Ontology-based Understanding of Everyday Activity Instructions

Sebastian Höffner a,*, Robert Porzel a, Maria M. Hedblom c, Mihai Pomarlan b, Vanja Sophie Cangalovic b, Johannes Pfau a, John A. Bateman b and Rainer Malaka a

a Digital Media Lab, TZI, University of Bremen, Germany
b Computational Linguistics, FB10, University of Bremen, Germany
c Institute for Artificial Intelligence, TZI, University of Bremen, Germany

E-mail: {hoeffner,porzel,hedblom,pomarlan,vanja,jpfau,bateman,malaka}@uni-bremen.de

1. Introduction

Performing household activities such as cooking and cleaning have, until recently, been the exclusive provenance of human participants. However, the development of robotic agents that can perform different tasks of increasing complexity is slowly changing this state of affairs, creating new opportunities in the domain of household robotics. Most commonly, any robot activity starts with the robot receiving instructions for that specific activity. For everyday activities, such instructions can be given either verbally from a human or through written texts such as recipes and procedures found in online repositories, e.g., from wikiHow. From the perspective of the robot, these tend to be vague and imprecise as natural language generally employs ambiguous, abstract, and non-verbal cues. Often, instructions omit vital semantic components such as determiners, quantities, or even the objects they refer to [1, 2].

Still, asking a human to, for instance, “take the cup to the table” will typically result in a satisfactory outcome. Humans excel despite many uncertain variables existing in the environment: for example, the cups might all have been put away in a cupboard or the path to the table could be blocked by chairs.
In comparison, artificial agents lack the same depth of symbol grounding between linguistic cues and real objects, as well as the capacity for insight and prospection to reason about instructions in relation to the real world and the changing states of that world. In order to turn an underspecified text, issued or taken from the web, into a detailed robotic action plan, various processing steps are necessary, some based on symbolic reasoning and some on numeric simulations or data sets. The research question in this paper is therefore the following: How can we use ontological knowledge to extract and evaluate parameters from a natural language instruction in order to simulate it formally? The solution proposed in this work is the Deep Language Understanding (DLU) processing pipeline. This pipeline employs the ontological Socio-physical Model of Activities (SOMA) [3], which serves not only to define interfaces of the multi-component pipeline but also to connect numeric data and simulations with symbolic reasoning processes.

2. Semantic Grounding and Ontologies

Natural language understanding for artificial agents dates back at least to the late 1940s. Recent examples include various artificial neural networks such as BERT [4] and GPT-2 [5] which can extract information from written text, answer questions, or tell stories when presented with a short teaser. These and similar machine learning techniques power chatbots and virtual assistants that rely on natural language understanding to answer questions or control smart home appliances [7]. Additionally, there are systems that generate scenes from natural language [8, 9] and methods to control non-player characters in computer games with unstructured natural language commands [10]. All of these systems present various possible solution strategies to natural language understanding, but the problem of symbol grounding remains. The symbol grounding problem can be described with the semiotic triangle, which relates concepts, symbols, and objects to each other [11, 12].

Dealing with the transition from symbols in language to actions and objects in the real world is partly a problem due to the fact that the underlying deep understanding of natural language is not a direct mapping from either syntax or grammar. Instead, semantics appears in the form of linguistic constructs that correspond to particular mental patterns of meaning [13]. A theory for how these constructs manifest in the cognitive sphere can be constructed by grounding it in the embodied experiences of agents, thereby offering a seamless transition to simulations and robotics research. By looking to the embodied theories of cognition [14], recorded human activity, simulation, and action execution of robots can be used as a foundation for how to construct meaningful ontological structures. One proposal for how experiences turn into meaningful constructs is through conceptual patterns called image schemas [15]. They are spatiotemporal relationships that to varying degrees capture the specifics of particular static relationships and dynamic transformations of objects, agents and environments. Relationships like movement between points (SOURCE_PATH_GOAL (SPG))2, relative distance in the NEAR_FAR schema and VERTICALITY are important image schemas to consider when performing object manipulations. While image schemas are predominantly studied in cognitive and linguistic settings, turning the theory into a formally applicable method has been initiated in terms of using them as logically formulated theories, hierarchically structured based on their internal compositionality [16].

