10 July 2002

RemoteDCL

A component-based environment to carry out experiments with mobile robots over the Internet

HASSO-PLATTNER-INSTITUT
für Softwaresystemtechnik

Teammembers

- **Human**
  - Helge Issel
  - Kai Köhne
  - Michael Richter
  - Stefan Henze
  - Kai Müller

- **Artificial**
  - Hunter
  - Runner

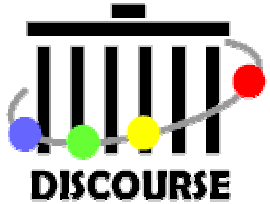
Agenda

- **Introduction**
  - Involved projects
  - Aim of the project
  - Existing Tools
- **Overview**
  - The whole system
  - The three main components
- **Interesting Parts**
  - Configuration
  - Compiler
  - LNPoI
  - Tracker
- **Testing**
- **Other platforms**
- **Living example**

Involved Projects

- **DCL**
  - ***D**istributed **C**ontrol **L**ab*
  - part of the professorship „Betriebssysteme und Middleware“ / HPI
  - aim is the research of software paradigm and design patterns allowing a connection between middleware-based components and embedded (mobile) systems
  - experiments with component software and mobile devices

- **DISCOURSE**
  - ***Di**stributed & **Co**llaborative **U**niversity **R**esearch & **S**tudy **E**nvironment*
  - “a distributed laboratory for distributed computing featuring advanced middleware technology”
  - FU, TU, HU Berlin, HPI are involved
  - a testbed for research and a reference platform for teaching
  - usage of Microsoft .NET technology



Aim of the Project

- **To develop a component-based environment which allows for creating experiments easily**
  - experiments involve to control and communicate with mobile devices
- **further requirements**
  - The components should be simple to replace
  - new experiments should require minimal changes
  - existing tools & programs should be integrated
  - the primary framework should be Microsoft .NET

Existing Tools

- **RCX**
  - of the "Lego Mindstorms Robotics Invention System" in-a-box robot-kit
  - a productized version of the MIT Programmable Brick
  - uses the Hitachi H8/3292 microcontroler
  - RAM size 512 K / speed 16Mhz@5V
  - supports Infrared communication

- **legOS**
  - a free (primitive) OS for he RCX
  - consists of a kernel as a replacement for the original firmware and several utilities (f.ex. to download new programs)
  - user tasks are executed as native code
  - supported language is C and C++
  - user programs are compiled with GCC on the PC and downloaded as binaries to the RCX
  - Remote Control Patch: kernel extension to support downloading, starting and stopping programs remotely

Existing Tools II

- **Cygwin**
  - provides a UNIX environment under Windows
  - consists of
    - library to emulate UNIX syscalls (cygwin1.dll)
    - various UNIX programs (bash, gcc …)
  - needed by legOS to compile & download programs under Windows

- **LNPol**
  - ***L**ego **N**etwork **P**rotocol **o**ver **I**nternet*
  - a router for LNP (Lego Network Protocol) messages
    - LNP messages consist of a header used for addressing and a body
    - equivalent to UDP packages
  - communicates
    - with programs on the PC over TCP/IP
    - with the RCX over an Infrared Tower

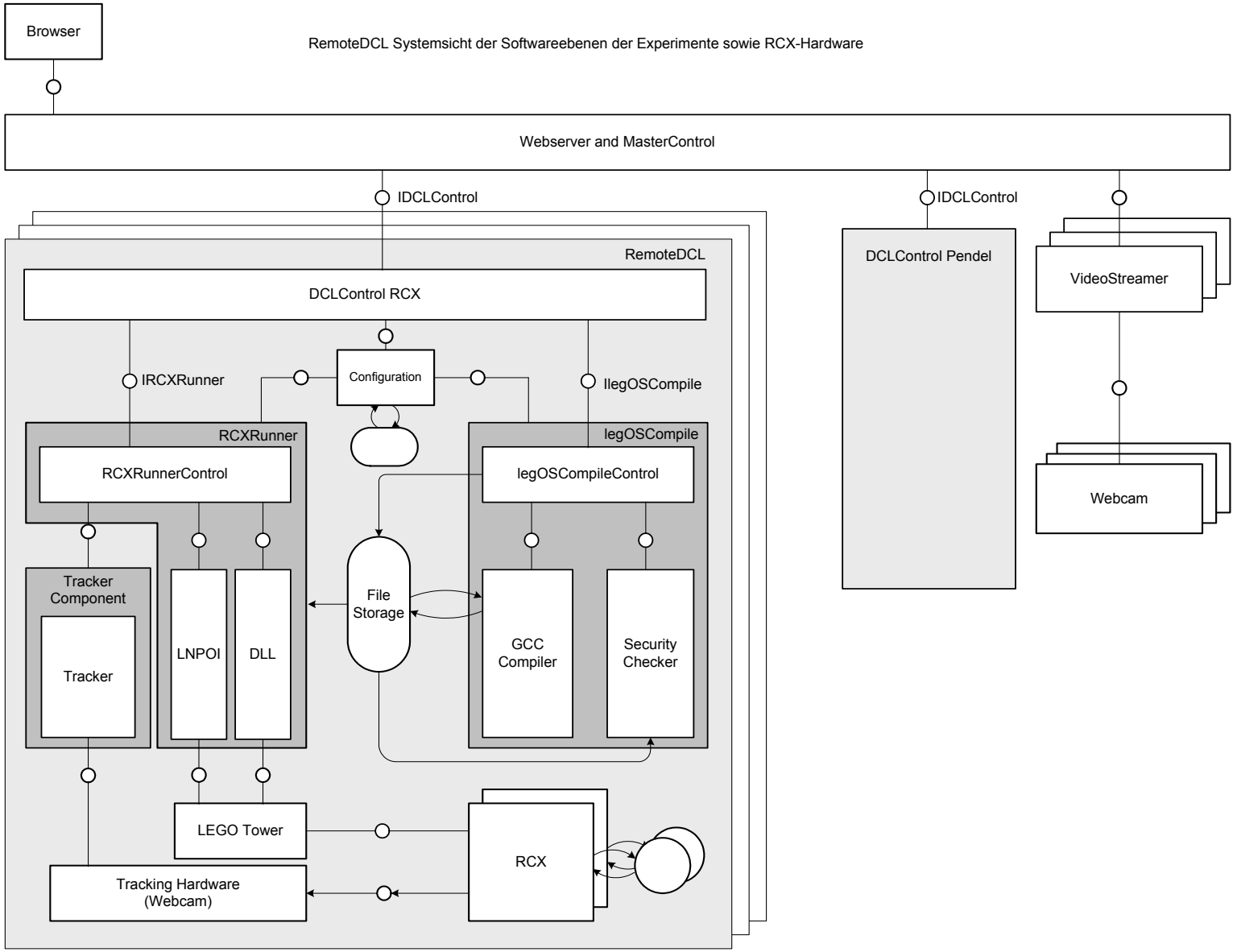
The Web-Interface

- **was already designed for another DCL experiment (a programmable pendulum)**
- **consists of**
  - various Active Server Pages
  - a controller for registering new experiments
- **handles user authentication & authorisation**
- **Interface for an experiment to implement:**

