A Study on the Application of the Software Framework MASSiVE in KAIST's Intelligent Sweet Home System

Oliver Prenzel*, Sang Wan Lee, Zeungnam Bien**, and Axel Graeser***

*Institute of Automation, Department of Electrical Engineering and Information Technology, University of Bremen, 28359 Bremen, Germany
(Tel: +49-421-218-3594, Fax: +49-421-218-4596 ; E-mail: {ag, prenzel}@iat.uni-bremen.de)

** Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea
(Tel: +82-42-350-5419, Fax: +82-42-350-8750 ; E-mail: bigbean@ctrsys.kaist.ac.kr, zbien@ee.kaist.ac.kr)

***Abstract* — MASSiVE (Multi - Layer Architecture for Semi-Autonomous Service Robots with Verified Task Execution) is a software framework that provides an infrastructure concept for distributed sensor and actuator systems such as service robots, operating in environments that are equipped with smart components. Besides this modular and extensible architecture, a principle of task knowledge specification and verification with process-structures is included in MASSiVE that is able to guarantee task planning in real time along with verified and thus robust system runtime behavior. In this paper the framework is presented briefly. The focus of this contribution is to study how to apply the MASSiVE principles for a typical sample application scenario in KAIST's Intelligent Sweet Home System. First, the required software components have to be derived, and second the exemplary application of MASSiVE's specification and verification methods and tools is demonstrated. The flexibility and the structuring capabilities of the MASSiVE framework as well as the impact of fast system deployment will be worked out.**

***Index Terms* — Software architecture, task knowledge specification and verification, semi-autonomy, service robots in intelligent environment.**

1. INTRODUCTION

A service robot is, unlike its industrial counterpart, intended to act in mostly unstructured and dynamic environments, like domestic ones or at work places. It has to be able to pursue a certain mission goal as commanded from its user on the one hand but also needs to react flexibly to dynamic changes within the workspace, like caused by variations in the lighting conditions or moving obstacles. To meet these requirements, hybrid multi-layer control architectures have been applied here [1], [2], [3]. These architectures usually consist of three layers:

- A *deliberative layer*, which contains a task planner to generate a sequence of operations to reach a certain goal with respect to the user's input command.
- A *reactive layer*, which has access to the system's sensors and actuators and provides reactive behavior which is robust even under environmental disturbances, e. g. with the help of closed-loop control.
- A *sequencer* that mediates between deliberator and reactive layer, i. e. activates or deactivates reactive operations according to the deliberator's specification.

The software framework MASSiVE [4], developed at the Institute of Automation (IAT, Germany) is a special kind of hybrid multi-layer control architecture which is tailored to the requirements of semi-autonomous and distributed systems, like the rehabilitation robots FRIEND [4] or KARES [5], acting in environments with distributed smart components. These intelligent wheelchair mounted manipulator systems allow to benefit from the inclusion of the user's cognitive capabilities into task execution and consequently lower the system complexity compared to fully autonomous system as conventionally intended to be developed. The semi-autonomous control requires a sophisticated integration of a human-machine-interface (HMI) which is able to couple input devices according to the user's impairment [6], [7], as e. g. a haptic suit, eye-mouse, speech-recognition, chin joystick or a brain-computer interface (BCI). The resulting MASSiVE control architecture with special emphasis on the HMI component is depicted in Fig. 1. Here, the deliberator has been moved to the sequencer component, and the HMI has direct access to control the actuators in the reactive layer during user interactions.

Besides the focus on semi-autonomous system control, the MASSiVE framework includes a second main paradigm, namely the pre-structuring of task knowledge. This task planner input is specified offline in a scenario-driven approach with the help of so-called process-structures on two levels of abstraction, the abstract level and the elementary level. After specification and before being used for task execution, the task knowledge is verified offline, to guarantee a robust runtime behavior. As discussed in larger detail in [7], the offline verified task knowledge drives all user interaction processes during task execution. It specifies the resources to be used (e. g. a camera), limits the user's interaction freedom based on a user interaction context, which is derived from the given task knowledge, and finally assures the availability as well as conformity of requested information after successful completion of a user interaction. Possible examples of user interactions are the camera-based identification of objects as described in [7] or the fine adjustment of the gripper to an object to be manipulated.

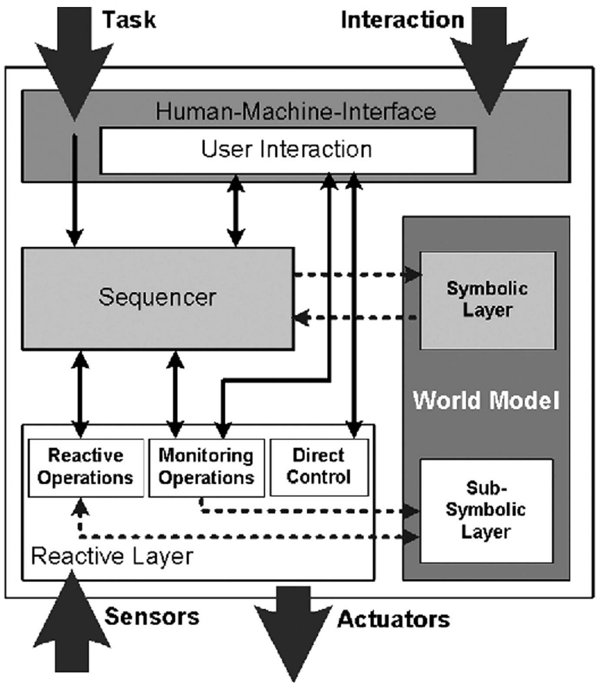

Fig. 1. Hybrid multi-layer control architecture MASSiVE for semi-autonomous service-robots with verified task execution.

