We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

25

Programming of Intelligent Service Robots with the Process Model "FRIEND::Process" and Configurable Task-Knowledge

Oliver Prenzel[1], Uwe Lange[2], Henning Kampe[2],
Christian Martens[1] and Axel Gräser[2]
[1]*Rheinmetall Defence Electronics,*
[2]*University of Bremen*
Germany

1. Introduction

In Alex Proyas's science fiction movie "I, Robot" (2004) a detective suspects a robot as murderer. This robot is a representative of a new generation of personal assistants that help and entertain people during daily life activities. In opposition to the public opinion the detective proclaimed that the robot is able to follow his own will and is not forced to Isaac Asimov's three main rules of robotics (Asimov, 1991). In the end this assumption turned out to be the truth.

Even though the technological part of this story is still far beyond realization, the idea of a personal robotic assistant is still requested. Experts predicted robotic solutions to be ready to break through in domestic and other non-industrial domains (Engelberger, 1989) within the next years. But up to now, only rather simple robotic assistants like lawn mowers and vacuum cleaners are available on the market. As stated in (Gräfe & Bischoff, 2003), all these systems have in common that they only show traces of intelligence and are specialists, designed for mostly a particular task. Robots being able to solve more complex tasks have not yet left the prototypical status. This is due to the large number of scientific and technical challenges that have to be coped with in the domain of robots acting and interacting in human environments (Kemp et al., 2007).

The focus of this paper is to describe a tool based process model, called the "FRIEND::Process"[1], which supports the development of intelligent robots in the domain of personal assistants. The paper concentrates on the interaction and close relation between the FRIEND::Process and configurable task-knowledge, the so called process-structures. Process-structures are embedded in different layers of abstraction within the layered control architecture MASSiVE[2] (Martens et al., 2007). Even though the usage of layered control architectures for service robots is not a novel idea and has been proposed earlier (Schlegel &

[1] The name FRIEND::Process is related to the FRIEND projects (Martens et al., 2007). It has been developed within the scope of these projects, but is also applicable to other service robots.

[2] MASSiVE – Multilayer Control Architecture for Semi-Autonomous Service Robots with Verified Task Execution

Woerz, 1999; Schreckenghost et al., 1998; Simmons & Apfelbaum, 1998), MASSiVE is tailored for process-structures and thus is the vehicle for the realization of verified intelligent task execution for service robots, as it is shown in the following. The advantages of using process-structures shall be anticipated here:

- **Determinism**: Process-structures represent the complete finite sequence of actions that have to be carried out during the execution of a task. Due to the possibility of a bijective transformation from process-structures to Petri-Nets, a-priori verification with respect to deadlocks, reachability and liveness becomes possible. Thus, the task planner and executor, as part of the layered architecture, operate deterministically when using verified task-knowledge.
- **Real-time capability***:* Additionally, the complexity of the task planning process satisfies real-time execution requirements, because this process is reduced to a graph search problem within the state-graph of the associated Petri-Net.
- **Fault-Tolerance**: Erroneous execution results are explicitly modeled within process-structures. Additionally, redundant behavior is programmatically foreseen. If an alternative robotic operation, which shall cope with the unexpected result, is not available, the user is included as part of a semi-autonomous task execution process.

To be able to provide a user-friendly configuration of process-structures and to guarantee consistency throughout all abstraction levels of task-knowledge, a tool-based process model – the FRIEND::Process – has been developing. The process model, on the one hand, guides the **development and programming** of intelligent behavior for service robots with process-structures. On the other hand, process-structures can be seen as a **process model for the service robot** itself, which guides the task execution of the robot during runtime. The unique feature of the FRIEND::Process in comparison to other frameworks (Gostai, 2011; Microsoft, 2011; Quigley et al., 2009) and the above mentioned control architectures is to completely rely on configurable process-structures and thus on determinism, real-time capability and fault tolerance.

The FRIEND::Process consists of the following development steps:

- **Analysis of Scenario and Task Sequence:** A scenario is split up into a sequence of tasks.
- **Configuration of Object Templates and Abstract Process-Structures:** The task participating objects are specified as Object Templates and pictographic process-structures on the symbolic (abstract) level are configured and verified.
- **Configuration of Elementary Process-Structures:** Process-structures on the level of system resources and sub-symbolic (geometric) information are configured and verified with the help of function block networks.
- **Configuration and Testing of Reactive Process-Structures:** Process-structures on the level of algorithms and closed loop control, operating sensors and actuators, are configured and tested, also with configurable function blocks.
- **Task Testing:** Task planning and execution is applied on all levels of process-structures and a complete and complex task execution is tested.

In the following Section 2, the motivation for the introduction of process-structures is explained in more detail by discussing the complexity of task planning for service robots with the help of an exemplary scenario. The description of the FRIEND::Process development steps is subject of Section 3. Throughout this description, exemplary process-structures of the sample scenario of Section 2 are introduced for each development step.

Finally, Section 4 summarizes and concludes the description of the FRIEND::Process for programming intelligent service robots.

2. Task planning on basis of process-structures

In this section, the complexity of classical task planning approaches is discussed first, before the introduction of process-structures is motivated. The discussion is carried out with the help of task execution examples from the field of rehabilitation robotics and the rehabilitation robot FRIEND III (IAT, 2009; Martens et al., 2007).

2.1 The complexity of classical task-planning approaches

With respect to one exemplary task – a service robot is supporting the preparation and the eating of a meal by a disabled person – the complexity of robotic task execution shall be illustrated. For this purpose the figures Fig. 1 to Fig. 3 are introduced. Fig. 1 shows the rehabilitation robot FRIEND III which is used as exemplary target system. In Fig. 2 snapshots of the task sequence "Meal preparation and eating assistance" are depicted. Finally, Fig. 3 shows the decomposition of this task sequence according to the principles to be presented in detail in this paper.

FRIEND III is a general purpose semi-autonomous rehabilitation robot suitable for the implementation of a wide range of support tasks. As depicted, FRIEND III consists of an electrical wheelchair which is equipped with several sensors and actuators: A stereo camera system mounted on a pan-tilt-head, force torque sensor, robotic arm and gripper with force control. FRIEND III has been developed by an interdisciplinary team of engineers, therapists and designers and has been tested with disabled users within the AMaRob project (IAT, 2009).

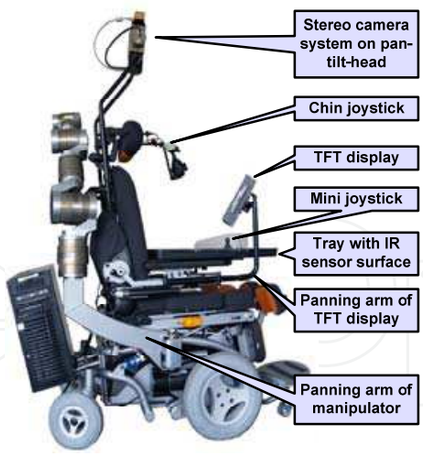


Fig. 1. FRIEND III rehabilitation robot

To perform "meal preparation and eating assistance", the robot system has to execute the following actions:

- Locate the refrigerator, open the refrigerator door, locate the meal inside the refrigerator, grasp and retrieve the meal from the refrigerator, close the refrigerator door
- Open the microwave-oven, insert the meal, close the oven, start the heating process

- Open the microwave-oven door again, grasp and retrieve the meal, close the microwave-oven door
- Place the meal in front of the user, take away the lid
- In a cycle, take food with the spoon and serve it near the user's mouth, finally put the spoon back to the meal-tray
- Clear the wheelchair tray

Fig. 2. Task sequence for meal preparation and eating assistance

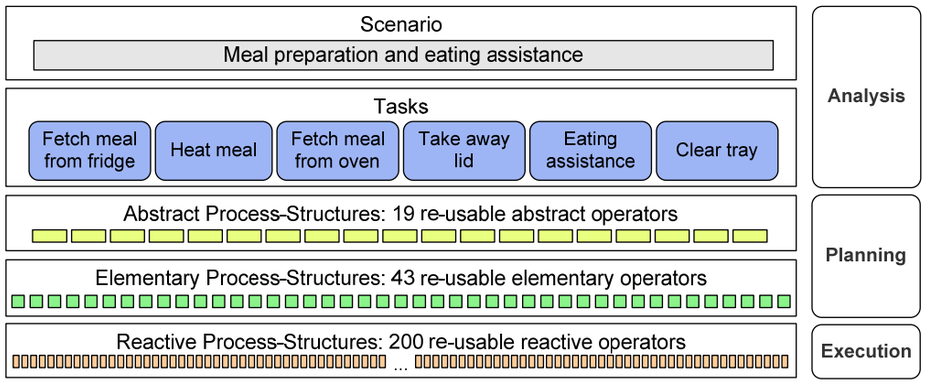


Fig. 3. Decomposition of a scenario on four abstraction levels, illustrated with the sample scenario "Meal preparation and eating assistance"

As shown in Fig. 3, the overall scenario is decomposed into tasks, abstract operators, elementary operators and reactive operators according to the layered control architecture MASSiVE. Abstract process-structures (PS_A[3]) model behavior on task planning level and elementary process-structures (PS_E) model behavior on system planning level. The reactive process-structures (PS_R) define reactive operations on the executable algorithmic level. From viewpoint of task planning, the "meal preparation and eating assistance" scenario is split up into 6 tasks, 19 abstract operators and 43 elementary task planning operators. Additionally, a large set of reactive operators is required within the execution layer.
In typical human environments, it is impossible to predefine a static sequence of operators beforehand. Many dynamic aspects resulting from dynamic environmental changes have to be

