INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED11
15 - 18 AUGUST 2011, TECHNICAL UNIVERSITY OF DENMARK

AUTONOMOUS VISUALIZATION AGENTS TO ENHANCE THE ANALYSIS OF VIRTUAL PROTOTYPES

Rafael Radkowski[1], Jürgen Gausemeier[1]
(1) Heinz Nixdorf Institute, University of Paderborn

ABSTRACT

A virtual prototype specifies the shape and the behavior of a product under development. However, it is a computer-internal representation of this product and suitable visualization techniques are necessary to understand the structure and behavior of it. Today visualization techniques are selected manually by the developers; that is time consuming and error-prone. In this paper, we introduce the concept of autonomous visualization agents: software agents that select autonomously a proper visualization for a certain task. To realize such agents an agent model to represent the knowledge of the agent, as well as a reasoning mechanism are necessary; both are introduced. Experiments have been carried out to test the correctness and the usability of the models and the reasoning mechanism. The experiments show that the models and the reasoning mechanism facilitate the autonomous selection of visualizations by software agents.

Keywords: Virtual Prototyping, Software Agents, Resource Description Framework

1 INTRODUCTION

A virtual prototype (VP) is the computer internal representation of a real prototype or product [1]. It is based on the digital mock-up (DMU). The DMU represents the shape and the structure of the product. Two kinds of models are the base of the DMU: 3D-CAD models and the logical structure of the product. The VP extends the DMU by further aspects that are taken into account, aspects like the kinematic, dynamic, tension or information processing. A computer-internal model represents each of these aspects.

Today, VPs are an essential element during the development of technical products. Technical products become more and more complex and cannot be developed without computational assistance. VPs facilitate the analysis of technical systems. They help the engineer to understand the system, to ensure the functionality and finally to develop the system. Thus, long time before the first prototype will be built. VPs are integrated into virtual environments (VE) in order to analyze and evaluate their behavior. A VE is a computer-generated environment, which simulates and presents the visual, haptic and auditive features of the area of operations of the VP. [1], [2]

However, the analysis of a VP requires proper visualization to show the shape of the VP and to explain its behavior. The amount of data, generated by a VP is too large. This makes an analysis without any further visualization techniques impossible [3], [4].

Nowadays developers themselves select the necessary visualization, and design the entire visual interface. Common software tools for the analysis of VPs facilitate the interactive assembly of graphical user interfaces: a user can select a visualization from a set of visualizations. That way the user can assemble an own graphical interface following the concept of plug & play. Unfortunately this proceeding has two disadvantages:

- First, a user can select an inadequate visualization for a certain task. Particularly by multivariate methods of analysis the user gets to his limits: The resulting graphical interfaces become unclear and overstated. Thus, significant events can be overseen on the screen.
- Second, the user, mostly a developer of a technical system, needs to spend time and effords for that task. Meanwhile, they cannot focus on their intrinsic development task.

To overcome these disadvantages, we introduce the concept of autonomous visualization agents. Visualization agents are software agents that act as an expert for a certain visualization in a virtual environment. They represent a certain visualization, they “know” their visualization and the task that can be processed by it. In the context of software agents, “knowing” means an automatic interpretation

of data, which leads to an autonomous actions [5]. The visualization must be prepared and saved in an appropriate data structure. The visualization agents select the visualization automatically if, they decide as representative for the user. A simple example points this out: If a VP violates a desired threshold value, the visualization agent autonomously shows a visualization that explains this violation and its reasons. An agent model represents the necessary knowledge for this behavior. For this model a common Semantic Web Techniques is used: the Resource Description Framework (RDF). RDF-based models represent the needed knowledge as agent-internal models.
This paper introduces the concept of the visualization agents and presents the models that facilitate the autonomous selection of visualizations. It is structured as following: First some related work is presented and RDF is introduced. Section three explains the concept of visualization agents. Then some first results are presented. The paper closes with a summary and an outlook.

2 RELATED WORK

The related work is separated into two sections. First the related work in the field of software agents supported engineering is reviewed. The second part introduces the utilization of RDF to model technical systems.

2.1 Agent-supported engineering

In engineering, software agents are utilized in many different application fields. Mainly agents are used to support the design process by making decisions, which are based on a large amount of data. In the following a choice of agent application and frameworks is introduced that drive the concept of visualization agents.
Mendez et al. describe an agent-based software architecture for agents in virtual environments [6]. They introduce the concept of expert agents. Expert agents are software agents with an expert knowledge in a specific technical domain. Based on this knowledge, the expert agent is capable to find a solution to solve a specific problem. The paper introduces a similar idea: the authors propose to use software agents with a domain specific knowledge to fulfill tasks. However, their desired tasks are training tasks.
Galea et al. present a framework for an intelligent design tool that assists a designer while working on micro-scale components [7]. They do not label their framework as software agent, but they use a similar artificial intelligence technique to model the knowledge and the reasoning system: production rules of type if (condition) then (action) to get an automatic decision.
Multi-agent systems have been used to support engineers in time-critical tasks, too [8]. An agent aggregates relevant information from other agents that represent different members of an engineering team. Thus, an engineer gets the right information at the right time.
Baolu et al. propose the so-called Multi-Agent Cooperative Model (MACM) [9]. It is a product design system that facilitates an easy access to similar data of different products. The system facilitates the product design and manages product data. With its aid the product design cycle will be shortened.
Geiger et al. introduce the agent modeling language SAM (Solid Agents in Motion), a language to describe 3D models in virtual environments and their behavior. This work is similar to our approach. However the agents generate rule-based animations to explain the kinematic behavior [10].
In summary, software agents support the engineer by gathering and processing data and supporting them within time-critical or fuzzy decisions. A concept for a visualization agent, like it is proposed in this paper, has not been realized yet.

2.2 Semantic Web and the Resource Description Framework

The Semantic Web (SW) facilitates machines to “understand” the content of web pages and other similar documents [11]. Thus, the machines can automatically link the information of different sources. The Resource Description Framework (RDF) plays a decisive role for the SW. RDF is a description language, which is used to annotate the content of a web page; it is syntax for meta-data of a web page. The underlying model is based on a directed graph. The nodes of the graph are denoted as resources, the edges are denoted as properties. The idea of RDF to is describe complex facts by a network of simple RDF statements. A RDF statement consists of a subject, a predicate and an object. Figure 1 shows an example. It shows an RDF statement, which describes a digital business card of a person named Karl Mustermann.

