Diploma Thesis

Porting the MIRO Middleware to a Mobile Robot Platform

Daniel Krüger
<daniel.krueger@web.de>

Faculty of Computer Science
Chair of Real Time Systems

CHEMNITZ UNIVERSITY OF TECHNOLOGY

September 6, 2005

Table of Contents

Introduction

Context

- Miro has been already ported to the robot platform Pioneer 2-AT

Objectives

- porting Miro to the Mobile Platform “Franz”
- analysis of security aspects in robot control

Introduction

Context

- Miro has been already ported to the robot platform Pioneer 2-AT

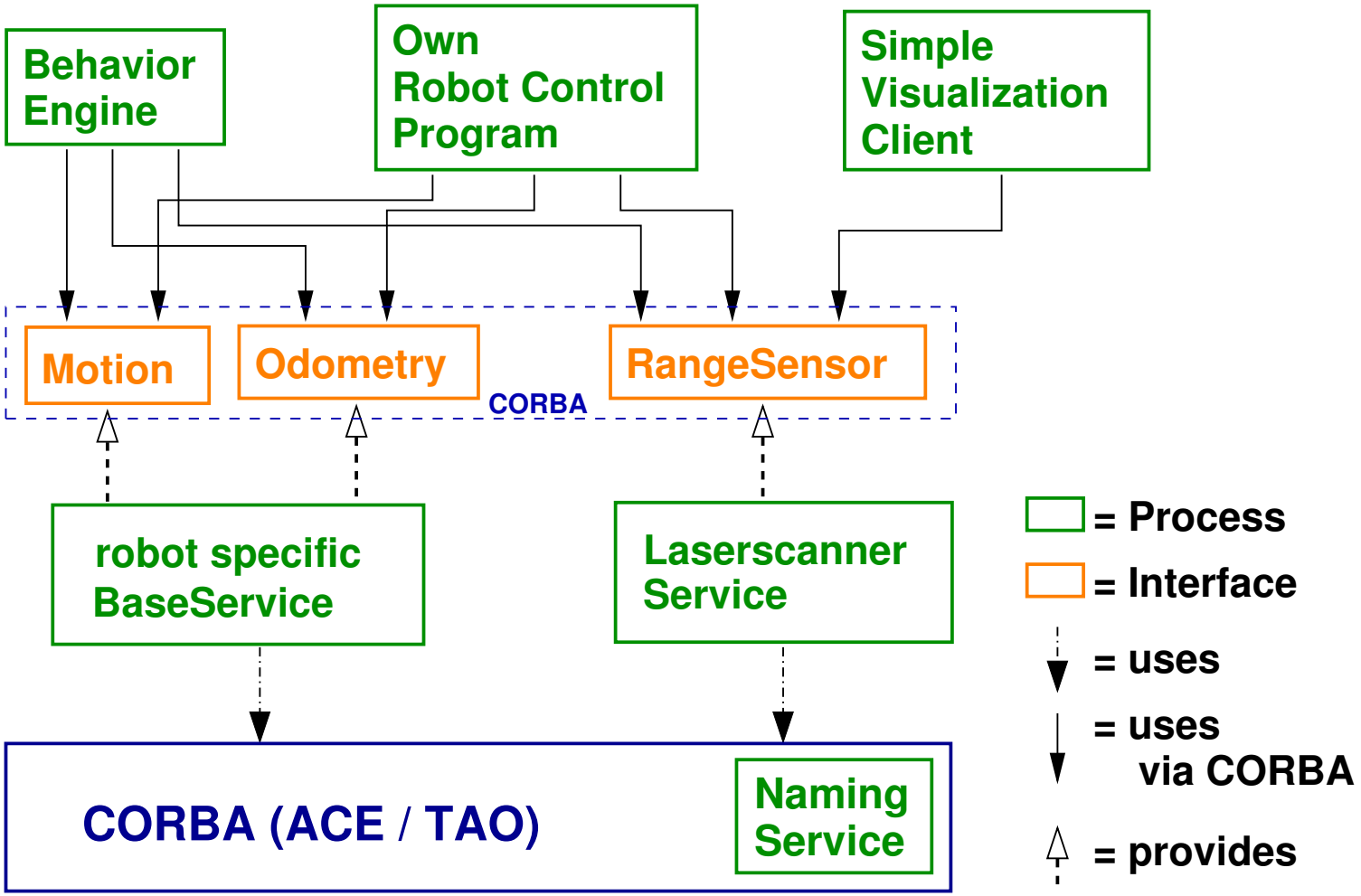
Objectives

- porting Miro to the Mobile Platform “Franz”
- analysis of security aspects in robot control

Miro is ...