Within this contribution the applicability of the MASSiVE framework for the realization of a typical support task from KAIST's Intelligent Sweet Home environment is discussed. The necessary architectural elements within the reactive layer of the control architecture are worked out and the steps of scenario-driven programming are illustrated. Especially, the recently developed approach of elementary process-structure specification with the help of function block networks is presented. This method makes the programming task more ergonomic and meanwhile maintains the verification capabilities with the help of automatic conversion of function block networks into verifiable Petri-Nets.

2. Sample Task in KAIST's Intelligent Sweet Home

2.1. User Transfer Scenario

The Intelligent Sweet Home system at KAIST has been built up to provide a realistic test environment of various solutions and concepts of rehabilitation robotics [8]. One of scenarios that have been developed is the transfer of the disabled user via robotic hoist from bed to wheelchair, or vice versa. This scenario is depicted in Fig. 2.

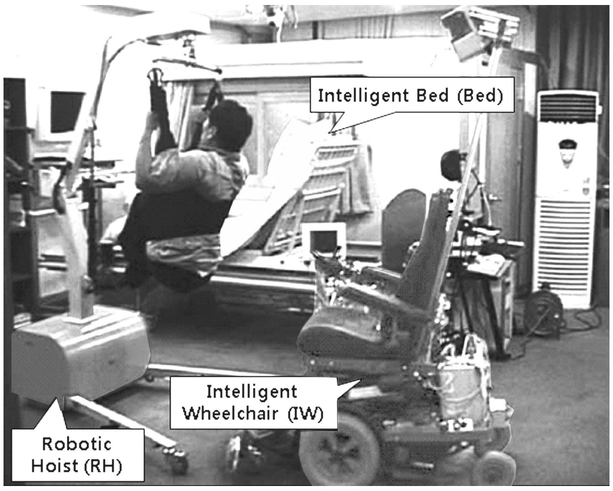

Fig. 2. User transfer scenario as typical support scenario in KAIST's Intelligent Sweet Home (ISH) System (Figure is taken from:[8]).

Three intelligent robotic agents are cooperating in this scenario:

- An *intelligent bed*, equipped with pressure sensor system to determine the user's posture, as well as bar-type robotic arm for actively supporting the user when intending to change posture and position,
- a *robotic hoist* for lifting and lowering the user, able to navigate in the room with a ceiling-mounted artificial star navigation system and
- an *intelligent wheelchair* with robotic arm and a tilted scanning mechanism of laser range finder, to navigate autonomously and to be able to automatically dock at the robotic hoist.

A detailed discussion of the user transfer task is described in [8]; and it is summarized in the following:

As Fig. 3 illustrates, the robotic hoist moves from the recharge station to the intelligent bed, which aids the user to sit up with help of the movable bar and automatic adjustment of the upper bed part. After the robotic hoist lifts the user, the intelligent wheelchair automatically docks to the hoist system, so that the user can be lowered into the wheelchair. Finally, the wheelchair navigates to the target location as commanded by the user (e. g. to the room door), whereas the robotic hoist moves back to the recharge station.

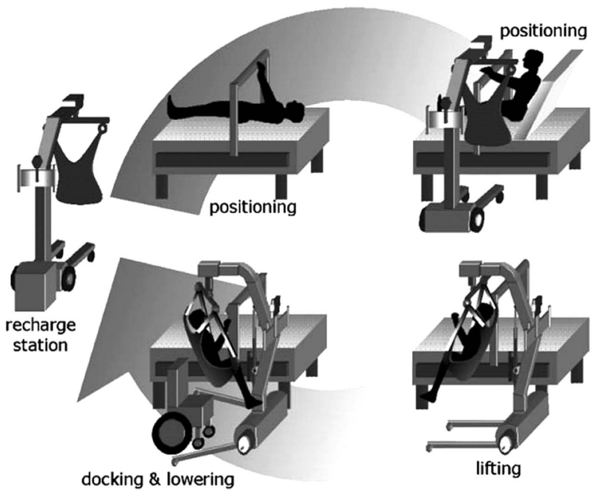

Fig. 3. Task steps of user transfer with robotic hoist, intelligent bed and intelligent wheelchair robots. (Figure is taken from [8].)

2.2. Problem

The execution of user transfer scenario is well-demonstrated by relevant description of the robots' behavior, i.e., sequence of the robots' actions. However, this is case-dependent; the sequence is required to be re-organized by developers when an addition of a new scenario is required.

Besides, ISH system lacks an automatic architecture which effectively manages task knowledge specification from abstract level to elementary level; relevant tasks are organized in abstract level, while related functions are executed in elementary level.

Another problem is that, since the resulting process map in the elementary level may require many actions for complex tasks, their relations inevitably become complicated; it is thus difficult to modify the relevant tasks in elementary level. Therefore, a flexible design concept [8] - organization that considers nature of the disabilities of the user and customization that makes the design much faster - is limited to simple scenarios.

Therefore, the problem to be tackled is an automatic realization of the given task from abstract level to elementary level under the MASSiVE framework, which is able to effectively structure task knowledge with high usability and flexibility.

In the following chapters, an application to MASSiVE framework will be demonstrated to improve the task management skill of ISH by focusing on a user transfer task, which is a typical scenario of ISH.