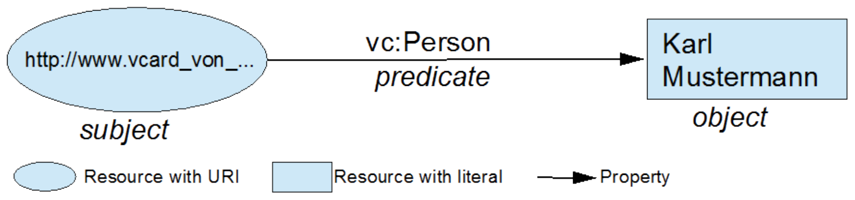


Figure 1: RDF statement of a business card

The subject is a Unified Resource Identifier (URI). It is linked to the data of the business card. The object is the name of the person, who is related to the business card. The predicate expresses the relation between the subject and the object. Here the predicate *vc:person* is used. The predicate is the most important thing of the semantic. It is a W3C (World Wide Web Consortium)-standardized predicate for the description of business cards. The SW works only, if everybody has the same understanding of these predicates and interprets them in the same manner.

A reasoning system is necessary to identify relations between two RDF-annotated web pages. RDF represents the database only. For that purpose query languages are used to query the necessary information. The queries need to be prepared in a way, that reasoning is possible by production rules. Therefore, queries like select [object] where [predicate] as well as a set of if…then clauses are used.

Some researchers have already used RDF and the related reasoning mechanism for the engineering of technical systems. For instance, Bludau and Welp have developed a framework, which supports engineers during the development of mechatronic systems [12]. Their framework searches for active principles and solution elements, which meet a given specification. Restrepo uses the SW techniques to search for design solutions for a given problem [13]. He has developed a database, which contains different design solutions; every solution is annotated by RDF. A reasoning mechanism searches for solutions, which meets the given design problem. Ding et al. use XML-based annotations to annotate CAD models with design constrains, goals, relationships, and bounds [14]. They mainly annotate geometric, topological and kinematic properties of a given design. Their approach can be utilized to find an optimal design solution during the product development process. The authors use XML as notation basis, but their notation is similar to a RDF notation. Ding et al. developed an XML-based product representation that allows an annotation of geometrical properties, too [15]. Li et al. present a classification of different annotation approaches [16]. In summary, the work supports the importance of annotation techniques in engineering design.

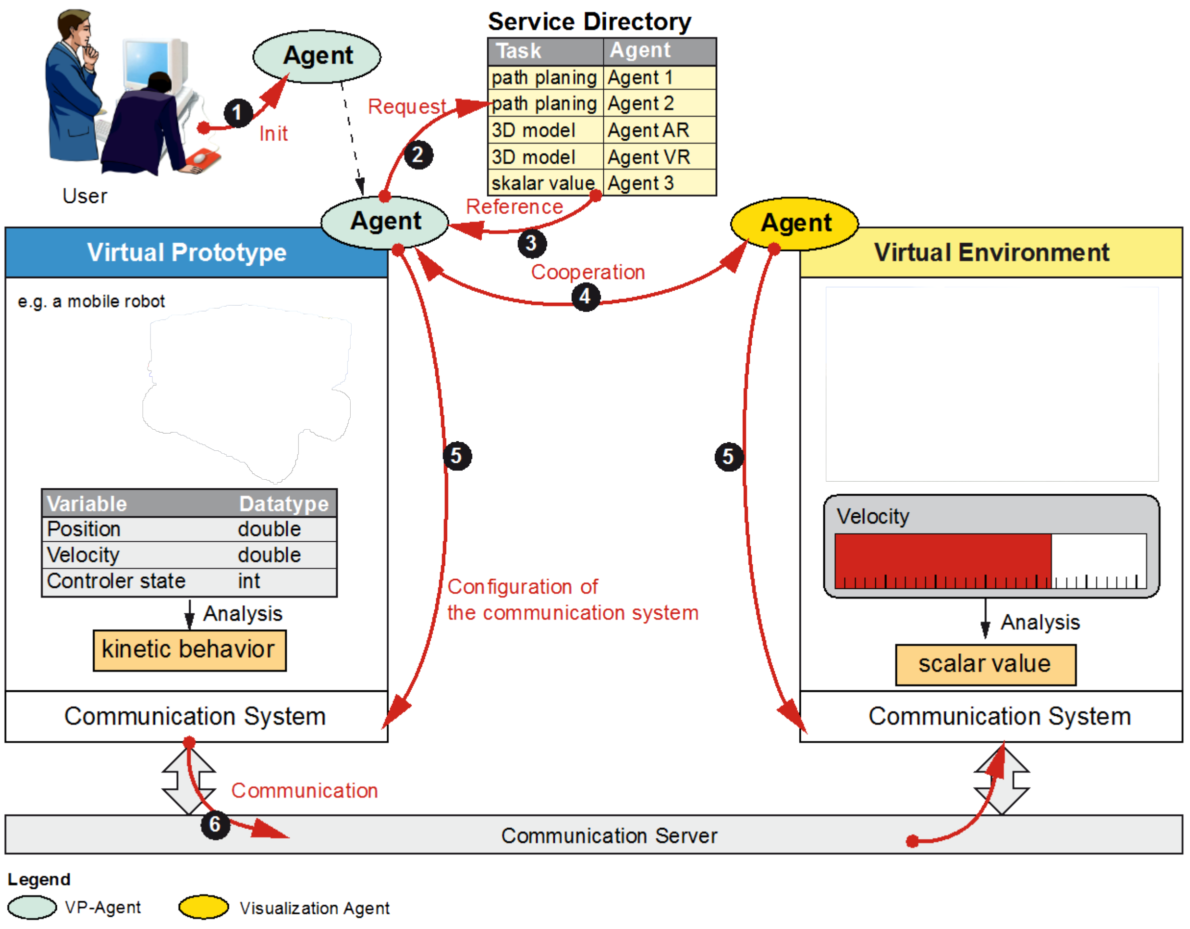
RDF and the reasoning mechanism from the field of SW do not have been utilized for autonomous identification of visualizations by software agents. The presented approaches are a good starting point and are used for the development of the visualization agents.