- developed at Ulm University
- a middleware (software architecture) for robots
- a distributed object oriented interface specification, based on CORBA (Common Object Request Broker Architecture)
  - Miro uses TAO (The ACE ORB) as CORBA implementation.
  - TAO uses ACE (The ADAPTIVE Communication Environment) as operating system abstraction layer.
- an implementation of core components in C++ under Linux

Architecture of Miro



The Mobile Platform "Franz"

Characteristics of "Franz"

- based on an electrical wheel chair
- Ackermann steering
- rear wheel drive
- drive control by a DSP which is programmed by WIN-DDC
- 2 computers with VIA EPIA Mini-ITX mainboards, 1 GHz C3 Nehemia processors, 512 MB RAM, 40 GB hard disks, ...
- existing sensors
  - incremental position encoders on both front wheels
  - ultra sonic range finders SRF08 connected with I^2C bus
  - 2 SICK LIDAR (one with tilt unit)
- many more sensors are planned

The Mobile Platform “Franz”

The Base Service `FranzBase`

- communication with WIN-DDC user program in the DSP
  - reading user program variables via visualization function
  - setting user program variables to control the robot
- design and implementation of interface `Miro::AckermannMotion`, which is derived from interface `Miro::Motion`, for drive control
- transform user program variables for integration into interface `Miro::Odometry` (translational / rotational velocity, position and heading)

The Mobile Platform "Franz"

Ultra Sonic Range Finder SRF08

- attached at I^2C bus
- service provides two objects of interface `Miro::RangeSensor` via CORBA Naming Service:
  1. `Sonar`: multiple groups of range sensors, which contain the ranges of subsequent echos, i.e. first group represents first echo and so on
  2. `Light`: one group of range sensors, which contains the readings of the light sensors as range from 0 to 255
- is completely separated from `FranzBase`
- triggers the sensors in an endless loop

The Mobile Platform “Franz”

Developed Clients and Tools

- `QtFranzControl`: GUI client to control robots, especially “Franz”
- `CtxIOR` / `CtxAdd`: tools to bind a Miro Naming Context to another Naming Context

Miro and Security

Why to Deal with Security Aspects

- communication between CORBA server and clients is not encrypted and insecure
- everybody who has got an Interoperable Object Reference can invoke methods of this object

Miro and Security

Solution

- CORBA Security Service and SSL Inter ORB Protocol (SSLIOP)
- but this is not fully implemented in TAO yet
- furthermore encryption resp. decryption has a significant processing overhead

"Work-Around"

- concept of CORBA Portable Interceptors
- enable encryption (SSLIOP) on selected CORBA objects (e.g. for actuator interfaces)
- access control is done by SSL certificates

Miro and Security

Solution

- CORBA Security Service and SSL Inter ORB Protocol (SSLIOP)
- but this is not fully implemented in TAO yet
- furthermore encryption resp. decryption has a significant processing overhead

"Work-Around"

- concept of CORBA Portable Interceptors
- enable encryption (SSLIOP) on selected CORBA objects (e.g. for actuator interfaces)
- access control is done by SSL certificates

Miro and Security

Feasibility Study

- demonstrates this solution
- server registers the objects `BarSecure` and `BarInsecure` in the CORBA Naming Service
- client tries to invoke a method on both objects secure and insecure

Conclusion

- good decision to use Miro
- “Franz” is qualified for outdoor robotics
- access control should be implemented in Miro, if TAO fully supports the CORBA Security Specification

Bibliography

- *Miro - Middleware for Robots*
  `http://smart.informatik.uni-ulm.de/MIRO/`
- *The ACE ORB*
  `http://www.theaceorb.com/`
- *The ADAPTIVE Communication Environment (ACE)*
  `http://www.cs.wustl.edu/~schmidt/ACE.html`
- *The OMG's CORBA Website*
  `http://www.corba.org/`
- *CORBA EXPLAINED SIMPLY*, Ciaran McHale
  `http://www.elet.polimi.it/upload/picco/Teaching/distsys/slides/CORBAbook.pdf`
- *Advanced CORBA Programming with C++*,
  Michi Henning and Steve Vinoski,
  Addison-Wesley Professional, 1999

Was ist CORBA?

