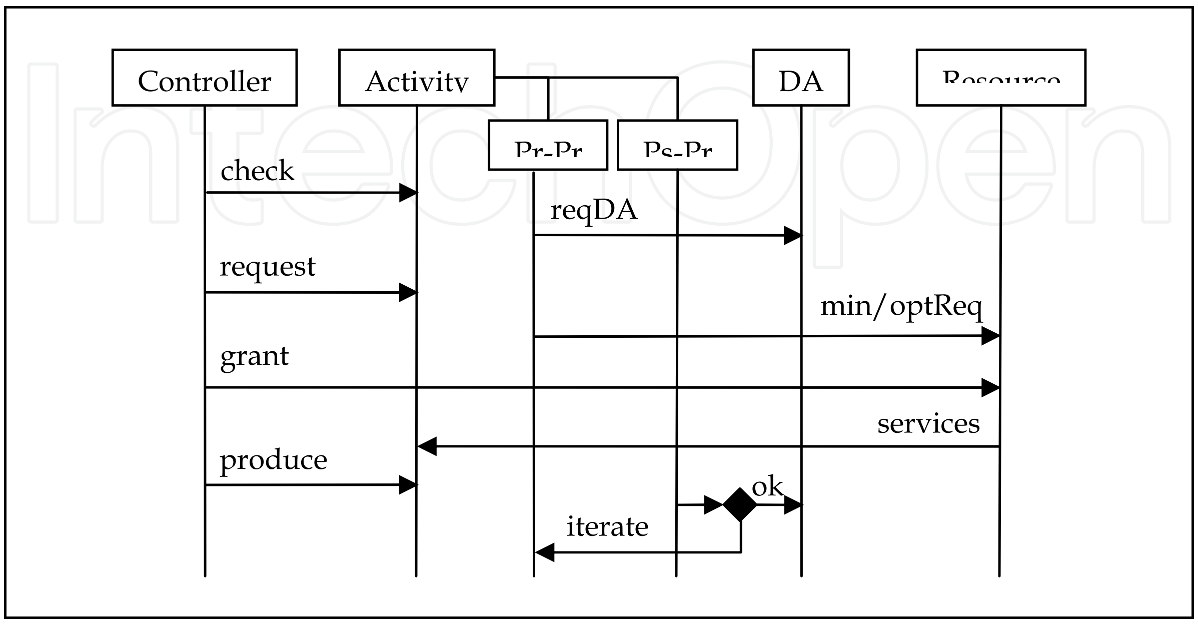


Fig. 13. Principal simulation steps

## 5. Calibration and results

Calibration of the simulator, e.g. calculation models for activities' duration and quality, is a continuous and recursive process. To illustrate, three data sets have been investigated and integrated into the simulator by means of the PSP notation to determine behaviour of some activities.

$Set_1$ is composed of five digital designs used by an industry partner to configure new tools or tool versions: A scalable design in four different sizes ranging from 200 to 1,600 kGates and a 42 kGates sized design. Duration, memory usage and normalized CPU factors (CPUF) are available. $Set_2$ consists of data for logical synthesis of 31 industrial digital designs that have been or are intended to be produced. The design codes contain up to 70,000 lines. $Set_3$ is composed of nine free downloadable[9] non-synthesized verilog codes of digital designs with lines of code (LoC) ranging between 2,300 and 94,000 lines.

From each set, some designs were not used in the investigation. Untried designs served as a benchmark to evaluate the accuracy of the determined metrics. The statistical computing language R[10] has been used to determine metrics to predict

- memory usage and duration of some physical synthesis activities dependent on design size and CPUF ($Set_1$) and

[9] Opencores, www.opencores.org.
[10] The R Project for Statistical Computing, www.r-project.org.

- logical synthesis' duration and the number of nets generated thereby dependent on code's design attributes and the CPUs of the computing resources used ($Set_2$ and $Set_3$).

The metrics determined for $Set_1$ show very accurate prediction of activity duration and memory usages. However four of the five investigated designs represent a scalable design and thus, the metrics are primarily relevant for comparable designs. $Set_2$ and Set $_3$ which contain more heterogeneous data also produce accurate predictions especially for longer activities and more complex designs (Fig. 14).