3. Behavioral Specification for User Transfer Task in MASSiVE

In the following section it is analyzed how to realize the user transfer task based upon the components provided by MASSiVE. This includes the specification of the architectural elements first and afterwards the programming and verification of required pre-structured task knowledge.

3.1 Architectural Components

Fig. 4 depicts the MASSiVE reactive layer with components as used in the ISH. To be able to re-use software components and to exchange hardware components without re-programming effort throughout the whole system, MASSiVE distinguishes between skill layer and hardware layer. The skill layer provides the basic system operations that are called by the task planner within the sequencer. They encapsulate algorithmic implementations based on the pure hardware functionality as provided by the hardware layer. As Fig. 1 points out, the reactive layer contains reactive operations which access sensors as well as actuators, direct control operations, with access only to the actuators, and finally monitoring operations based on sensorial input. The skills in the skill layer are provided by skill servers and the hardware functionality is encapsulated in hardware servers within the hardware layer. One skill server is responsible for the management of a certain set of hardware. For the sake of re-configurability, scalability of computer power and to be able to build up distributed systems, all components in the reactive layer are able to communicate via the middleware CORBA [9]. However, also direct communication within the same software process is possible, if required, e. g. within control loops that rely on a fast throughput of sensorial data. The basic system design layout as well as the communication infrastructure is software-technically provided by common base classes for skill servers as well as for hardware servers.

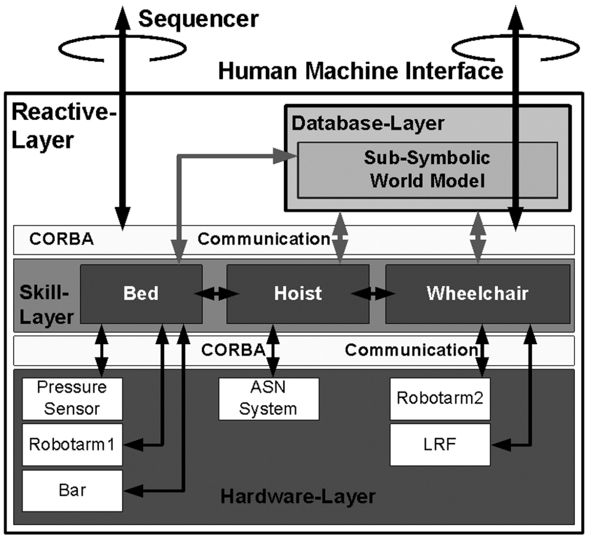

Fig. 4. Reactive layer components for ISH.

Besides containing the skill and hardware layer, the MASSiVE reactive layer has CORBA communication access to the sequencer, the human machine interface and to the sub-symbolic world model component. The latter one serves as data buffer for all non-symbolic (e. g. geometrical) information about the environment and the system that is exchanged by the skills operating in the skill layer. (The symbolic world model layer is accessed by the symbolic planner in the sequencer, as depicted in Fig. 1 and further explained below.)

According to Fig. 4 the skill servers *Bed, Hoist* and *Wheelchair* are required for the user transfer task in ISH. The intelligent bed skill server manages the hardware servers *PressureSensor*, *Robotarm1* and *Bar*. The hoist skill server is connected to the *ASN system* (artificial star navigation system) and the intelligent wheelchair skill server maintains the hardware servers *Robotarm2* and *LRF* (laser range finder).

In the subsequent section the required task knowledge for the user transfer will be analyzed. This includes the exemplary derivation of necessary skill server interfaces.

3.2 Task Knowledge Specification on the Abstract Level

Every task in MASSiVE that can be commanded by the user on high abstraction level, like "Pour in a drink" or "Transfer me to the room door", is modeled with the help of an abstract process-structure (PS_A). The system programmer has to decide whether to split up one task into several sub-tasks, to produce highly re-usable and less complex task specifications. A PS_A models a task on the abstract, i. e. symbolic level. First, the task participating objects, TPO, have to be determined. TPOs are combined to object constellations (OCs) which describe the different physical contact states of the objects throughout the task execution process. MASSiVE's PS_As are basically AND/OR nets, being used in assembly task planning, but also having been proposed as compact task description for service robotics [10]. They have been enhanced with first order predicate logic facts [11] to be embeddable in the MASSiVE control architecture [12]. Due to the PS_As origin in assembly planning the contained OCs are connected via assembly or disassembly operations (AOP, DOP) or internal state transitions (IST).

The PS_A for the user transfer task is defined in Fig. 5. Task participating objects are the user (User), the intelligent bed (Bed), the robotic hoist (RH) and the intelligent wheelchair (IW). The given PS_A contains 12 object constellations which are connected with each other by 8 pairs of composed operators (COPs). The COP pairs are specified in Table 1 according to the encircled numbers in Fig. 5. They are named composed operators since they constitute a high abstraction operation specification and will be decomposed into elementary process-structures as described further on. Besides the task participating objects the COPs require symbolically represented sub-symbolic information [12], which is denoted as information descriptors like InFrontOfUser, UserLoc[1], DockingLoc and RoomDoor in this sample task. To specify an abstract process-structure in a logically correct form, the task participating objects are formally introduced in MASSiVE's task knowledge database and the programming of the PS_A is done with a specially developed graphical programming interface [13].

[1] Loc = Abbreviation for location.