CORBA: Common ORB Architecture

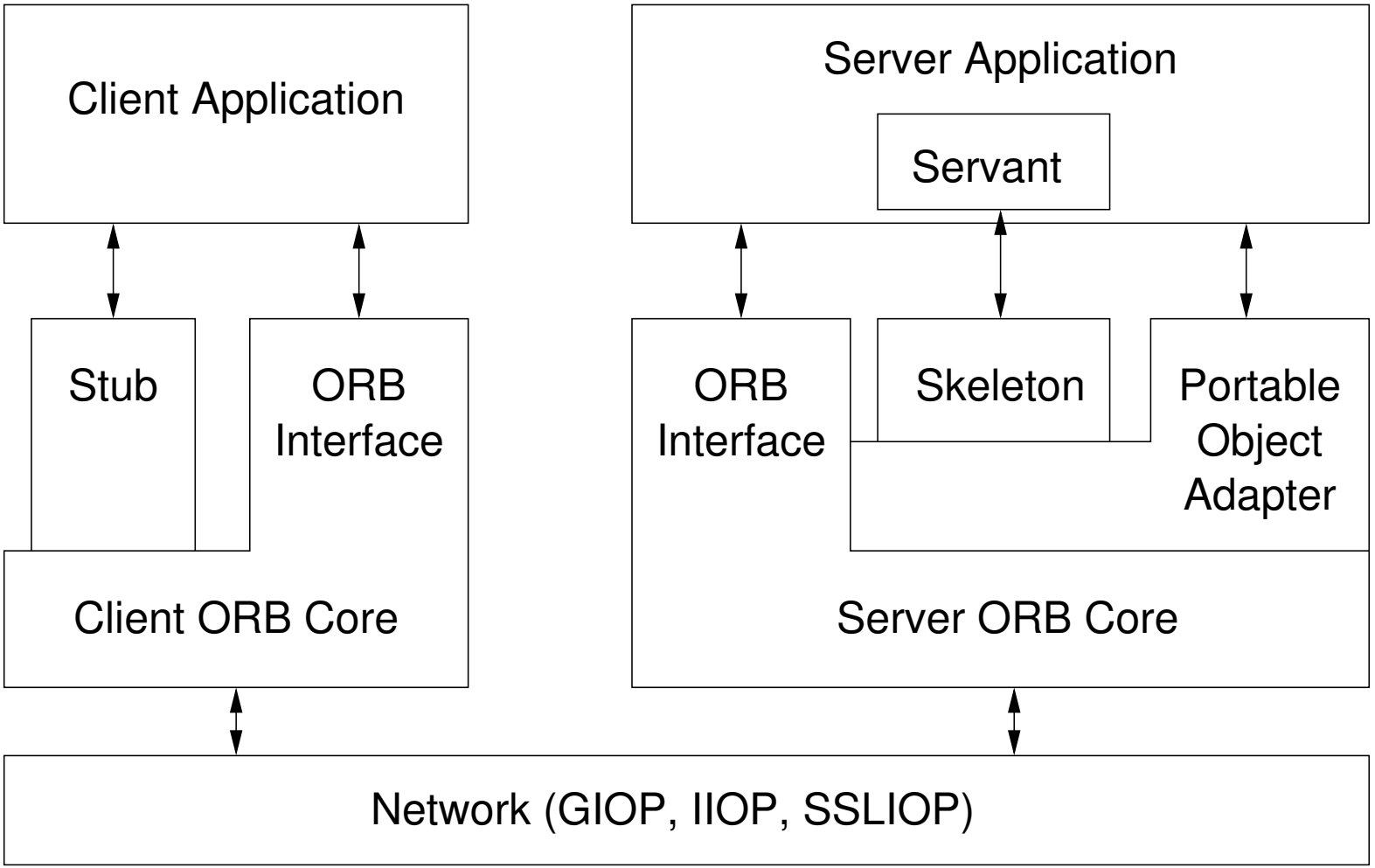
ORB: Object Request Broker

- überträgt und verteilt Aufrufe von Methoden auf Objekte
- Prozesse, die Objekte bereitstellen, heißen *Server*
- Prozesse, die mittels des ORB auf Objekte zugreifen, heißen *Clienten*
- ein Prozess kann aber auch gleichzeitig Server und Client sein
- diese Prozess können auch auf verschieden Rechnern in einem Netzwerk laufen