3 CONCEPT OF THE VISUALIZATION AGENTS

This section describes the concept of visualization agents. Two different agents with two different roles exist: One agent that represents the virtual prototype and a second agent, which represents the visualization. The entire agent platform is formed by a set of agents. They communicate and interact and finally by this, they form the virtual environment. Each agent contains an internal data model. This data model describes the represented object like meta-data. The models are based on the RDF notation. Our approach is similar to the approach that is utilized by the Semantic Web (SW): at the SW, web pages are annotated by an RDF notation. A reasoning mechanism compares the annotations of different web pages and identifies similarities. Our approach: we use RDF to describe the VP, the visualization, and the task of the user. A reasoning mechanism identifies similarities, too. If the models are similar, we assume that the visualization is suitable to explain the data of a VP. In the following, an overview of the entire concept is presented. Then the necessary agent models are described and finally, the reasoning mechanism is introduced.

3.1 Overview

Figure 2 shows a schematic overview of the function principle. On the left side of the figure a box represents a VP of a mobile robot. The box on the right side indicates a virtual environment. The environment contains the visualizations. A software agent represents both the VP and the visualization. To simplify the further reading the agent of the VP is denoted as VP-agent. The visualization agent is denoted as Vis-agent.



Service Directory

Task	Agent
path planing	Agent 1
path planing	Agent 2
3D model	Agent AR
3D model	Agent VR
skalar value	Agent 3

Variable	Datatype
Position	double
Velocity	double
Controler state	int

Figure 2: Concept of the visualization agents

The objective of both agents is to identify an appropriate visualization by communication and cooperation. This visualization should help a user to accomplish a certain task. At the beginning, the VP-agent does not know any visualization agent in the virtual environment. In the following the activities are explained, which are necessary to identify a suitable visualization. The steps are labeled with the numbers 1 to 6 (Figure 2).

In the first step (1), a user needs to initialize the VP, the simulation, and the virtual environment. At the same time the agent platform is initialized and the agents start to operate. The user needs to specify the task, which the user wants to carry out, e.g. to *analyze the kinetic movement* or to *inspect the parts of the VP*.

In the second step (2), the VP starts to search for an appropriate visualization for the VP and its data. For this purpose the VP-agent contacts a service directory and queries for reachable Vis-agents. The virtual environment provides this service directory. It contains a list of all reachable agents, sorted by a category of tasks. The VP-agents submit a desired category, which meets the kind of visualization the VP-agent searches for. Normally, more than one visualization facilitates the visualization of the data of the VP. Thus, in the third step (3), the VP-agent receives a list of potential Vis-agents.

In the fourth step (4), the VP-agent contacts each Vis-agents, which reference it receives from the service directory. Thereby it submits data about the functions of the VP and data about the task the user desired to apply to every Vis-agent. Each Vis-agent compares this data with two internal data models. These data models characterize the capabilities of a Vis-agent. A similarity-vector E_{vis} is calculated. This vector and its numerical values represent the capability of the agent to visualize the queried task and data. This vector is returned to the VP-agent. At the end of this step the VP-agent has a set of similarity-vectors, one from every Vis-agent.

Next, the VP-agent compares these different similarity-vectors and by this, it compares the different Vis-agents. Therefore, a reasoning mechanism is used. After the VP-agent has decided for one Vis-agent (4) they start to cooperate.

In the fifth step (5), the visualization is realized. Therefore, the data of the VP needs to be submitted to the Vis-agent and its represented visualization. Figure 2 shows a simple example: the VP has a value "velocity" that needs to be visualized. The Vis-agent on the right side can visualize this by a bar chart. In order to do this, the data "velocity" needs to be sent to the Vis-agent. To realize this data exchange, a communication server is used [17]. This communication server managed the data exchange between different connected programs. In this case: a program that simulates the VP and its behavior and a virtual environment that hosts the visualization (6). The task of both agents is to configure this communication server and by this, to configure the data exchange. The VP-server informs the communication server about the attributes it wants to allocate. The Vis-agent informs the server, what data it wants to receive. If the requested data are available, the exchange of the data starts until an agent stops its operation.

3.2 Data models

Altogether the agents utilize three different data models to represent their knowledge: a so-called task model, a function model, and a visualization model. The VP-agent uses the task model and the function model, the Vis-agent uses a task model and a visualization model. The models base on the RDF-notation. The challenge during the development of these models was to identify objects and attributes, which facilitate an automatic identification of capable visualization: what should be annotated?

3.2.1 Task Model

A task is defined as *"the application of methods, techniques, and tools to add value to a set of inputs – such as material and information – to produce a work product that meets fitness for use standards established by formal or informal agreement [18]."* A common technique to specify a task is a block diagram where each block represents a certain activity and the entire diagram represents the task (Figure 3). A string inside the block denotes the activity, e.g. "Check the impulse response". The arrows that lead into the block represent input data. Input data are objects or information, which are processed during the activity. An information can be the velocity of a mobile robot for instance, an object of the computer-internal representation of the shape of the virtual prototype. The arrows that leave the block represent the output object and output information. In addition, an activity includes a method and a tool. The attribute method names the utilized method itself, the systematic proceeding of the activity; for instance a parameter identification. The tool names an additional software tool, which supports the user during the activity. To concretize the task, boundary conditions can be specified. For instance, this can be a minimal necessary amount of data.

To describe this task model as computer-internal representation, an RDF-notation has been developed. To develop this notation, a set of resources and properties need to be defined. The set of resources and properties describes the task. Figure 3 shows an extract of the resulting RDF-notation for one action. It simplifies the entire RDF schema in order to point out how a task is modeled. The following resources and properties are used:

- Activity (1): The activity itself is the main element. It is specified by a resource, which keeps a string of the action itself.
- Input information (2): The activity has a property *input_information* to specify the incoming information. The property refers a resource, movement in the shown example. This resource keeps a link to the computer-internal data of this information.
- Input object (3): The activity uses a property *input_object* to refer to the incoming objects. The property points to an additional resource, which contains a link to the computer-internal representation of this object.

The properties *output_information* and *output_object* (4) are used to refer the outgoing information and objects. The properties refer to resources, too. Every activity can use multiple input and output objects and information.