[3] Find all abbreviations in the glossary at the end of this paper.

considered, e. g. caused by changing lighting conditions, arbitrarily placed and filled objects, changing locations of objects and the robotic platform, various obstacles, and many more. Consequently, a strategy to plan a sequence of actions that fulfills a certain task is mandatory.
Many task planners are based upon deliberative approaches according to classical artificial intelligence. Typically, the robotic system models the world with the help of symbolic facts (e. g. first order predicate logic, (Russel & Norvig, 2003)), where each node of a graph represents a state (snapshot) of the world. The planner has to find a sequence of operations which transforms a given initial state into a desired target state. In the worst cases this leads to NP-complete problems, as there is an exponential complexity of classical search algorithms (Russel & Norvig, 2003). If we consider breadth-first search as a simple example, a calculation time of hours results at search depth 8; and with a depth of 14, hundreds of years are required for exhaustive search (branching factor 10 and calculation time of 10.000 nodes/s are assumed). The search depth is related to the number of required operators for a certain task and the branching factor results from the number of applicable operators in one node. Compared to the number of required and available operations shown in Fig. 3 it becomes obvious that only trivial problems can be solved on this basis. Certainly, the mean search time can be improved in comparison to breadth-first search, with e. g. heuristic approaches like A*, with hierarchical planning, search in the space of plans or successive reduction of abstraction (Russel & Norvig, 2003; Weld, 1999), but in worst cases a planning complexity as mentioned has to be faced. Even though the improvements of deliberative task planners are notable, it is still questionable whether they are efficient (real-time capable) and robust (deterministic and fault-tolerant) enough for the application in real world domains (Cao & Sanderson, 1998; Dario et al., 2004; Russel & Norvig, 2003).

2.2 Process-structures as alternative to classical planning approaches

An alternative to deliberative systems are assembly planning systems. Cao and Sanderson proposed such an approach for the application to service robotics (Cao & Sanderson, 1998). Based on this idea, Martens developed a software-technical framework (Martens, 2003) that operates on pre-structured task-knowledge, called *process-structures*. Table 1 summarizes the concept of process-structures and the distinction of task level, system level and algorithmic level.
Fig. 4 shows an example of an abstract process-structure that models the fetching of a cup from a container. The object constellations (OC) model the physical contact situation of the involved objects box (B), container (C), gripper (G) and table (T). The object constellations are connected via composed operators (COPs). These are in most cases (i. e. where this is physically meaningful) bi-directional operators. To be able to perform task planning based on an abstract process-structure, a set of OCs defines an initial situation and another set of OCs defines the target situation. Thus, task planning on abstract level means to find a sequence of COPs from initial to target situation. The initial situation is usually dynamically determined at runtime with the help of an initial monitoring procedure (Prenzel, 2005). The target situation is pre-determined for a certain PS_A.
A process-structure contains a context-related subset of task-knowledge. The finite size of a process-structure makes planning in real-time with short time intervals as well as a priori verification possible. The logical correctness of a structure is checked against a set of rules. A positive result of this check guarantees that no system resource conflicts exist. It also guarantees the correct control and data flow. Altogether, the concept of process-structures is the basis for a robust system runtime behavior. Despite pre-structuring, the process-

structures are still flexible to adapt to diverse objects, so that their re-usability in different scenarios is achieved. Technical details of process-structures beyond this summarized concept description can be found in (Martens et al., 2007).

PS_A	**Task Level**
Defines what happens	Models e. g. the fetching of an object
Is configured by:	Non-technical personnel or the user
PS_E	**System Level**
Defines how something happens from system perspective	Models the usage of system resources and the control and data flow
Is configured by:	System programmer
PS_R	**Algorithmic Level**
Defines how something happens from perspective of reactive algorithms	Models the combined usage of hardware sensors and actuators
Is configured by:	System programmer

Table 1. Summarized concept of process-structures

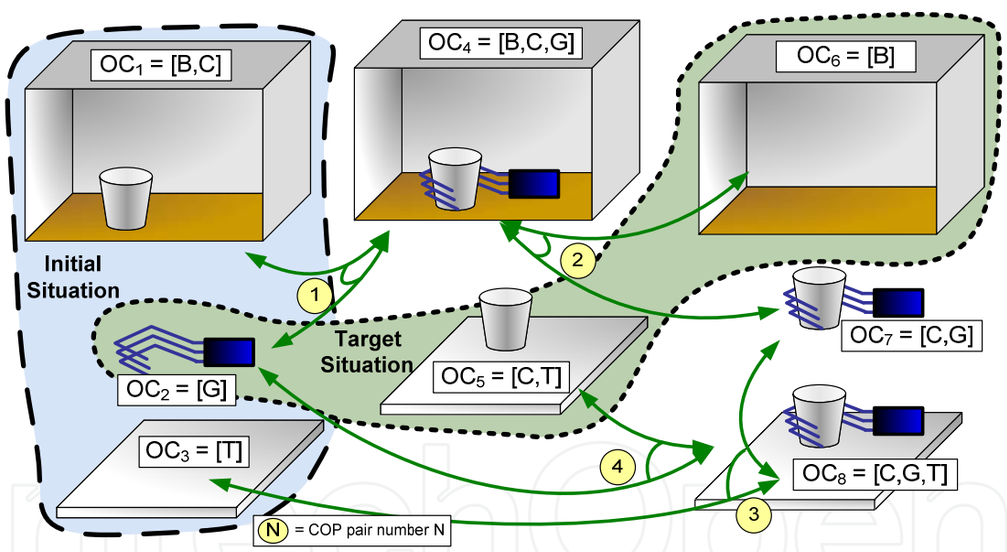


Fig. 4. Schematic illustration of an abstract process-structure (PS_A) which models the fetching of a cup from a container-like place as e. g. a fridge or a cupboard

The applicability of process-structures for the programming of service robots has been shown in (Martens, 2003) with the help of several representative rehabilitation robotic scenarios. As anticipated in the introduction this approach has been extended during the AMaRob project (2006 – 2009) and within (Prenzel, 2009) to embed the process-structure-based programming into a process model – the FRIEND::Process. From task analysis to final testing of implemented system capabilities, the FRIEND::Process guides through the complete development cycle of a service robot based on a closed chain of user-friendly configuration tools. Enhancements of the FRIEND::Process are matter of ongoing developments.

3. The FRIEND::Process

Process models structure complex processes in manifold application areas. With respect to system- and software-engineering, a process model shall organize the steps of development, the tools to be used and finally the artifacts to be produced throughout the different development stages. The overall scheme of the FRIEND::Process is depicted in Fig. 5. Central elements of the process and consequently the specialty in comparison to other process models are the process-structures. Within the development steps, the building blocks of process-structures are decomposed as shown in Table 2. In the following sections the five development steps of the FRIEND::Process are discussed in detail. Thus, the contents of Table 2, i. e. the composition of process-structures and the decomposition on the next level as well as the abbreviations will be explained. Also, the application of the FRIEND::Process for the development of the sample task of "meal preparation and eating assistance" is shown in each step.

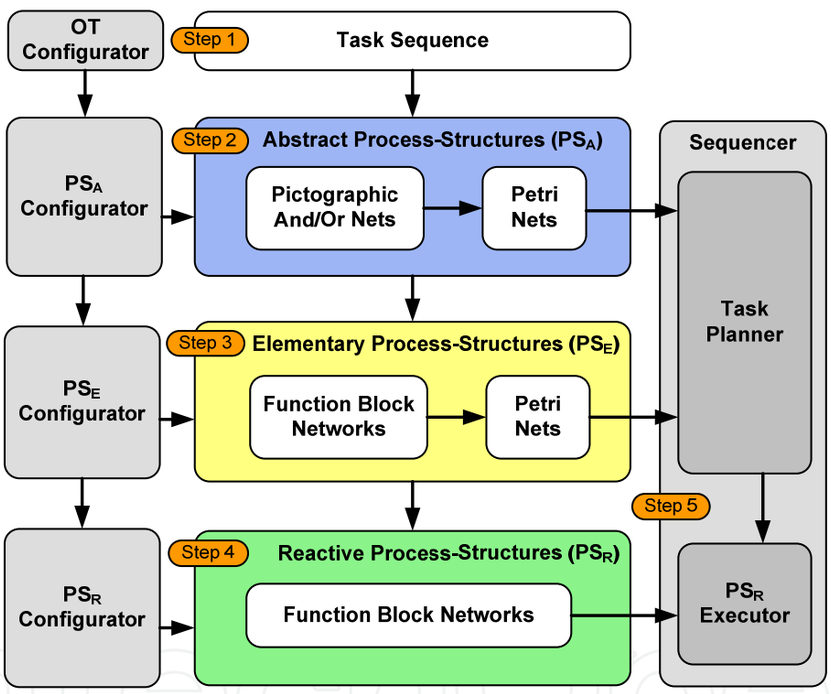


Fig. 5. Scheme of the FRIEND::Process with five development steps and the respective process-structure levels as well as the involved tools for configuration, planning and execution

Process-Structure Decomposition	Process-Structure Building Blocks
Scenario → Task Sequence	Tasks
Task → PS_A	System, Object Templates (OTs), Object Constellations (OCs), Facts, Composed Operators (COPs)
COP → PS_E	System, Object Templates (OTs), Facts, Skills
Skill → PS_R	System, Object Templates (OTs), Reactive Blocks

Table 2. Decomposition and building blocks of process-structures

3.1 FRIEND::Process step 1: Analysis of scenario and task sequence

Development according to the FRIEND::Process starts with the "Scenario Analysis" as step 1. Unlike the subsequent steps, this step is not (yet) tool-supported. The scenario analysis splits up a complex scenario like "meal preparation and eating assistance" into a sequence of re-usable tasks. Also, a structured collection of the objects takes place that are in the focus of a certain scenario.

