An Active Perception Control Architecture for Autonomous Robots

Speaker: J.P. Bandera

March 2009, Coimbra (Portugal)

Ingeniería de Sistemas Integrados
Departamento de Tecnología Electrónica
Universidad de Málaga (Spain)

Instituto de Sistemas é Robótica
Universidade de Coimbra (Portugal)

Index

- Introduction
- Related work
- Components of the architecture
- Implementation details
- Conclusion

Index

Motivation

ISIS group research objective: a social robot

- **Autonomous** agent (use limited perceptual capabilities)
- Help people solving everyday tasks
- Work in real indoor environments
- Interact with people (including untrained users)

Introduction

Control architecture for a robot

- Organization of its actuation, perception and processing capabilities
- Necessary to develop the appropriate capacities and integrate them in an efficient and robust way
- Perception is a set of functions that obtain an abstract representation of the environment

Introduction

Different control architectures

- Deliberative: sense-model-plan-act paradigm
- Behaviour-based control: reactive rules
- Hybrid architectures: three layers

- Autonomous agent (use limited perceptual capabilities)
- Help people solving everyday tasks
- Work in real indoor environments
- Interact with people (including untrained users)

Not only reactive

Needs to adapt

Index

- Introduction
- Related work
- Components of the architecture
- Implementations details
- Conclusion

Index

- Introduction
- Related work
- Components of the architecture
- Implementations details
- Conclusion

Related work

Hybrid architectures

- AuRA, XAVIER, RHINO and ATLANTIS
  - Deliberative layer only activated when any contingence arise
- 3T, BERRA and LAAS
  - Reactive layer controlled by the highest level

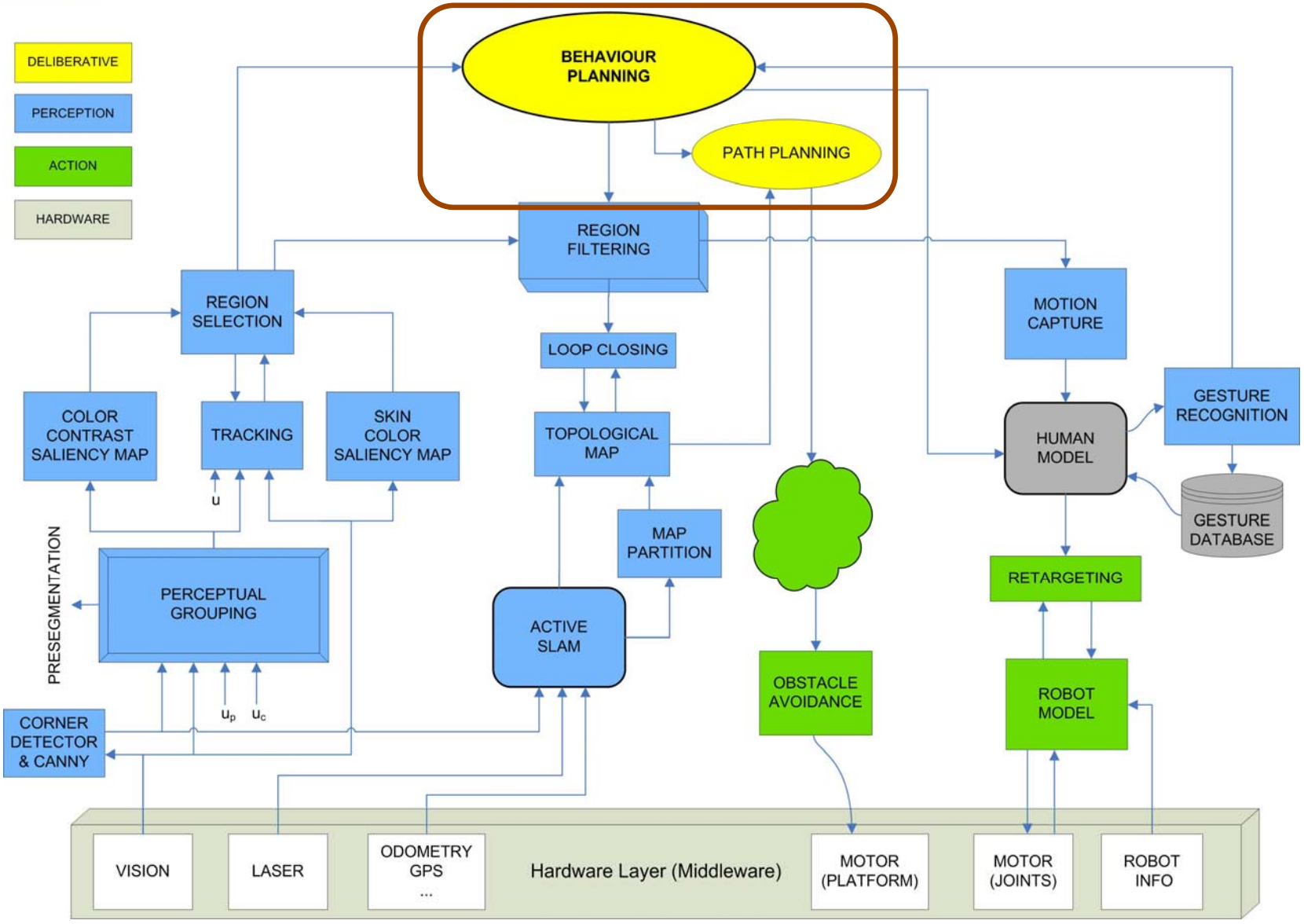
Commercial platforms do not provide the desired level of autonomy (Saphira, Teambots, ...)