- Fitness value (5): Every input and output object and information uses a property *fitness_value* to express a numerical value or a set of numerical values that quantifies the objects and information. It is an optional property. It refers to a literal that contains the numerical value.
- Tool (6): The vocab *tool* labels a property of the activity to describe an additional software tool. This software tool is utilized to carry out the named activity. The property refers to a resource containing a link to this certain tool. This property is optional.
- Method (7): At last every activity needs one method, which is utilized to process the activity. The vocab *method* is used to express this property. It refers to an additional resource. At this

time, the resource keeps a name of the method only. The methods are provided in a database. The user can only select a method.

- Conditions (8): Every method can be concretized by additional conditions. Two conditions are used. The first one is a requirement value. It denotes a minimum amount of data that is necessary to process this method. This property is expressed by the keyword *req_min_value*. It refers to a literal containing a numerical value. The value quantifies the requirement. The second condition expresses whether a user input is necessary during this activity or not. Therefore the keyword *user_input* is used. It labels a property, which refers to a resource. This resource contains a statement that expresses the type of user input. For instance a boolean decision (yes / no). The conditions are optional properties.

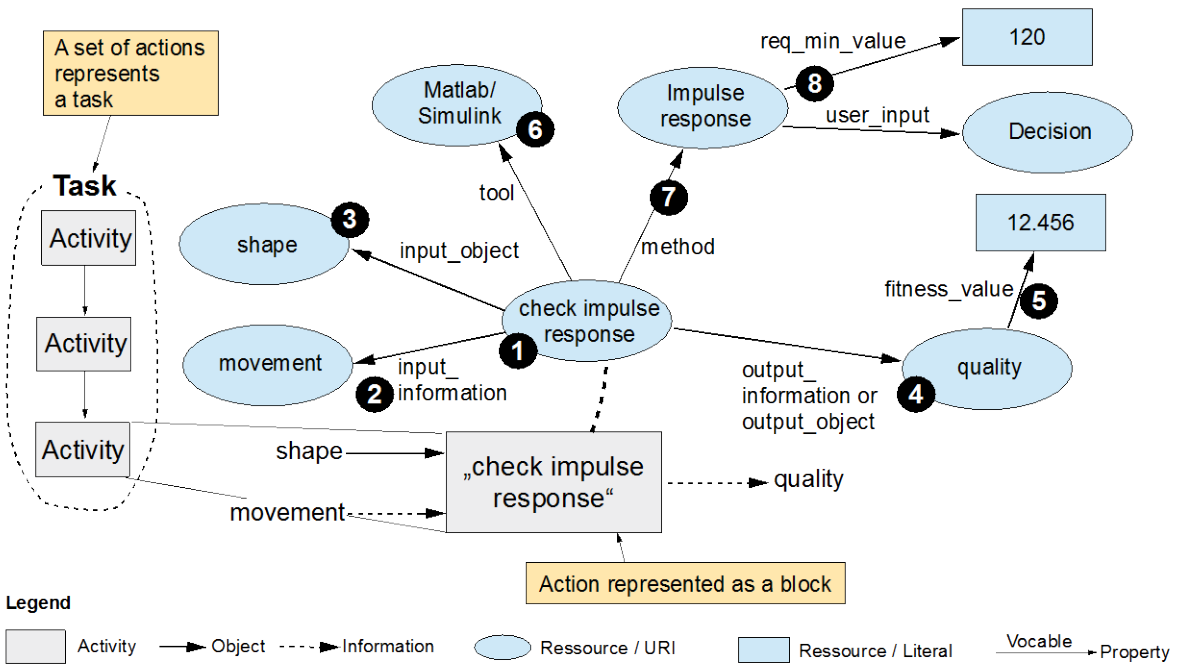


Figure 3: RDF-Description of a task model

3.2.2 Function Model

Figure 4 shows a schematic overview of the function model. It is utilized to specify the functionality of a virtual prototype respectively the product under development. Therefore a function structure according to Pahl and Beitz is used [19]. For the graphical presentation of the function structure a block diagram is a common technique. Each block represents a function. A function is defined as “An Operation, activity, process, or action performed by a system element to achieve a specific objective within a prescribed set of performance limits". According to Pahl / Beitz it is expressed by a substantive and a verb. The substantive names the object that is processed by the function. The verb names the process or the activity the technical system carries out. To build up a function structure the functions of a technical system are connected by the flow of material, energy, and information. The arrows in figure 4 show these flows. The entire function structure represents a model of the functionality of the technical system.

To use the function structure as a knowledge model for an agent a formal computer internal representation has been developed. Therefor we have developed a RDF notation, too. Figure 4 shows an extract of the developed RDF schema in order to introduce the resources and properties and to demonstrate its usage. The following notation is used:

- Function (1): The function itself is expressed by a resource. The resource keeps a character string of the function. It is the main resource of every function and it is required.
- Function term (2): To facilitate an automatic processing of the function term, the function uses a property *function_term*. This vocable refers to an empty resource that points to the substantive and the verb of the function. The property *substantive* refers to a literal of the substantive. The property *verb* refers to the function verb. It is a literal, too.

- Flow of energy, information, and material: To model these three types of flow the function uses the properties *link_x_material* (4), *link_x_information* (5), and *link_x_energy* (6), where x is a wild-card for *in* our *out*. The property refers to an empty resource.
- Attributes: The flow of energy, information, and material need to be specified by three additional properties. These properties are a label, the unit of the technical dimension and the dimension of the value. The property *label* (7) refers to a literal, it contains a character string that names the flow. The property *unit* (8) points to a resource. This resource keeps a value of a technical dimension; in the example the unit "V" for voltage is shown. The last property depicts the dimension of the flow. A scalar, a vector, or an array can model the flow and this needs to be declared. For this, the property *dimension* (9) is used. It refers to a literal to characterize the dimension.
- Source and drain (10): Every flow has its source and a drain. To specify them the properties *source* and *drain* exist. Both properties refer to a resource that contains a link to the related function.