3.1.1 Description of the process step

The development step 1 is dedicated to a first analysis of the desired task execution scenario. As shown in Fig. 6 a sequence of re-usable tasks is specified. Besides the strictly sequential concatenation of tasks, cyclic repetitions are also possible, as e. g. required for the eating assistance scenario introduced at the beginning of the paper.

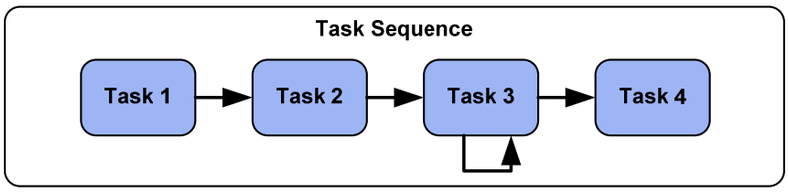


Fig. 6. A complex task sequence consists of several tasks

The FRIEND::Process defines criteria for task splitting:

- **Modularity, low complexity and re-usability:** One task is focusing on a set of objects. This set shall be kept as small as possible to limit the task's complexity and to ensure re-usability of a task. It shall be possible to use the tasks independently, but also to concatenate them to more complex action sequences.
- **The typical physical location of the objects:** If movement of the robotic platform is required, this is a clear indicator to switch the task context, e. g. when moving from fridge to microwave oven in the meal preparation scenario. After moving the platform, relative locations between platform and objects have to be re-assessed.

Currently, the process step 1 is not yet supported by a dedicated tool. Therefore, to still achieve a certain level of formality, the results of scenario analysis are collected in a UML use case diagram as seen in Fig. 7. For each task a use case with verbal task description is specified. This includes the objects involved in the task, the so-called task participating objects (TPO).

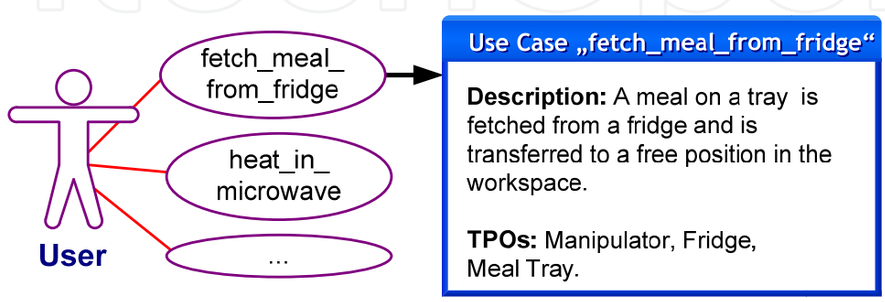


Fig. 7. Use case diagram with tasks (use cases) of the sample scenario. For each task, a detailed description as well as the set of task participating objects (TPO) is specified

The objects involved in task execution are the elements that are relevant in all subsequent development steps. To follow the principle of re-usable task-knowledge, the TPOs are specified as abstract object classes. For example, a task that describes the fetching of an object from a container-like place (see Fig. 4) can be re-used to fetch either a bottle or a meal from the refrigerator. In the FRIEND::Process the re-usable classes of objects are specified as hierarchical UML ontology. An exemplary ontology for the scenario "meal preparation and assistance to eat" is depicted in Fig. 8. It is depicted that the TPOs are constructed from basic geometric bodies (cuboid and cylinder) and more complex objects are created with inheritance and aggregation.

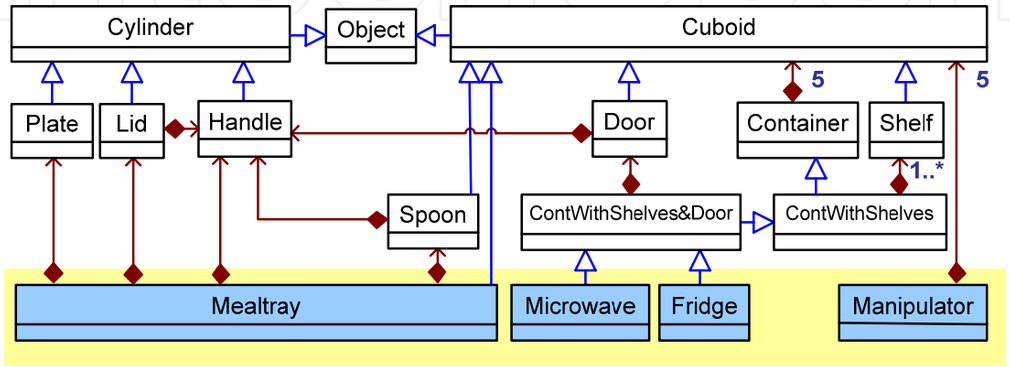


Fig. 8. Ontology of task participating objects (TPO) for the scenario "Meal preparation and assistance to eat"

To embed the TPOs in the tool-chain that covers all further development steps, the concept of "Object Templates" (OT) has been introduced (Kampe & Gräser, 2010). The configuration of Object Templates and their integration into the different levels of process-structure configuration will be discussed in more detail within the following process steps.

3.2 FRIEND::Process step 2: Configuration of object templates and abstract process-structures

In this development step the task participating objects are formally specified and configured with the help of Object Templates. Subsequently, an abstract process-structure (PS_A) is configured based on pictographic And/Or-Nets. This means that physical object constellations (OC) and physical transitions between the object constellations are specified. Besides configuration of PS_A, the logical correctness of the abstract process-structures is guaranteed by the configuration tool. Finally, the pictographic PS_A are converted to Petri-Nets according to (Cao & Sanderson, 1998) for the input into the task planner.
In the following, a description of the process step is introduced first. Afterwards, the configuration concept for Object Templates is shown. Finally, the configuration of an abstract process-structure is exemplified.

3.2.1 Description of the process step

As shown in Fig. 9 the FRIEND::Process decomposes each task into an abstract process-structure (PS_A). A schematic exemplary pictographic PS_A for the task "Fetch cup from container" has already been introduced and discussed in Fig. 4. Within the FRIEND::Process, the configuration of PS_A is carried out within a pictographic configuration

environment, the so-called PS_A-Configurator. Fig. 10 shows the PS_A-Configurator with the PS_A "Fetch meal from fridge".

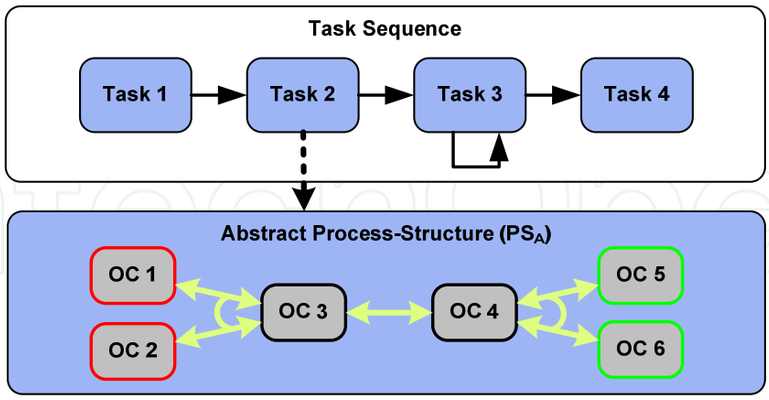


Fig. 9. Decomposition of a task as abstract process-structure with object constellations (OC) and composed operators (COP)

The procedure of PS_A configuration is as follows:

- Selection of task participating objects (TPOs)
- Composition of object constellations (OCs)
- Connection of OCs via composed operators (COPs)
- Selection of default initial and default target situation

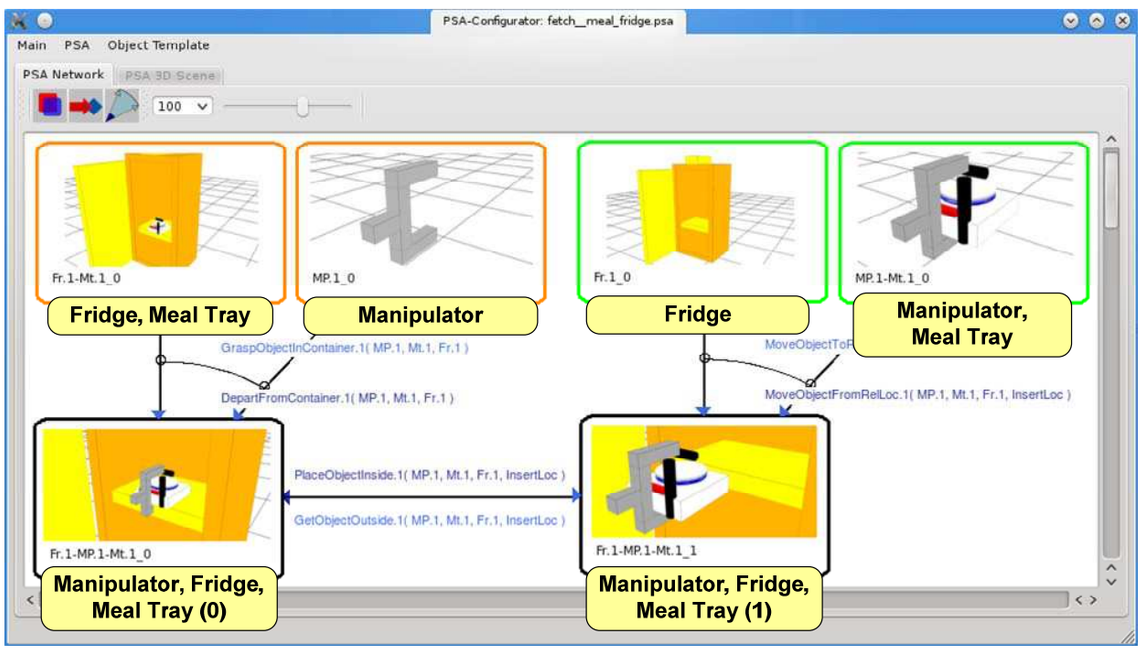


Fig. 10. PS_A-Configurator with the pictographic abstract process-structure modeling the Task "Fetch meal from fridge"[4]

[4] For better readability, overlays have been added in this illustration.

