Towards a Component-based System Architecture for Autonomous Mobile Robots[*]

Stefan A. Blum
Institute for Real-Time Computer Systems
Technische Universität München
Munich, Germany
Stefan.Blum@rcs.ei.tum.de

ABSTRACT
This paper presents the component-based system architecture OSCAR (Operating System for the Control of Autonomous Robots), a general purpose control system for the exploration of indoor environments with an autonomous mobile robot. The requirements and basic concepts of the proposed architecture are discussed and compared with other existing systems. OSCAR relies on efficient communication structures employing the CORBA middle-ware standard. Basic insights about the applied component concept are given. A behavior-based coordination mechanism is adopted for sequential control. In an example scenario, the mobile robot MARVIN is employed to explore a doorway using geometric object descriptions for retrieving topological and symbolic information of the environment.

KEY WORDS
Intelligent system architecture, behavior-based robotics, mobile robot, component-ware

1. Introduction

Information processing in autonomous mobile robots comprehends perception of the environment with partial consideration of a-priori knowledge to generate reasonable actions within a given task. It is easy to argue that information processing is hardly done in a single monolithic block. Sensor data may be obtained from several sensor devices in parallel and may be processed in several ways (e.g. feature extraction from camera images). A mobile robot may have several computers on-board that permit distribution of processes. Furthermore, modularity supports the development of algorithms in developer teams and facilitates testing. It is needless to say that these criterions leads to software architectures. Arkin [1] further specifies timelineness in development (e.g. provided using specialized tools), niche targetability, robustness regarding exceptions of components, run-time flexibility and reconfigurability as well as performance effectiveness.

Software Engineering aspects, particularly the encapsulation and reusability aspect lead to component-based systems [2]. This paper describes the system architecture OSCAR [3, 4] that is developed for the autonomous mobile robot MARVIN at the Institute for Real-Time Computer Systems (RCS). OSCAR was designed from a defined system structure for the exploration of indoor environments. OSCAR is currently further developed with the goal to support a wide range of different component types.

The remainder of the paper is structured as follows: Section 2 gives an overview about related work, Section 3 describes the overall system structure OSCAR is embedded in. In Section 4 we present the component-base architecture in detail. In Section 5 the behavior-based approach to coordinate the robot's behavior is described. An application example is provided in Section 6. The paper concludes with our perspectives about future work in Section 7.

2. Related Work

By inspecting intelligent (mobile) robot systems one will find in most cases a software architecture which defines a more or less restrictive framework for mapping of functionality. Unfortunately is the term "architecture" often used in several different ways: it may describe the chosen approach to control the robot (deliberative, behavior-based or hybrid) [1] or relate to how algorithms have to be integrated and focuses more on communication issues (see e.g. [5, 6]).

Since the variety of architectures is wide, we have chosen to mention only a small subset: In [7] RCS (Realtime Control System) is proposed as a reference architecture for intelligent systems. RCS consists of hierarchically layered processing nodes with a strong distribution of deliberative and reactive functionality. A similar architecture called TCA (Task Control Architecture) [8] introduces a system that is able to handle information processing in tasks. TCA provides a high-level method for passing messages between distributed systems and capabilities to schedule tasks and manage their resources. Saphira [9], with its C-like programming language COLBERT [10], provides a flexible "interface" for task composition on a middle level of abstraction.

[*] The work presented in this paper was supported by the *Deutsche Forschungsgemeinschaft* as part of the project *Exploration von Innenräumen mit optischen Sensoren auf mehreren, aufgabengerechten Abstraktionsebenen* (Fa109/14).

Behavior-based control mechanisms, first introduced in [11], find their representative in several software architectures: In [12] a schema-based behavior coordination approach is proposed. BERRA [13] is a behavior-based architecture for service robots integrating a human-computer interface in its deliberative planner. In [14] an architecture is described in which behavior modules are controlled by an arbiter deploying a real-time operating system. A similar approach provides the DD-Designer environment for behavior engineering for RoboCup mobile platforms [15]. The related Dual Dynamics architecture includes a framework for a specification-centered design approach.

Mobility [16] is a commercial available system that is shipped with RWI robot platforms and supports unfortunately only a certain class of robots. Beyond it, component software [2] is not yet available for robotic systems since there is no standardization for domain-specific interfaces. Another problem is that the term “component” is not restrictly defined [17]. We adopt the definition provided in [2].