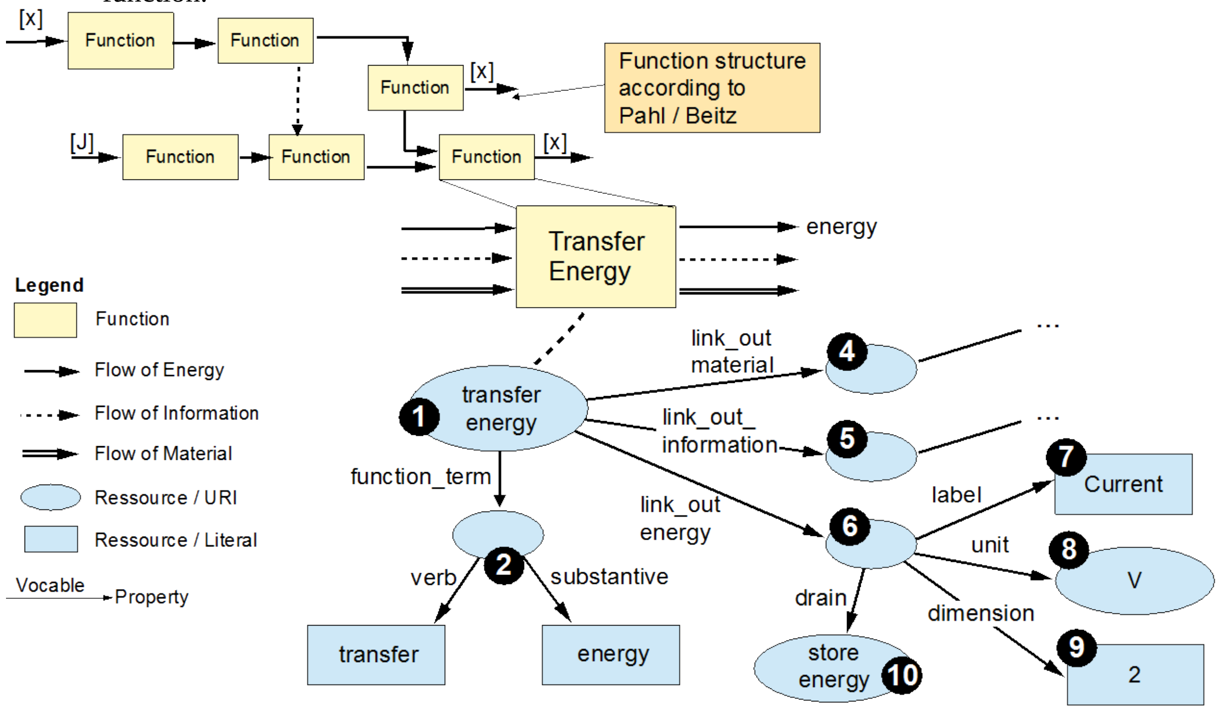


Figure 4: Schematic presentation of the RDF notation of a function structure

The example explains how a function model can be build up and which properties are necessary to describe the functionality of a VP by RDF.

3.2.3 Visualization Model

A visualization is defined as a technique to create images, diagrams, 3D models, and animations to communicate and to explain abstract data. For instance it can be a bar chart, which is a visual representation of a scalar value (Figure 5). In the context of the visualization agent, visualizations are diagrams and 3D models. Both of them are a part of the virtual environment.

A visualization is annotated be an RDF-notation, too. Figure 5 shows an overview of the used resources and properties and how they are applied. As an example, a bar chart is used. To define the RDF-notation it was necessary to identify elements and attributes that specify a visualization and its capabilities. The following resources and properties are used:

- Visualization (1): The visualization itself is modeled as resource. The entry of this resource refers to the internal data model of the visualization. This key element is required.
- Visualization type (2): To specify the type of visualization the related resource has a property *type*. This property refers to a resource that denotes the visualization by a keyword. If the example, the keyword BAR_CHART specifies a bar chart. Other keywords are SYMBOL, ICONS, NET, TREE, and some more. Each of them represents a certain type of visualization.

- Dimensions (3): Every visualization has a set of visual variables. These visual variables are modified to express abstract data by a graphical representation. The property *visualization_dimensions* specifies the number of visual variables each visualization provides. It refers to a resource that contains the number of modifiable visual variables.
- Visual variable (4): This property is used to specify the visual variables itself. To describe them, the visual variables according to Bertin are used [20]. These variables are the size of a visualization, the position, the orientation, the grey scale value, the color, the texture, and the shape. They are transferred to properties like *size_1D*, *size_2D*, *size_3D*, *position*, *orientation*, *color*, etc. For instance *size_1D* specify a visualization, which size can be modified in one dimension. At the example of the bar chart, it is the length of the bar. The property refers to a resource. This resource keeps a link to a variable of the visual variable, which represents it length inside the computer-internal data model. In the shown example it refers to the double my_length.