Index

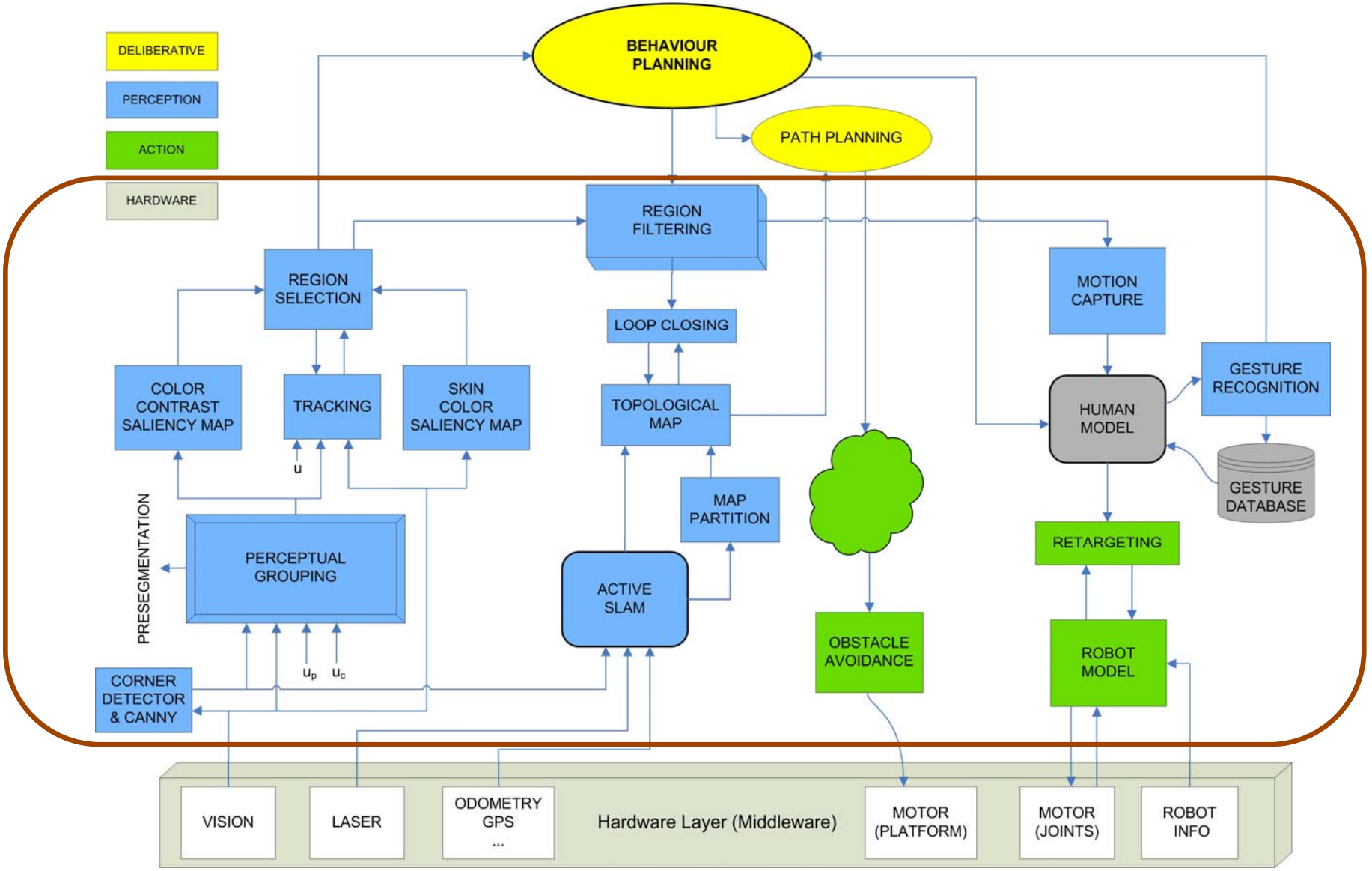
- Introduction
- Related work
- Components of the architecture
- Implementation details
- Conclusions