3. System Structure

Overview For the exploration of indoor environments, a general purpose system structure was defined for the autonomous robot MARVIN (Figure 1). MARVIN is equipped with four PCs running under the Linux operating system. As actuators, the robot utilizes the platform Labmate (TRC) and an AMTEC camera head. We employ two b/w CCD cameras and a 360 degree laser range scanner from Accurange as sensors. Additionally, a hardware board to calculate optic flow vectors in real-time is available.

The system structure (Figure 1) defines different processing units as the basic elements. We distinguish *physical sensors*, *logical sensors*, *interpretation modules*, *integration modules* and *actuator modules*. As a global data base, we employ *GEM* (Generalized Environmental Model, see below) [18]. Data stored in GEM can be accessed by *GEM access modules* that serve as local data caches. Consequently, the different processing units are organized hierarchically in several layers. Within the sensor and interpretation layer, a hierarchy of processing units is also possible.

Information processing in the system structure can be described as follows: Sensor raw data is handled by physical sensors. Logical sensors extract and abstract features in a way that interpretation modules can interpret it employing GEM access modules for obtaining model data. As a result, e.g. different object and localization hypotheses are generated that may be fused in integration modules, that themself supply GEM with new achieved environmental information. Sequential control and activation, deactivation and configuration is performed by the coordination layer. A *coordination module* controls the available actuators that are encapsulated by *actuator modules*.

A key design decision is to unify all involved processing units as much as possible, i.e. to share common interfaces for data flow and configuration. This is also associated with standardized data formats (see Section 4.). Although this may cause some overhead, the unification strategy is the only possibility to benefit from component-based systems.

This concept makes the system employable for different devices and robots by exchanging physical sensor and actuator modules. In general, the design turns the system structure into a highly scalable architecture that supports exchanging modules with same functions, but different implementations, processing speed, accuracy, etc.

Generalized Environmental Model GEM serves as object-oriented environmental model in the OSCAR system architecture. GEM offers the possibility to store environmental features in several levels of abstraction: single features such as a wall-floor line or a single surface may be stored as well as complex objects in boundary representation *(B-Rep.)*. GEM supports data access with access modules (see below) for any applied sensor. Hereby, the referring sensor model also is retained. Prediction e.g. of visible features by executing a z-buffering algorithm is as well supported as indexing in a context of generation of hypotheses.

Data access is possible on world, object and feature level, whereby the abstraction level *world* itself is stored as a topological graph containing *islands of geometric models* referred to as *geometric islands* in the following. Nodes of the graph point to mission relevant world positions within geometric islands, edges are attributed with a sequence of instructions for passing them. Depending on the target being in the same geometric island, driving instructions range from cartesian way points to qualitative behaviors. A geometric island consequently contains at least one node, but normally several nodes. New geometric islands are generated, if a driving path’ geometric confidence is below a certain threshold and must therefore be described topologically.

Objects and features may be stored in a hierarchical order, i.e. objects generally can encompass member objects and features may be aggregated to more complex ones respectively. The applied storing form also takes modeling of possible degrees of freedom into account. GEM may be pre-loaded with already achieved environmental information before a mission is performed. A generic mission expert supplies GEM with mission relevant object models. GEM consequently is in charge of providing the persistence of environmental information for future (service) tasks of the mobile robot.

4. Communication Issues and Component-based Framework

4.1 Communication

The communication issue is one of the most crucial topics related to distributed system architectures. While some