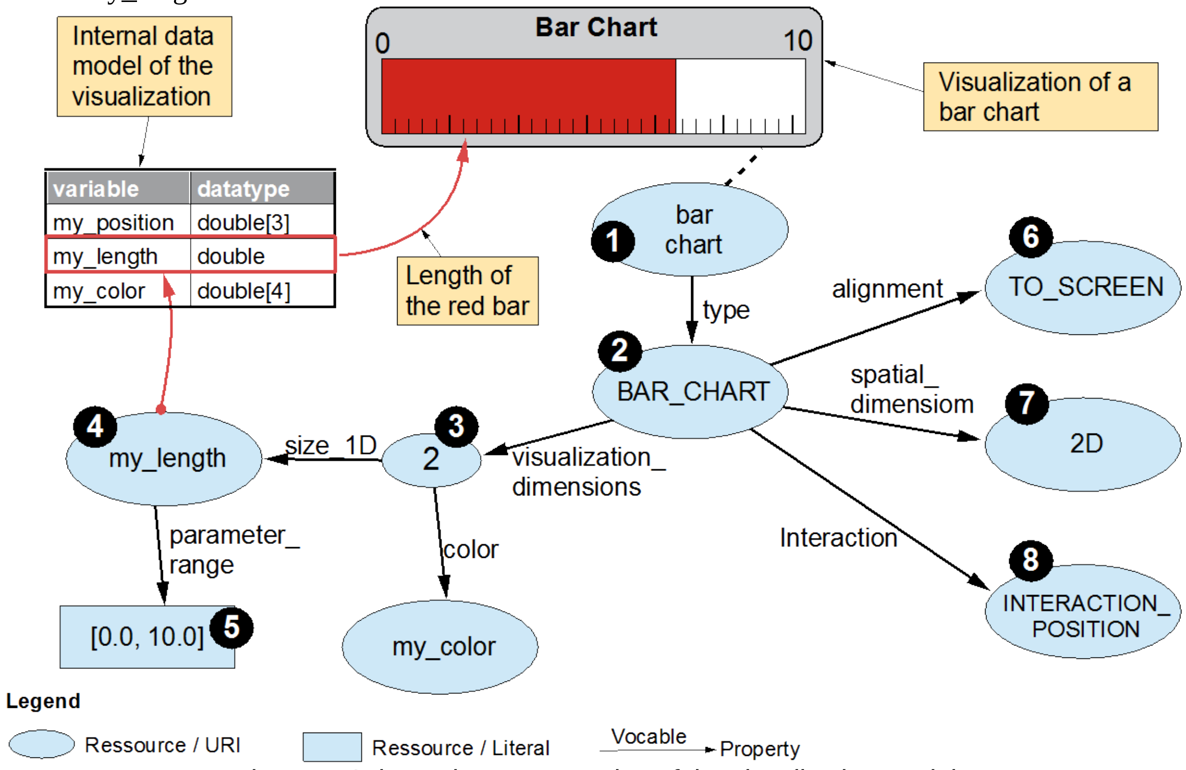


Figure 5: Schematic representation of the visualization model

- Parameters (5): To concretize the visualization the visual variable can be limited by a set of parameter. At this time two parameters respectively properties are used: *range* and *threshold*. The property *range* specifies the boundaries of a dimension. For instance, the bar of the bar chart is limited by a minimum and a maximum value. In the shown example it ranges from 0 to 10. The property *threshold* names a threshold, which is shown by the visual variable.
- Alignment (6): The property *alignment* specifies the spatial alignment of the visualization. It refers to a resource that contains a keyword. Used alignments are HUD (head-up display), TO _SCREEN, TO_MODEL, and some more. For instance, TO_SCREEN means that the visualization is rotated into the viewing direction of the user automaticaly. Thus, the user sees the right face of the visualization every time.
- Spatial Dimension (7): A visualization can be distinguished by its spatial dimension. This feature is specified by the term *spatial_dimension*. The property refers to an additional resource, it contains the dimension: 0D (Points), 1D (Lines), 2D (Surfaces), 3D (Volumes).
- Interaction (8): The property *interaction* needs to be specified if input data from the user is necessary or possible. Maybe a visualization should be moved on screen or the range of a bar needs to be adapted interactively. The property refers to a resource that contains the keyword INTERACTION_x, where x is a wild-card for RANGE, POSITION, and some more. For instance INTERACTION_RANGE means that the user can modify the boundaries of a visual attribute.

This data is sufficient to specify a visualization with a set of annotations. Its computer-internal representation has been integrated into an agent model to specify the visualization.

3.4 Reasoning Mechanism

The reasoning mechanism identifies the Vis-agent, which associated visualization is adequate to visualize the data of the VP or the VP itself. In general the reasoning mechanism compares the models and convert the results to a numerical value. This numerical value expresses the capability of a Vis-agent to visualize the data of a VP.

Altogether three steps are necessary to identify a proper visualization. The first step is processed by the Vis-Agent. The second and the third steps are processed by the VP-agent. Prerequisite is that the VP-agent has submitted its models to the Vis-agent.

In the first step, production rules are used to determine the similarity between different models. A Vis-agent keeps a set of production rules to evaluate the request. Each production has the form

$$IF\ (Condition\ C_1\ \&\ Condition\ B_1\ \&\ldots\ \&\ Condition\ C_n\ \&\ Condition\ B_m)\ THEN\ A_1;\ \ldots;\ A_o$$

Conditions of type C_n are related to the function model and the task model of the VP-agent. Conditions of type B_m are related to the visualization model and task model of the Vis-agent. Each visualization agent contains a set of production rules. These rules compare the referred models and determine whether the Vis-agent fulfills the requirements of the VP-agent. If the capabilities meet the requirements, action A_o is processed. Each action is an equation of the form

$$A_o = a\,g + E$$

With the term $a = 1$ if the production rule is passed and $a = 0$ if the production rule fails. The value g is a weight that indicates how important the production rule is. The term E is an offset; it represents the experience of the agent and describes how useful this action was during previous uses. The value A_o represents the result. The results of every production rule are combined in one vector:

$$E_{Vis} = \{A_1, A_2, \ldots, A_o\}$$

This vector is a rating scale for the quality of the visualization in a certain task. Every Vis-agent calculates this vector and returns it to the VP-agent.

In the second step the VP-agent compares all results $E_{Vis,i,}$ where the index i refers to a certain Vis-agent. A statistical method is used for this comparison, the so-called linear ranking. This method calculates a likelihood value $p(i)$ for each visualization:

$$p(i) = \sum_{j=0}^{m} \cdot \left(E_{max} - (E_{max} - E_{min}) \cdot \frac{E_{Vis,j} - 1}{size - 1} \right)$$

With to rating values E_{max} and E_{min}. These values express the estimated amount of minimal and maximal fulfilled production rules. During the development of a VP-agent, it needs to be estimated how many production rules need to be fulfilled in order to identify a suitable visualization. This estimation needs to be evaluated by the developer of a certain visualization. The equation assigns a numerical value to each production rule and expresses the fulfilled rules by a numerical value. A high value indicates that the Vis-agent is adequate to visualize the VP and the generated data of the VP.

In the third step, the VP-agent decides, which visualization agent is used: the VP-agent takes the Vis-agent with the highest value $p(i)$. One constraint is the equation:

$$p(i) > p_{threshold}$$

The value $p(i)$ needs to cross a threshold $p_{threshold}$. At this time the threshold is determine empirically.

4 RESULTS

The concept of visualization agent has been implemented. However, we are in an early stage of development. But the results prove the usability of the developed RDF notation. The following subsections describe the test platform and the experiments.

4.1 Test platform

To test the concept of visualization agents and the developed models, a software prototype has been developed and an application example has been build. To test the agents a virtual prototype of a mobile robot has been used, which is under development at the Heinz Nixdorf Institute: the so-called BeBot. The BeBot is an autonomous driving robot; it can fulfill different tasks in a team. The task of the visualization agents has been to show the behavior of the robot using catchy visualizations.