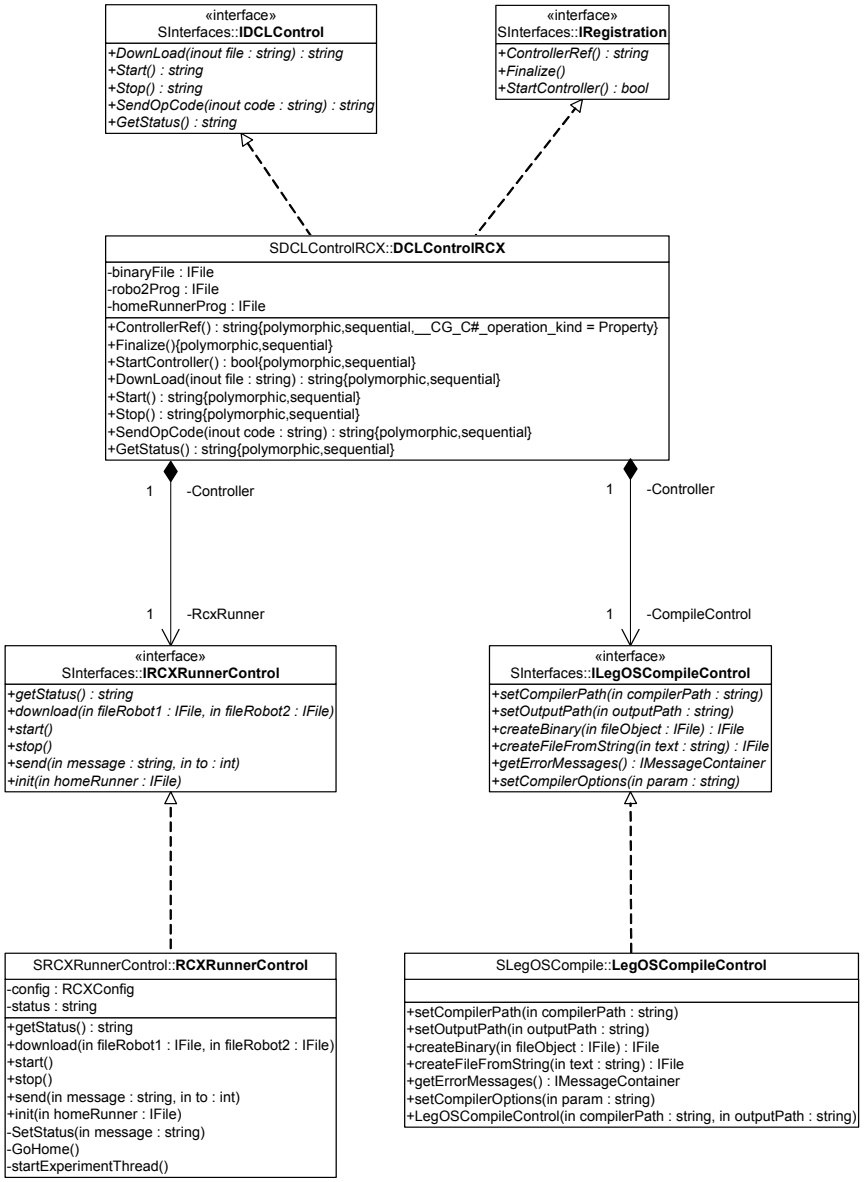
Agenda

- **Introduction**
  - Involved projects
  - Aim of the project
  - Existing Tools
- **Overview**
  - The whole system
  - The three main components
- **Interesting Parts**
  - Configuration
  - Compiler
  - LNPoI
  - Tracker
- **Testing**
- **Other platforms**
- **Living example**

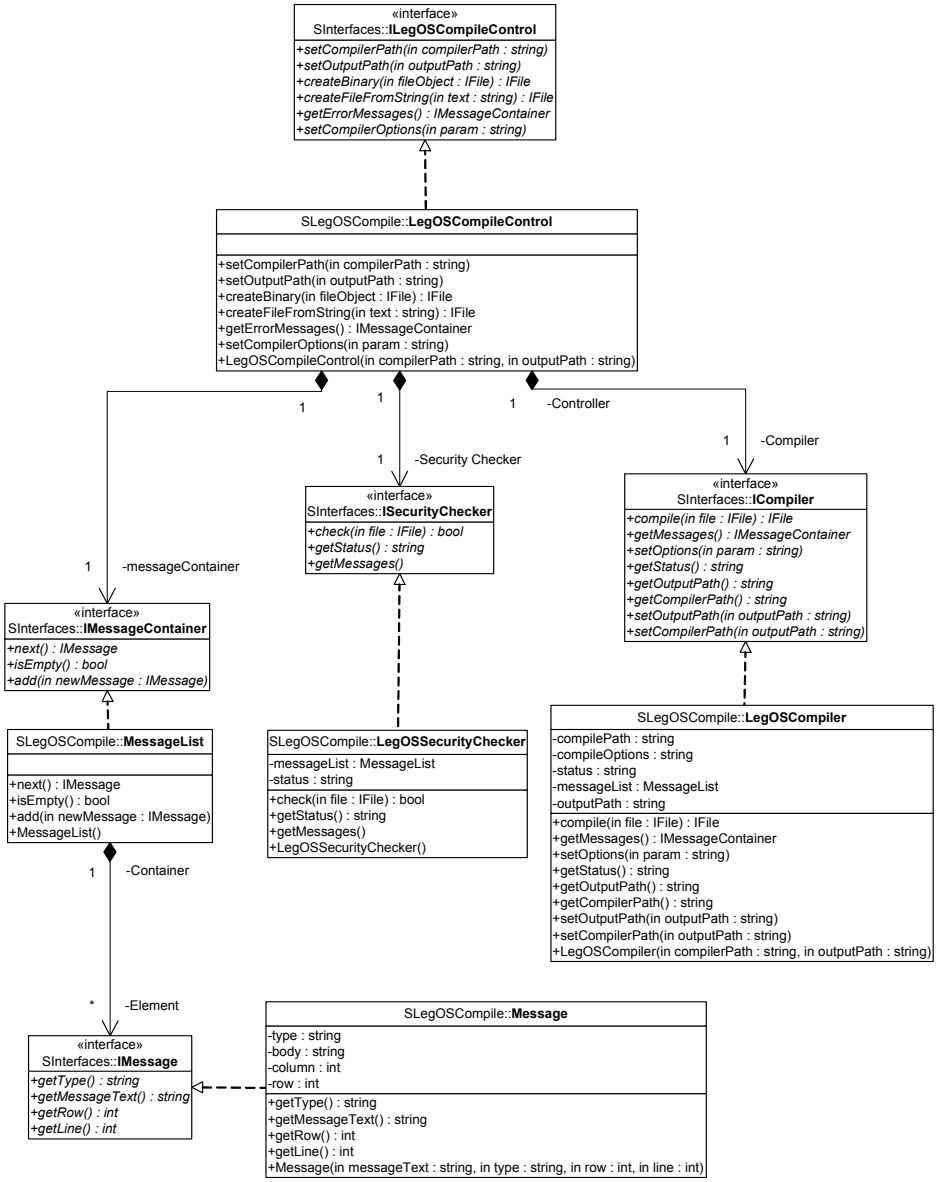
Overview


Browser
RemoteDCL Systemsicht der Softwareebenen der Experimente sowie RCX-Hardware
Webserver and MasterControl
IDCLControl
IDCLControl
RemoteDCL
DCLControl RCX
DCLControl Pendel
VideoStreamer
IRCXRunner
Configuration
IlegOSCompile
RCXRunner
legOSCompile
RCXRunnerControl
legOSCompileControl
Webcam
Tracker Component
Tracker
LNPOI
DLL
File Storage
GCC Compiler
Security Checker
LEGO Tower
RCX
Tracking Hardware (Webcam)

