



ARC Centre of Excellence for

Autonomous Systems

Orca: Components for Robotics

**Alexei Makarenko,
Alex Brooks,
Tobias Kaupp**

**ARC Centre of Excellence for
Autonomous Systems
The University of Sydney**

Uni. of Sydney: where we are coming from

- **Distributed applications**
 - Sensor networks, decentralized information fusion
 - Naturally leads to components
- **Industry funded projects**
 - Need for robust implementations
- **Complexity of robotic software**
 - Robotic software – the bottleneck of what is possible today (not computing hardware, sensors, algorithms)
 - “The PhD problem”

- **Advantages**
 - Modularization: “collection of small problems is easier to solve than one big problem”
 - Minimizing source-code cross-dependencies
 - Maximizing software reuse
 - Parallel development by large distributed teams

Component Model

- **Fundamentals**
 - Interface definition
 - Communication mechanism
- **Options**
 - .NET (Microsoft)
 - Enterprise Java Beans (Sun)
 - CORBA (OMG and implementers)
 - Ice (ZeroC)
 - Custom (e.g. Player, Orca-1)

- **Middleware: Off-the-shelf vs. Custom**
 - Player uses custom and is very successful
 - We believe there is a limit to what can be done with custom middleware (reliability, features, performance)
- **Off-the-shelf Middleware: CORBA vs. Ice**
 - The concept is similar if not identical
 - For comparison of design, features, performance, etc. see ZeroC.
 - We've used Ace/Tao and found it difficult
 - We find Ice much more straightforward

Ice – Internet Communication Engine

- **Background**

- Company: ZeroC, since 2001
- <http://www.zeroc.com>
- Open-source, proprietary

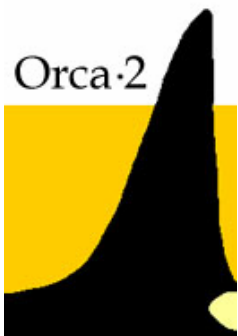


- **Support**

- OS: Linux, Win, MacOS X, Solaris
- Language: C++, Java, C#, Visual Basic, Python, PHP
- Transport: TCP/IP, UDP/IP, SSL

- **Licensing**

- GPL for open-source projects
- LGPL for Orca
- Commercial license otherwise

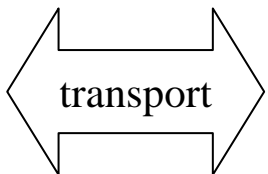
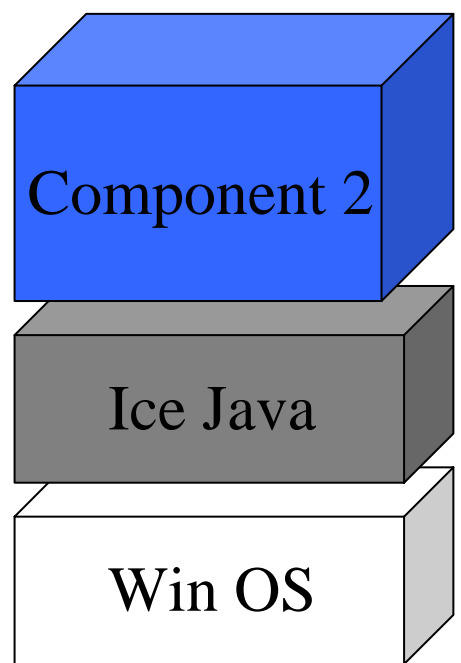
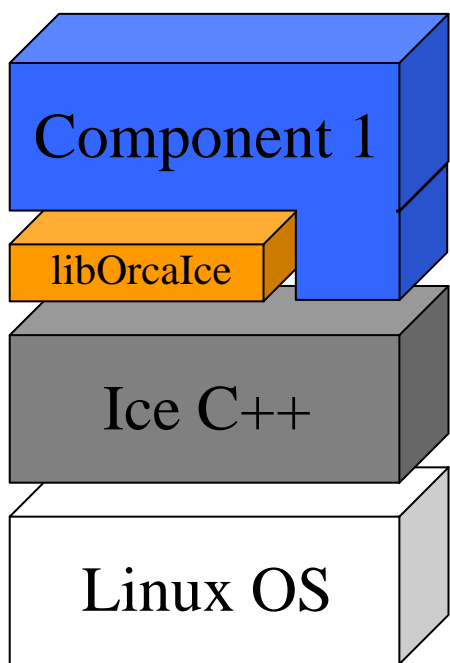
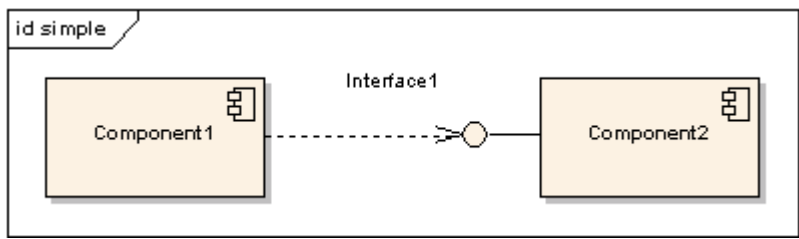