Index

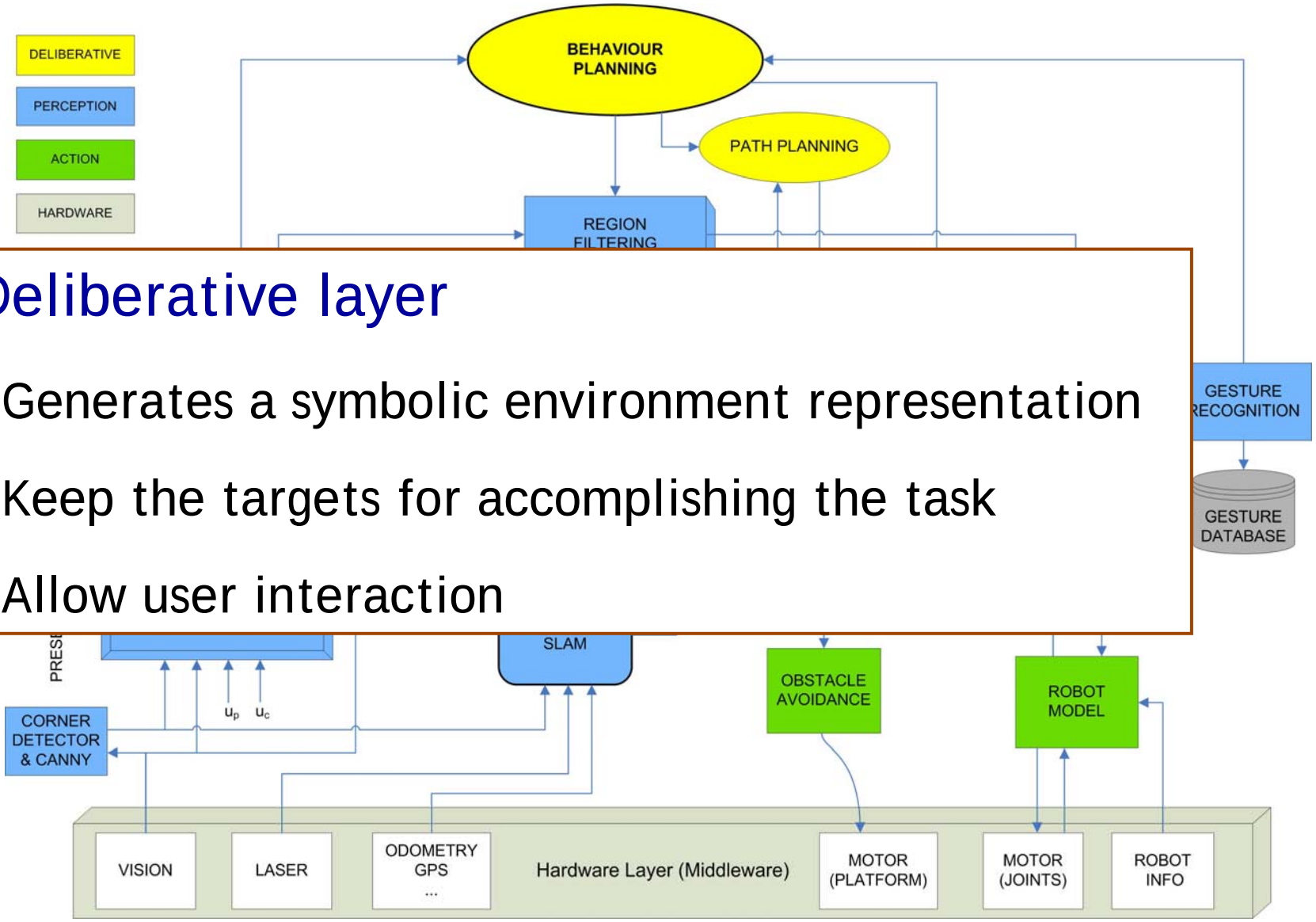
Components of the architecture



Components of the architecture



Components of the architecture

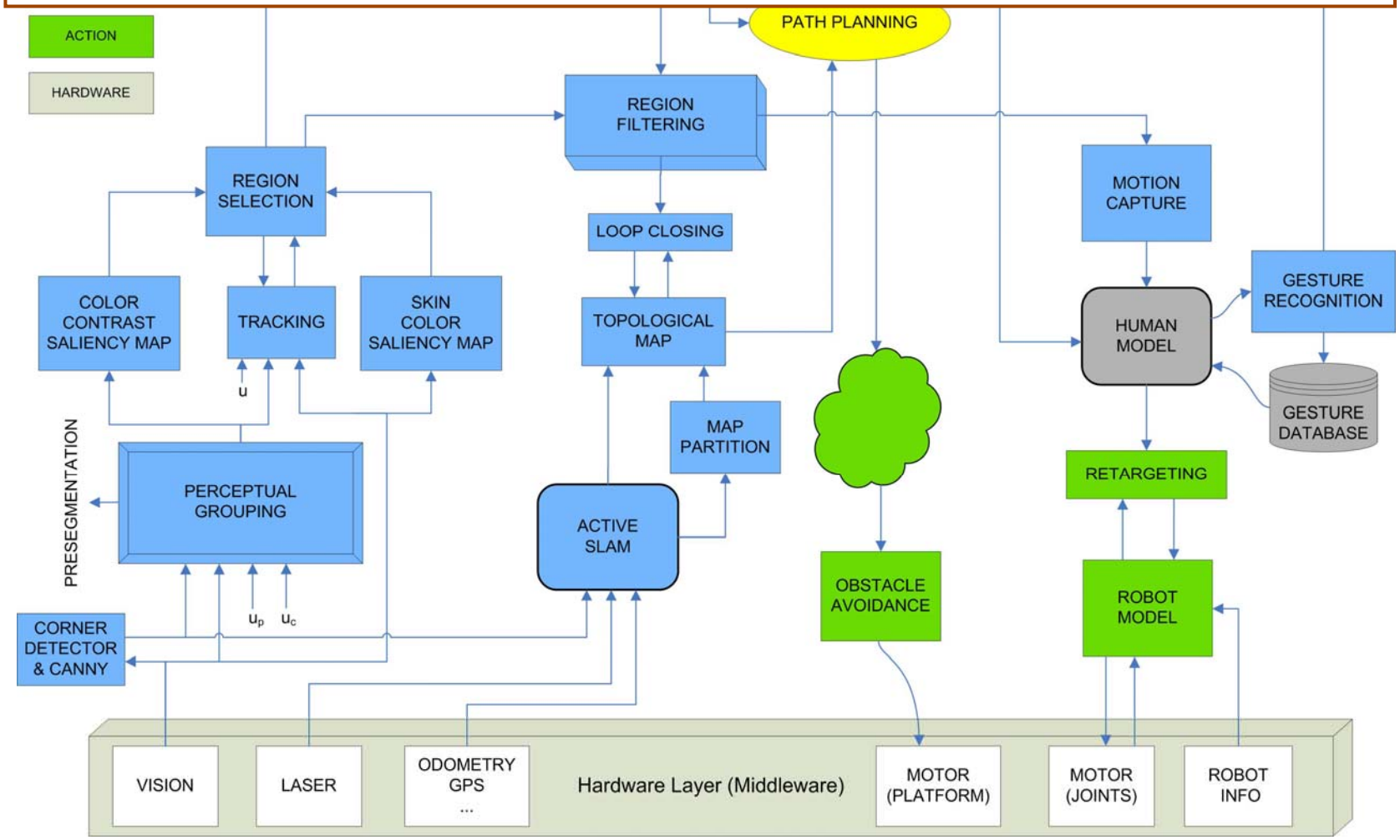


Deliberative layer

- Generates a symbolic environment representation
- Keep the targets for accomplishing the task
- Allow user interaction

Reactive layer: Perceptions

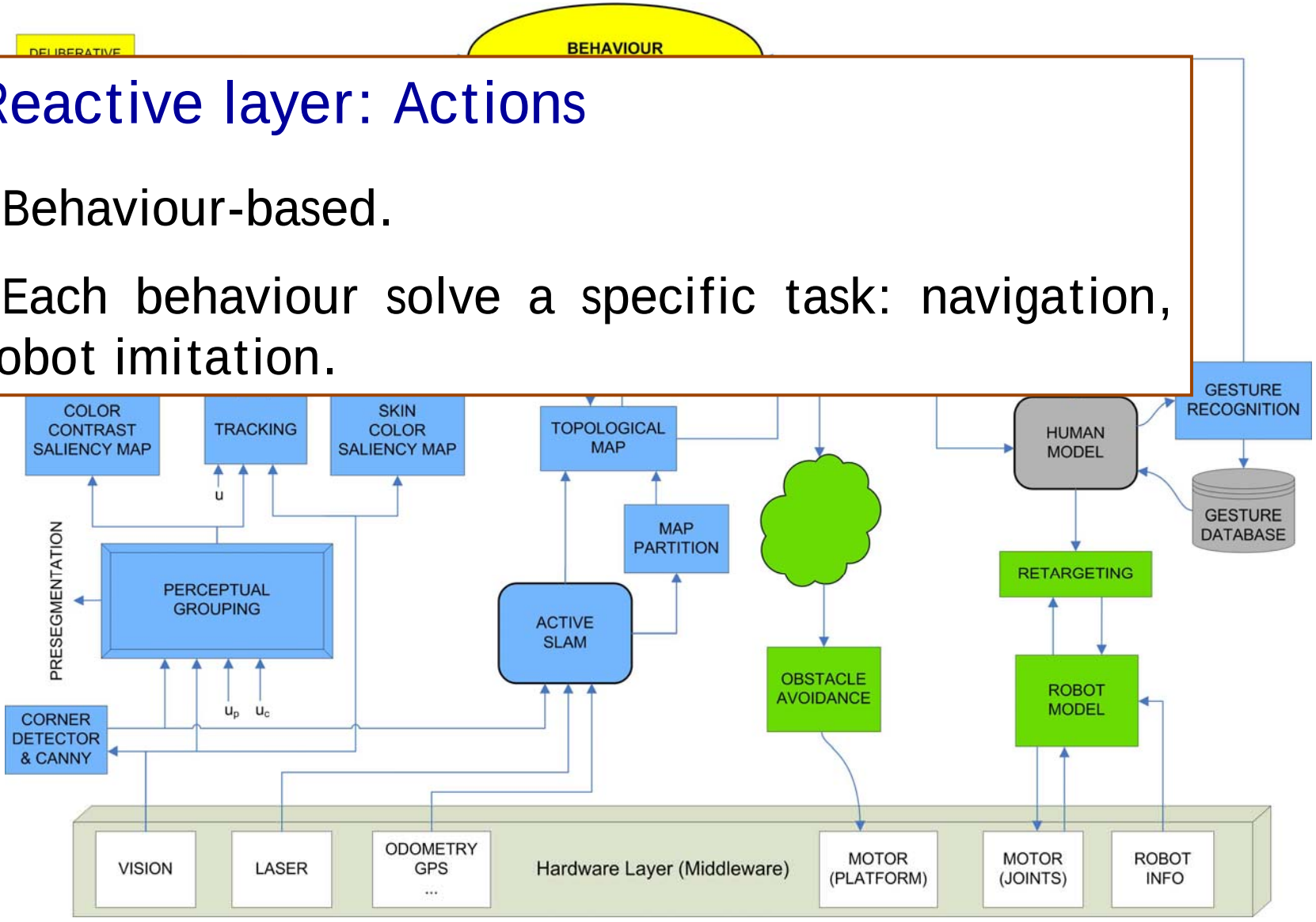
- Set of modules that represent stimulus
  - Primitive: Localization and Attention Mechanism,...
  - Compound: Active SLAM, Motion Capture, Gesture Recognition,...