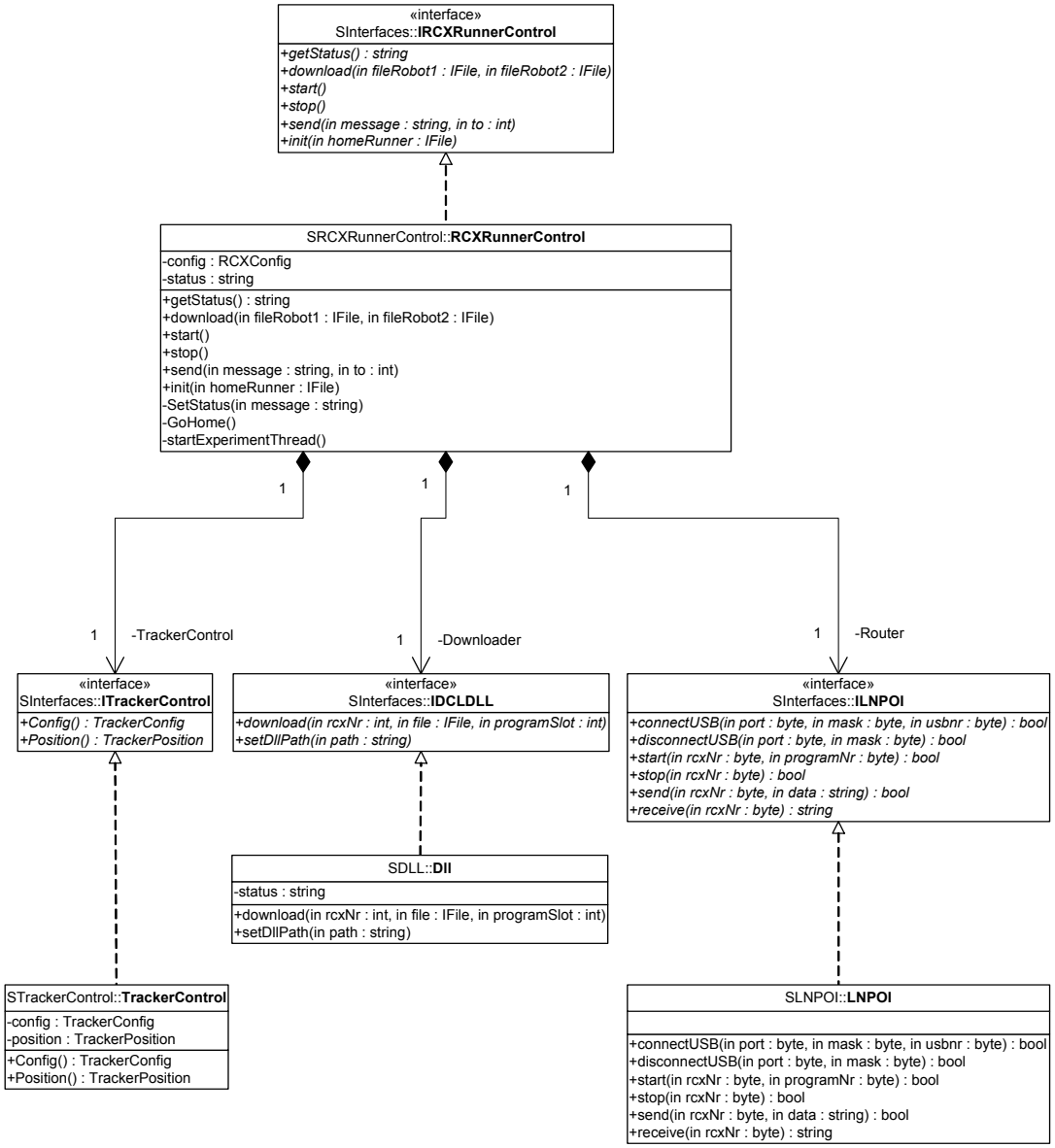
The Experiment Controller



The Compiler



The RCXRunner



Agenda

- **Introduction**
  - Involved projects
  - Aim of the project
  - Existing Tools
- **Overview**
  - The whole system
  - The three main components
- **Interesting Parts**
  - Configuration
  - Compiler
  - LNPoI
  - Tracker
- **Testing**
- **Other platforms**
- **Living example**

Configuration - Concept

- **Problem**
  - interfaces abstract from implementation, nevertheless often implementation-specific configuration data needed by implementing class

- **Idea:**
  - a single object which encapsulates all configuration data reduces coupling between components
  - configuration may be stored in an configuration file or generated at runtime
  - allows design of small, „clean“ interfaces which concentrate on functional requirements
  - as default: data should be stored in XML-files

Configuration - Implementation

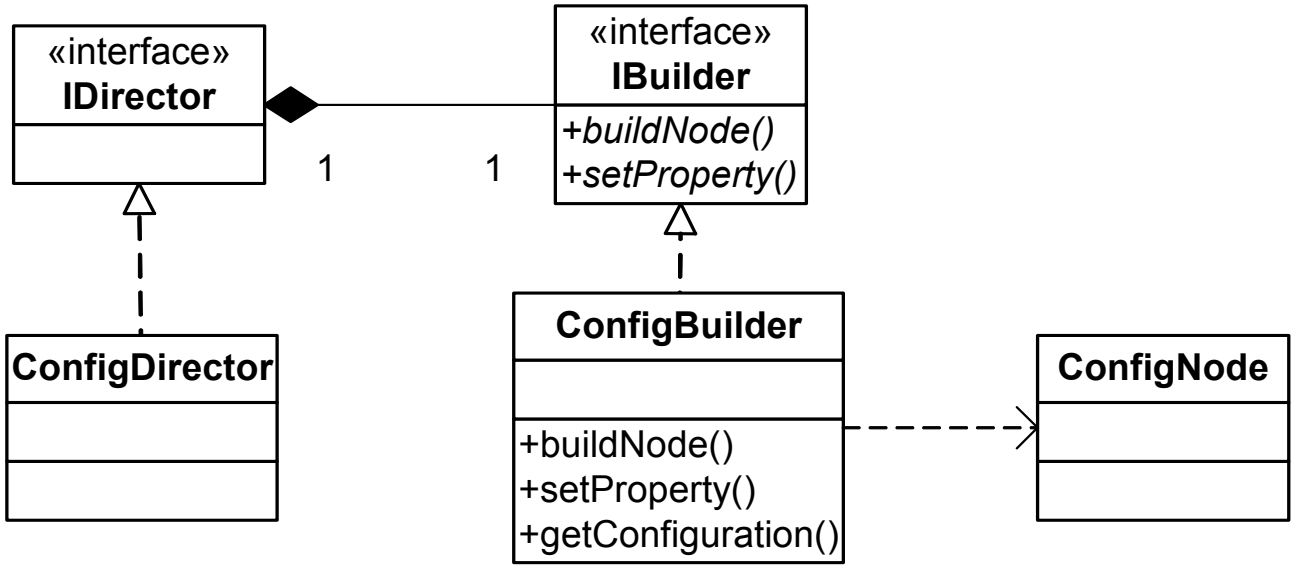
Realisation

- configuration may only be set once
- config-objects builds in-memory lightweight object-tree representing current configuration
- components may only access well-known sub trees, hence component data clearly separated
- each component gets a private copy of the (sub-) tree → data tamper-proof