The pictographic representation of an OC is configured within a sub-dialog within the PS_A-Configurator. Within this configuration dialog, the predicate logic facts, which are assigned to an OC, can be inspected. These facts are the pre- and post-condition facts of the COPs that interconnect the OCs. Within the constraints given by the COP facts, the pictographic appearance of an OC can be adjusted within the PS_A-Configurator within a 3D scene. The rendering of object constellations is based on "Object Templates".

3.2.2 Object templates

Objects play a central role in process-structures. The different levels of process-structures model different aspects of objects. On abstract level, a symbol is associated with an object for the purpose of task planning (e. g. "Mt.1" for the meal tray in the sample scenario). On system level, i. e. on the level of elementary process-structures, so-called sub-symbolic (i. e. geometric) object information is processed. With respect to the meal tray this is, for instance, the location to grasp the tray. To model the different aspects of objects and to assure an information consistency throughout the different information layers, the concept of Object Templates has been introduced.

Object Templates comprise the following aspects:

- A 3D model of the object, used for pictographic rendering of object constellations on PS_A level as well as for motion planning and collision avoidance on PS_R level
- Associated sub-symbolic (geometric) data for planning and execution on PS_E and PS_R level, e. g. the grasping location of an object
- Complex objects can be composed of simpler objects; e. g. a meal tray consists of a tray, a plate, a lid and a spoon
- Object Templates are configured with natural parameters of the composed object, e. g. width, height, depth and wall thickness for a container, instead of separate specification of all geometric primitives
- The 3D appearance of Object Templates is associated with task-knowledge like symbolic facts and characteristics. For example the fact "IsAccessible(MicrowaveOven)" renders the opening status of the door of the oven's 3D model.

An exemplary Object Template is the meal tray depicted in Fig. 11. It consists of a base tray, a plate with a lid and a spoon. Both the lid and the spoon are detachable from the meal tray. The different stages of separation are depicted in Fig. 12.

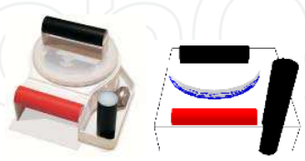


Fig. 11. The meal tray of the eating scenario as photo (left) and modeled by means of an Object Template (right)

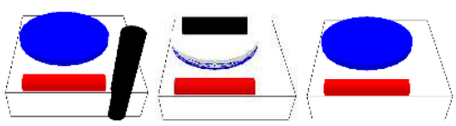


Fig. 12. The different separation stages (detached lid, detached spoon, both lid and spoon detached) of the meal tray

The configuration of Object Templates takes place within the Object-Template-Configurator (OT-Configurator) which is part of the PS_A-Configurator as shown in Fig. 13.

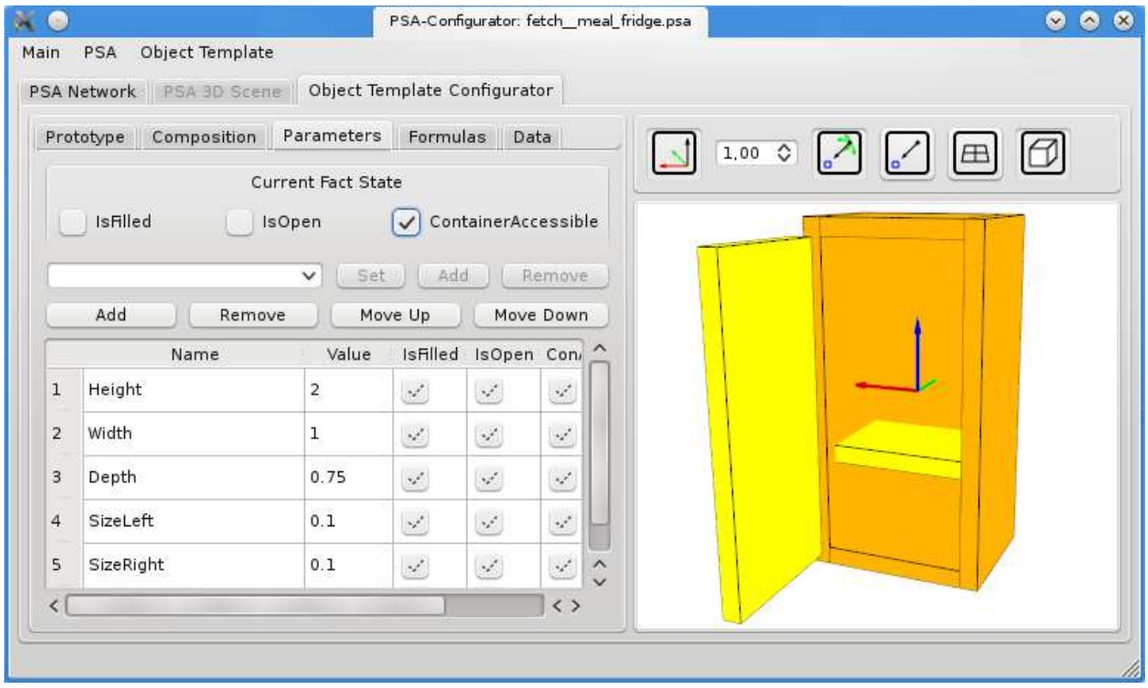


Fig. 13. Object-Template-Configurator (OT-Configurator) as part of the PS_A-Configurator

Within the screenshot in Fig. 13 the Object Template of the refrigerator is modeled. On the left side the parameters and their association to symbolic facts are specified. On the right side the 3D model of the object is rendered according to the current configuration. To render the 3D model of a composed object, the aggregated sub-objects are composed with formulas within the Object-Template-Configurator tool. Frequently required and complex formulas like alignment and rotation of Object Templates are provided with the help of assistive functions.
Certain aspects of the 3D geometry have a fix association with object characteristics as given in the following table:

Characteristic	Associated sub-symbolic element
IsGrippable	Coordinates to grasp the object
IsPlatform	Limits to place other objects onto this object
IsContainer	Limits to place other objects within this object

Table 3. Relations between characteristics and sub-symbolic elements

3.2.3 Exemplary abstract Process-Structure: Fetch meal tray from fridge

The exemplary PS_A that shall be discussed in detail has already been introduced within the PS_A-Configurator frontend in Fig. 10. In this PS_A the task participating object are a fridge (symbol "Fr" with instance number "1" → "Fr.1"), a meal tray ("Mt.1"), the manipulator ("MP.1") and the abstract symbol for a relative location ("InsertLoc"). In this PS_A the initial

situation consists of two object constellations. The first one models the manipulator in a free position in the workspace (instance number "0" is assigned to this object constellation → "MP.1_0"). The second object constellation models the already opened fridge containing the meal tray ("Fr.1-Mt.1_0"). The two OCs are connected via the assembly COP "GraspObjectInContainer(MP.1, Mt.1, Fr.1)". If physically possible, a complementary disassembly operator is assigned to model the reverse operation for re-usage of the PS_A in another scenario context. In this case this is the COP "DepartFromContainer(MP.1, Mt.1, Fr.1)". The assembled object constellation is depicted on the bottom left side and the associated abstract planning symbol is "Fr.1-MP.1-Mt.1_0". Due to the associated symbolic facts, which are imposed within the object constellation by the post-condition facts of the COP, the pictographic representation is rendered so that the manipulator grasps the meal tray in the fridge.

Besides assembly and disassembly operators, the And/Or-Net syntax provides operators modeling the internal state transition (IST) of object constellations (IST COPs). IST COPs are applied when the physical contact state of the involved objects is not changed. From the viewpoint of planning on abstract level, objects being in close relative locations to each other are considered to be in a physical contact situation. Therefore, the IST COP "GetObjectOutside(MP.1, Mt.1, Fr.1, InsertLoc)" is applied to transform the OC "Fr.1-MP.1-Mt.1_0" on the left side into the OC "Fr.1-MP.1-Mt.1_1" on the right side. Finally, the COP "MoveObjectFromRelLoc(MP.1, Mt.1, Fr.1, InsertLoc)" models the disassembly operation and results in two object constellations which model the target situation of this abstract process-structure: "Fr.1_0" is the empty fridge and "MP.1-Mt.1_0" is the manipulator with the gripped meal tray in a free position in the work space.