IDL: Interface Definition Language

- Sprache zur Beschreibung der Schnittstellen bzw. Objektklassen
- standardisiertes *Language Mapping* spezifiziert Umsetzung in einzelne Programmiersprachen

Aufbau von CORBA



IDL - Interface Definition Language

```
module Miro {
  struct VelocityIDL {
    long translation;
    double rotation;
  };
  interface Motion {
    void setVelocity(in VelocityIDL velocity)
                     raises(EOutOfBounds, EDevIO);
    void getMinMaxVelocity(out long minTranslation,
                           out long maxTranslation,
                           out double minRotation,
                           out double maxRotation);
    VelocityIDL getTargetVelocity();
  };
  exception EOutOfBounds { string what; };
};
```

IDL - Interface Definition Language

```
module Miro {
  struct VelocityIDL {
    long translation;
    double rotation;
  };
  interface Motion {
    void setVelocity(in VelocityIDL velocity)
                     raises(EOutOfBounds, EDevIO);
    void getMinMaxVelocity(out long minTranslation,
                           out long maxTranslation,
                           out double minRotation,
                           out double maxRotation);
    VelocityIDL getTargetVelocity();
  };
  exception EOutOfBounds { string what; };
};
```

IDL - Interface Definition Language

```
module Miro {
  struct VelocityIDL {
    long translation;
    double rotation;
  };
  interface Motion {
    void setVelocity(in VelocityIDL velocity)
                          raises(EOutOfBounds, EDevIO);
    void getMinMaxVelocity(out long minTranslation,
                           out long maxTranslation,
                           out double minRotation,
                           out double maxRotation);
    VelocityIDL getTargetVelocity();
  };
  exception EOutOfBounds { string what; };
};
```

IDL - Interface Definition Language

```
module Miro {
  struct VelocityIDL {
    long translation;
    double rotation;
  };
  interface Motion {
    void setVelocity(in VelocityIDL velocity)
                          raises(EOutOfBounds, EDevIO);
    void getMinMaxVelocity(out long minTranslation,
                           out long maxTranslation,
                           out double minRotation,
                           out double maxRotation);
    VelocityIDL getTargetVelocity();
  };
  exception EOutOfBounds { string what; };
};
```

Language Mapping und Implementation

1. Der IDL-Compiler (`$ACE_ROOT/bin/tao_idl`) erzeugt den *stub*-Klasse (z.B. in `NameC.h`) und den *skeleton*-Klasse (z.B. in `NameS.h`, `NameS.cpp`).
2. Server-Implementation
   - *skeleton*-Klasse ableiten und mit "'Sinn"' füllen
   - Haupt-Programm schreiben, welches ein Objekt dieser Klasse erzeugt und dessen *IOR* (Interoperable Object Reference) im *Naming Service* registriert und den *ORB* aufruft.
3. Der Client benötigt für den Zugriff auf ein Objekt dessen *IOR*, welchen er am besten vom *Naming Service* bekommt
4. Aus diesem *IOR* kann ein Objekt der *stub*-Klasse erzeugt werden, welches dann wie ein lokales Objekt funktioniert.

einfacher CORBA-Client

```
#include "idl/MotionC.h"
#include "miro/Client.h"

int main(int argc, char * argv[]) {
  Miro::Client client(argc, argv);

  try {
    Miro::Motion_var motion =
          client.resolveName<Miro::Motion>("Motion");
    motion->limp();
  } catch (const CORBA::Exception& e) {
    return 1;
  }
  return 0;
}
```

einfacher CORBA-Client

```
#include "idl/MotionC.h"
#include "miro/Client.h"

int main(int argc, char * argv[]) {
  Miro::Client client(argc, argv);

  try {
    Miro::Motion_var motion =
          client.resolveName<Miro::Motion>("Motion");
    motion->limp();
  } catch (const CORBA::Exception& e) {
    return 1;
  }
  return 0;
}
```

einfacher CORBA-Client

```
#include "idl/MotionC.h"
#include "miro/Client.h"

int main(int argc, char * argv[]) {
  Miro::Client client(argc, argv);

  try {
    Miro::Motion_var motion =
          client.resolveName<Miro::Motion>("Motion");
    motion->limp();
  } catch (const CORBA::Exception& e) {
    return 1;
  }
  return 0;
}
```

