A Component Based Approach for Robotics Software based on Communication Patterns: Crafting Modular and Interoperable Systems

(Position Paper Workshop W-M02, ICRA 2005)

Christian Schlegel
University of Applied Sciences
Prittwitzstrasse 10
D-89075 Ulm
Germany
schlegel@fh-ulm.de

18. April 2005

1 Context

Vital functions of robots are provided by software and software dominance is still growing. Mastering the software complexity is not only a demanding but also indispensable task towards an operational robot. Component based software approaches provide suitable means to master the complexity issue. Nevertheless, shareable, distributable and reusable off-the-shelf software components for robotics are still a dream. The statements are derived from more than ten years of experience in implementing any kind of robotics demonstrators. The focus has been on mobile platforms which have to execute a variety of tasks autonomously within an office-like indoor environment. The focus is not on hard realtime control frameworks.

2 Problem

One of the reasons for missing standard robotics software components is the lack of a software component model taking into account robotics needs. The challenge of component based software approaches for robotic systems is to assist in building a system and to provide a software architecture without enforcing a particular robot architecture. In particular in robotics, there are many demanding functional and non-functional requirements.

3 Forces

- Different requirements from different user groups

  End users operate an application based on the provided user interface. They focus on the functionality of their application and use a readily provided system with a given functionality to fulfill the required tasks. They do not care on how the application has been built by the application builder and mainly expect reliable operation.

Application builders assemble applications based on suitable and reusable components. They customize them by adjusting parameters and sometimes even fill in application dependent parts at *hot spots*. They expect the framework to ensure clearly structured and consistent component interfaces for easy assembling of approved off-the-shelf components.

Component builders focus on the specification and implementation of a single component. They expect the framework to provide the infrastructure which supports their implementation effort in such a way that it is compatible with other components without being restricted too much with regard to component internals. They want to focus on algorithms and component functionality without bothering with integration issues.

Framework builders design and implement the framework such that it matches the manifold requirements at its best and that the above types of users can focus on their role.

- *Framework builders* are the software engineering experts who provide the framework and easy-to-use patterns based on state-of-the-art software technology. They hide all the difficult details of complex middleware systems like CORBA from the other user groups.
- *Component builders* accept only few restrictions with respect to the component internal structures. On the other hand, they expect the framework to ensure that afterwards everything fits together nicely and can be exchanged / modified / reused. A software framework has to provide obvious additional value with easy-to-use access to state-of-the-art software technology.
- *Project coordination* requires means to specify interfaces, responsibilities, guarantees and commitments.
- From the technical point of view, robotics always requires to cope with
  - inherent complexity due to concurrent activities
  - distributed development
  - deployment of software components on networked computers ranging from embedded systems to PCs
  - only some parts require hard realtime, for many parts soft realtime or even no realtime is sufficient

4 Solution (The SMARTSOFT Approach)

Component based approaches already proved that they provide the right level of granularity. However, there is a need for a framework on top of a component based approach that assists in system level integration taking into account robotics requirements without enforcing a particular architecture.

Components are technically implemented as processes. A component can contain several threads and interacts with other components via predefined communication patterns. Components can be wired dynamically at runtime.

Communication Patterns assist the component builder and the application builder in building and using distributed components in such a way that the semantics of the interface is predefined by the patterns, irrespective of where they are applied. A communication pattern defines the communication mode, provides predefined access methods and hides all the communication

and synchronization issues. It always consists of two complementary parts named *service requestor* and *service provider* representing a *client/server*, *master/slave* or *publisher/subscriber* relationship.

Communication Objects parameterize the communication pattern templates. They represent the content to be transmitted via a communication pattern. They are always transmitted *by value* to avoid fine-grained intercomponent communication when accessing an attribute. Furthermore, object responsibilities are much simpler with locally maintained objects than with remote objects. Communication objects are ordinary objects decorated with additional member functions for use by the framework.

Service Each instantiation of a communication pattern provides a service. A service comprises the communication mode as defined by the communication pattern and the content as defined by the communication objects.

The service based view comes along with a specific granularity of a component based approach. Services are not as fine-grained as arbitrary component interfaces since they are self-contained and meaningful entities and not only dependencies spanning across components.

The basic idea of the SMARTSOFT approach is as follows:

- A fixed set of communication patterns which provide predefined component interaction modes is used to compose all component interfaces. Communication patterns provide the only link of a component to the external world and can therefore ensure decoupling at various levels. Communication patterns decouple structures used inside a component from the external behavior of a component. Decoupling starts with the specific level of granularity of component interfaces enforced by the communication patterns which avoids too fine-grained interactions and ends with the message oriented mechanisms used inside the patterns between components.