To be able to develop and verify the three levels of process-structures independently, i. e. in a modular manner, the consistency of task-knowledge on all levels has to be assured. This is achieved with common building blocks of the different process-structures as shown in the decomposition chain in Table 2. The common elements are the interfaces to the next level of process-structures. The important interfacing elements between PS_A and PS_E are the pre- and post-condition facts of the COP to be decomposed as PS_E in the next process step. For the COP "GraspObjectInContainer" the facts are shown in Table 4.

Pre-Facts	**Post-Facts**
HoldsNothing(Manipulator) = True	HoldsNothing(Manipulator) = False
IsInFreePos(Manipulator) = True	IsInFreePos(Manipulator) = False
-	IsGripped(Manipulator, Object) = True
ContainerAccessible(Container) = True	-
IsInsideContainer(Object, Container) = True	-

Table 4. Pre- and Post facts of COP "GraspObjectInContainer(Manipulator, Object, Container)"

3.3 FRIEND::Process step 3: Configuration of elementary process-structures

In the third process step, each composed operator (COP) of an abstract process-structure (PS_A) is decomposed into an elementary process-structure (PS_E). To achieve user-friendly configuration of PS_E, configurable function blocks are assembled to function block networks (FBN). Each function block models a reactive robot system operation, also called skill. A

priori verification of task-knowledge on this level takes place with the help of Petri-Nets, which result from automatic conversion of FBNs.

3.3.1 Description of the process step

Fig. 14 depicts the decomposition principle of COPs into elementary process-structures, consisting of skill blocks. An elementary process-structure, as first introduced by (Martens, 2003), is a Petri-Net with enhanced syntax and superordinated construction rules. The advantage of Petri-Nets is their ability to model parallel activities. This is useful for the behavioral modeling on robot system level, for instance, if a manipulator action is guided by input from a camera system or another sensor. Furthermore, Petri-Net-based PS_E offer mathematical methods for analysis of the reachability of a certain system state, for verification of the correctness of control and dataflow and for the exclusion of resource conflicts (Martens, 2003).

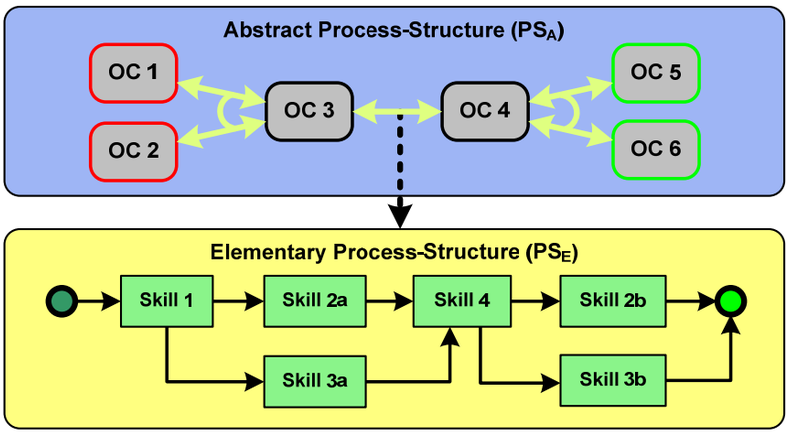


Fig. 14. Decomposition of a composed operator (COP) as elementary process-structure

Besides these conceptual advantages, from the viewpoint of implementation it turned out that the programming of elementary process-structures with Petri-Nets is a time consuming and error prone procedure. The setup of a correctly verified Petri-net-PS_E usually takes several hours. Even with strong modularization of the networks, the large number of places and transitions leads to hardly manageable Petri-Nets in real-life applications. This is the reason why the FRIEND::Process introduces the configuration of PS_E on the basis of function block networks (FBN). Similar to the PS_A-Configurator, a configuration frontend, called PS_E-Configurator, has been created. This tool subsumes all logical and syntactical rules that are required for PS_E-configuration. Furthermore, a conversion algorithm has been developed (Prenzel et al., 2008), which converts an FBN into a Petri-Net for automatic execution of verification routines, like a reachability analysis. A screenshot of the PS_E-Configurator with the PS_E "GraspObjectInContainer" is given in Fig. 15.

With respect to their representative function for Petri-Nets, the control flow within the FBN structures is token-oriented. The execution starts from the "Start" block and ends at the "Target Success" block. In-between, reactive skills are executed, including manipulative operations as well as sensor operations or user interactions. Each function block has one input port, and several output ports according to the possible execution results of the skill (see e. g. block "CoarseApproachToObjectInContainer" in Fig. 15 with the output ports "Success", "Failure", "Abort" and "UserTakeOver"). The output port "Abort" is not

explicitly connected to an abort block to increase the readability of the network structure. The typical construction rule for a semi-autonomous system (like FRIEND) is to provide user interactions as redundant action for autonomous system operations. As shown in Fig. 15, the failure of an autonomous operation (e. g. "AcquireObjectBySCam") is linked to the user interaction "DetermineObjectBySCam", replacing the failed system action.

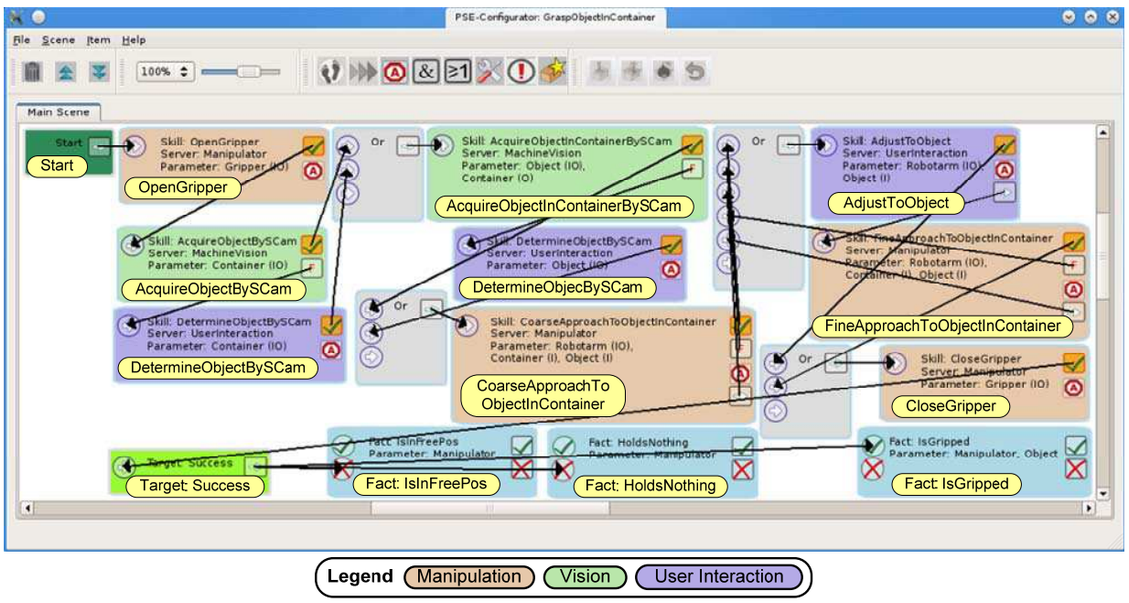


Fig. 15. PS_E-Configurator with elementary process-structure as function block network, modeling the COP "GraspObjectInContainer".[5]

The configuration of PS_E on the basis of function block networks does not only achieve a decisive increase of development comfort (configuration instead of programming), but it also decreases the required task-knowledge engineering time significantly. By building the PS_E directly in the correct manner, the time-consumption for the construction of one PS_E is reduced from hours to 10-15 minutes per network. On this basis, real world problems like the "Meal preparation and assistance" task, become manageable in their complexity.

3.3.2 Exemplary elementary process-structure: Manipulator grasps meal tray in fridge

The exemplary PS_E "GraspObjectInContainer", as shown in Fig. 15, models the grasping of an object in a container-like place in a general way. In the sample scenario "meal preparation" this PS_E is applied to fetch the meal tray from the refrigerator and also from the microwave oven after heating of the meal.

The objects (Object Templates) "Manipulator", "Object" and "Container", which are involved in this PS_E, are the input artifacts handed over as COP parameters from the previous step of the FRIEND::Process. The first skill block that follows the "Start" block is the manipulator skill "OpenGripper". Subsequently, the container (fridge) is located with the help of the vision skill "AcquireObjectBySCam(Object)". This skill calculates the location and size of the given object with the help of a stereo camera (SCam). In the sample scenario the COP parameter "Container" (i. e. the fridge) is inserted at the skill's placeholder

[5] For better readability, overlays have been added in this illustration.

"Object" according to the principle of type-conform parameter replacement (Martens, 2003). The Object Template of a fridge provides the according two sub-symbolic parameters location and size. A successful execution of the skill guarantees that the container's location and size are stored in the system's world model and can serve as input parameters for subsequent skills. After verification of the associated Petri-Net of this PS_E the correctness of the data flow between all skill blocks is assured. If the recognition of the fridge is successful and the user has not to be involved, the skill "AcquireObjectInContainerBySCam" is executed to determine the location of the meal tray in the fridge. Afterwards, a "CoarseApproachToObjectInContainer" follows. This skill roughly directs the manipulator in front of the meal tray in the fridge based on the location information calculated beforehand. Fig. 15 depicts that this manipulator skill is followed by an enforced user interaction, since all output ports are connected to the Or-block preceding the user interaction. The confirmation by the user is included at this place due to testing purposes to assure a correct execution of the first skill. For real task-execution a quick reconfiguration of the PS_E will change the system behavior and directly execute the next manipulator skill "FineApproachToObjectInContainer". This skill leads to a final grasping of the meal tray handle, while avoiding collisions of the manipulator with the fridge with the help of dedicated methods for collision avoidance and path planning (Ojdanic, 2009). The final action necessary to complete the grasping is to close the gripper. The PS_E ends with setting the post-facts of the COP as specified in Table 4.