einfacher CORBA-Client

```
#include "idl/MotionC.h"
#include "miro/Client.h"

int main(int argc, char * argv[]) {
  Miro::Client client(argc, argv);

  try {
    Miro::Motion_var motion =
          client.resolveName<Miro::Motion>("Motion");
    motion->limp();
  } catch (const CORBA::Exception& e) {
    return 1;
  }
  return 0;
}
```

einfacher CORBA-Client

```
#include "idl/MotionC.h"
#include "miro/Client.h"

int main(int argc, char * argv[]) {
  Miro::Client client(argc, argv);

  try {
    Miro::Motion_var motion =
            client.resolveName<Miro::Motion>("Motion");
    motion->limp();
  } catch (const CORBA::Exception& e) {
    return 1;
  }
  return 0;
}
```

einfacher CORBA-Client

```
#include "idl/MotionC.h"
#include "miro/Client.h"

int main(int argc, char * argv[]) {
  Miro::Client client(argc, argv);

  try {
    Miro::Motion_var motion =
            client.resolveName<Miro::Motion>("Motion");
    motion->limp();
  } catch (const CORBA::Exception& e) {
    return 1;
  }
  return 0;
}
```

ACE / TAO

ACE = The ADAPTIVE Communication Environment

- objektorientiertes Framework zur betriebssystemunabhängigen Interprozess-, Netzwerk- und Echtzeitkommunikation
- C++ (kompilierbar mit gängigen C++-Compilern wie GCC, MS Visual C++, Borland C++)
- für viele Betriebssystem verfügbar (Linux, Windows, BSD, ...)

TAO = The ACE ORB

- betriebssystemunabhängige CORBA-Implementation
- verwendet nur ACE
- Language Mapping für C++
- sehr standardtreu, d.h. problemlose Kommunikation mit anderen CORBA-Implementationen

ACE / TAO

ACE = The ADAPTIVE Communication Environment

- objektorientiertes Framework zur betriebssystemunabhängigen Interprozess-, Netzwerk- und Echtzeitkommunikation
- C++ (kompilierbar mit gängigen C++-Compilern wie GCC, MS Visual C++, Borland C++)
- für viele Betriebssystem verfügbar (Linux, Windows, BSD, ...)

TAO = The ACE ORB

- betriebssystemunabhängige CORBA-Implementation
- verwendet nur ACE
- Language Mapping für C++
- sehr standardtreu, d.h. problemlose Kommunikation mit anderen CORBA-Implementationen

Miro - Interface Samples

- Motion: `setTargetVelocity(translation, rotation)`
- Odometry:
  `getPosition()` returns current coordinates and heading
  `getVelocity()` returns current rotational / translational velocity
- RangeSensor:
  `getScanDescription()` returns the layout of the range sensor
  `getFullScan()` returns a vector of sensor readings
- Battery
- Video, Video Filter, Video Broker

Kernkomponenten von Miro

- Basisimplementationen der spezifizierten Schnittstellen
- Implementationen für spezielle Roboterarchitekturen:
  - RWI B21
  - ActivMedia Pioneer
  - Sparrow (entwickelt an der Universität Ulm)
- Parameterverwaltung von roboterspezifischen Konstanten über XML-Dateien
- Behavior Engine
- viele kleine und einfache Clientprogramme zum Testen der Schnittstellen und zur Visualisierung von Sensorwerten

Vorteile von Miro

alle Vorteile von CORBA / TAO / ACE

- Verteilbarkeit (Server und Clienten können auf verschiedenen Rechnern im Netzwerk laufen, aber auch mehrere Clienten können gleichzeit auf ein Serverobjekt zugreifen)
- Portabilität 1: relativ betriebssystemunabhängig

weiterhin

- Portabilität 2: leicht auf neue Roboterarchitekturen portierbar
- sehr mächtiges und flexibles Framework
- offen für Erweiterungen

Vorteile von Miro

alle Vorteile von CORBA / TAO / ACE

- Verteilbarkeit (Server und Clienten können auf verschiedenen Rechnern im Netzwerk laufen, aber auch mehrere Clienten können gleichzeit auf ein Serverobjekt zugreifen)
- Portabilität 1: relativ betriebssystemunabhängig

weiterhin

- Portabilität 2: leicht auf neue Roboterarchitekturen portierbar
- sehr mächtiges und flexibles Framework
- offen für Erweiterungen

Nachteile von Miro

- sehr mächtiges und flexibles Framework,
  wodurch es einen hohen Einarbeitungsaufwand gibt
- Overhead durch CORBA