The software prototype consists of four components: The first component, a virtual environment is based on OpenSceneGraph (www.openscengraph.org), an open source scenegraph library for the development of 3D graphic applications.
The second component is a simulation for mobile robots. This simulation is carried out with Open Steer [21], an open source software library. Open Steer provides a robot model and a set of functions to drive this model; functions like seek, evade, path following, leader following and some more.
The third component is JADE (Java Agent DEvelopment Framework). JADE is a software framework that facilitates the implementation of multi-agent systems through a middleware that complies with the FIPA (Foundation for Intelligent Physical Agents) specifications, a standard specification for software agents. Furthermore, it provides a set of tools that supports the debugging and deployment phases of agents. The described agent behavior has been implemented using the JADE framework.
The fourth component is a communication server. It realizes the exchange of data between the three components, mentioned before. The entire system works in real time. The technical aspects of the server are described in [17].
In addition to the four components, the software SchemaAgent from Altova is used to annotate the models. It provides a graphical user interfaces to model the resources, properties and the entire RDF graph. The RDF model is stored in an XML notation.
Finally, the software library Jena is used to implement the RDF vocabulary for the annotation, the RDF queries, and the reasoning system (http://jena.sourceforge.net/). Jena is a framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDF-Schemas and includes a rule-based inference engine. The inference engine has been extended to realize the method, which is described in section 3.2.

4.2 Test application

Until now some tests have been carried out to prove the usability of the test infrastructure and the developed RDF models. Therefore, two test applications have been prepared: a Capture the Flag (CtF) application and a path-following application. In the following the CtF application is described only.
Originally CtF is a game where a hunter needs to capture a flag, the other players chasing the hunter and try to prevent him from capturing the flag. In our case the players are the BeBots; one is the hunter, the other try to chase it. The BeBots operate autonomously without any interactions from a user. Figure 6 shows two screenshots from the application. The left figure shows an overview of the virtual environment. The flag stands in the middle of the environment. Spheres are placed as obstacles. The BeBots need to avoid them. The right figure shows a detailed view to the scene. The BeBot with the yellow diamond on top tries to capture the flag.

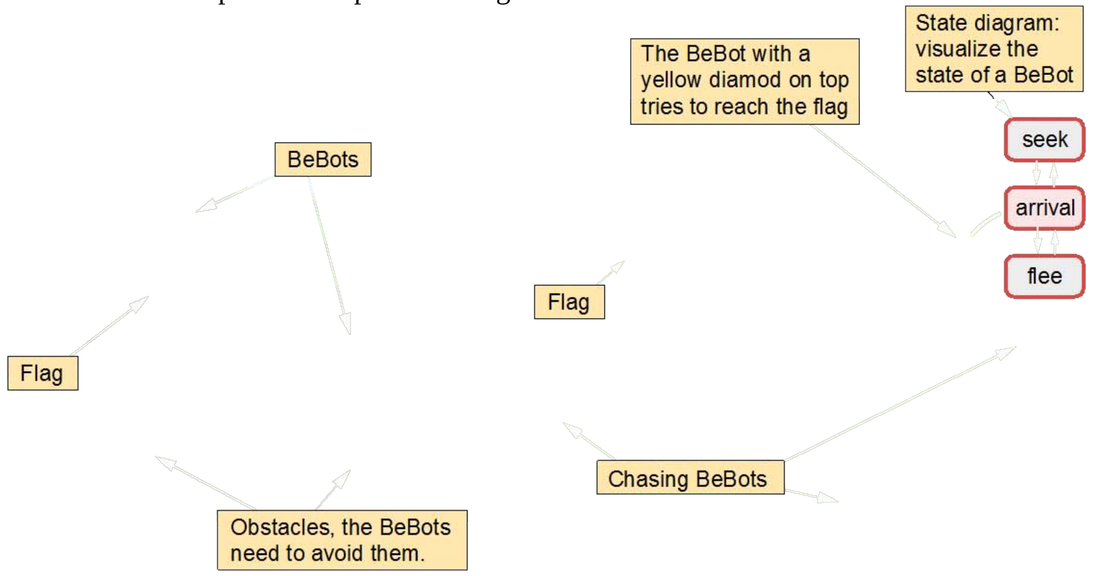

The BeBot with a yellow diamod on top tries to reach the flag
State diagram: visualize the state of a BeBot
seek
arrival
flee
BeBots
Flag
Flag
Chasing BeBots
Obstacles, the BeBots need to avoid them.

Figure 6: Overview of the virtual environment (left), detail view of the test (right)

A state machine with six states models the behavior of the BeBots. Each state represents a type of behavior: seek, flee, obstacle avoidance, robot avoidance, pursuit, and arrival. The BeBots decide on their own which state is active; the decision base on a rule system.

To test the visualization agents the BeBots and one visualization (state diagram) have been implemented and represented by software agents. The task, the behavior, and the visualizations have been specified by the introduced RDF notation. The CtF task has been specified by a task model, the behavior by a function model, and the visualizations by a visualization model.
The task for the Vis-agents is to visualize the different states by a state diagram. Therefore the VP-agent needs to identify the correct Vis-agent. At this time, the test should prove the correctness of the models. It should show, whether it is possible to identify a visualization by the developed RDF notation or not. Thus, we want to know, whether it is possible to reach a high likelihood value *p(i)* or it only low values can be calculated.
The tests show that the models are suitable to describe the visualization, the task, and the function of the BeBots. The likelihood values become high enough to identify a correct visualization in future.