From the viewpoint of the system's task planner, each skill-function-block represents an elementary (executable) operation. Within the execution level of the system, the operations are not seen as atomic units. The execution of one skill means to activate reactive system functionality, for instance the sensor-controlled approach of an object to be grasped in the skill "FineApproachToObjectInContainer". These basic system skills have to couple sensors and actuators on the algorithmic level. To pursue the paradigm of configurable process-structures also on this level, the FRIEND::Process introduces reactive process-structures.

3.4 FRIEND::Process step 4: Configuration and testing of reactive process-structures

Historically, during the elaboration of the FRIEND::Process, the elementary operators (skills) have been implemented directly in C++. Subsequently, when appropriate CASE-tools became available, the elementary operators have been implemented with model driven development techniques (Schmidt, 2006) as executable UML models. Then, a configuration tool has been developed, which makes user-friendly configuration of process-structures possible also on this development level. With the help of this tool it is assured that the verified interfaces from the PS_E-layer are respected and the robustness assertion throughout the complete system architecture is maintained.

3.4.1 Description of the process step

Fig. 16 depicts the decomposition of a skill block from PS_E-layer into a reactive process-structure (PS_R) consisting of algorithmic blocks. Similar to the PS_E function blocks, PS_R are also based on configurable function block networks. The PS_R-Configurator tool results from the Open-Source Image Nets Framework[6], which originally has been developed for configurable image processing algorithms.

[6] http://imagenets.sourceforge.net/

The PS_R Configuration Framework consists of the following five parts (see Fig. 17):

- PS_R-Configurator,
- Embedding of PS_R into any C++ code via PS_R-Executor,
- Reactive process-structures (PS_R), which are executable function block networks,
- Extensible set of Plug-Ins and
- Configurable function blocks

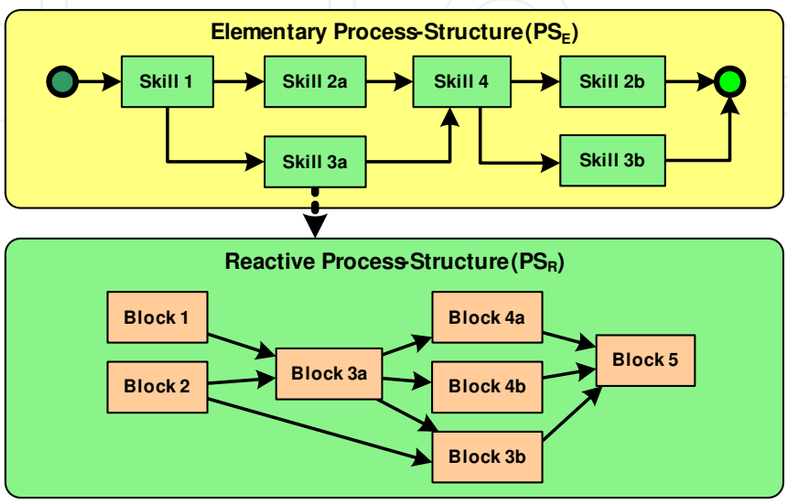


Fig. 16. Decomposition of a skill into a reactive process-structure

The "PSR-Configurator" is a graphical user interface, which can be used to rapidly create a "function block network", namely a reactive process-structure (PS_R). With the PS_R-Executor, it is possible to load and execute the previously configured PS_R. The PS_R itself is a directed graph, connecting configurable "function blocks". Each block can execute code to process its input (image data or other data) and save its outputs. One or more blocks are grouped in a "Plug-In" and an arbitrary number of Plug-Ins can be loaded dynamically by the PS_R. In this way, the PS_R-Framework can be easily extended by new independent Plug-Ins. This independency of the algorithmic modules results in completely independent development within a team of developers. In addition, the strong modularization leads to a technically manageable amount of code within a single block and reduces the time of inspecting an erroneous block. The PS_R execution library can save a PS_R in human readable XML format. Thus, on the one hand the PS_R-Configurator can configure, load and save a PS_R, but on the other hand also external C++ code can load a PS_R file.

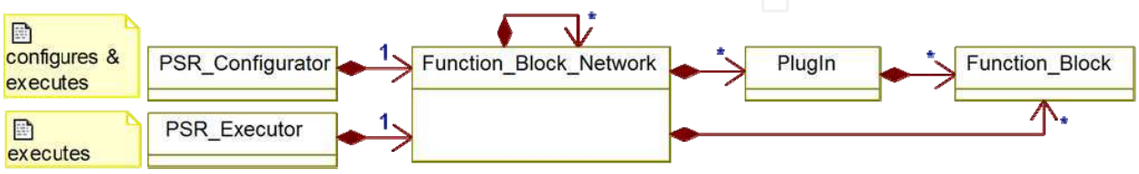


Fig. 17. The UML structure of the PS_R-Configuration Framework

Hierarchical modeling is a common method to subdivide algorithms into separate parts - it breaks down the complexity and facilitates reusability. In the PS_R-Framework, parts can be constructed as separate PS_R and can be combined afterwards to constitute a complete

algorithm. The PS_R-Executor is in fact also a function block, which can load and process a PS_R. The connection between the PS_R inside an Executor and the outer net is established by special input and output blocks. For example the PS_R "Color2Color3D" shown in Fig. 18 calculates a colored point cloud out of a stereo image pair. On the left side there are two input blocks, which hand over the images from the block in orange. This block only exists in this PS_R for testing the net and will be ignored on execution if this PS_R is loaded by a PS_R-Executor (see Fig. 19, right side).

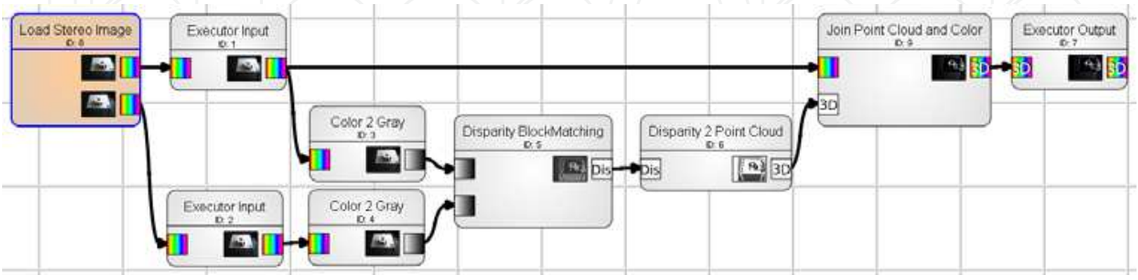


Fig. 18. The functionality of calculating a colored point cloud out of a stereo image pair is depicted in this PS_R, called "Color2Color3D"

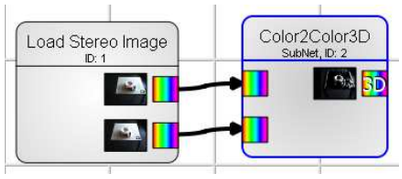


Fig. 19. The previously shown PS_R can be loaded as one PS_R-Executor block

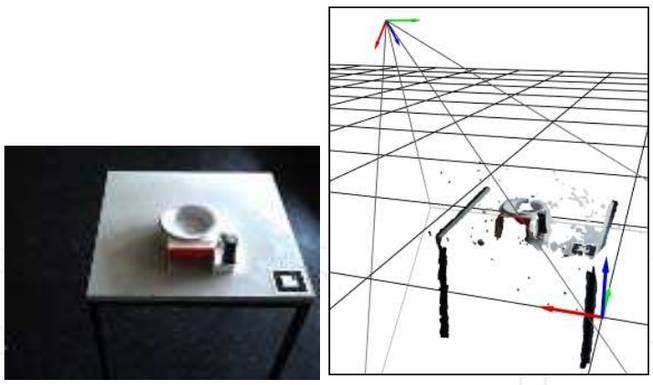


Fig. 20. Left: original image, right: resulting point cloud of the stereo camera images visualized in 3D by the PS_R-Configurator

The PS_R in Fig. 19 shows the use of a subnet of an image acquisition together with the calculation of the extrinsic matrices of a stereo camera, which describe the relation of the cameras to the robot. These matrices depend on an invariant transformation frame inside the pan-tilt-head (see Fig. 1) and its rotation angles. By combining the two subnets, a live view of the stereo camera's point cloud can be calculated (depicted in Fig. 20, in the center of the images the meal tray can be seen). As a visually guided robot is a real world object, which moves in the three dimensional Cartesian space, it is useful to display the vision

results in the same space. While configuring a PS_R with the PS_R-Configurator, intermediate results can be visualized in two and three dimensions; depending on the data type, for example scalar values can only be visualized in 2D, camera matrices can be visualized in 2D and 3D (using OpenGL (Wright et al., 2010), see Fig. 20, right).
To be able to execute a PS_R as skill block within the context of the PS_E layer and to guarantee that the PS_E interfaces are respected, a special type of "Verified PS_R-Executor block" is created. During configuration of this kind of block, the PS_R-Configurator checks that the used resources as well as input and output parameters match the specification of a certain PS_E skill to be modeled as PS_R. For example in the case of the PS_R "AquireObjectBySCam(Object)" the allowed resource is the stereo camera system. The input parameter is the Object Template of the given object and the output parameter are the return values "Success" and "Failure".