In DLU, the first steps of using image-schematic relationships are introduced as ontological micro-theories included in SOMA as a means to ground robotic action plans into the large body of linguistic research on embodied semantics. The theories define slots, such as trajector for an SPG, which can be filled using parameters (cf. Section 3.5), motivated by Bergen and Chang’s embodied construction grammar [17]. To extract the theories from natural language, the Streaming Construction Grammar (SCG) parser [18] is employed.

While this application is a novel research contribution, employing ontologies for knowledge representation and reasoning in autonomous robot control is an established field of research in developments both in service and industrial robotics (see [19] for an overview). One example in the industrial robotics domain is the ROSETTA project [20, 21]. The initial scope of this project was reconfiguration and adaptation of robot-based manufacturing systems.
cells, however, the authors have further developed their activity modeling for coping with a wider range of industrial
tasks. Other authors have focused on modeling industrial task structure, part geometry features, or task teaching
from examples (e.g. [22–24]). These industrial tasks tend to be more structured and less demanding in terms of
flexibility compared to the everyday activity domain that is the focus of this paper.

An approach to activity modeling in the service robotics domain is presented by Tenorth and Beetz [25]. The
scope of their work is similar to this work, as the authors also consider how activity knowledge can be used to fill
knowledge gaps in abstract instructions given to a robotic agent performing everyday activities. However, the scope
of the work presented here is wider, as it is also considered here how knowledge can be used for the integration of
numeric approaches and reasoning. Another difference is that, in Tenorth and Beetz’ modeling, there is no distinction
between physical and social context, and therefore, it is less expressive than SOMA which is used here.

A more general approach to activity modeling for robotic agents is presented by the IEEE-RAS working group
Ontologies for Robotics and Automation (ORA) [26]. The group has the goal of defining a standard ontology
for various sub-domains of robotics, including a model for object manipulation tasks. It has defined a core ORA
ontology [27], as well as additional modules for industrial tasks such as kitting [28].

3. System Overview

The DLU processing pipeline consists of the interconnected components depicted in Fig. 1. In this section, the
most important components are described.

DLU starts by parsing a sentence into semantic specifications (Fig. 1 c) based on SOMA (a) using the Stream-
ing Construction Grammar (SCG) parser (b) [18]. Next, DLU determines referents (f) for the extracted semantic
specifications in a pre-defined semantic map (d) of a kitchen scene. After that, the pipeline extracts image schema theories (e) from the semantic specifications. Each image schema theory has a set of parameters which DLU fills with values. These values are either references to objects or other image schemas, or they are values from evaluating pre-trained models. For example, one model discussed below contains information on where glasses can be placed on top of a table. Eventually, all knowledge from the previous steps—parsing, grounding, theory extraction—is combined into a sequence of trajectories (g), which DLU executes inside a game engine (h).

### 3.1. SOMA and other Ontologies

Across the DLU pipeline the Socio-physical Model of Activities (SOMA) is employed as a common interface. SOMA is based on the DOLCE+DnS Ultralite (DUL) foundational framework and its plugin Information Objects ontology (IOLite) [3, 29, 30]. Consequently, SOMA has two knowledge branches; one physical and one social [3], which leads to a distinction between objects and actions in the physical branch on the one hand, as well as roles and tasks in the social branch on the other. Beßler et al. explain that axiomatizations in the physical branch express physical contexts which can be classified by axiomatization in the social context [3]. For example, a cup and its properties of being a designed physical artifact would be described using parts of the physical branch, but its potential usage or affordances would be axiomatized within the social branch. SOMA is built out of multiple modules for different aspects. For example, the SAY module defines the linguistic theories and knowledge required by SCG, and the HOME module defines typical household objects. For DLU, SOMA is an optimal choice ontology as it is designed to describe actions with high precision.

To keep the ontology prefix definitions in the article at a minimum, all of the used prefixes are defined in Listing 1.

### 3.2. SCG / Semantic Specification

To extract a semantic specification from a sentence, the Streaming Construction Grammar (SCG) parser [18] is employed. Given a textual input, it produces a set of triples which describe the sentence’s syntax as well as semantics. The underlying grammar maps the semantics onto SOMA concepts where possible, and employs a custom set of ontologies for linguistic features. The produced entities are tagged as syntactic or semantic concepts to allow for straightforward filtering. This is currently used in DLU, which operates on semantics as discussed in Section 3.5. An example of semantic triples can be found in Listing 2.