- **Main objective: Software Reuse**
 - *Enable*: by defining a set commonly-used interfaces
 - *Simplify*: by providing libraries with a high-level convenient API
 - *Encourage*: by maintaining a repository of components
- **What makes it different?**
 - adopts a Component-Based Software Engineering approach
 - uses an industrial-strength library (Ice) for communication and interface definition
 - aims to be general, flexible and extensible
 - provides optional tools to assist in the development of individual components and the management of large systems
 - maintains a repository of free, reusable components

“An object-oriented middleware platform”

- **Developer**
 - Defines interfaces in an Interface Definition Language (IDL)
 - A bunch of remote procedure calls
 - Writes server application
 - Implements RPC's
 - Writes client application
 - Calls RPC's
- **Ice**
 - Takes care of communication, serialization, etc.

Software Architecture

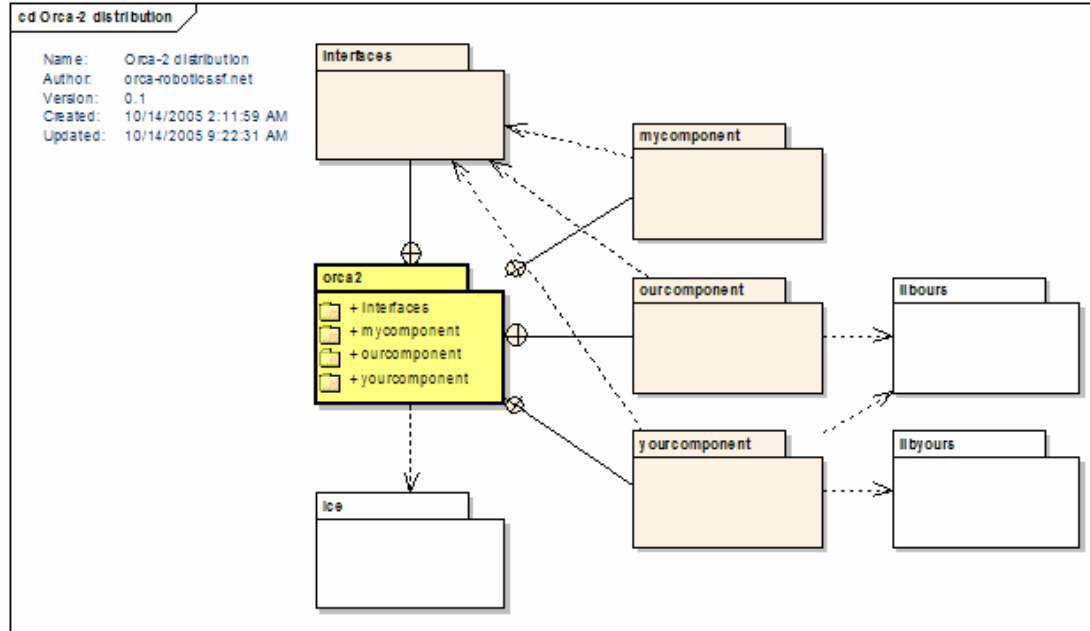


e.g. TCP/IP

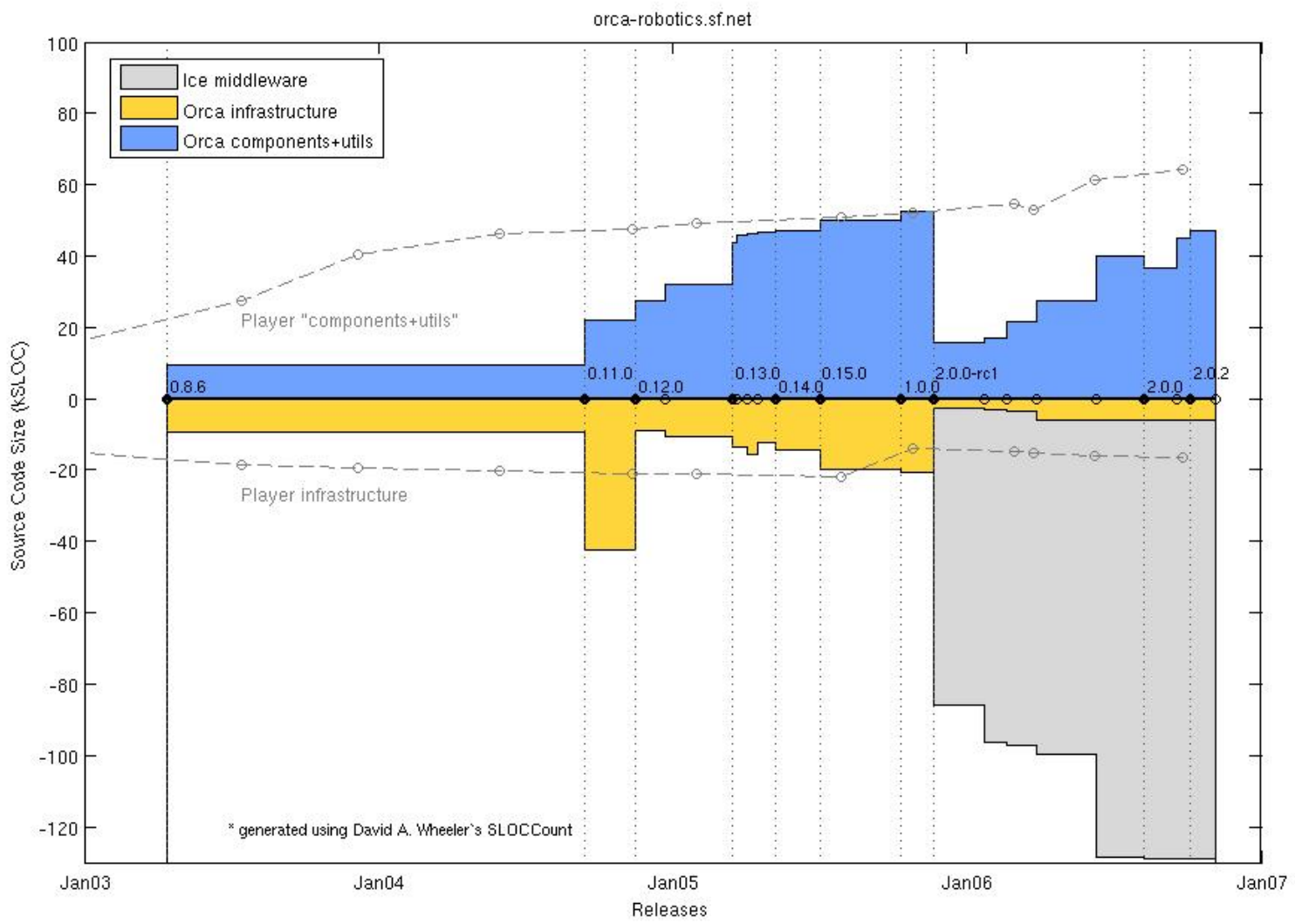
Ice Services

- **IceGrid – grid computing**
 - Registry (centralized)
 - On-demand activation, reactivation
 - Replication, etc.
- **IceBox – application server**
 - Internal comms are optimized to a function call (optional)
- **IceStorm – event service (an IceBox service)**
 - Publish/subscribe of any interface
 - Federation
- **IcePatch2 – file distribution**
- **Freeze – persistence service**
- **Glacier2 – firewall**
- **Ice-E – “embedded Ice”**
 - Light-weight version for PDA’s.

- **Hosted on SourceForge**
- **Distribution includes**
 - Common interface definitions (Slice IDL)
 - Utility libraries
 - Component repository



Project History and Size



- **Slice: an Interface Definition Language (IDL)**

```
// Slice

class Position2dData extends OrcaObject
{
    Frame2d pose;
    Twist2d motion;
};

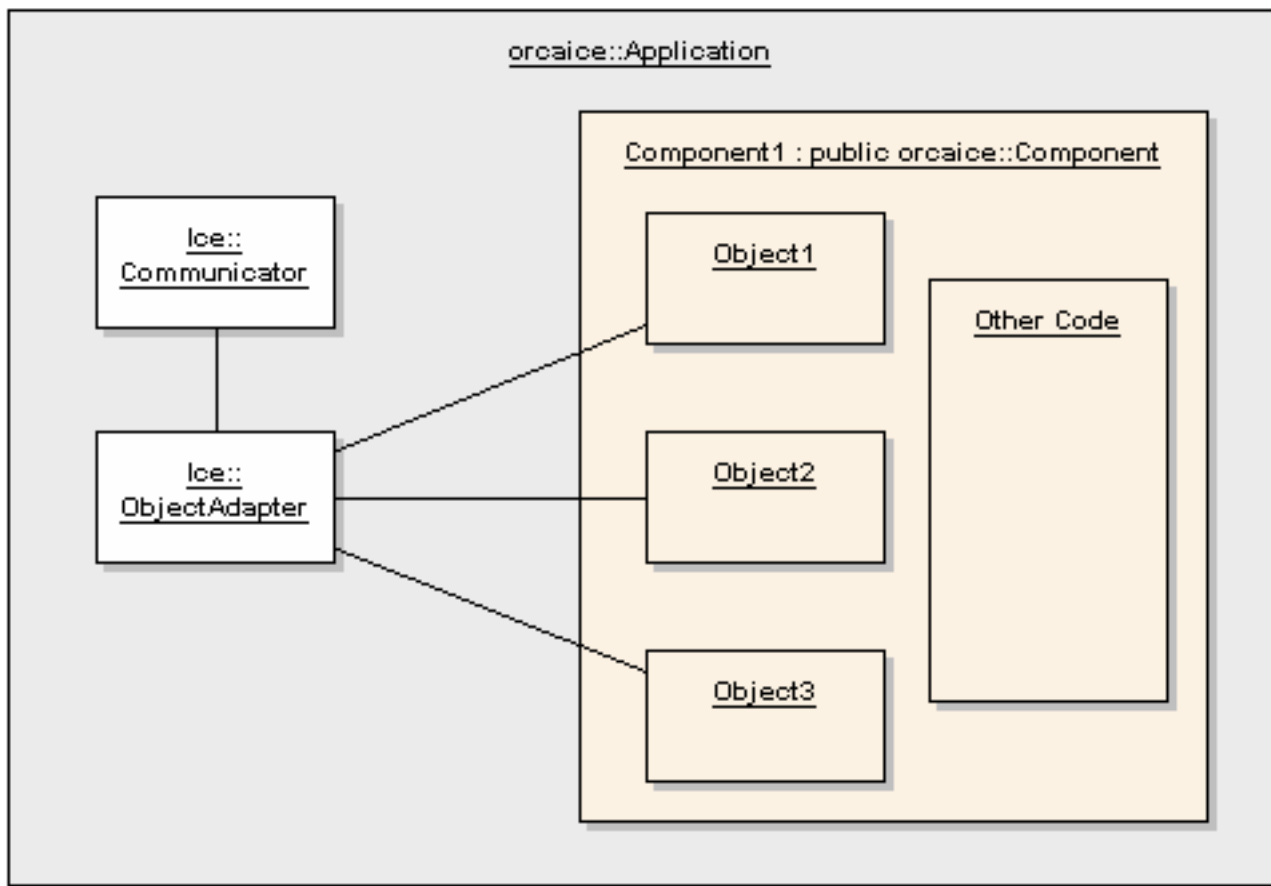
interface Position2d
{
    nonmutating Position2dData getData()
        throws DataNotExistException, HardwareFailedException;

    void subscribe( Position2dConsumer* subscriber )
        throws SubscriptionFailedException;

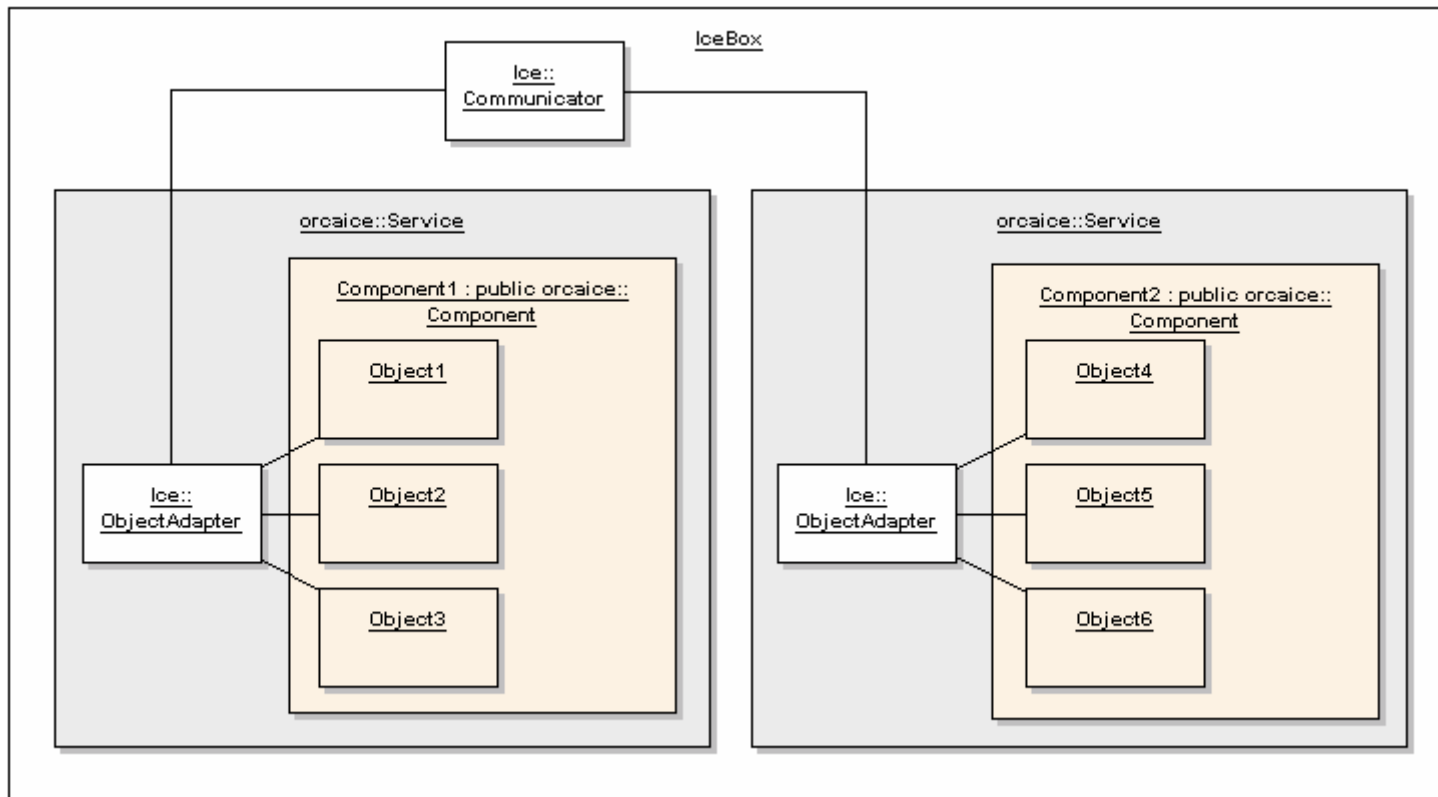
    idempotent void unsubscribe( Position2dConsumer* subscriber );
};
```

- **Aims**
 - To set naming conventions
 - To simplify component development
- **Provides**
 - Classes to derive from
 - Classes to use
 - Helper functions
- **Strictly optional**
 - Can choose not to use it and call `libIce` directly

Stand-Alone Component



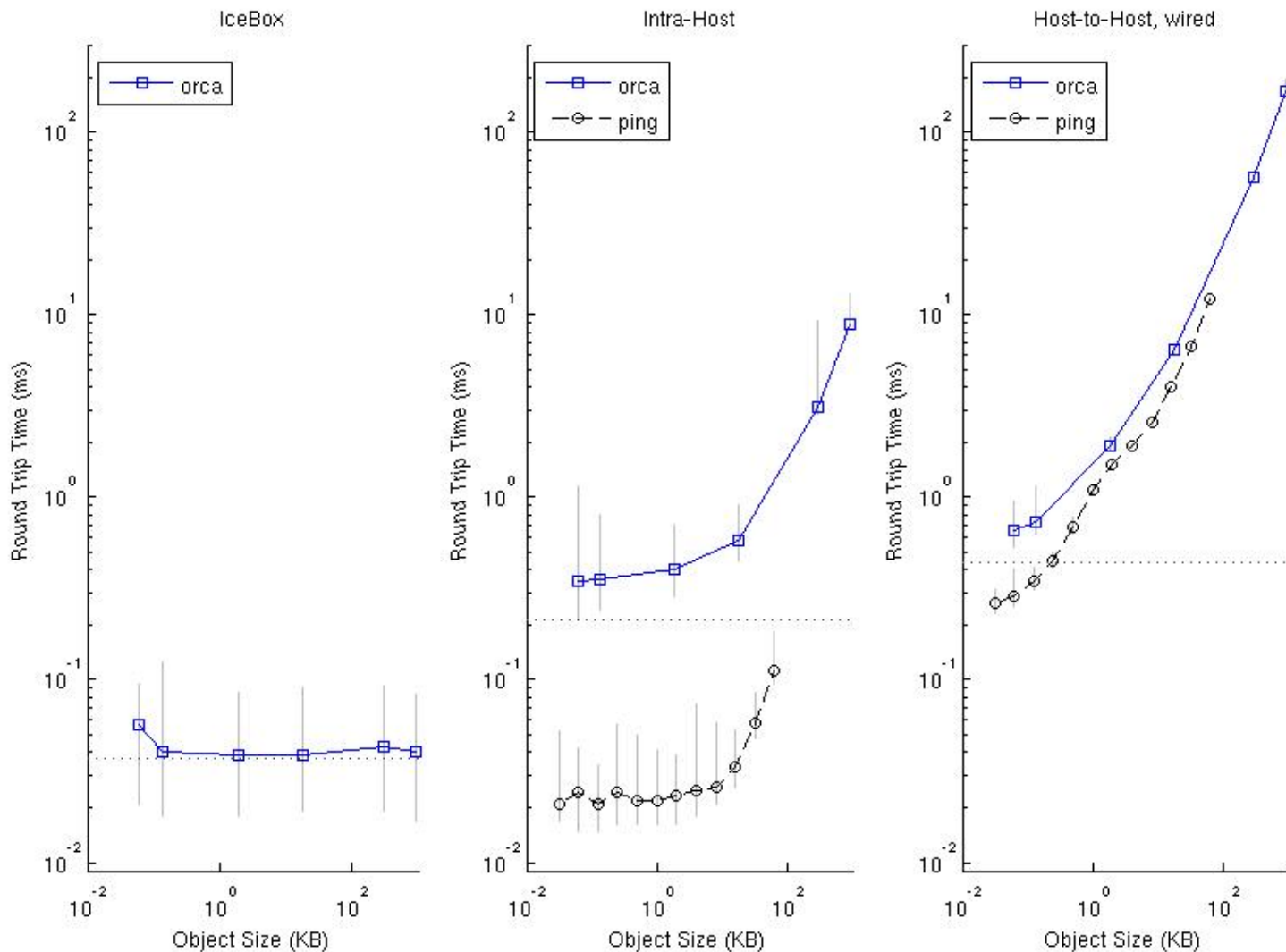
Component as an IceBox Service



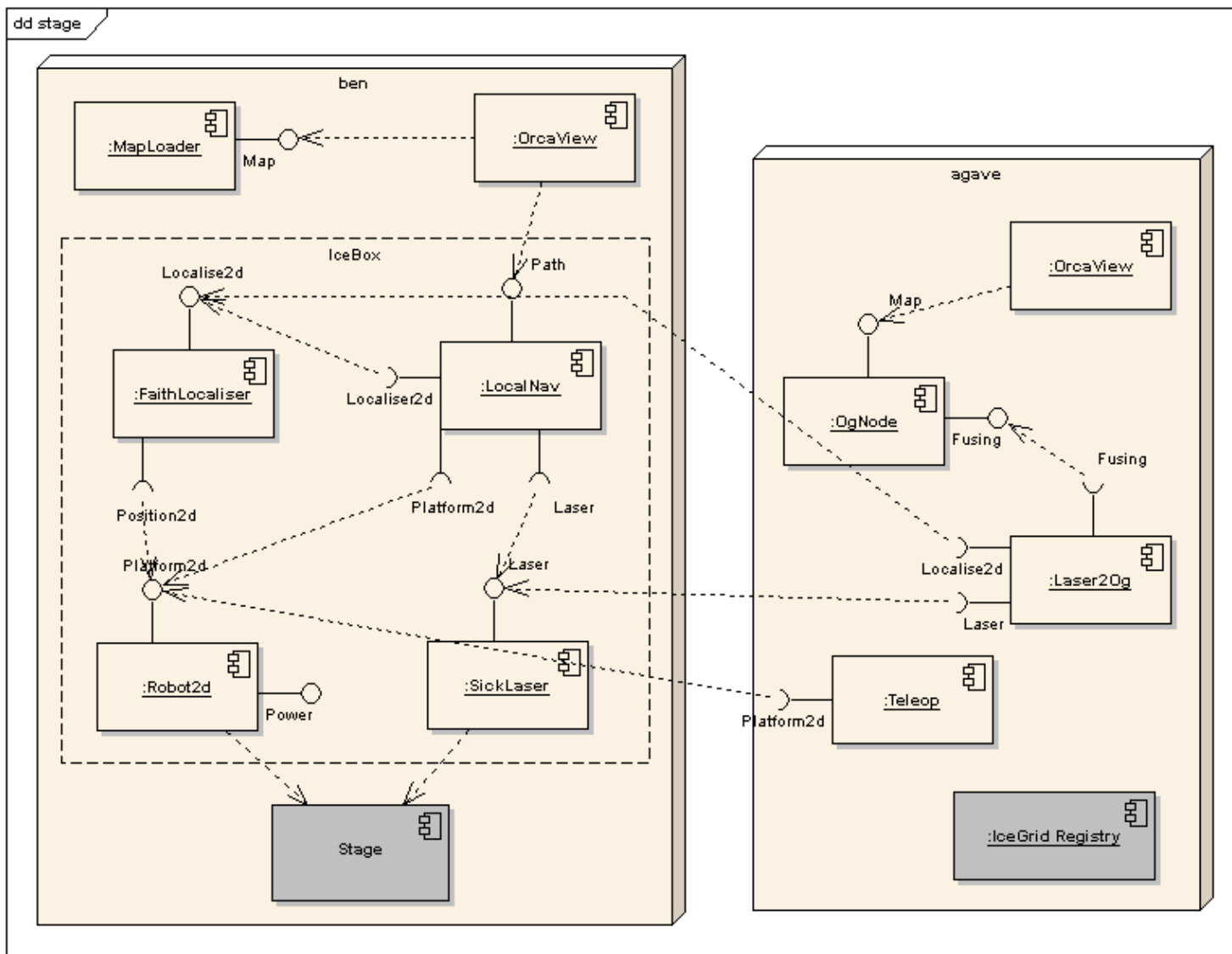
Component Repository

- **FaithLocaliser**
- **FeatureMapLoader**
- **LaserFeatureExtractor**
- **LocalNav**
- **Logger**
- **LogPlayer**
- **OgMapLoader**
- **OrcaView**
- **RegistryView**
- **SegwayRmp**
- **SickLaser**
- **SimLocaliser**
- **Teleop**

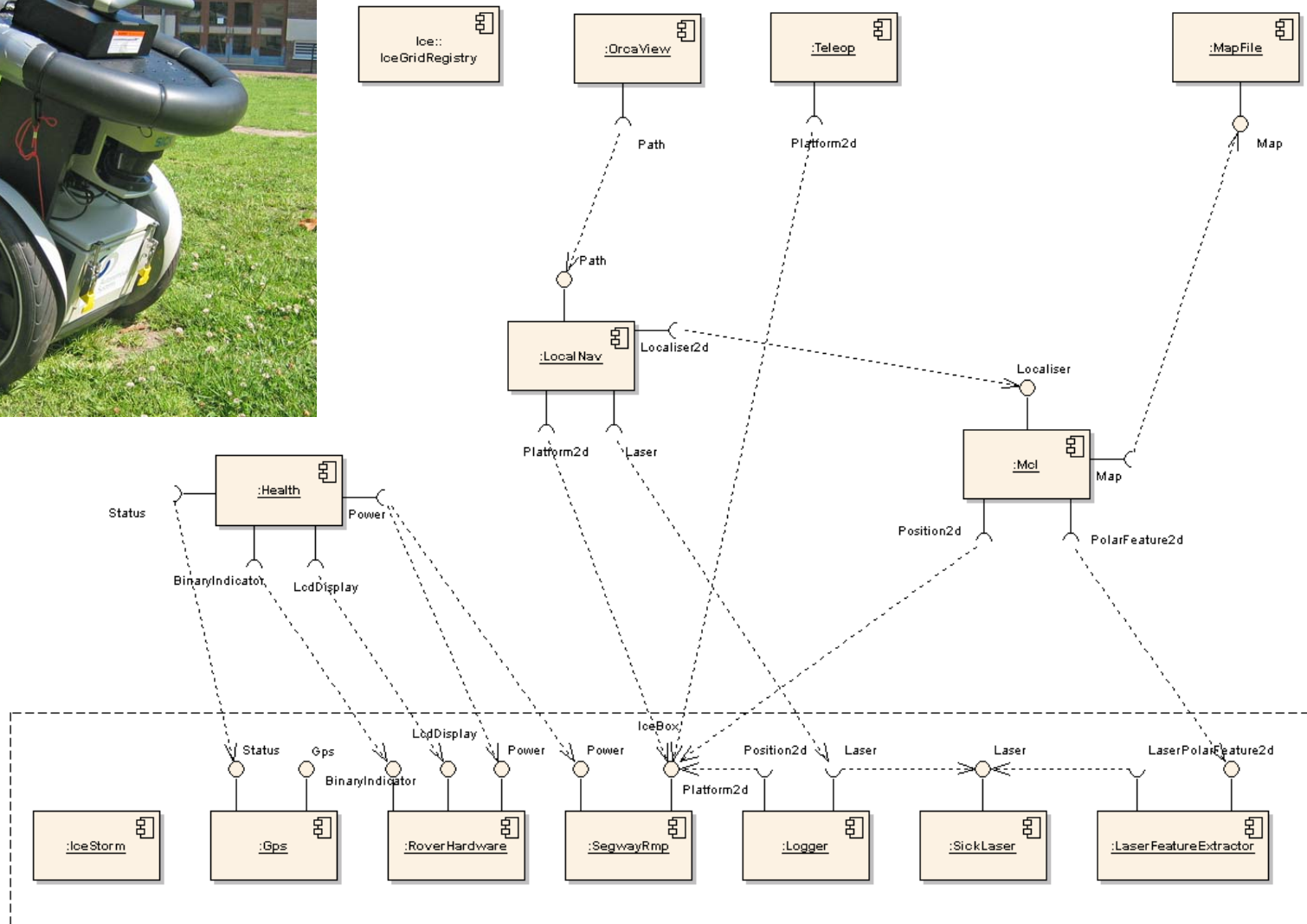
Performance



Example: Simple 2-D Demo with Stage



Current work : ACFR Segway Project

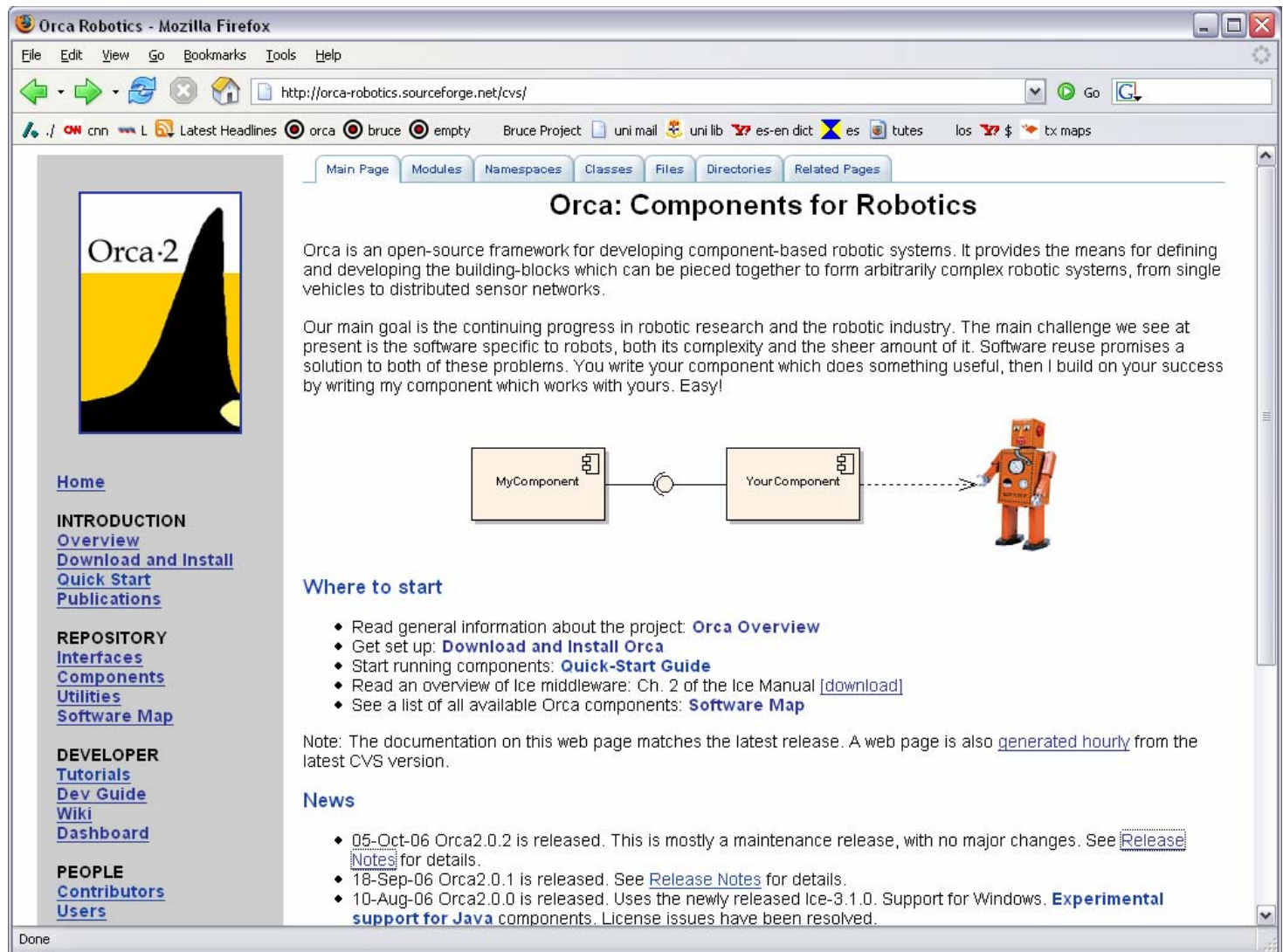


Current work: ACFR/Berkeley Team for 2007 Urban Grand Challenge



Potential Issues to Be Aware of

- **Developer skills**
 - Harder than monolithic applications
 - Requires certain discipline in
 - Structuring the project
 - Writing individual components
 - But ... for large projects, CBSE is a solution not a problem
- **Testing**
 - Harder than monolithic applications
 - But ... using a middleware package removes many testing issues.



The screenshot shows a Mozilla Firefox browser window displaying the Orca Robotics website. The address bar shows the URL `http://orca-robotics.sourceforge.net/cvs/`. The page title is "Orca: Components for Robotics".

Orca: Components for Robotics

Orca is an open-source framework for developing component-based robotic systems. It provides the means for defining and developing the building-blocks which can be pieced together to form arbitrarily complex robotic systems, from single vehicles to distributed sensor networks.

Our main goal is the continuing progress in robotic research and the robotic industry. The main challenge we see at present is the software specific to robots, both its complexity and the sheer amount of it. Software reuse promises a solution to both of these problems. You write your component which does something useful, then I build on your success by writing my component which works with yours. Easy!

Where to start

- Read general information about the project: [Orca Overview](#)
- Get set up: [Download and Install Orca](#)
- Start running components: [Quick-Start Guide](#)
- Read an overview of Ice middleware: Ch. 2 of the Ice Manual [[download](#)]
- See a list of all available Orca components: [Software Map](#)

Note: The documentation on this web page matches the latest release. A web page is also [generated hourly](#) from the latest CVS version.

News

- 05-Oct-06 Orca2.0.2 is released. This is mostly a maintenance release, with no major changes. See [Release Notes](#) for details.
- 18-Sep-06 Orca2.0.1 is released. See [Release Notes](#) for details.
- 10-Aug-06 Orca2.0.0 is released. Uses the newly released Ice-3.1.0. Support for Windows. **Experimental support for Java** components. License issues have been resolved.

The diagram shows two component boxes, "MyComponent" and "Your Component", connected by a solid line. A dashed line extends from "Your Component" to a small orange robot icon.



ARC Centre of Excellence for

Autonomous Systems