3.4.2 Exemplary reactive process-structure: Acquire meal tray by stereo camera

To show the capabilities of the reactive process-structures, a simplified example is discussed in the following, namely the machine vision skill to acquire an object by the stereo camera with the configuration "Meal Tray". This example of a PS_R is non-reactive, as no actor is involved. Though, in a more complex PS_R, it is possible to combine the camera and the robot in a feedback loop to implement visual servoing to achieve reactive behaviour.
In Fig. 21 several general blocks are used to find the red meal tray handle in an image. The processing chain starts with the detection of highly saturated, red parts. It is followed by a 9x9 closing operation to eliminate noise. Afterwards, contours are detected and filtered according to a priori knowledge of the size of the handle. Then, the minimum rectangles around the contours are determined and the major axes and their end points are calculated. For testing the current PS_R, again the orange blocks have been added to visualize intermediate testing results and they are not executed during task execution.

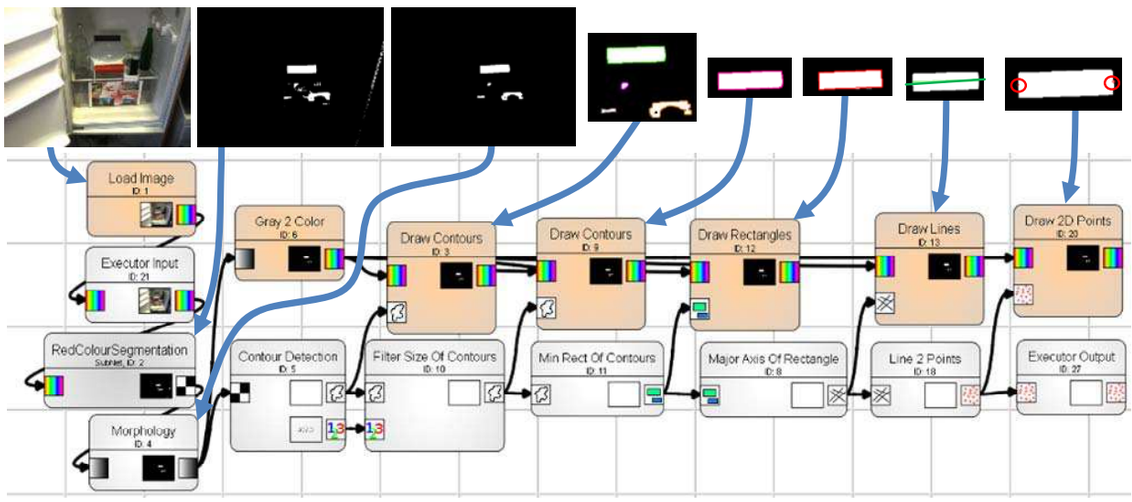


Fig. 21. PS_R "MajorAxisPoints" which detects red areas of a certain size and calculates the major axes of these areas. Orange blocks are omitted when this PS_R is used in a PS_R-Executor

To grasp the meal tray handle with the manipulator, the determination of its location in 3D is required. Thus, a 2D detection of the meal tray is not sufficient. However, the previously created and tested PS_R "MajorAxisPoints" can be used twice, one for each image of the stereo camera. Fig. 22 depicts the usage of the previous net to calculate the 3D line,

describing the handle of the meal tray. The block *Optimal Stereo Triangulation* computes a 3D contour based on key feature points, extracted from a stereo image. With the known camera matrices and the 2D feature correspondences, the 3D points are found by the intersection of two projection lines in the 3D space using optimal stereo triangulation, as described in (Natarajan et al., 2011).

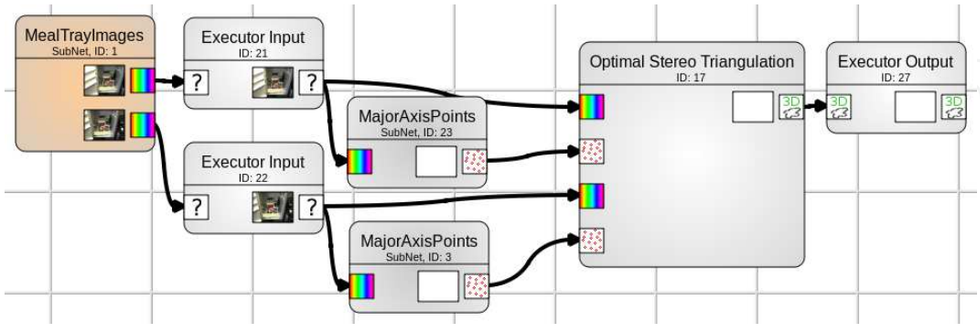


Fig. 22. PS_R which detects the meal tray handle in 3D

Next, the 3D line of the 3D handle detection is used to calculate a transformation frame, having the position of the right 3D point and the rotations to point the y-axis in line direction. Using the a priori knowledge that the meal tray should be parallel to the world coordinate system, only rotation around z-axis has to be calculated. Fig. 23 displays (top, from left to right) the 3D line, the calculated frame, the meal tray Object Template and the placed meal tray, based on the frame. For the fulfillment of the specification of the calling PS_E, the Object Template has to be written to the World Model (a service to read from and write data to) with the "Write to World Model" block. This ensures that the detected object is globally available for later processing steps and is the actual result of this PS_R.

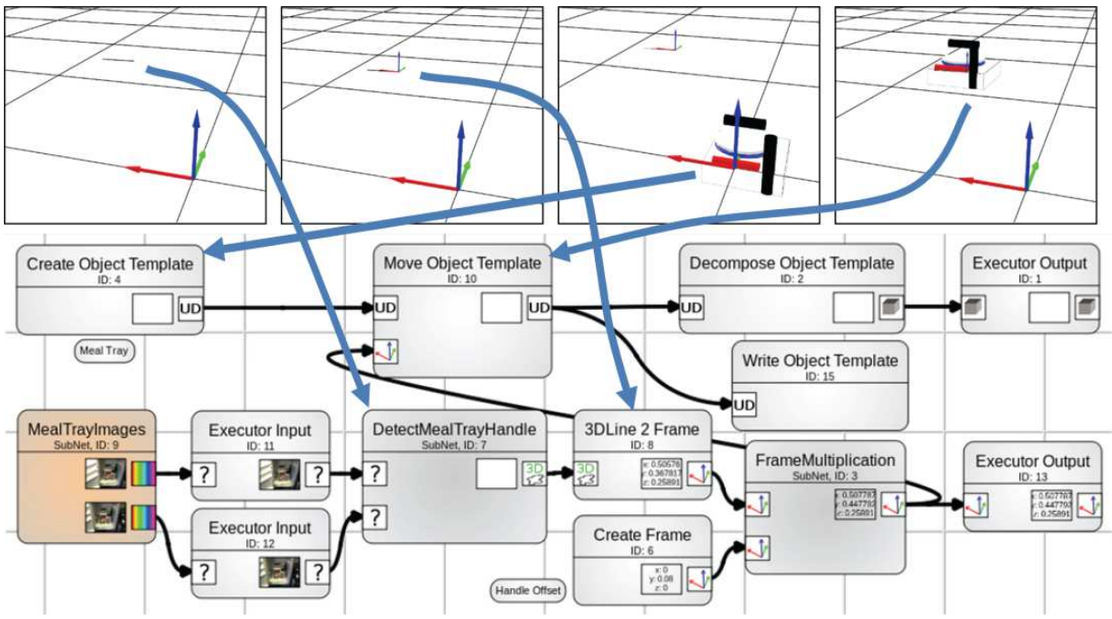


Fig. 23. Meal tray detection and Object Template placement based on 3D handle detection, frame calculation and Object Template movement (UD = user data)

For simulation and PS_R unit testing in the PS_R-Configurator, the fridge, the static environment (wheelchair, monitor and user) and the robot with its current configuration can be placed in the same 3D scene with the meal tray. Fig. 24 and Fig. 25 show the real scene and the 3D simulation result in comparison.

Fig. 24. Real scene of this PS_R

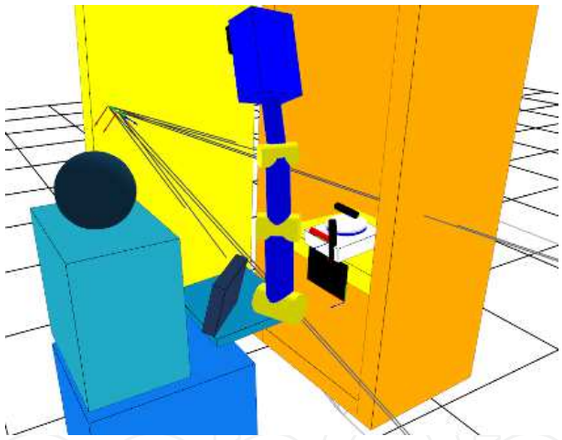


Fig. 25. Simulated scene of this PS_R

3.5 FRIEND::Process step 5: Task testing

After finishing the configuration of process-structures on all three levels, the planning and execution of a task (PS_A) has to be tested. The modularly configured, verified and tested process-structures of lower abstraction (PS_E and PS_R) are involved in this final process step.