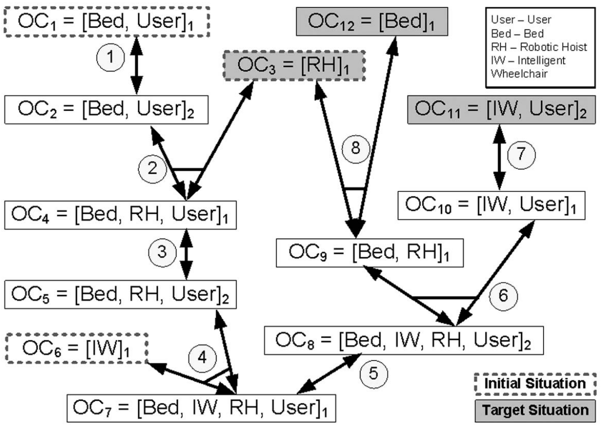

Fig. 5. Abstract process-structure (PS_A) for user transfer task. The encircled numbers denote pairs of applicable composed operators (COPs). They are given in Table 1.

Table 1. Composed operator pairs (COPs) for PS_A from Fig. 5. The COP numbers refer to the encircled numbers in the PS_A. The marked AOP 4 is used for further more detailed illustrations.

COPNo.	COPType	COP Name and Parameter
1	AOP	MoveBar(Bed, InFrontOfUser)
	DOP	RemoveBar(Bed, InFrontOfUser)
2	AOP	MoveToRelLoc(RH, Bed, UserLoc)
	DOP	MoveFromRelLoc(RH, Bed, UserLoc)
3	IST	Lift(RH, User)
	IST	Lower(RH, User)
4	**AOP**	**MoveToRelLoc(IW, RH, DockingLoc)**
	DOP	MoveFromRelLoc(IW, RH, DockingLoc)
5	IST	Lift(RH, User)
	IST	Lower(RH, User)
6	AOP	MoveToRelLoc(IW, RH, DockingLoc)
	DOP	MoveFromRelLoc(IW, RH, DockingLoc)
7	AOP	MoveToLocation(IW, RoomDoor)
	DOP	MoveFromLocation(IW, RoomDoor)
8	AOP	MoveToRelLoc(RH, Bed, UserLoc)
	DOP	MoveFromRelLoc(RH, Bed, UserLoc)

Certain sets of OCs in Fig. 5 are marked as *Initial Situation* respectively *Target Situation*. A situation in a PS_A contains OCs that uniquely include all task participating objects TPO and are part of the *Situation Graph* as defined in [14]. I. e. they have to define a valid intermediate state that is transformable via the COPs as specified for the PS_A. Initial and target situation are determined online according to the current execution context. The initial situation is the result of the initial monitoring process [14] and the target situation is set according to the user input command respectively according to a valid initial situation in a subsequently executed PS_A. In the given example the initial situation is set to the state where the user is lying in the bed, the robotic hoist is in its recharge location and the intelligent wheelchair is also in some default or arbitrary but free location. In the desired target situation the user has been transferred to the wheelchair and has been brought to the room door, while the bed is left empty and the robotic hoist left the location beside the bed. In this setup, the task planner will generate the following action sequence:

1) MoveBar(Bed, InFrontOfUser)
2) MoveToRelLoc(RH, Bed, UserLoc)
3) Lift(RH, User)
4) MoveToRelLoc(IW, RH, DockingLoc)
5) Lower(RH, User)
6) MoveFromRelLoc(IW, RH, DockingLoc)
7) MoveToLocation(IW, RoomDoor)
8) MoveFromRelLoc(RH, Bed, UserLoc)

3.3 Task Knowledge Specification on the Elementary Level

To demonstrate the task knowledge specification process on the elementary level, the assembly operation number 4 as marked in Table 1 is decomposed into an elementary process-structure (PS_E). A PS_E gets the input parameters as specified in the COP's parameter list (i. e. IW, RH and DockingLoc in this case). Furthermore, pre- and post facts are associated with each COP as mentioned in Section 3.2 or discussed in more detail in [4].

The task of the system programmer who is responsible for task knowledge input on the elementary level is to specify how to execute a certain COP on the level of basic skills of a robotic system, i. e. as seen from the basic system perspective. He has to define

- The resources of the system to be used,
- the elementary operations (skills) to be executed,
- the necessary control flow and flow of sub-symbolic data and
- decides how to establish the connection to the abstract task knowledge layer via setting the COP's post-facts.

First, PS_Es are specified, and secondly they are transformed into Petri-Nets for a formal verification on a well-defined mathematical basis. The subject of verification is the correctness of the just listed items as well as to detect possible resource conflicts, deadlock situations, unreachable targets or further modeling errors.

For the sake of user-friendly programming on this level, an approach with function blocks has been developed [15]. In comparison to the previous approach of manual direct specification of a PS_E on Petri-Net level, it offers the following advantages and therefore leads to a more ergonomic programming procedure:

- Hide the PS_E's complexity,
- simplify the programming process,
- avoid semantic errors and
- intelligently and automatically support the programming procedure.

The developed function block based programming interface offers the programming elements as specified in Fig. 6. The EEOPBlock[2] encapsulates all required information that is necessary for skill execution (skill server, skill name, parameters, resources, possible return values) and FactBlocks set the predicate logic

[2] EEOP = elementary executable operation, since from the viewpoint of the symbolic task planner these operations are elementary.

facts as post-condition of the COP on the abstract level. (The facts are stored in the symbolic layer of the world model, see Fig. 1.) Furthermore, two types of logical blocks are used to define AND respectively OR connections within the control flow between the function blocks and three kinds of ControlBlocks mark the start, target or abort condition of execution.

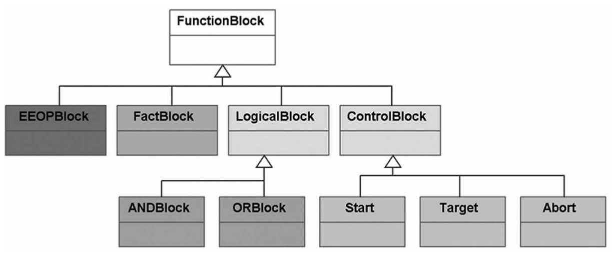

Fig. 6. Elements of function-block-based programming approach.

The function block network for the definition of the COP *MoveToRelLoc(IW, RH, DockingLoc)* is depicted in Fig. 7, also showing the programming interface. This rather simple case of function block network sequentially executes the following skills:

1) A self-localization of the robotic hoist is done with the help of the hoist's camera and the artificial star navigation system. The result is stored in the sub-symbolic world model.

2) The intelligent wheelchair navigates in front of the hoist. The target location of this operation is generated based upon the previously determined hoist location information.

3) The wheelchair docks to the hoist by using its laser range finder.

Finally, the COP's post-facts *IsInRelLoc()* and *IsInFreePos()* are set to TRUE respectively FALSE.

3.4 Function Block Transformation into Petri-Nets

To understand the algorithm that converts a function block network into a Petri-Net the counterpart elements in the two different network types are summarized in Table 2.

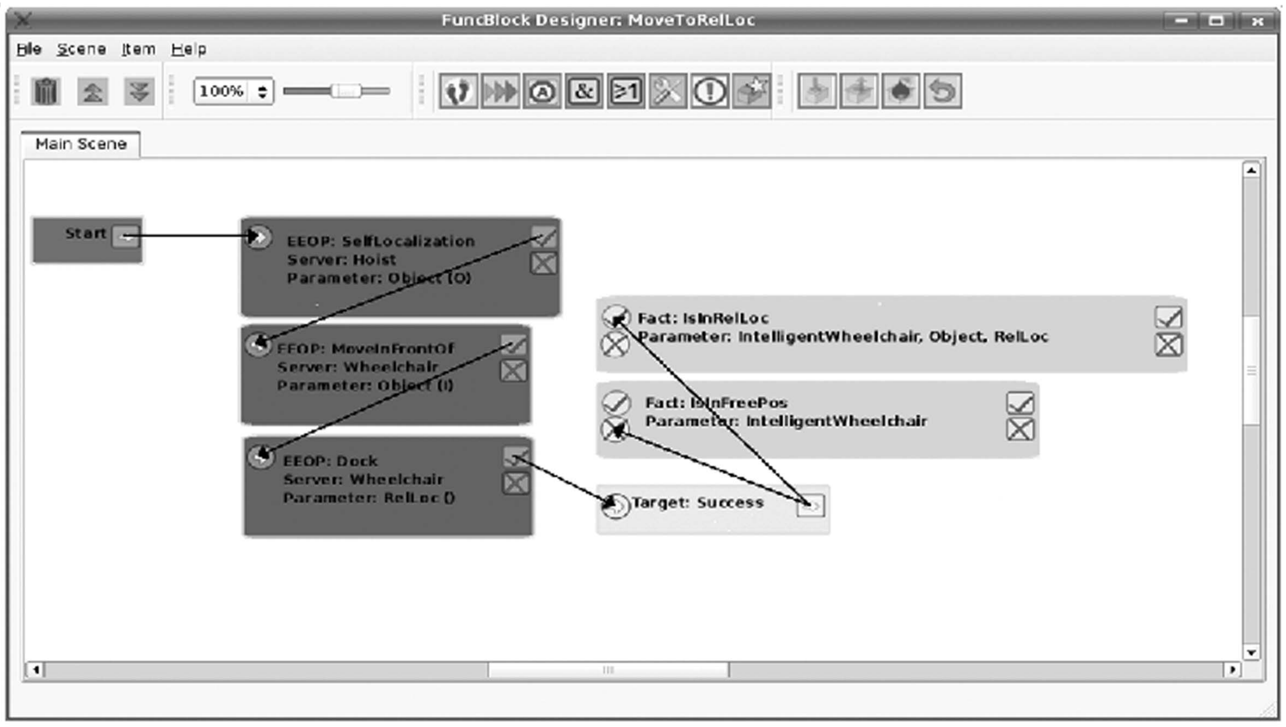

Fig. 7. Function-block-based programming interface with network specifying the COP *MoveToRelLoc(IW, RH, DockingLoc).*