### 3.3. Context / Semantic Map

For a command to be executed, an environment with objects to manipulate is mandatory. This environment is a context or a scene. Fig. 2 shows a kitchen scene in which actions in DLU are simulated. Each object in the kitchen scene is internally annotated with unique identifiers and semantic labels aligned with SOMA. This allows for storing knowledge about the scene in the same format as the semantic specifications. In essence, the context description resembles a semantic map of the scene. However, no semantic mapping of the scene is performed – instead, for the time being, a semantic map is assumed to be provided.
Fig. 2. The kitchen scene. All objects—chairs, bowls, pots, cupboards, etc.—have semantic labels aligned with SOMA and unique identifiers, e.g., <urn:cup_0>.

3.4. Grounding

In DLU, the concepts of the semiotic triangle are defined in SOMA. The symbols are part of the natural language instruction, e.g., the word “cup”. The objects are the entities in the kitchen scene, e.g., the table entity has a mesh, textures, and defined physical properties. And the mapping from the symbols to the concepts is performed by SCG. With the semantic map, the mapping from the objects to the concepts are given a priori. The missing link, the mapping between the objects and symbols, is called **grounding** in DLU; this is also known as **referencing**.

Grounding is implemented using the following heuristic in DLU: A symbol references an object if the mean path depth of their common types to **dul:PhysicalObject** is maximal. This means, that after parsing “take the cup to the table” with SCG and storing the results alongside the semantic map in the knowledge base, DLU might have—among others—the triples in Listing 3 available. In Listing 3, there is a symbol <urn:Cup> from the semantic specification and the two objects <urn:cup_0> and <urn:fridge_0> from the semantic map. During the grounding step, DLU has to find a mapping between <urn:Cup> and <urn:cup_0>, but avoid matching the symbol cup to the fridge. Given the examples in Listing 3, the SPARQL query Listing 4 returns pairs and object types as can be seen in Table 1.

The Prolog predicate\(^3\) `rdf_db:rdfreachable/5` finds the depth of a path in a SPARQL query, which is used for the heuristic. For each type, its path depth from **dul:PhysicalObject** is determined. The path depth is the minimal required number of subclass relations from the object type to **dul:PhysicalObject**. In the example, **soma:DesignedContainer** requires three relations, while **dul:Cup** requires four. For each symbol-

---

\(^3\)https://www.swi-prolog.org/pldoc/man?section=semweb-rdf11#rdfreachable/5

---

Listing 2: An excerpt from SCG output: Semantic triples involving a cup in n3/turtle-notation. Note: SCG uses different prefixes, here they are adjusted for consistency.
Listing 3: Semantic specifications for the entity referred to as `<urn:Cup>` and for the objects `<urn:cup_0>` and `<urn:fridge_0>` from the semantic map. Here, the tag `cg:semantic` distinguishes symbols from objects denoted with the type `owl:NamedIndividual`.

```
<urn:Cup> rdf:type dlu:Cup .
<urn:Cup> rdf:type soma:DesignedContainer .
<urn:Cup> scg:tag cg:semantic .
<urn:cup_0> rdf:type soma:DesignedContainer .
<urn:cup_0> rdf:type dlu:Cup .
<urn:cup_0> rdf:type owl:NamedIndividual .
<urn:fridge_0> rdf:type soma:DesignedContainer .
<urn:fridge_0> rdf:type soma:Fridge .
<urn:fridge_0> rdf:type owl:NamedIndividual .
```

Listing 4: SELECT statement to select all symbol-object pairs with the same assigned object types.

```
SELECT DISTINCT ?symbol ?object ?objtype
WHERE {
}
```

object pair, e.g., `<urn:Cup>` and `<urn:cup_0>`. DLU can thus find a mean path depth value and arbitrarily select one of the highest valued ones, thus referencing the cup with `<urn:cup_0>` but not with the fridge, see Table 1. Finally, this knowledge is asserted by introducing a `dul:isReferenceOf` to the knowledge base for further computations.

### 3.5. Theory and goal extraction

"Take the cup to the table" is a `soma:Command` (cf. Fig. 3). In SOMA, a command classifies a state transition, i.e., the intention of a command is to transform a scene from one scene state, the initial scene, into another scene state, the terminal scene [31]. For each state, a set of schemas or theories has to be satisfied. Most commands leave the initial scene free of constraints except for ensuring the existence of several objects, e.g., a cup and a table. The focus of DLU lies in the terminal scene, as this defines which schemas or theories need to be satisfied in the final state after the command. In the example case, this is a theory of proximity: i.e., the cup has to be in the proximity of the table. It is not possible to be more specific here on linguistic grounds because "take to" does not constrain

Table 1 Candidate groundings for the symbol `<urn:Cup>` to different objects. The grounding heuristic assigns mean depth values to each pair of candidate groundings. The pair `<urn:Cup>, <urn:cup_0>` has the highest, and thus the most specific, value. Hence, this pair is selected.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Object</th>
<th>Object type</th>
<th>Depth</th>
<th>Mean depth</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;urn:Cup&gt;</code></td>
<td><code>&lt;urn:cup_0&gt;</code></td>
<td>dlu:Cup</td>
<td>4</td>
<td>4/3 = 3.5</td>
</tr>
<tr>
<td><code>&lt;urn:Cup&gt;</code></td>
<td><code>&lt;urn:cup_0&gt;</code></td>
<td>soma:DesignedContainer</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td><code>&lt;urn:Cup&gt;</code></td>
<td><code>&lt;urn:fridge_0&gt;</code></td>
<td>soma:DesignedContainer</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
whether the cup should be on top of or below the table, or in any other relation. A possible resolution to this dilemma is human computation, as described in Section 3.6. Here, the focus is on the extraction of theories which give information on how the new state should be achieved.

![Diagram of command execution](https://osf.io/dmcjq/)

The command “take the cup to the table” evokes a SPG schema [15], denoted by a **soma:SourcePathGoalTheory**. In DLU, SPGs are modeled similarly to Bergen and Chang’s version in their embodied construction grammar by introducing a theory with the roles trajector, source, path, goal, and means [17]. Each of the roles can be filled by a reference to an object in the scene, a property of an object, a reference to another theory, or some specific numerical value. The initial roles in the example instruction, “take the cup to the table,” can be seen in Listing 5. The trajector is a reference to the cup in the scene and the source is the pose of that cup, extracted from the knowledge base. Note that the path is an empty list—DLU has no information about the path and so determines the cup’s trajectory in subsequent steps. The goal is a reference to another entity in the knowledge base: the proximal theory, which should be satisfied at the terminal scene, after executing the command.

By having theories reference other theories, the tree of theory references can be traversed to filter those theories which lead towards our goal state. In the example sentence, this applies only to the SPG theory, which references the proximal theory. The proximal theory satisfies the terminal scene of the command and has no further references.
Thus, it is possible to infer that DLU only needs to process the SPG and the proximal theory itself to calculate the required actions to reach the goal of the command.

3.6. Action Creation

There are several accounts of what “actions” are (e.g., [32]). In this work, an atomic action is the process of a trajector following a trajectory. It is possible to generate such actions from the extracted schemas in order to reach the goal of the command. For this, a distinction between two types of theories has to be made: **resolvable** and **executable** theories. While this is mostly an implementation detail, it allows for picking theories from which actions can be built: each executable theory eventually describes a trajectory, while each resolvable theory adds information to such trajectories.

In “take the cup to the table,” there is one executable theory, the SPG mentioned above. Eventually, it should describe a trajectory from the original location of the cup to a pose somewhere in the proximity of the table, as is defined by the proximal theory. Finding this trajectory requires two steps: First, resolving the proximal theory to a proper pose, then using the initial pose and the resolved pose to plan a path between the two.

One can see the proximity theory as a specialization of Johnson’s **NEAR_FAR** schema [15], as it conveys some information about nearness. However, nearness in itself is context dependent and relies on a reference frame. To solve the problem of precision in spatial closeness, DLU uses human computation [33]. Pfau et al. collected human activity data in a virtual reality (VR) kitchen environment [34] for various tasks. Inside the VR environment, users could chop vegetables, move objects around, and perform other cooking related tasks. While the users performed tasks in the VR kitchen, their hands and the object positions were tracked, as well as collision data between individual objects. The collisions between glasses and the table surface were used to gather data where cups are in a proximity relation to a table, as the kitchen did not contain any cups. Several Gaussian mixture models (GMMs) were fit on the collision positions and the one with lowest Bayesian information criterion (BIC) was selected as the model for “glasses in proximity of a table”. Figure 4 depicts the raw collisions and the GMM on a normalized table surface.

Having distinct models for proximity relations between cups and tables, between lamps and chairs, between houses and trees, etc., offers a way to select the correct model that is required. To achieve this in DLU, the model of proximity relations between cups and tables is stored in the ontology as an instance of a **soma:SoftwareImplementation** which concretely expresses an instance of a proximal theory. To store the model, it is serialized using **dill** and written as a **xsd:base64Binary** into the knowledge base. This allows SPARQL queries to be run to select a model and, after deserialization, evaluate it. For the proof of concept of DLU, cups and glasses are treated as the same object class. Otherwise, an additional reasoning step to select models for objects similar to other objects would be required.

---

4 The complete dataset can be found online: https://osf.io/rx9jg/
5 GMMs with 1–6 Gaussians, using different covariance types. Lowest BIC was 143.82 for three Gaussians with spherical covariances. Adapted from https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm_selection.html
6 A full example can be found at https://osf.io/c63w4/
7 The standard way to serialize Python objects, pickle, cannot handle anonymous functions well. Hence, dill (https://dill.rtfd.io) is used.
Fig. 4. The positions of the collisions of glasses with the table surface in VR, together with the resulting GMM. Collisions were projected onto the same horizontal area and show the collisions from the top. The table width and length are normalized to $[-1, 1]$, so that the data can be mapped onto arbitrary (rectangular) tables. Multiple collisions at $x = -1$ were removed prior to calculating the GMM, as they formed a straight line at the edge of the table.

After selecting a candidate model for the proximity theory, it is evaluated to sample a goal pose for the cup in the example and replace the reference to the proximity theory in the SPG (cf. Listing 5) with the concrete sampled pose. As a final step, a trajectory between the start and goal pose is generated to build a path. Currently, this means a linear interpolation between the start and goal pose, but DLU provides an occupancy grid of the scene to allow for more sophisticated path planning in the future, e.g., using the A* algorithm.

3.7. Simulation

Physics engines are popular tools to simulate real life phenomena. Computational fluid dynamics (CFD) solvers such as OpenFOAM [35] or Autodesk CFD\(^8\) focus on high accuracy but are not necessarily suited for real time simulations. In contrast, there are many physics engines which focus on speed, such as the PhysX SDK\(^9\) or the Bullet SDK [36]. They are embedded in a variety of applications, e.g., Robot Operating System (ROS) employs the Bullet SDK, the OpenAI Gym [37] offers bindings to MuJoCo\(^10\), and Unity3D\(^11\) integrates the PhysX SDK. In addition to these general purpose physics engines, there exist specialized tools, e.g., to predict folk psychological phenomena [38].

In DLU, Unity3D is used to perform naive physics simulations. It executes the actions described above in the scene and records the actual trajectories of all entities in the kitchen scene. After a simulation run, the results are evaluated visually.

\(^{8}\)Autodesk Inc., Autodesk CFD, https://www.autodesk.com/products/cfd/overview
\(^{10}\)Roboti LLC, MuJoCo, http://www.mujoco.org
\(^{11}\)Unity Technologies, Unity, https://unity.com
4. Discussion of System Performance and Limitations

To evaluate the DLU pipeline, the instruction “take the cup to the table” is used, as it has several interesting properties: a) “Take” makes no specific assumptions about the means, but together with “to” it entails a movement with a trajectory. b) “The cup” could easily be replaced with other items, based on similar functional properties or affordances. c) “To the table” does not explicitly specify that the cup should be place on top of it, but only that it needs to be close to it. d) While two items are involved in the instruction, one remains stationary. e) Additionally, the task is rather forgiving in terms of precision. While covering a pot requires placing a lid exactly on top of it, placing the cup on the table allows for more freedom in location (albeit not in orientation). f) Lastly, taking a cup to a table is usually a mundane exercise for humans. The sentence is thus prototypical of everyday activities and their instructions. The evaluation criterion is that the final simulation matches common sense expectations.

DLU is built upon a few assumptions. First, it assumes error free natural language instructions and the existence of a semantic map. Further, it expects that each instruction maps to an action as defined above and can be achieved by moving objects along a trajectory in the environment. Also, it presumes that a real robot can perform that action when given a trajectory and a trajector, allowing it to leave out the robot—instead, objects “fly” around. For the model of proximity relations between glasses and tables, it is assumed that collision events are a good enough indicator of proximity. Lastly, the hypothesis is that once a sufficiently large set of functional relationships and image-schematic micro-theories is modeled in DLU, the pipeline will scale up and be able to understand more instructions.

When the DLU pipeline is evaluated under these assumptions and given the above criterion, “take the cup to the table” results in a simulation that human observers, such as DLU’s authors, would consider an appropriate execution of the task. Still, individual components of the pipeline can be observed in isolation to reveal their strengths and weaknesses.

SOMA and SCG have been evaluated by their authors [3, 39]. The grounding step (Section 3.4) works for many objects in the scene, given the parser provides enough information. It does not work for complex object descriptions or when multiple objects are present, e.g., “the red cup” might select a blue cup. In the schema or theory extraction step, only those theories for which queries were written can be found. To date, these are the ones which are relevant in “take the cup to the table,” namely SPG, CAUSEDMOTIONTHEORY, and PROXIMALTHEORY. There is also only one model available for the proximal theory of glasses being in the proximity of tables. Under the assumption that one can exchange glasses and tables for other objects, this makes it possible to move objects around the scene and place them on top of each other. In DLU, it is shown that it is possible to store such a model with proper annotations and retrieve it to use it without enforcing specific constraints on the model. This allows sharing and reusing models across different ontologies. As path planning is not the focus of this work, DLU interpolates linearly between the start and goal pose of an object. However, since DLU provides occupancy information as well as object data such as meshes available, future versions can employ existing path planning solutions.

5. Conclusion and Future Work

This paper introduced DLU, a natural language processing pipeline which is capable of simulating natural language instructions using SOMA as a common interface for individual components. On the basis of “take the cup to the table,” this work showed that the architecture is successful in simulating an everyday activity task.

DLU runs on a live instance at https://litmus.informatik.uni-bremen.de/dlu/12, and all required resources to run DLU locally are compiled at the Open Science Framework (OSF)13. The source code, software, and hardware information used to build and run DLU are available at https://osf.io/nbxsp/. The ontologies and their metrics are available at https://osf.io/e7uck/.

The external modules, SOMA and SCG, are under active development and as they advance to become more advanced and complete, DLU benefits by being able to parse more instructions and harvest better ontological de-

---

12 A video demonstration is available at https://osf.io/t2mnw/
13 https://osf.io/nm86g/
scriptions. For the grounding module, other systems to replace the heuristic are currently being evaluated. Possible options are a system to ground unknown synonyms [40] and end-to-end machine learning models to ground instructions such as “go to the red pole” [41, 42].

Other future work includes building models on different relations between different objects to scale up the knowledge base, e.g., SUPPORT or functional relationships such as coverage. In addition to broadening the application, it also offers to validate model selection procedures, e.g., whether exchanging a cup for a glass is an option, and if one can resort to a model of SUPPORT between two physical objects. To decide which object classes are possible replacements candidates, it is planned to use semantic similarities, e.g., word2vec [43], and affordance-based models, fueled by human computation tasks. One major step in future work will be to reiterate the last two steps: that is, if the simulation fails, to re-sample the parameters and try again. For this, an integration of schemasim is planned, which allows checking whether a configuration of objects in a scene satisfies a set of expectations [44], and thus to decide whether the simulation succeeded.

Acknowledgements

The authors wish to thank Susanne Putze, Dmitry Alexandrovsky, Nina Wenig, and four anonymous reviewers for their valuable feedback. The research reported in this paper has been (partially) supported by the FET-Open Project #951846 “MUHAI – Meaning and Understanding for Human-centric AI” funded by the EU Program Horizon 2020 as well as the German Research Foundation DFG, as part of Collaborative Research Center (Sonderforschungsbereich) 1320 “EASE – Everyday Activity Science and Engineering”, University of Bremen (http://www.ease-crc.org/).

The research was conducted in subprojects P01 and H02.

References