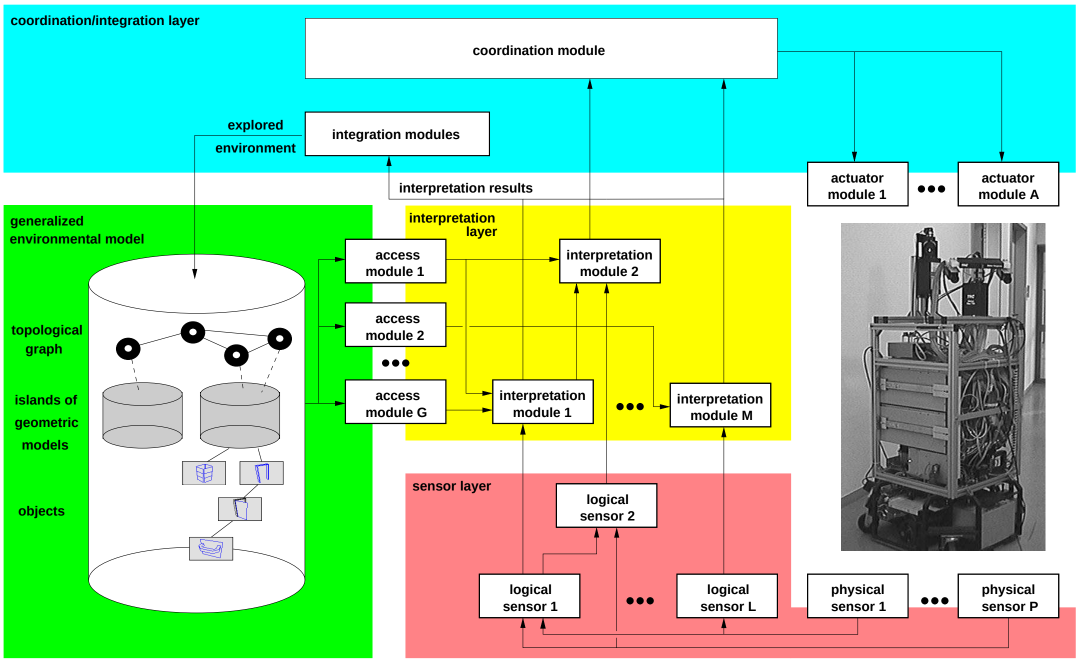

Figure 1. System structure

architectures come with their own communication layer (see e.g. [19]), OSCAR fully relies on the middle-ware standard CORBA 2.3 [20] specified by the OMG (Object Management Group) consortium. The benefits of CORBA are – besides the incorporation of the object-oriented design paradigm to the client-server approach – its interoperability, scalability and independence of programming language and hardware platform. This simplifies the portation to e.g. a real-time operating system in the future. As implementation, we employ Orbacus from IONA Technologies [21] that incorporates a full support for threaded applications. Performance measurements related to data transport can be found in [4].

4.2 Data Flow

In the following, two different data categories are applied: *Cues* are data that are mainly processed by components (see below), i.e. in the hierarchical architecture (Figure 1), cues are streaming from the physical sensors to the integration modules. Consequently, cues emcompass sensor raw data, extracted features, hypotheses, etc. Cues are container-like typed data and contain always a sequence of timestamps to monitor e.g. the point in time of their generation or fusion with other cues. Moreover, the cue type is defined in CORBA IDL and therefore is easy to standardize. Cue flow is realized between two components using the pull (data flow by request) or the push (automatic data flow) paradigm. In contrary, *configuration data* is used to (re-)configure components. A configuration data is a pair of a token and a numeric or string value. Thus, configuration data is specific for particular components. Data flow from/to GEM, data flow inside the coordination layer as well as control flow to the actuator modules is standardized using specialized interfaces.

4.3 OSCAR Components

Within the OSCAR context, a component is defined as the smallest data processing unit. All unit types defined in Section 3. are therefore mapped in components. For the developer's perspective, a component defines a lean application programming interface (API) that is realized with an abstract C++ class. To implement a component, several virtual methods have to be overridden: the main information processing (*Step* method) and actions to be performed for reconfiguration, activation and deactivation, etc. Furthermore, a component can allocate input and output channels. A specific channel is needed for each different cue type. Besides pull and push, it is possible to access cues with a timestamp as index and chronologically ordered arrays of cues. Non-blocking triggering of cue processing from lower-level components is also supported.

The OSCAR framework specifies different component types for processing units in the system's hierarchy, whereby only the amount of interfaces provided is concerned: physical sensor do not provide input channel facilities as well as integration modules and behavior components do not provide output channel facilities; behavior components, integration and interpretation modules may employ an interface to GEM access modules.

Component Embedding The granularity of a component, i.e. the "amount" of data processing algorithms has to be determined considering the trade-off between reusability and the possibility to be distributed on the one hand side and the time and memory consuming overhead caused by data copying, communication, thread or process context switch on the other side. The design of the OSCAR framework tries to introduce a possibility to keep this overhead small in a way that fine-grained components are possible.

A single component may run in two different modes: *continuous mode* means that the component is actively invoking its cue processing method *Step* periodically, while *single-step mode* leads to a reactive behavior where cue processing is only performed if the component is triggered from "outside". Thus the single-step mode induces a synchronous coupling of components, where the continuous mode may lead to an asynchronous delivery of cues. The implementation of both modes and the form of cue flow (pull/push) as well as the storage of a limited amount of cues to keep track of the history make a buffering of cues in communication channels necessary. Furthermore, those buffers have the meaning of a local cache, e.g. if a component performs fusion of cues over time. Since cue data flow in a channel may underlay a 1-n or n-1 connection relationship (for pull and push resp.), in some cases a double buffering of cues prevents the loss of data especially if cues are processed with strong different time rates. This necessity leads to a more or less tight coupling between components.

Module Framework Tighter coupled components can be composed to component agglomerations with respect of certain constraints. Emerging component agglomerations are referred from now on as *modules*[1]. Within a module the following rules have to be considered:

- Since a module is embedded in a single process, all components must be runnable device-constrained on the same host.
- Processing within a module always is synchronous, i.e. only up to one component is running in continuous mode, all other are deactivated or are running in single-step mode.
- Either the application of pull or push cue transport mechanism is allowed.

A component is always embedded in a module. Within the OSCAR framework components may be instantiated several times in different modules e.g. for parallel processing. The module infrastructure also takes care of cue buffering. Therefore, private and public accessible ring buffers are implemented. For specific internal cue channels, the ring buffer may be left out to avoid cue copying. For the distribution of components to modules, it also has to be considered that avoiding communication overhead in larger modules induces to longer responding times of data requested to be processed. In contrary, older cues can be requested ad hoc, since the module consists of two threads.

[1]The term module is used in another sense as for interpretation, integration or actuator module in Section 3..

Realization of modules and components As stated before, an OSCAR module runs in one process. Since the composition of components in modules is not defined at compile time, modules employ a dynamic plugin concept relying on the dynamic linking library *libdl*. Therefore components as well as cue type stubs and skeletons are only available as shared objects.

4.4 Meta-Infrastructure

OSCAR that can been seen as an abstract operating system provides an assemblage of meta-infrastructure. A boot routine starts and configures all modules, related components are registered in a centralized registry. Monitors installed on each PC hosting a OSCAR module keep the related processes under surveillance. Components may be exchanged at run-time, if e.g. a component run into an error state.

From the developers point of view, several facilities are provided for analyzing, component testing and debugging purposes including generic cue loggers and a simulation environment.

5. Behavior-based Coordination

One goal that the OSCAR architecture tries to consider is the possibility to design the overall behavior of a mobile robot for a given task by defining elementary behaviors. An elementary behavior depends on a set of given physical and logical sensors, interpretation and integration modules and actuator configurations. The connection structure of sensor, interpretation and integration components together with cue and configuration flow emerges from the employed behavior components for a scenario. This approach is therefore behavior specification-centered, in contrary to most architectures that define their system structure by parsing a configuration file that contains a list of processing units (see e.g. [13]).

For behavior coordination, we apply the dynamic approach defined by Steinhage and Bergener [22]. Hereby, pairwise relationships (inhibition and requirement) between each two elementary behaviors can be defined. Both relationships are coded in two $n \times n$ matrices, where n is the number of elementary behaviors. The activation of each elementary behavior is controlled by a set of differential equations evaluating both matrices and considering the individual context for a each behavior.

In our architecture, each behavior is encapsulated in a behavior component. Every behavior component is controlled by the arbitration module that implements the integration of the differential equations. Additionally, a finite automaton is in charge of sequential control.

6. Application Example

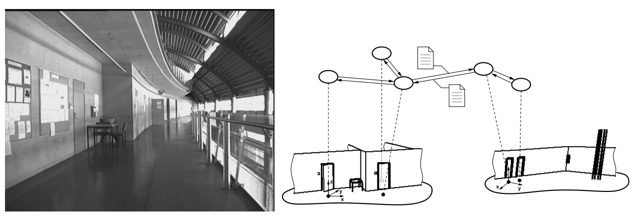

Figure 2. Scenario: Doorway and structure of the model to be explored

As an use-case of OSCAR, we have realized the scenario *exploration of a doorway* (Figure 2). Hereby the goal is to detect the doors of the doorway and register them with their room number in GEM. Thereby a node in the topological graph is to be created at every door. Apart from a geometric description of the object class door, which is registered a-priori in the model, the robot has no information about its environment. (After the exploration, the robot should be able to accept a room number as input and drive directly to it.)

The typical sequence of action to register an instance of the object "door" in the model is executed as follows: First, the mobile platform is searching a wall considering line segments extracted from range data from the panoramic laser scanner[23]. When a wall was detected, the platform is following the wall creating a temporary local map where static and moving obstacles sensed with the laser and an optic flow sensor[24] are stored. The map is also applied to determine possible movements of the platform. Moreover, the laser line segments are used to predict possible vertical video line segments [25] to detect doors. When a door is detected, the inaccurate relative coordinates of the door are used to determine a position for the mobile robot to localize the door correctly. This position is targeted and the coordinates of the door are determined correctly. Then the robot is moving in front of the door and reads the door plate using optical character recognition. The coordinates of the door and the room number are stored together with the position of the robot determined by the odometry in GEM.

For the realization we have defined the following behaviors: `search_wall`, `follow_wall`, `detect_doors`, `localize_door`, `target_goal_point`, `read_door_plate` and `update_model`.

As described above, each applied elementary behavior depends on a certain set of components. Components may depend recursively on other components. Figure 3 shows the used components with the related cue flow for our design (Configuration flow is not considered.). Components inside dashed line limited region are composed within a module. Of course, the unification of cue processing components makes the agglomeration of components

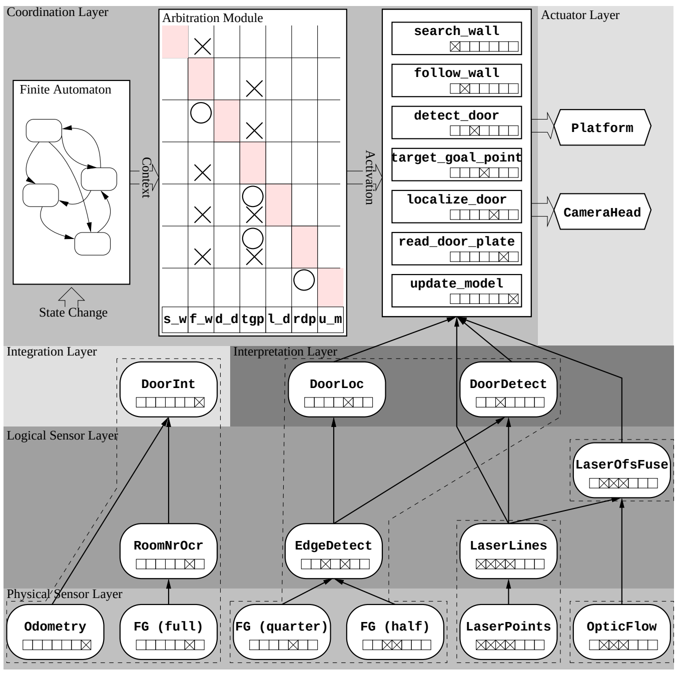

Figure 3. Applied component configuration

not residing in the same layer feasible.

The tic box array shows the dependencies of the behaviors: E.g. the behavior `detect_door` depends on the `DoorDetect` component in the interpretation layer. The `DoorDetect` component depends itself on the logical sensor components `EdgeDetect` and `LaserLines` etc. Furthermore, inhibition and requirement relationships are stored in the arbitration module. The activation of the behavior `detect_door` requires the behavior `follow_wall` (marked as circle in the matrix of the arbitration module) and inhibits the behavior `target_goal_point` (marked as cross).

The sequential control is fulfilled using a finite automaton. Hereby, exception handling (e.g. if a door could not been localized) is also considered. State changes are induced from components in the integration, interpretation and sensor layer and forwarded as contexts to the arbitration module. Note that the application the arbitration module reduces the number of required states. E.g. it is not necessary to insert a state for `update_model` since this behavior only can be activated when the behavior `read_door_plate` was active. Another advantage is hereby that updating the model is done while the robot is about to follow the wall again, i.e. parallel execution of behaviors is possible which cannot easily be implemented using only a state automaton.

7. Conclusion and Future Work

We have presented the system architecture OSCAR for autonomous mobile systems. The outlay of the architecture proposes a definition for domain-specific interfaces for data processing and behavior units. The related framework provides efficient embedding structures for components and takes care for all communication issues. Although the architecture is still under development and only a few components are available, we are confident that more components will emerge by realizing more use-cases also for other robotic platforms. Beyond it, we hope that a larger amount of components will emerge when OSCAR will be utilized by several institutes [26].

Besides, future work will include an automated mapping of components regarding load balancing issues. Furthermore, a formalism is to be specified to define exploration and service tasks even beyond behavior composition level. Therefore, an elimination of the still hard-wired finite automaton is considered.

References

[1] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[2] Clemens Szyperski. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley Publishing Company, 1997.

[3] Stefan Blum, Tobias Einsele, Alexa Hauck, Norbert O. Stöffler, Georg Färber, Thorsten Schmitt, Christoph Zierl, and Bernd Radig. Eine konfigurierbare Systemarchitektur zur geometrisch-topologischen Exploration von Innenräumen. In *Autonome Mobile Systeme*, Informatik aktuell, pages 378–387. Springer-Verlag, November 1999.