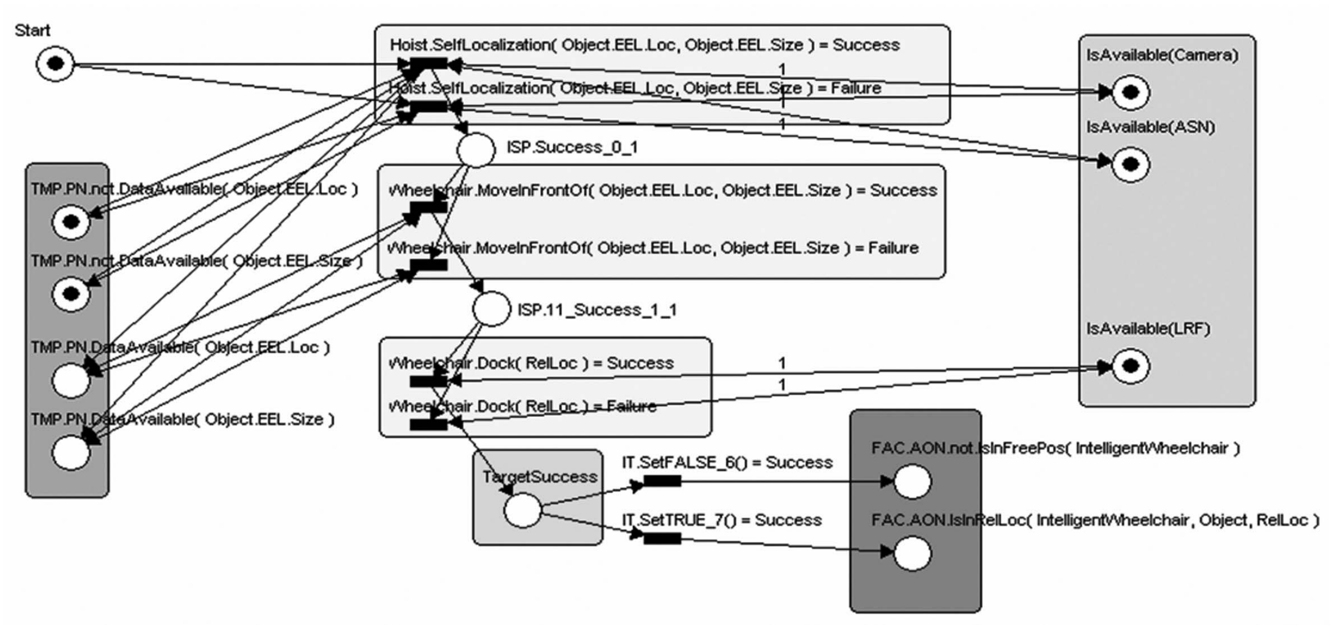
Start
Hoist.SelfLocalization(Object.EEL.Loc, Object.EEL.Size) = Success
Hoist.SelfLocalization(Object.EEL.Loc, Object.EEL.Size) = Failure
ISP.Success_0_1
Wheelchair.MoveInFrontOf(Object.EEL.Loc, Object.EEL.Size) = Success
Wheelchair.MoveInFrontOf(Object.EEL.Loc, Object.EEL.Size) = Failure
ISP.11_Success_1_1
Wheelchair.Dock(RelLoc) = Success
Wheelchair.Dock(RelLoc) = Failure
TargetSuccess
IT.SetFALSE_6() = Success
IT.SetTRUE_7() = Success
IsAvailable(Camera)
IsAvailable(ASN)
IsAvailable(LRF)
TMP.PN.not.DataAvailable(Object.EEL.Loc)
TMP.PN.not.DataAvailable(Object.EEL.Size)
TMP.PN.DataAvailable(Object.EEL.Loc)
TMP.PN.DataAvailable(Object.EEL.Size)
FAC.AON.not.IsInFreePos(IntelligentWheelchair)
FAC.AON.IsInRelLoc(IntelligentWheelchair, Object, RelLoc)
1
1
1
1
1

Fig. 8. Elementary process-structure of the COP *MoveToRelLoc(IW, RH, DockingLoc)* in Petri-Net format.

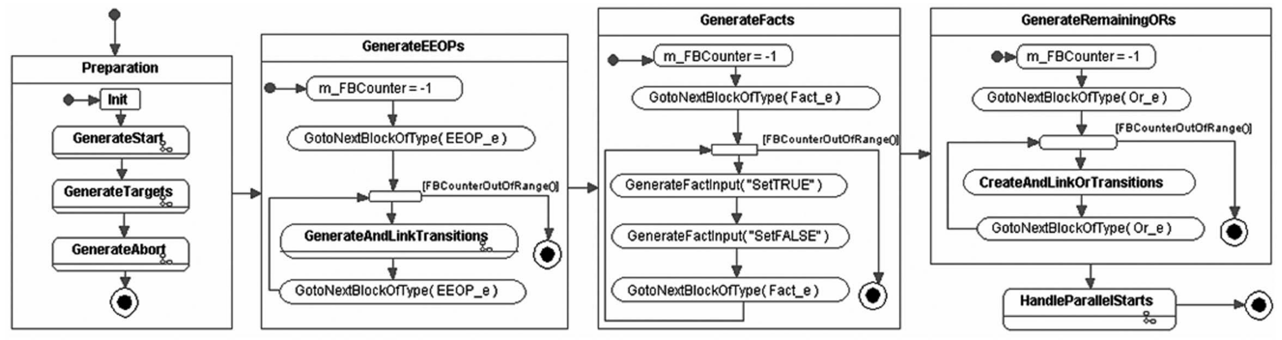
Preparation
Init
GenerateStart
GenerateTargets
GenerateAbort
GenerateEEOPs
m_FBCounter = -1
GotoNextBlockOfType(EEOP_e)
[FBCounterOutOfRange()]
GenerateAndLinkTransitions
GotoNextBlockOfType(EEOP_e)
GenerateFacts
m_FBCounter = -1
GotoNextBlockOfType(Fact_e)
[FBCounterOutOfRange()]
GenerateFactInput("SetTRUE")
GenerateFactInput("SetFALSE")
GotoNextBlockOfType(Fact_e)
GenerateRemainingORs
m_FBCounter = -1
GotoNextBlockOfType(Or_e)
[FBCounterOutOfRange()]
CreateAndLinkOrTransitions
GotoNextBlockOfType(Or_e)
HandleParallelStarts

Fig. 9. Hierarchical flowchart of algorithm for converting function block networks into Petri-Nets.

Table 2. Counterpart elements of function block networks and Petri-Nets.

Function Block Network	**Petri-Net**
Start, Target, Abort	Place(s)
EEOP	Transition(s)
Fact	Place(s)
OR	Place
AND	Link to Transition

Also, the Petri-Net resulting from the conversion of the sample COP *MoveToRelLoc* is given beforehand in Fig. 8. The conversion algorithm has been implemented – like the complete MASSiVE architecture – in the model-driven development (MDD) environment Telelogic Rhapsody [16]. The hierarchical flowchart model as depicted in Fig. 9 can be automatically translated into C++ code that builds the executable conversion module. This hierarchical algorithm specification illustrates how the complexity of an algorithm can be handled with the help of an easy to understand flowchart model whose details can be accessed on demand by browsing through the software model in the CASE tool. Furthermore, only flowcharts that adhere to the structuring rules as introduced by Nassi/Shneidermann [17] produce structured code in the target programming language. Thus, a strict modularization of the algorithm is enforced.

As the conversion algorithm is discussed in more detail in [15], it is summarized here briefly. Some notations have to be mentioned, beforehand: Bold actions (e. g. the *Init* action) summarize C++ code specifications, bold actions with sub-flowchart symbol at the right corner of the action box (e. g. *GenerateStart*) subsume sub-flowcharts, whereas all other actions with normal font weight directly display C++ code.

In the algorithm's first stage, the places for the start, target and abort blocks are introduced. As specified in Table 2 the start block may be represented not only with one place, but in case of parallel control flow links leaving the start block (parallel executable operations), the start place splits into helper transition plus place, according to the number of leaving branches. (See [15] for such an example.)

The handling of EEOPs, Facts and OR blocks is processed in a very similar manner. The algorithm iterates over all function blocks in the network and checks for the specific type (EEOP, Fact or Or) and generates the respective elements according to Table 2. This handling includes, if necessary, the introduction of helper elements, as a place in a Petri-Net always has to be connected to a transition and vice versa. Special consideration is required for EEOP blocks, as they need places to mark data availability (left side in Fig. 8) as well as resource availability (right side in Fig. 8). AND blocks are indirectly modeled by direct linking of the preceding place to the blocks input transition.

Finally, a special handling for parallel branches leaving the start block takes place. Since it is possible that one of these starting branches has not been executed when the PS_E's target is reached (e. g. due to a fact state that never became true during the execution), this branch has to be deactivated. This is done by connecting the target place to the start branch activation place with a helper transition and thus to force the consumption of tokens from potentially marked places by the target place.

After conversion of a function block network into an equivalent Petri-Net, the verification algorithms [18] are applied to conduct the remaining checking procedures as listed in Section 3.3.

If the specification of elementary process-structures is finished, the interfaces for all skill servers are known and the algorithmic skill implementation, including e. g. the design of control loops, takes place. The skill server interfaces for the hoist skill server and the wheelchair skill server are exemplarily depicted in Fig. 10, but only contain the skills as defined in the discussed sample PS_E *MoveToRelLoc*. Fig. 10 also shows how the derived skill server classes inherit basic skill server functionality from the common base class, which are e. g. methods to indicate the connection state within the reactive layer network or to reset the skill server. The *CallBack* argument, which is the last argument in each asynchronous skill method, serves as communication mechanism to the sequencer as introduced in [4].

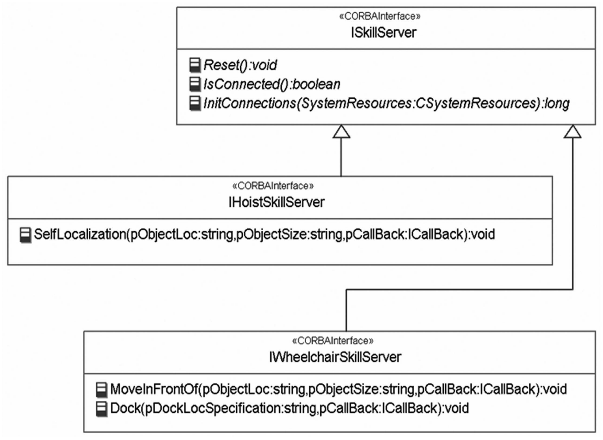
«CORBAInterface»
ISkillServer
Reset():void
IsConnected():boolean
InitConnections(SystemResources:CSystemResources):long
«CORBAInterface»
IHoistSkillServer
SelfLocalization(pObjectLoc:string,pObjectSize:string,pCallBack:ICallBack):void
«CORBAInterface»
IWheelchairSkillServer
MoveInFrontOf(pObjectLoc:string,pObjectSize:string,pCallBack:ICallBack):void
Dock(pDockLocSpecification:string,pCallBack:ICallBack):void

Fig. 10. Resulting skill server interfaces as derived from the sample PS_E *MoveToRelLoc* depicted in Fig. 7 and Fig. 8.

4. Conclusion and Outlook

In order to improve a task management skill of KAIST's Intelligent Sweet Home system, the MASSiVE framework has been applied. The discussion included the specification of required infrastructural components as well as the necessary steps of specification of verifiable task knowledge on the behavioral programming level. This is, besides the application of MASSiVE in various support scenarios at the IAT, the first concrete example of concept transfer. However, with respect to the special capability of system-controlled integration of user-interactions into task execution, this approach is in general suitable for any human-centered service robotic system or for systems that have access to a human operator.