3.5.1 Description of the process step

For the purpose of task testing the "Sequencer" is used, which embeds a task planner for process-structures and the PS_R-Executor (see Fig. 5). The Sequencer is part of the process-structure-based control architecture MASSiVE mentioned in Section 1. The Sequencer interacts with skill servers, which offer the functionality that has been configured and

verified as reactive process-structures beforehand. The layered system architecture organizes a hardware abstraction via skill layer, so that there is a unique access-point on the sensors and actuators from a certain responsible skill server.
Task tests can be performed in the following execution modes:

- *Probabilistic simulation:* the skill interfaces and the communication infrastructure are tested and skill return values are simulated,
- *Skill simulation:* the skill's functional core is simulated,
- *Motion simulation:* the motion governed by manipulative skills is simulated and visualized within a virtual 3D space as shown in Fig. 25,
- *Hardware simulation:* the sensors and actuators are simulated,
- *Real execution:* the skill is executed with access of sensors and actuators.

Based on the process-structures, a complete task is planned and executed in one of the listed skill execution modes. This means that the Sequencer first plans a sequence of COPs and subsequently decomposes each COP into an elementary process-structure. Planning on this level results in a sequence of skills to be executed then. Step by step and based on the execution result of each skill, the once planned skill sequence is pursued, or re-planning takes place if an unexpected result is obtained.

4. Conclusion

As shown in Section 2.1 it is a challenging task to establish intelligent behavior of service robots operating in human environments. Typical operation sequences of support tasks in daily life activities seem to be simple from human understanding. However, to realize them with a robotic system, a huge complexity arises due to the variability and unpredictability of human environments.
In this paper the FRIEND::Process – an engineering approach for programming robust intelligent robotic behavior – has been presented. This approach is an alternative solution in contrast to other existing approaches, since it builds on configurable process-structures as central development elements. Process-structures comprise a finite-sized and context-related set of task knowledge. This allows a priori verification of the programmed system behavior and leads to deterministic, fault-tolerant and real-time capable robotic systems.
The FRIEND::Process organizes the different stages of development and leads to consistent development artifacts. This is achieved with the help of a tool chain for user-friendly configuration of process-structures.
The applicability of the here proposed methods has been proven throughout the realization of the AMaRob project (IAT, 2009) where task execution in three complex scenarios for the support of disabled persons in daily life activities has been solved. One of theses scenarios is the "Meal preparation and eating assistance" scenario, used for exemplification throughout this paper. The most error prone and thus challenging action in this scenario is the correct recognition of smaller objects (e. g. the handle of the meal tray) under extreme lighting conditions. However, with the inclusion of redundant skills in the elementary process-structures, the system's robustness has been raised in an evolutionary manner. In cases where even redundant autonomous skills did not execute successfully, the accomplishment of the desired task was achieved via inclusion of the user within a user interaction skill.
Currently, the methods and tools discussed in this paper are continuously developed further and are applied in the project ReIntegraRob (IAT, 2011). The mid-term objective is to integrate the different configuration tools for process-structures into one integrated

configuration environment. The PS_R Configuration Framework, which is the most elaborated tool, will build the basis for this.

5. Glossary

COP	**C**omposed **Op**erator
FBN	**F**unction **B**lock **N**etwork
FRIEND	**F**unctional **R**obotarm with user-fr**IEN**dly interface for **D**isabled people
MASSiVE	**M**ultilayer Control **A**rchitecture for **S**emi-Autonomous **S**ervice Robots w**i**th **V**erified Task **E**xecution
OC	**O**bject **C**onstellation
OT	**O**bject **T**emplate
PS	**P**rocess-**S**tructure
PS_A	**A**bstract **P**rocess-**S**tructure
PS_E	**E**lementary **P**rocess-**S**tructure
PS_R	**R**eactive **P**rocess-**S**tructure
TPO	**T**ask **P**articipating **O**bject

6. References

Asimov, I. (1991). Robot Visions, Roc (Reissue 5th March 1991), ISBN-10: 0451450647

Cao, T. & Sanderson, A. C. (1998). AND/OR net representation for robotic task sequence planning, In: *IEEE Transactions on Systems, Man, and Cybernetics - part C: Applications and Reviews*, 28(2)

Dario, P., Dillman, R., and Christensen, H. I. (2004). EURON research roadmaps. Key area 1 on 'Research coordination', Available from http://www.euron.org

Engelberger, J. F. (1989), *Robotics in Service*, MIT Press, Cambridge, MA, USA, 1st ed, 1989

Gostai. (2010). Urbi 2.0. Available from http://www.gostai.com

Gräfe, V. & Bischoff, R. (2003). Past, present and future of intelligent robots, *Proceedings of the 2003 IEEE International Symposium on Computional Intelligence*, In: *Robotics and Automation* (CIRA 2003), volume 2, ISBN 0-7803-7866-0, Kobe, Japan

IAT (2009). *AMaRob Project*, Institute of Automation, University of Bremen, Germany. Available from http://www.amarob.de

IAT (2011). *ReIntegraRob Project*, Institute of Automation, University of Bremen, Germany. Available from http://www.iat.uni-bremen.de/sixcms/detail.php?id=1268

Kampe, H. & Gräser, A. (2010). Integral modelling of objects for service robotic systems, Proceedings for the joint conference of ISR 2010 (41st International Symposium on Robotics) und ROBOTIK 2010 (6th German Conference on Robotics), 978-3-8007-3273-9, Munich, Germany

Kemp, C. C., Edsinger, A. & Torres-Jara, E. (2007). Challenges for robot manipulation in human environments, In: *IEEE Robotics and Automation Magazine*, vol. 14, pp. 20-29

Martens, C. (2003). Teilautonome Aufgabenbearbeitung bei Rehabilitations-robotern mit Manipulator - Konzeption und Realisierung eines software-technischen und algorithmischen Rahmenwerks, PhD dissertation, University of Bremen, Faculty of Physics / Electrical Engineering, (in German)

Martens, C., Prenzel, O. & Gräser, A. (2007). The rehabilitation robots FRIEND-I & II: Daily life independency through semi-autonomous task-execution, In: *Rehabilitation Robotics* (Sashi S Kommu, Ed.), pp. 137-162., I-Tech Education and Publishing, Vienna, Austria, Available from http://www.intechopen.com/books/show/title/rehabilitation_robotics

Microsoft. (2011). Microsoft Robotic Studio, Available from http://www.microsoft.com/robotics

Natarajan, S.K., Ristic-Durrant, D., Leu, A., Gräser, A. (2011). Robust stereo-vision based 3D-modeling of real-world objects for assistive robotic applications, in *Proc. of IEEE/RSJ International Conference on Robots and Systems (IROS), San Francisco, USA*

Ojdanic, D. (2009). Using cartesian space for manipulator motion planning - application in service robotics, PhD dissertation, University of Bremen, Faculty of Physics and Electrical Engineering

Prenzel, O. (2005). Semi-autonomous object anchoring for service-robots, *in B. Lohmann (Ed.), A. Gräser, Methods and Applications in Automation,* pp. 57 - 68, Shaker-Verlag, Aachen, 2005, ISBN 3-8322-4502-2

Prenzel, O., Boit, A. and Kampe H. (2008) Ergonomic programming of service robot behavior with function block networks, in *Methods and Applications in Automation*, Shaker-Verlag, pp. 31-42

Prenzel, O. (2009). *Process model for the development of semi-autonomous service robots*, PhD dissertation, University of Bremen, Faculty of Physics and Electrical Engineering

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y. (2009) ROS: an open-source Robot Operating System, In: *Proc. Of ICRA Workshop on Open Source Software*

Russel, S., and Norvig, P. (2003). *Articial Intelligence - A Modern Approach*, Prentice Hall, Upper Saddle River, New Jersey, 2nd ed.

Schlegel, C., and Woerz, R. (1999) The software framework SmartSoft for implementing sensorimotor systems, In: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1610-1616

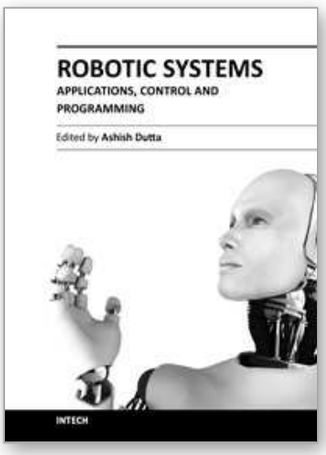
Schmidt, D. C. (2006). Model-driven-engineering, In: *guest editor's introduction*, pp. 25-31, IEEE Computer

Schreckenghost, D., Bonasso, R., Kortenkamp, D., Ryan D. (1998) Three tier architecture for controlling space life support systems, In: *Proc. of IEEE SIS'98*, Washington DC, USA

Simmons, R., Apfelbaum, D. (1998) A task description language for robot control, In: *Proc. of Conference on Intelligent Robotics and Systems*

Weld, D. S. (1999). Recent advances in AI planning, in *AI Magazine*, vol 20, pp. 93-123

Wright, R. S., Lipchak, B., Haemel, N. & Sellers, G. (2010). *OpenGL SuperBible: Comprehensive Tutorial and Reference* (5th Edition), Addison-Wesley, ISBN 978-0321712615



Robotic Systems - Applications, Control and Programming
Edited by Dr. Ashish Dutta

ISBN 978-953-307-941-7
Hard cover, 628 pages
Publisher InTech
Published online 03, February, 2012
Published in print edition February, 2012

This book brings together some of the latest research in robot applications, control, modeling, sensors and algorithms. Consisting of three main sections, the first section of the book has a focus on robotic surgery, rehabilitation, self-assembly, while the second section offers an insight into the area of control with discussions on exoskeleton control and robot learning among others. The third section is on vision and ultrasonic sensors which is followed by a series of chapters which include a focus on the programming of intelligent service robots and systems adaptations.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen