A dynamic architecture to develop distributed software for multi-modal service robots

Daniel Westhoff
email: westhoff@informatik.uni-hamburg.de

Introduction

A novel concept for the programming of service robots is proposed by the group TAMS. The presented software architecture eases the development of applications for service robots. This software architecture is based upon the Roblet-Technology. It introduces the possibility to develop, compile and execute an application on one workstation. Since the Roblet-Technology uses Java development is independent of the operation system. With the feature of running programs as a distributed software, the framework allows running algorithms which need great computation power on remote machines which provide this power. In this way, it greatly improves programming of applications in service robotics. The concept is evaluated in the context of the service robot TASER of the TAMS Institute at the University of Hamburg. This robot consists of a mobile platform with two manipulators equipped with artificial hands. Several multimodal input and output devices for interaction round off the robot.

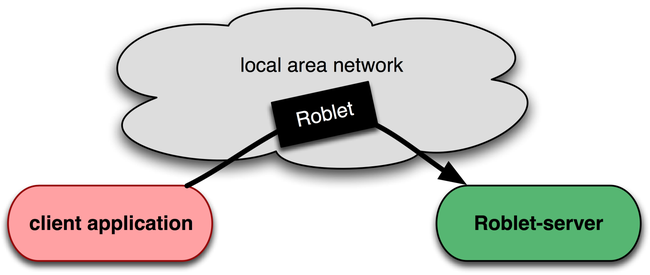

Figure 1: Application with one client and one server.

Roblets

The basic of the proposed framework is the Roblet-Technology, a concept firstly introduced by Westhoff *et.al.* in 2004. Roblet-Technology is a client-server architecture where clients can send parts of themselves, referred to as Roblets, to a server. The server, referred to as Roblet-server, then executes the Roblets with well-defined behavior in case of malfunctions. A basic setup is shown in figure 1 Not only data is transmitted between the client and server but complete executable programs. This can be compared to Java applets but with the difference that Roblets are not downloaded but sent.

Complex setups consist of multiple client applications and Roblet-servers. A Roblet terminates if the execution of its code finishes normally or returns an exception to the client in the case of failures. In addition, a Roblet can be terminated by a client remotely or by the Roblet-server directly. After a Roblet terminates, the Roblet-server resets itself. Roblet-Technology is applicable to all kinds of distributed systems but it has features that make its integration into robotic applications useful. In general, applications in service robotics are distributed systems. Besides one or multiple mobile robots, there are visualisation- and control applicatons that run on workstations in a local area network. Sometimes there is no direct access to the robot systems via keyboard, mouse and monitor but only through a wireless network. When a Roblet-application is executed it will send parts of itself to available servers and spread in the local network. Roblets may send parts of themselves to other servers as well. The network communication is hidden by the Roblet library, which simplifies the overall development. That means, the network is transparent and developing distributed applications based on Roblet-Technology is like developing an application for one workstation. Access to the remote servers is encapsulated in a client library, reducing the execution of a Roblet on the remote system to one method call.

Modules

For robotic applications we propose *modules* to extend a basic Roblet-server as shown in figure 2. A module is loaded when the Roblet server is started. A module encapsulates a class of similar functionality. For the robot TASER we developed several modules: One module merges the functionality of the mobile platform, a second module wraps the manipulator system including the robot arms and the hands. There are modules for the different vision systems, the pan-tilt unit, a speech module and other parts of the interaction subsystem. Notice that the map server and the pathplanning server don't run on the robot's control computer but on a workstation in the local network. This allows the integration of information gathered by multiple robots. For example, in the case of dynamic map adjustment this relieves the robot's onboard computer of some computationally expensive tasks.

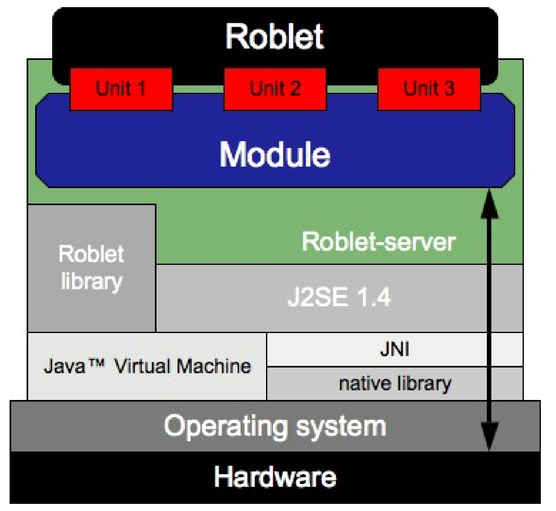

Figure 2: Chart of the generic Roblet server and how it is extended by a module.

Units

Modules are further devided into *units*. Units are Java interfaces that are implemented within the modules. Units build the hardware abstraction layer in our framework. For example, a module encapsulates the localization subsystem of a mobile robot and a Roblet wants to query the current pose estimate of the robot. The module implements a unit which defines a method to get the pose. On another robot there may be another localization system encapsulated by another module. But, if the module implements the same unit, the same roblet can be executed on both robots without changes. Nonetheless, special features of a subsystem are available to Roblets if module-specific units are implemented. A roblet has only access to units, it does not know anything about a module and a module's implementation of an interface. The whole concept is strictly object-oriented. By introducing units, the framework is able to generalize access to similar classes of subsystems without loosing access to their special features. Additionally, units introduce a possibility of versioning into the system. If new features are integrated into a module then new units will be introduced. As long as older units are still available, all Roblets using these old units still work. This has proven to be of great use since complex applications often consist of dozens of client applications and Roblet-servers.