Based on re-usage of existing software modules, the new components can be deployed quickly. The scenario driven and tool-supported programming approach guides through the task knowledge definition process and finally leads to the specification of the skill server interfaces, which encapsulate the algorithmic implementations. Especially on the level of elementary system level programming the function-block-based approach offers an ergonomic to use method which inherently avoids errors and enables verification of task knowledge via automatic conversion of function block networks into Petri-Nets. Due to stringent application of modularity within both levels of task knowledge specification, the scalability of the method can be guaranteed. One process-structure is always a limited set of task knowledge that guarantees verifiability.

Further steps to be conducted are ongoing improvements and enhancements of MASSiVE's programming tools, e. g. to achieve a more tight integration of the different tools for task knowledge specification on the different abstraction levels. Also, as a consequence of this first study of applicability, the final algorithmic implementations of skill and hardware servers and eventually the test on the real systems in the ISH have to be conducted.

Acknowledgement

Part of this work is supported by the BMBF (Bundesministerium fur Bildung und Forschung, German Ministry of Education and Research) under the grant "AMaRob" within the "Leitinnovation Service-Robotik". This work was partially supported by the SRC/ERC program of MOST/KOSEF (grant #R11-1999-008).

References

[1] R. Simmons, "Architecture, the backbone of robotic systems," Proceedings of the 2000 IEEE International Conference on Robotics and Automation, (San Francisco, CA), Apr. 2000.

[2] R. Bonasso, D. Kortenkamp, D. Schreckenghost, and D. Ryan, "Three tier architecture for controlling space life support systems," Proceedings of IEEE SIS'98, (Washington DC, USA), 21 -23 May 1998.

[3] C. Schlegel and R. Woerz, "The software framework SmartSoft for implementing sensorimotor systems," Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, vol. 3, pp. 1610-1616, Oct 1999.

[4] C. Martens, O. Prenzel, and A. Graeser: "The Rehabilitation Robots FRIEND-I & II: Daily Life Independency through Semi-Autonomous Task-Execution," in *Rehabilitation Robotics*, I-Tech Education Publishing, Vienna, Austria, ISBN 978-3-902613-01-1, 2007.

[5] Z. Z. Bien, W.-K. Song, D.-S. Kwon, M.-J. Chung, H.-S. Park, D.-J. Kim, J.-H. Kim, and K. Lee, "A Wheelchair Robot System and its Various Interface Methods for the Disabled Persons," Proceedings of the 1st Workshop on Technical Challenge for Dependable Robots in Human Environments, Seoul, Korea, May 21-22, 2001.

[6] C. Martens, D.-J. Kim, J.-S. Han, A. Graeser, and Z. Z. Bien, "Concept for a modified hybrid multi-layer control-architecture for rehabilitation robots," in Proceedings of the 3rd International Workshop on Human-friendly Robotic Systems, pp. 49- 54 Daejeon, Korea , January 21-22, 2002.

[7] O. Prenzel, C. Martens, M. Cyriacks, C. Wang, and A. Graeser, "System-controlled user interaction within the service robotic control architecture MASSiVE," Robotica, vol. 25, no. 2, pp. 237-244, 2007.

[8] K.-H.Park, Z. Z. Bien, J.-J. Lee, B. K. Kim, J.-T. Lim, J.-O. Kim, H. Lee, D. H. Stefanov, D.-J. Kim, J.-W. Jung, J.-H. Do, K.-H. Seo, C. H. Kim, W.-G. Song, and W.-J. Lee, "Robotic smart house to assist people with movement disabilities," Autonomous Robots, vol. 22, no. 2, Springer

Netherlands, ISSN 0929-5593, February, 2007.

[9] S. Vinoski, and M. Henning, *Advanced* CORBA® *Programming with C++*, Addison Wesley Professional, 1999.

[10] T. Cao and A. C. Sanderson, "AND/OR Net Representation for Robotic Task Sequence Planning," IEEE Transactions on Systems, Man, and Cybernetics - part C: Applications and Reviews, vol. 28, May 1998.

[11] S. Russel and P. Norvig, *Artificial Intelligence - A Modern Approach*, Upper Saddle River, New Jersey: Prentice Hall, second ed., 2003.

[12] C. Martens, "Teilautonome Aufgabenbearbeitung bei Rehabilitationsrobotern mit Manipulator – Konzeption und Realisierung eines softwaretechnischen und algorithmischen Rahmenwerks," PhD dissertation, University of Bremen, Faculty I Physics / Electrical Engineering, Nov. 2003. (In German).

[13] C. Martens, "Task oriented programming of service-robots on the basis of process-structures," in: *Methods and Applications in Automation*, B. Lohmann, A. Graeser, (Ed.), pp. 45-56, Shaker Verlag, ISBN 3-8322-4502-2, Aachen, 2005.

[14] O. Prenzel, "Semi-Autonomous Object Anchoring for Service-Robots," in *Methods and Applications in Automation*, B. Lohmann, A. Graeser, (Ed.), pp. 57-68, Shaker Verlag, ISBN 3-8322-4502-2, Aachen, 2005.

[15] O. Prenzel, A. Boit, and H. Kampe: "Ergonomic Programming of Service Robot Behavior with Function Block Networks," in: *Methods and Applications in Automation*, Shaker-Verlag, 2008.

[16] Telelogic, "Telelogic web-page," http://www.telelogic.com, 2008.

[17] I. Nassi, and B. Shneiderman, "Flowchart Techniques For Structured Programming," in ACM SIGPLAN (Special Interest Group on Programming Languages) Notices, vol. 8, August 1973.

[18] H. Kampe, "Verification of Petri-Net Based Process-Structures for the Programming of Service-Robots," tech. rep., University of Bremen, Institute of Automation (IAT), 2007.