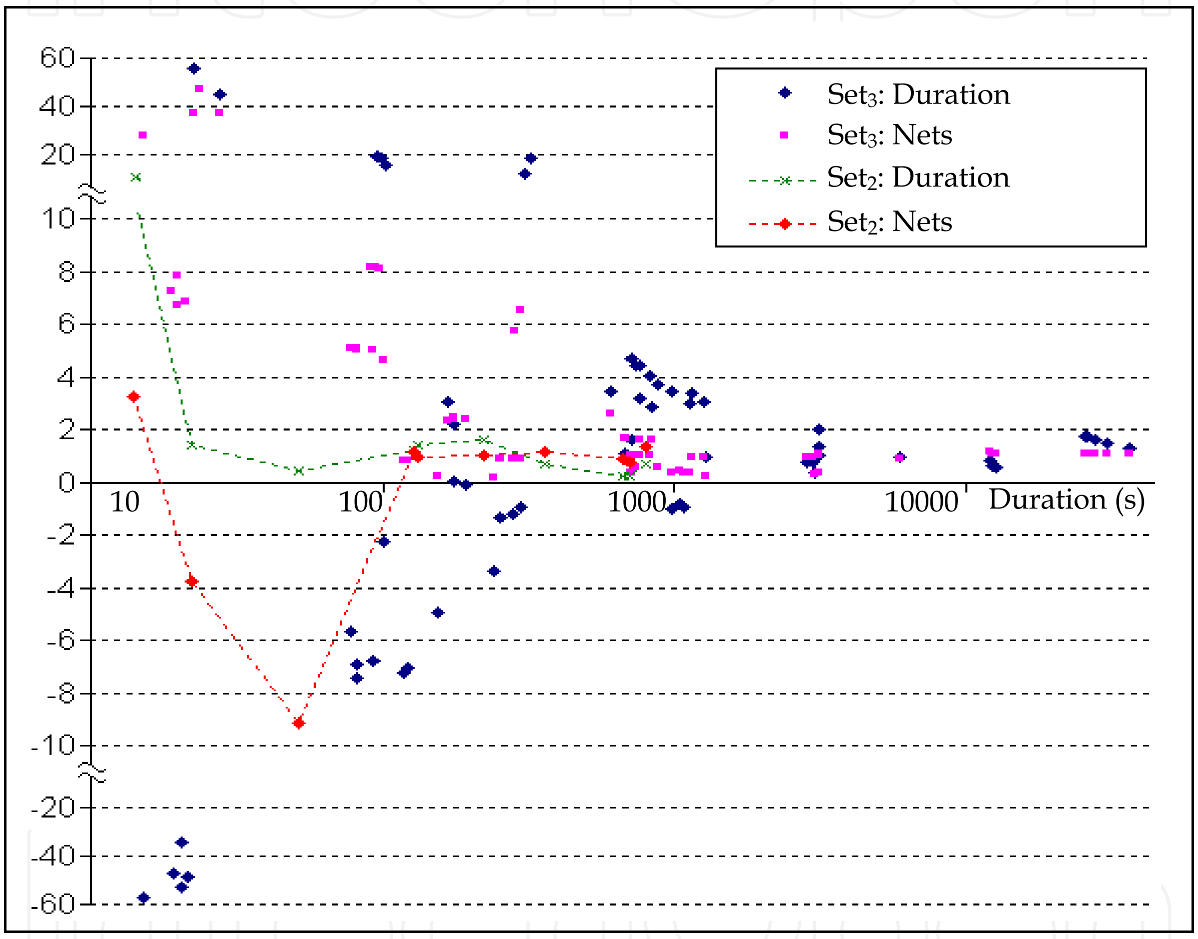


Fig. 14. Ratio of predicted to real values for $Set_2$ and $Set_3$

## 6. Conclusion

This work addresses the lack of approaches to model and simulate chip design processes. In cooperation with leading semiconductor industries, a simple though expressive model to represent design processes has been developed. The model is endowed with the formalism necessary to allow for computer aided simulation. Adrenalin, the simulator based on the RS model, gives users the possibility to try several process alternatives and compare them to each other in a simulative and thus cheap and fast way.

Generating calculation models is a continuous and recursive process. The interface offered by Adrenalin to read in calculation models from XML files and the formalized PSP notation allow for later adjustment without editing the simulator code.

The exemplary investigation of some industrial data has produced accurate results in estimating activity durations and thus attests to the predictability and ability to simulate chip design processes.

Future work will continue to generate further calculation models using more data for calibration and combining further input variables to substantiate the metrics. Furthermore, the focus will be set on automating re-arrangements of planned processes in order to determine controllable complexity and needed resources in case of given resources and complexity respectively.

## 7. References

Goodall, A.; Fandel, D., Alan, A.; Landler, P. & Huff, H. R. (1998). Long-term Productivity Mechanisms of the Semiconductor Industry, *Proceedings of American Electrochemical Society Semiconductor Silicon*, pp. 125–143

Hassine, A. & Barke, E., (2005). Measure Your Design Value to Improve It, *Proceedings of IEEE International Engineering Management Conference*, pp. 668–672

Collett, R. (2004). Benchmarking IC Development Capability - Why?, *Fabless Forum,* Vol. 11

Numetrics (2006), IC Product Lifecycle Management and Portfolio Optimization - Critical Elements of an Enterprise Solution, *White Paper, Numetrics Management Systems Inc.*

Numetrics (2000). Measuring IC and ASIC Design Productivity, *White Paper, Numetrics Management Systems Inc.*

Leppelt, P., Hassine, A. & Barke, E. (2006). An Approach to Make Semiconductor Design Projects Comparable, *Proceedings of Asia Pacific Industrial Engineering Management Systems Conference*, pp. 2067–2074

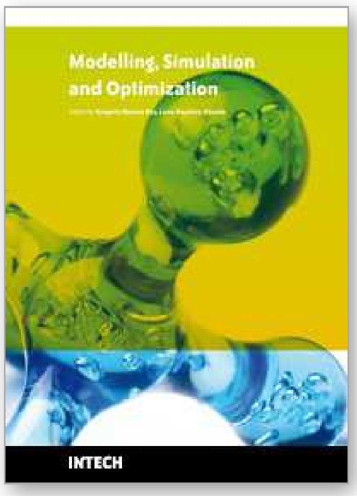
Fenstermaker, S.; George, D.; Kahng, A.,Mantik, S. & Thielges, B. (2000). METRICS: A System Architecture for Design Process Optimization, *Proceedings of IEEE Design Automation Conference*, pp. 705–710

Kahng, A. & Mantik, S. (2001). A System for Automatic Recording and Prediction of Design Quality Metrics, *Proceedings of International Symposium on Quality Electronic Design*, pp. 81–86

Sohnius, R.; Ermolayev, V.; Jentzsch, E. & Matzke, W.-E. (2007). An Approach for Assessing Design Systems: Design System Simulation and Analysis for Performance Assessment, *Proceedings of International Conference on Enterprise Information Systems*, pp. 231-236

Matzke, W.-E. & Strube, G. (2006). A Management Tool for the Performance Management of Distributed (global) Dynamic Engineering Design Processes, Proceedings of IEEE International Engineering Management Conference, pp. 146-151

Ermolayev, V.; Jentzsch, E.; Karsayev, O.; Keberle, N.; Matzke, W.-E.; Samoylov, V.; & Sohnius, R. (2006). An Agent-Oriented Model of a Dynamic Engineering Design Process, In: *Agent-Oriented Information Systems III*, Vol. 3529/2006, pp. 168-183, Springer Berlin/Heidelberg



**Modelling Simulation and Optimization**
Edited by Gregorio Romero Rey and Luisa Martinez Muneta

ISBN 978-953-307-048-3
Hard cover, 708 pages
**Publisher** InTech
**Published online** 01, February, 2010
**Published in print edition** February, 2010

Computer-Aided Design and system analysis aim to find mathematical models that allow emulating the behaviour of components and facilities. The high competitiveness in industry, the little time available for product development and the high cost in terms of time and money of producing the initial prototypes means that the computer-aided design and analysis of products are taking on major importance. On the other hand, in most areas of engineering the components of a system are interconnected and belong to different domains of physics (mechanics, electrics, hydraulics, thermal...). When developing a complete multidisciplinary system, it needs to integrate a design procedure to ensure that it will be successfully achieved. Engineering systems require an analysis of their dynamic behaviour (evolution over time or path of their different variables). The purpose of modelling and simulating dynamic systems is to generate a set of algebraic and differential equations or a mathematical model. In order to perform rapid product optimisation iterations, the models must be formulated and evaluated in the most efficient way. Automated environments contribute to this. One of the pioneers of simulation technology in medicine defines simulation as a technique, not a technology, that replaces real experiences with guided experiences reproducing important aspects of the real world in a fully interactive fashion [iii]. In the following chapters the reader will be introduced to the world of simulation in topics of current interest such as medicine, military purposes and their use in industry for diverse applications that range from the use of networks to combining thermal, chemical or electrical aspects, among others. We hope that after reading the different sections of this book we will have succeeded in bringing across what the scientific community is doing in the field of simulation and that it will be to your interest and liking. Lastly, we would like to thank all the authors for their excellent contributions in the different areas of simulation.