5 CONCLUSION AND FURTHER WORK

This paper introduces the concept of autonomous visualization agents: software agents that facilitate the autonomous identification of a suitable visualization for a certain task. To develop these software agents, computer-internal models are necessary, which represent the knowledge of an agent and let them act autonomously. Therefore, three different models have been developed that are suitable to act as a knowledge representation for the desired task: a task model, a visualization model, and a function model. A RDF notation has been developed. This notation facilitates a computer-internal and formal representation of the knowledge. In addition a reasoning mechanism has been introduces. This mechanism compares the described models by a set of production rules and transfers the results into a numerical likelihood value. To prove the suitability of these three models a test application has been developed and a couple of tests have been carried out. Objective of the tests was to figure out whether it is possible to achieve high numerical likelihood values by the models and the drafted reasoning mechanism. High values indicate a suitable visualization for a certain visualization task. The results show that the reasoning mechanism and the models are suitable for the visualization agents.
Two things could be shown by this work:
First, using a task model, a visualization model and a function model was the right decision. These models are suitable to identify a visualization for a desired task.
Second, the resources and properties of the developed RDF notation facilitate the computer-internal annotation of the task, the visualization, and the functions. They can be used as meta data for these aspects.
Altogether the results are a good starting point for further research. One disadvantage of the RDF notation is possible its dependency on a certain user. Eventually the models and the effectiveness of the reasoning mechanism differ too much when different users describe the models. This needs to be tested in future.
In further work the application example will be extended. First, a universal set of production rules needs to be identified. Until now the production rules have been determined empirically prior to the experiment. Second, the reasoning mechanism needs to be verified. The experiments show us that it is mathematically correct: it calculates high likelihood values when the models meet the production rules and low values when they do not meet the production rules. But the calculations have been carried out using a one to one relation between a visualization (state diagram) and a BeBot respectively its agent. The next step will be the verification by a set of experiments utilizing a large visualization database.

This contribution was developed and published in the course of the Collaborative Research Center 614 “Self-Optimizing Concepts and Structures in Mechanical Engineering” funded by the German Research Foundation (DFG) under grant number SFB 614.

REFERENCES

[1] Gausemeier, J.; Plass, C.; Wenzelmann, C.: *Zukunftsorientierte Unternehmensgestaltung*. Carl Hanser Verlag, München, 2009
[2] Krause, F.-L.; Jansen, C.; Kind, C.; Rothenburg, U.: *Virtual Product Development as an Engine for Innovation.* In: Krause, F.-L.: The Future of Product Development. Proceedings of the 17th CIRP Design Conference. Springer Verlag, Berlin, 2007
[3] Purschke, F.; Schulze, M.; Zimmermann, P.: *Virtual reality-new methods for improving and accelerating the development process in vehicle styling and design.* Computer Graphics International, 1998

[4] Lang, U.; Wössner, U.: *Virtual and Augmented Reality Developments for Engineering Applications.* In: European Congress on Computational Methods in Applied Sciences and Engineering ECCOMAS 2004 P. Neittaanmäki, T. Rossi, S. Korotov, E. Oñate, J. Périaux, and D. Knörzer (eds.) Jyväskylä, 24—28 July 2004

[5] Wooldridge, M.; Jennings, N. R.: *Applying Agent Technology*. In: Journal of Applied Artificial Intelligence, Special Issue on Intelligent Agents and Multi-Agent Systems, 1995

[6] Mendez, G.; de Antonio, A.: *An Agent-Based Architecture for Collaborative Virtual Environments for Training.* In: Proceedings of the 5th WSEAS Int. Conf. on Multimedia, Internet and Video Technologies, pp. 29-34, Corfu, Greece, August 17-19, 2005

[7] Galea, A.; Borg, J.; Grech, A.; Farrugia, P: *Towards Intelligent Design Tools for Micro-scale components.* In: International Conference on Engineering Design, ICED´ 09, pp. 5-73 - 5-84, 24-27 August 2009, Stanford, CA, 2009

[8] Payne, T.R.: *Agent-based Team Aiding in a Time Critical Task.* In: HICSS`00, Proceeding of the 44rd Hawaii International Conference on System Sciences, Vol. 1. 2000

[9] Baolu, G.; Shibo, X.; Meili, C.: *Research and Application of a Product Cooperative Design System Based on Multi-Agent.* In: Third International Symposium on Intelligent Information Technology Application, pp. 198 - 201 2009

[10] Geiger, C.; Lehrenfeld, G.; Mueller, W.: *Authoring communicating agents in virtual environments.* In: Proceedings of the Computer Human Interaction, pp. 22-29, Adelaide, SA, Australia, 1998

[11] Berners-Lee T., Hendler J.; Lassila, O.: *The Semantic Web*, Scientific American, 2001.

[12] Bludau, C.; Welp, E.: *Semantic Web Services for the Knowledge-based Design of Mechatronic Systems.* In: Proceedings of the International Conference on Engineering Design, ICED 2005, Melbourne, Australia, August 15-18, 2005

[13] Restrepo, J.: *A Visual Lexicon to Handle Semantic Similarity in Design Precedents*. In: Proc. of the 16th International Conference on Engineering Design, ICED`07, Paris, 2007

[14] Ding, L.; Matthews, J.; Mullineux, G.: *Annacon: Annotation with constrains to support design*. In: International Conference on Engineering Design, ICED´ 09, pp. 5-37 - 5-48, 24-27 August 2009, Stanford, CA, 2009

[15] Ding, L., Davies, D.; McMahon, C. A.: *The integration of lightweight representation and annotation for collaborative design representation*. In: Research in Engineering Design, 19 (4), pp. 223-238, 2009

[16] Li, C.; McMahon, C.; Newnes, L.: *Annotation in Design Processes: Classification of Approcches.* In: International Conference on Engineering Design, ICED´ 09, pp. 8-251 - 8-262, 24-27 August 2009, Stanford, CA, 2009

[17] Radkowski, R.; Wassmann, H.: *Software-Agent Supported Virtual Experimental Environment for Virtual Prototypes of Mechatronic Systems*. In: Proceedings of the ASME 2010 World Conference on Innovative Virtual Reality WINVR2010 , May 12-14, 2010, Ames, USA, 2010

[18] Wasson, C.: *System Analysis, Design, and Development*. Wiley-Interscience, Hoboken, USA, 2006

[19] Pahl, G.; Beitz, W.; Feldhusen, J.; Grote, K.-H.: *Engineering Design: A Systematic Approach,* 3rd ed. Springer Verlag, Berlin, 2006

[20] Bertin, J.: *Semiology of Graphics*, University of Wisconsin Press, 1983

[21] Reynolds, C. W.: *Steering Behaviors For Autonomous Characters.* in: Proceedings of Game Developers Conference 1999 held in San Jose, California. Miller Freeman Game Group, San Francisco, California. pp. 763-782, 1999

Contact: Rafael Radkowski
Heinz Nixdorf Institute (University of Paderborn)
Product Engineering
33102 Paderborn, Germany
Tel: Int +49 5251 606228
Fax: Int +49 5251 606268
Email: rafael.radkowski@hni.uni-paderborn.de