[4] Stefan Blum. OSCAR - Eine Systemarchitektur für den autonomen, mobilen Roboter MARVIN. In *Autonome Mobile Systeme*, Informatik aktuell, pages 218–230. Springer-Verlag, November 2000.

[5] M. Klupsch. Object-Oriented Representation of Time-Varying Data Sequences in Multiagent Systems. In N.C. Callaos, editor, *International Conference on Information Systems Analysis and Synthesis (ISAS'98)*, pages 833–839, Orlando, FL, USA, 1998. International Institute of Informatics and Systemics (IIIS).

[6] C. Schlegel, J. Illmann, H. Jaberg, M. Schuster, and R. Wörtz. Integrating Vision Based Behaviours with an Autonomous Robot. In *International Conference on Vision Systems (ICVS)*, volume 1542 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[7] J. S. Albus and A. M. Meystel. A Reference Model Architecture for Design and Implementation of Intelligent Control in Large and Complex systems. *Int. J. of Intelligent Control and Sytems*, 1(1):15–30, 1996.

[8] Reid Simmons. An architecture for coordinating planning, sensing, and action. In *Proc. of the DAPRA workshop*, pages 292–297, 1990.

[9] K. Konolige and K. Myers. The saphira architecture for autonomous mobile robots. Artifical Intelligence Center, SRI International, Menlo Park, California, 1996.

[10] Kurt Konolige. COLBERT: A Language for Reactive Control in Saphira. In *German Conference on Artificial Intelligence*, Freiburg, 1997.

[11] R. A. Brooks. A Robust Layered Control System For a Mobile Robot. *IEEE Journal of Robotics and Automatisation*, RA-2, No. 1:14–23, 1986.

[12] R. C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4), 1989.

[13] M. Lindström, A. Orebäck, and H.I. Christensen. Berra - a behaviour based robot architecture. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'00)*, San Francisco, CA, USA, 2000.

[14] Thomas Bergener and Axel Steinhage. An Architecture for Behavioral Organization using Dynamical Systems. In C. Wilke, S. Altmeyer, and T. Martinetz, editors, *Third German Workshop on Artificial Life*. Verlag Harri Deutsch, 1998.

[15] Ansgar Bredenfeld and Giovanni Indiveri. Robot behavior engineering using DD-designer. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'01)*, Seoul, Korea, May 2001.

[16] Real World Interfaces, Inc. Mobility. http://www.rwii.com/rwi/software_mobility.html, 1998.

[17] Manfred Broy, Anton Deimel, Juergen Henn, Kai Koskimies, Frantisek Plasil, Gustav Pomberger, Wolfgang Pree, Michael Stal, and Clemens Szyperski. What characterizes a (software) component? *Software Concept & Tools*, 19:49–56, 1998.

[18] A. Hauck and N. O. Stöffler. A Hierarchic World Model Supporting Video-based Localization, Exploration and Object Identification. In *Proc. 2nd Asian Conf. on Computer Vision (ACCV'95)*, volume 3, pages 176–180, 1995.

[19] C. Fedor. TCX - An Interprocess Communication System for Building Robotic Architectures. Carnegie Mellon University, Pittsburg, Pennsylvania, 1993.

[20] OMG. CORBA/IIOP 2.3 specification. http://www.omg.org/corba, 1998.

[21] Object Oriented Concepts. *Orbacus*. http://www.ooc.com/ob/.

[22] A. Steinhage and T. Bergener. Dynamical Systems for the Behavioral Organization of an Anthropomorphic Mobile Robot. In *From animals to animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, pages 147–152. MIT Press, 1998.

[23] T. Einsele. Real-Time Self-Localization in Unknown Indoor Environments using a Panorama Laser Range Finder. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'97)*, pages 697–703, Grenoble, France, September 1997.

[24] Norbert O. Stöffler and Georg Färber. An Image Processing Board with an MPEG Processor and Additional Confidence Calculation for Fast and Robust Optic Flow Generation in Real Environments. In *Proc. Int. Conf. on Advanced Robotics (ICAR'97)*, pages 845–850, Monterey, CA, USA, July 1997.

[25] G. Magin and C. Robl. A Single Processor Real-Time Edge-Line Extraction System for Feature Tracking. In *IAPR Workshop on Machine Vision Applications (IAPR MVA'96)*, 1996.

[26] Stefan Blum. OSCAR-Homepage. http://www.oscar-net.org.