- Using communication patterns with given access modes prevents the user from puzzling over the semantics and behavior of both component interfaces and usage of services. One can neither expose arbitrary member functions as component interface nor can one dilute the precise interface semantics and the interface behavior. Given member functions provide predefined user access modes and hide concurrency and sychronization issues from the user and can exploit asynchronicity without teasing the user with such details.

- Arbitrary communication objects provide diversity and ensure genericity with a very small set of communication patterns. Individual member functions are moved from the externally visible interface to communication objects.

- *Dynamic wiring* of intercomponent connections at runtime supports context and task dependent assembly of components.

Since component interactions are mapped onto predefined communication patterns, all component interfaces are composed of the same set of patterns. Therefore, looking at the external interface of a component immediately opens up the provided and required services, and looking at the communication pattern underlying a service immediately opens up the usage and semantics of this service.

5 Evaluation

This approach already proved its suitability over several years. The latest implementation is based on ACE/TAO (CORBA) and is available under GPL/LGPL (see accompanying workshop slides).

6 Technical Issues

Further hints on which aspects seem to be stable over all kinds of robotic applications can be found in section 4 of the accompanying slides. This shortly summarizes some aspects:

- Communication patterns result in component interfaces with clear semantics since all component interfaces are always composed out of the same patterns and thus all expose the same methods. This greatly simplifies commitments on component functionality and interfaces and allows testing and replacement with low effort.
- Dynamic wiring needs to be integrated into the component interfaces since disconnecting and reconnecting services requires to properly handle pending answers etc. Normally, this results in complex component internal structures. Since all these challenges are already handled inside the communication patterns, a great source of errors is already removed from the responsibility of a component builder.
- Dynamic wiring is a basic functionality needed in nearly any robotics application to dynamically compose complex functionalities out of distributed components. Too fine-grained component interfaces make it impossible to rearrange component connections due to unmanageable component internal states.
- *By-value* semantics of transferred objects greatly reduces the sphere of influence and local object responsibilities are much simpler to handle than complex object lifetime mechanisms. Furthermore, local object responsibilities greatly reduce network load since overloaded operators etc. do not unexpectedly expand across component boundaries.
- Communication objects can be extended by arbitrary interfaces without affecting the core data structures transmitted via the communication patterns. Thus, additional interfaces are privat to the component and need not be exposed globally.
- Separating the middleware data type description inside the communication objects from the user access methods allows to use any data type independent of the capabilities of the middleware data type description language. For example, one can easily transfer objects which contain heap memory or which use STL classes at their interface methods. Thus, middleware migrations only affect the description inside the communication objects and are *not* visible outside the communication objects. Furthermore, advancing middleware technology can be seemlessly applied within the communication patterns without affecting the framework semantics.
- User access modes at both parts of communication patterns (the service provider and the service requestor) are completely independent and at service requestors, all provided user access modes can be used concurrently. Thus, one can use the synchronous and the asynchronous access modes of a service concurrently without further coordination and without implying any processing mode at the other side of the communication pattern. This flexibility based on decoupling ensures that components can internally implement that structure that is most suitable without side effects on internals of other components.

- Tools and software should be under GPL (parts better LGPL) to ensure wide spread usage and even allowing commercial activities based on the at some time grown base of device drivers and components.

7 Bibliography

- For most up-to-date information on the communication patterns and a CORBA based implementation see the web site at

  http://www.rz.fh-ulm.de/∼cschlege

- The most comprehensive description of the communication patterns can be found in my thesis available on the above web page.

References

[1] C. Schlegel. A Component Approach for Robotics Software: Communication Patterns in the OROCOS Context. In *18. Fachtagung Autonome Mobile Systeme (AMS)*, Informatik aktuell, pages 253–263, Karlsruhe, December 2003. Springer.

[2] C. Schlegel. *Navigation and Execution for Mobile Robots in Dynamic Environments: An Integrated Approach*. PhD thesis, University of Ulm, 2004.

[3] C. Schlegel and R. Wörz. Der Softwarerahmen SMARTSOFT zur Implementierung sensomotorischer Systeme. In *14. Fachtagung Autonome Mobile Systeme (AMS)*, Informatik aktuell, pages 208–217, Karlsruhe, November 1998. Springer.

[4] C. Schlegel and R. Wörz. Interfacing different layers of a multilayer architecture for sensorimotor systems using the object-oriented framework SMARTSOFT. In *Proceedings 3rd European Workshop on Advanced Mobile Robots (EUROBOT)*, Zürich, Schweiz, September 1999.

[5] C. Schlegel and R. Wörz. The software framework SMARTSOFT for implementing sensorimotor systems. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1610–1616, Kyongju, Korea, October 1999.