Implementation

- uses „Builder-Pattern“
- configuration object calls a „Config-Builder“ object, configured with an „XML-Director“
- parsing of XML-file done by „Director“, tree build by „Config-Builder“ → allows (not implemented) for different file formats

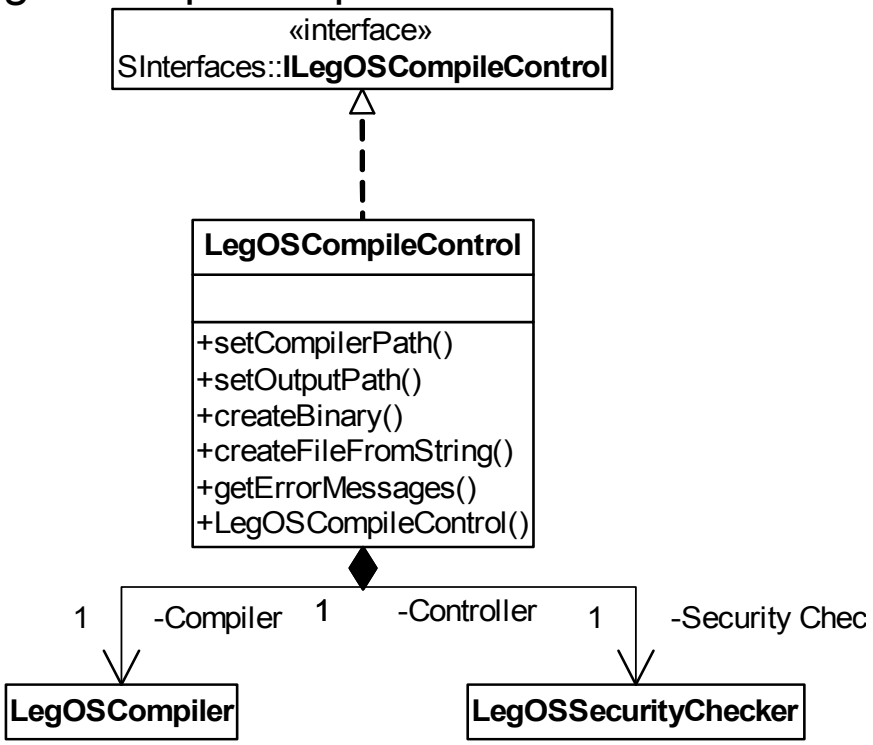
Configuration – Builder-Pattern



Compiler - Concept

Concept

- definition of interface to allow easy substitution with different compilers
- classes implement a compiler or encapsulate access to an external compiler running in a separate process



Factory method

- generation of source and binary files

Compiler - Implementation

- **using GCC**
  - RCX-native file loaded as kernel extension, therefore multiple creational steps necessary
    - compiling, relocation text segent, linking against kernel symbol table …
  - using of make → requires using bash-script

- **Facade**
  - our „compiler“ is an facade for class that actual encapsulates „gcc“ and a „security checker“ class

- **Why a Security Checker?**
  - using C as programming language is risky: some programmers may try to overwrite control routines on robots
  - to provide (at least) some safeguard mechanisms: scanning code for potentially harmful constructs
  - relies on regular expressions

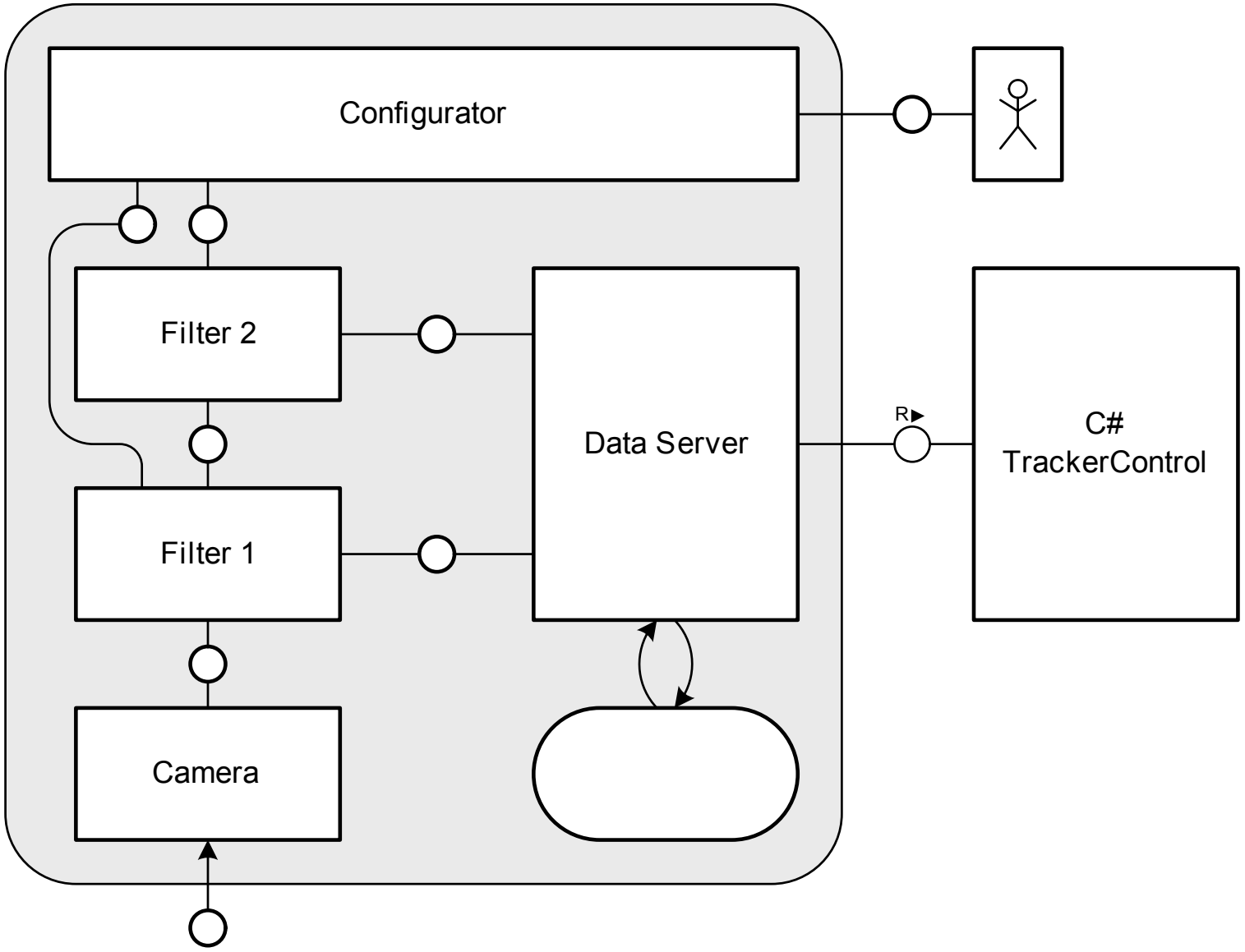
LNPol

- **LNPol → transport protocol used to communicate with the robots**
- **Situation:**
  - Sourcecode for handling LNPol available
- **Problem:**
  - Available only in C++
- **Solution:**
  - Wrapping code within a managed C++ class
- **The .Net managed C++ Extensions**
  - integration of C++ into .Net
  - class / function wrapping
  - data marshaling
- **Design pattern implemented: Wrapper**

Tracking Environment

- **How to know where we are?**
  - many experiments need the knowledge of position
- **RCX-Robots running in a flat world**
  - the world is a disc (a table is nearly a disc)
  - sudden death must be prevented (they must not fall down from the table)
- **World can be observed by a camera**
  - track the position of a robot in an image
  - calculate the Robot's position in the world
- **tracking data must be submitted**
  - to the experiment environment
  - and the Robots

Components working together



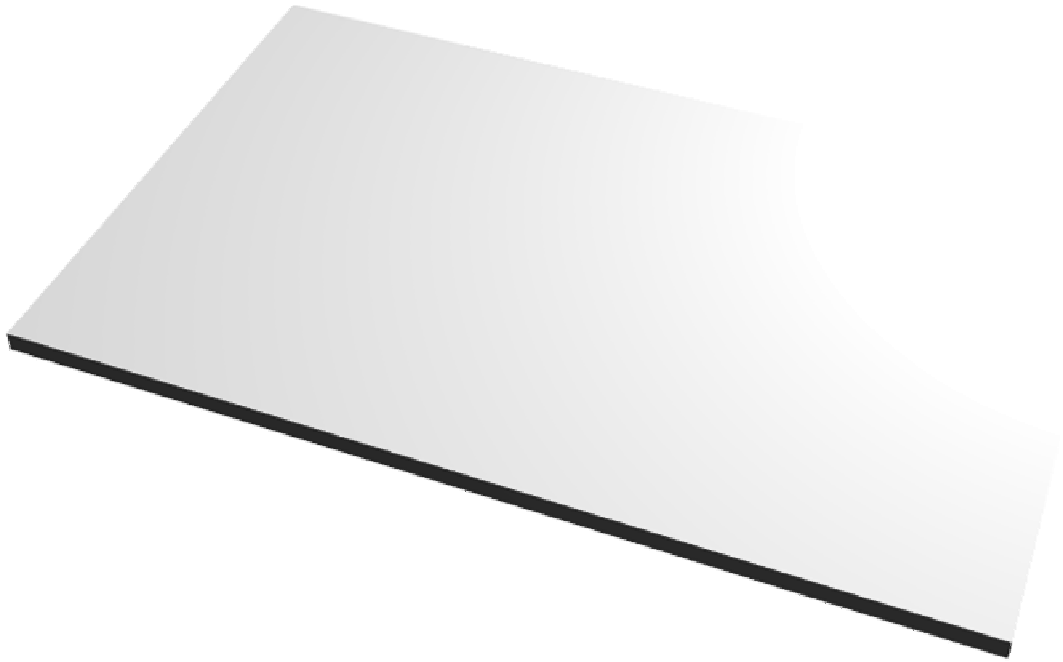
CamShift Filter

- **COM-Component**
- **DirectX**
  - DirectMedia used to get video stream from the camera
  - output on the screen
- **Filter Chains**
  - Configurator builds up filter chains to apply CamShift-Filter to the video
- **CamShift-Filter**
  - area selected by user
  - filter creates a histogram with matching colors for that area
  - based on this it looks for a rectangle filled with these colors
  - by moving the colored area the filter recognizes changes and calculates a new rectangle
  - points within this rectangle are submitted to a data server
- **integration**
  - we integrated the whole data-capturing to get coordinates of the tracked data out of the filter

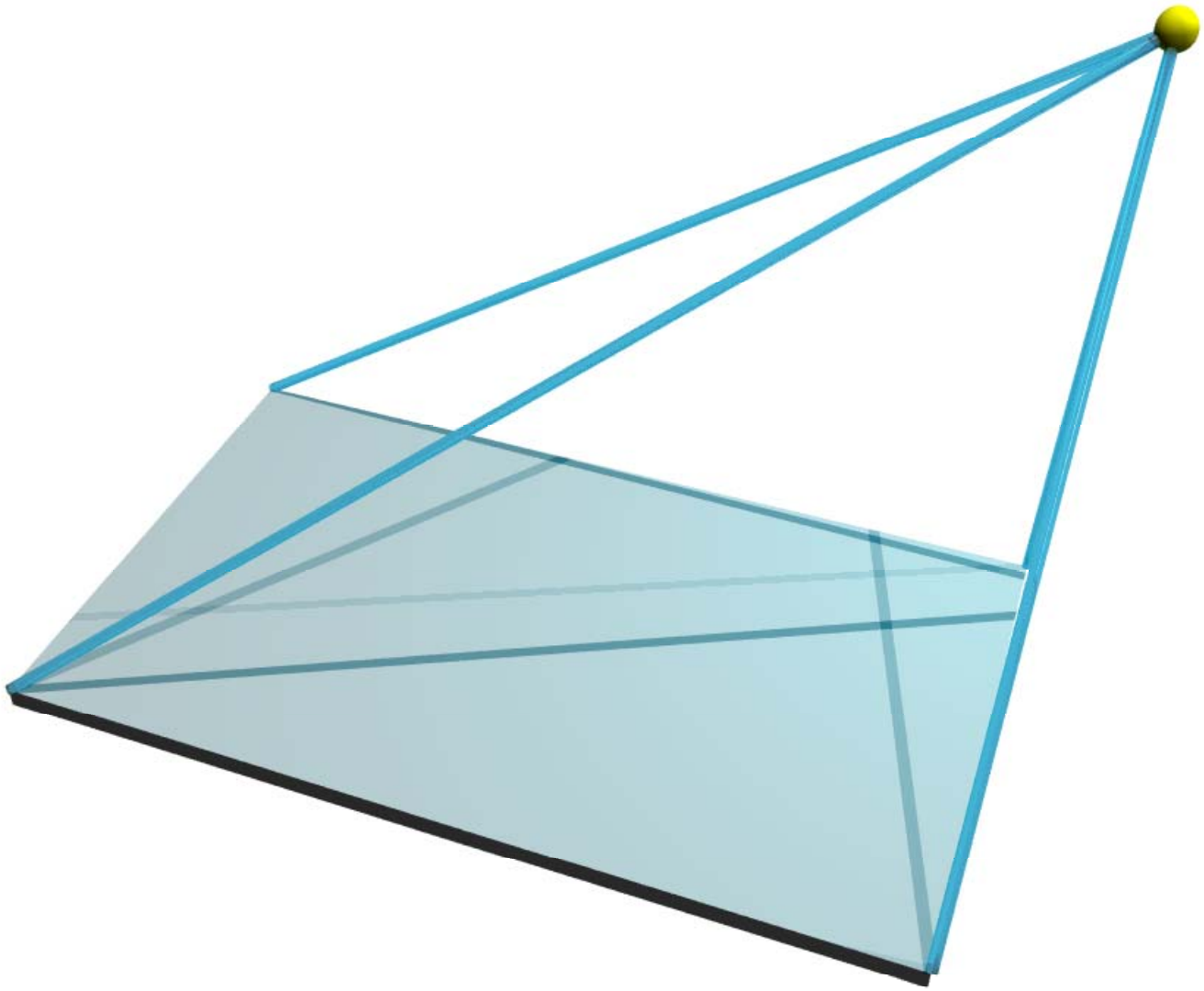
Tracking Data Server

- **COM-Object**
  - singleton
  - we have one component that holds all tracked data
  - filters register themselves to that object
- **integrating COM in C# and .NET Framework**
  - see Lecture
- **how it works**
  - filters send their data to the server every time they track a position
    - very often
  - experiment pulls the data from the server whenever it wants to
    - sometimes (about 2 times a second)
    - may be configured by experiment-configuration
- **the big deal**
  - we could use any data server that matches the interface
  - might be changed even at runtime

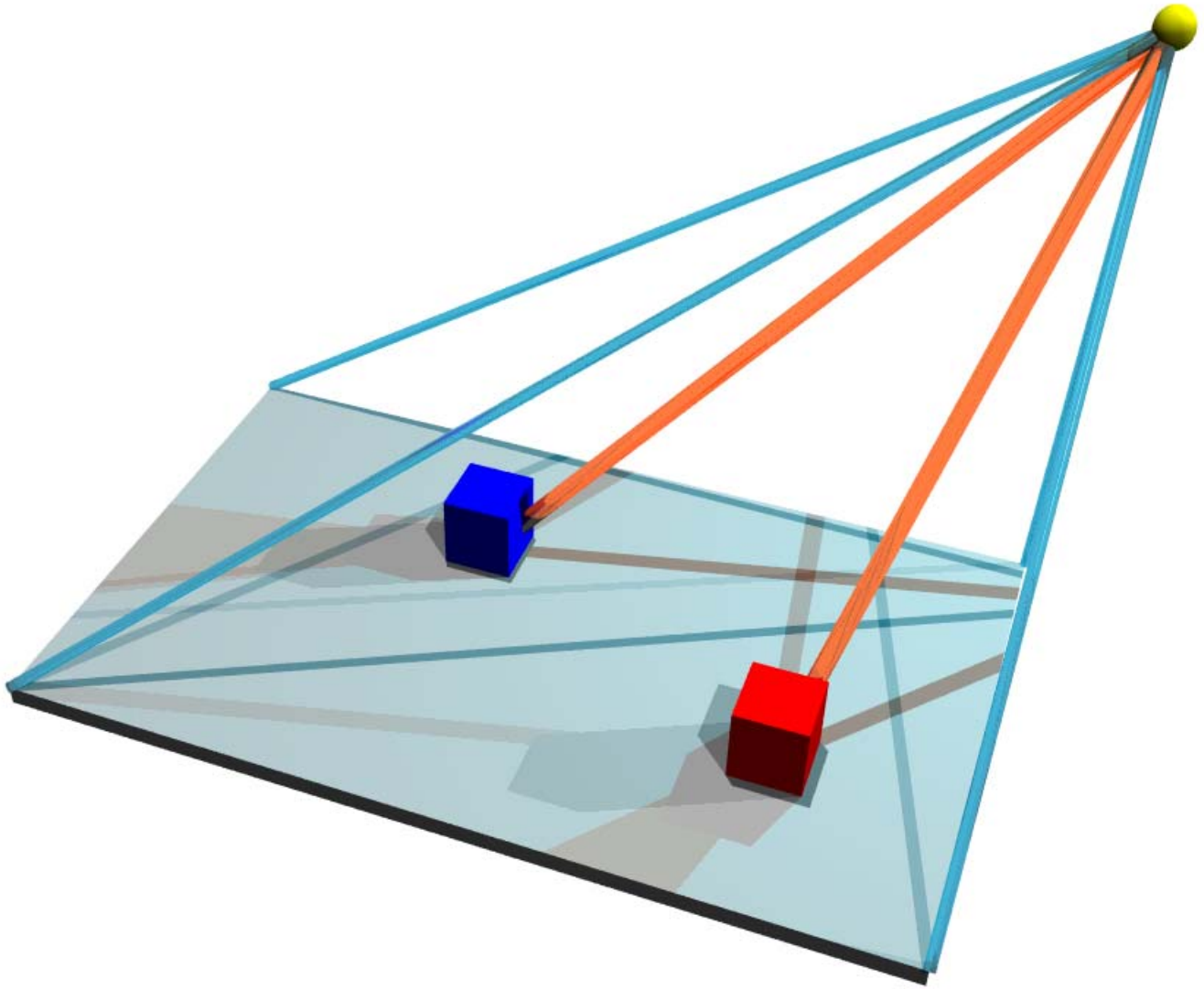
Tracking Theory



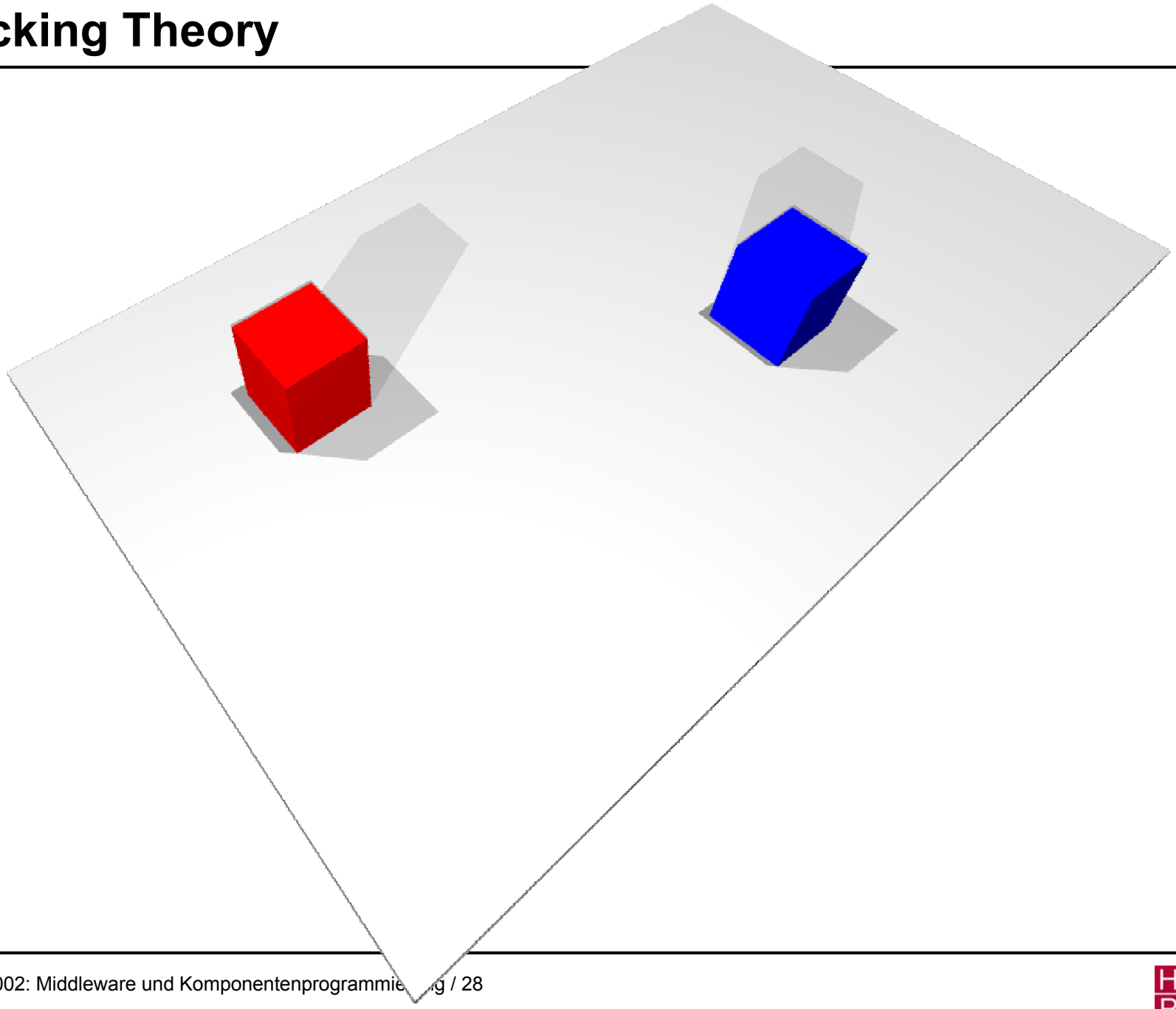
Tracking Theory



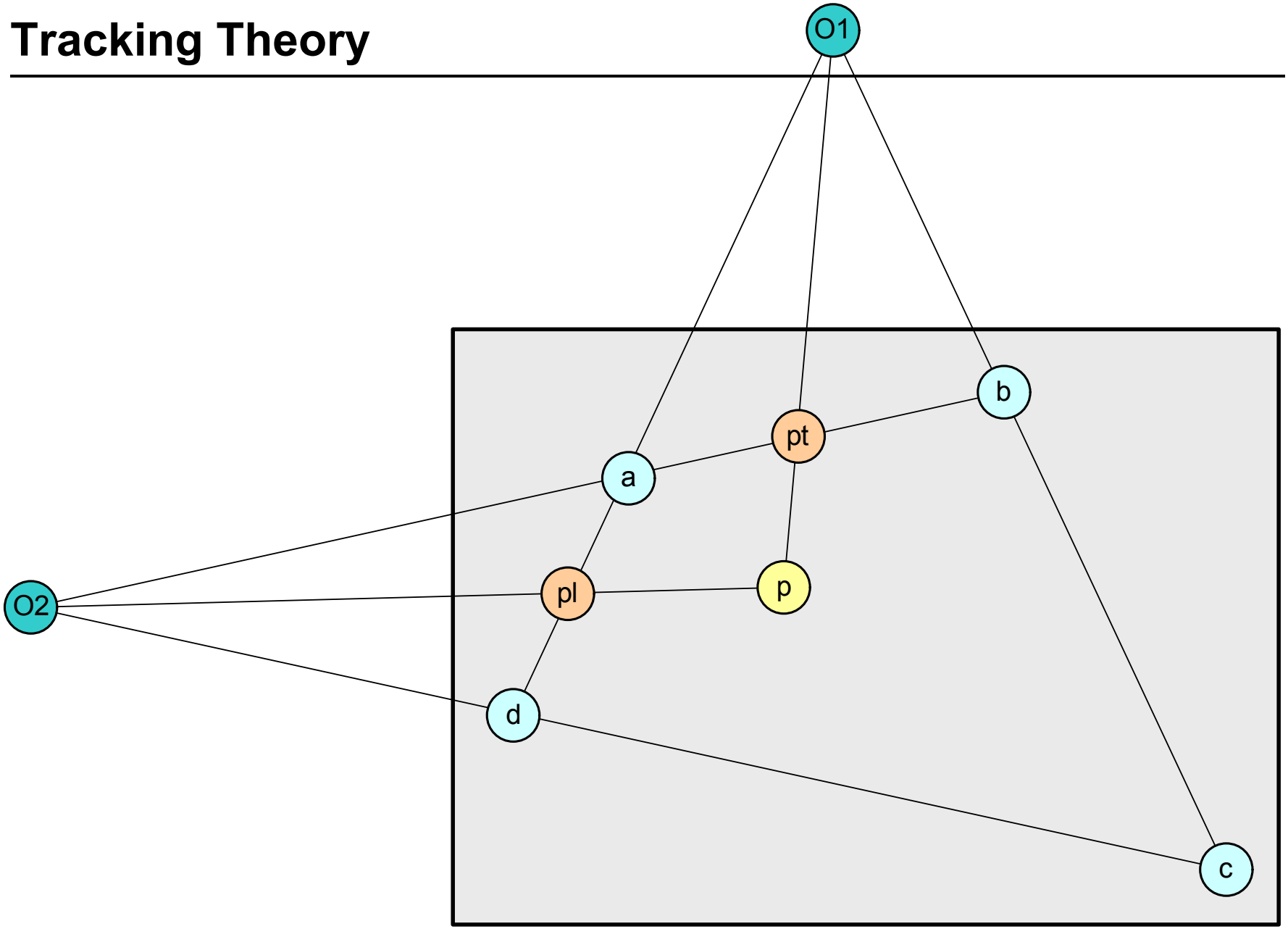
Tracking Theory



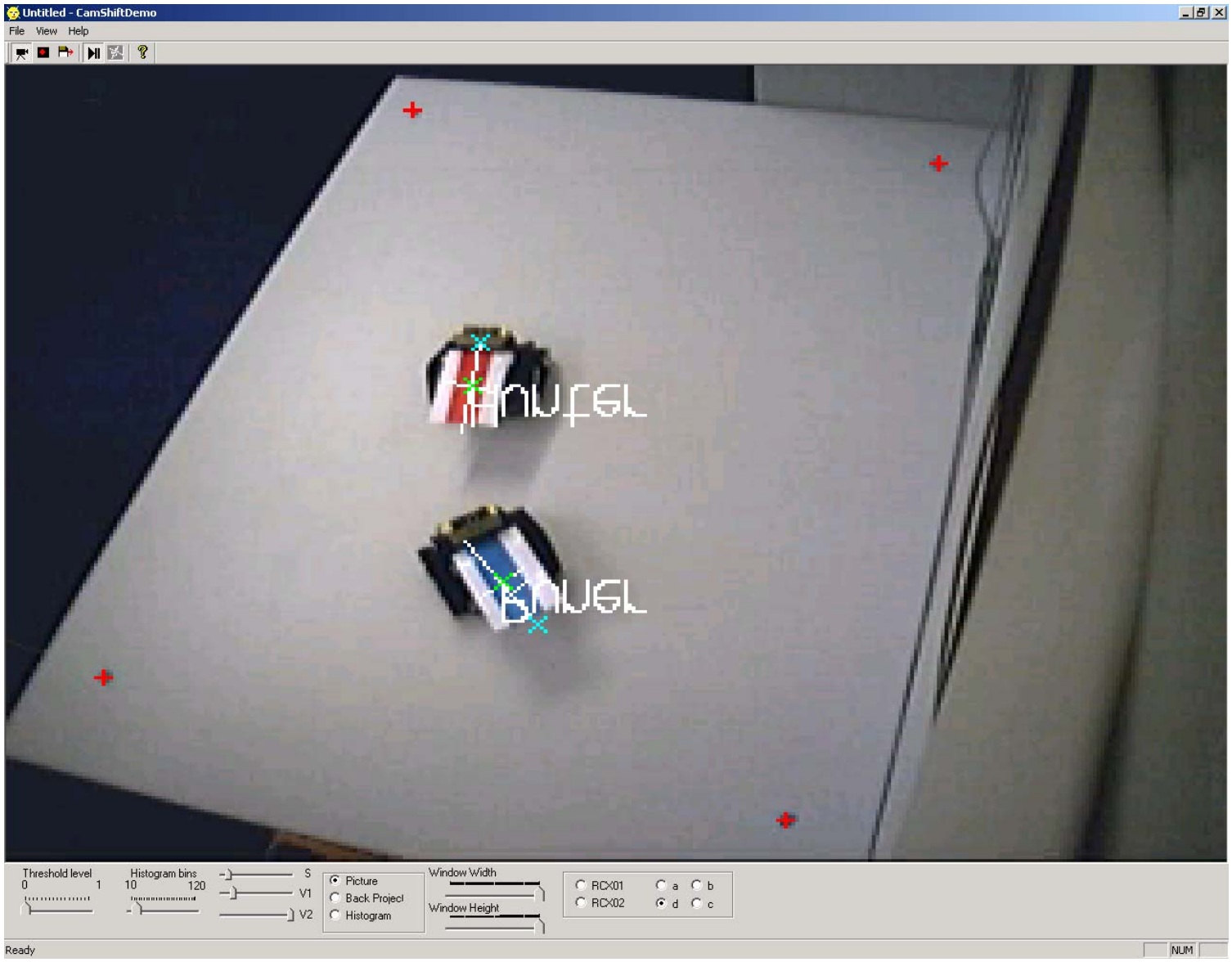
Tracking Theory



Tracking Theory



Configurator



Agenda

- **Introduction**
  - Involved projects
  - Aim of the project
  - Existing Tools
- **Overview**
  - The whole system
  - The three main components
- **Interesting Parts**
  - Configuration
  - Compiler
  - LNPoI
  - Tracker
- **Testing**
- **Other platforms**
- **Living example**

Testing

- **Testing-Environment for all components**
  - all self-written components but controllers have their own test-classes
    - each step during the experiment (compiling, downloading, tracking…) can be tested separately
  - controllers can use dummy-components for some components
    - creating a well defined world by simulating user, and robots
    - may be heavily improved
- **Log4NET**
  - thanks for org.apache.log4j
  - easy logging with several levels

Agenda

- **Introduction**
  - Involved projects
  - Aim of the project
  - Existing Tools
- **Overview**
  - The whole system
  - The three main components
- **Interesting Parts**
  - Configuration
  - Compiler
  - LNPoI
  - Tracker
- **Testing**
- **Other platforms**
- **Living example**

Other platforms?

- **Key features important for our project**
  - remoting
  - interoperability with native code
  - interoperability with COM-components
- **CORBA**
  - CORBA addresses only remoting, therefore it is not suited
- **Java**
  - very powerful all-purpose language / component software
  - discussed in more detail on following slides
- **COM**
  - is a mess

Java – Remoting / Processes

- **Java allows for Java-specific remoting (Java RMI)**
  - powerful, but usage more complicated
    - creation of special interface
    - creation of stub/skeleton with separate compiler (rmic)
    - rmi naming service has to be started on server
  - in contrast: C# a class has to inherit from MarshalByRef object – that‘s all
- **Java allows creating / using of CORBA-objects**
- **bridge between both worlds: RMI over IIOP**
  - makes RMI-objects „real“ CORBA-objects (language interoperability!)
- **Processes**
  - Java allows creation / destruction of processes as well as communication with them (via redirected stdin / stdout / stderr)

Java – interacting with native / C /C++ code

- **Using of C-functions / C-libraries**
  - JNI (Java Native Interface) allows calling of C-functions
  - works with Dlls (under Windows)
  - requires preliminary modifications to code
    - developer needs access to source code or
    - to encapsulate existing functions within own functions

- **Interacting with C++ objects**
  - feasible: C++ using Java objects
  - as far as we know no way the other way around: C++ objects can not be referenced under Java

- **COM-interoperability**
  - we did not found any tool / library addressing that problem
  - only Microsoft‘s Java compiler dated from 1998 allowed COM-interoperation

.NET vs. Java EJBs - Performance

- **Microsoft‘s „Pet Store“**
  - Microsoft implemented Sun‘s „Pet Store“ with .NET
  - several benchmarks, latest made by VeriTest: .NET about 10 times faster
- **„NILE“-benchmark (implementation of an online-shop)**
  - 3 times faster, better scaling on multiple CPUs
  - benchmark sponsored by Microsoft
- **Conclusions drawn**
  - no widely accepted benchmark available – Pet Store benchmark had to be repeated several times due to mistakes
  - benchmark results controversial discussed in the net
  - interesting: SUN pretends .Net doesn’t exists → maybe .NET really faster
  - „platform“-performance relies heavily on underlying middleware (application server, databases, ...)

Agenda

- **Introduction**
  - Involved projects
  - Aim of the project
  - Existing Tools
- **Overview**
  - The whole system
  - The three main components
- **Interesting Parts**
  - Configuration
  - Compiler
  - LNPoI
  - Tracker
- **Testing**
- **Other platforms**
- **Living example**

... Demonstration and further perspectives ...

What‘s next

- **short term goals**
  - open the experiment to the DISCOURSE-project
  - further generalize components
  - support alternative experiment

- **long term goals**
  - develop a new, more secure scripting language
  - use handhelds supporting .NET as mobile devices (instead of directly communicating with the RCX)

Links

- http://gotdotnet.com/team/compare/
- http://www.onjava.com/pub/a/onjava/2001/11/28/catfight.html