Components of the architecture

Reactive layer: Actions

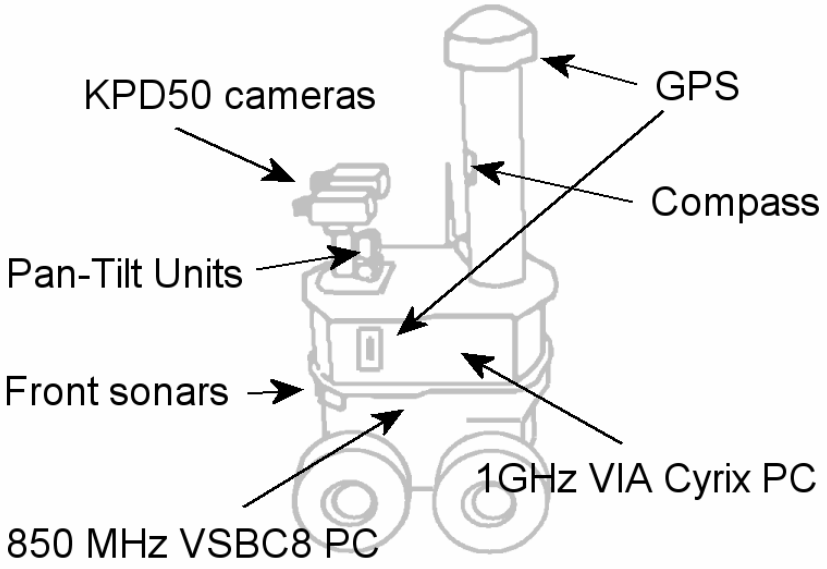
- Behaviour-based.
- Each behaviour solve a specific task: navigation, robot imitation.



Components of the architecture

Robot platform

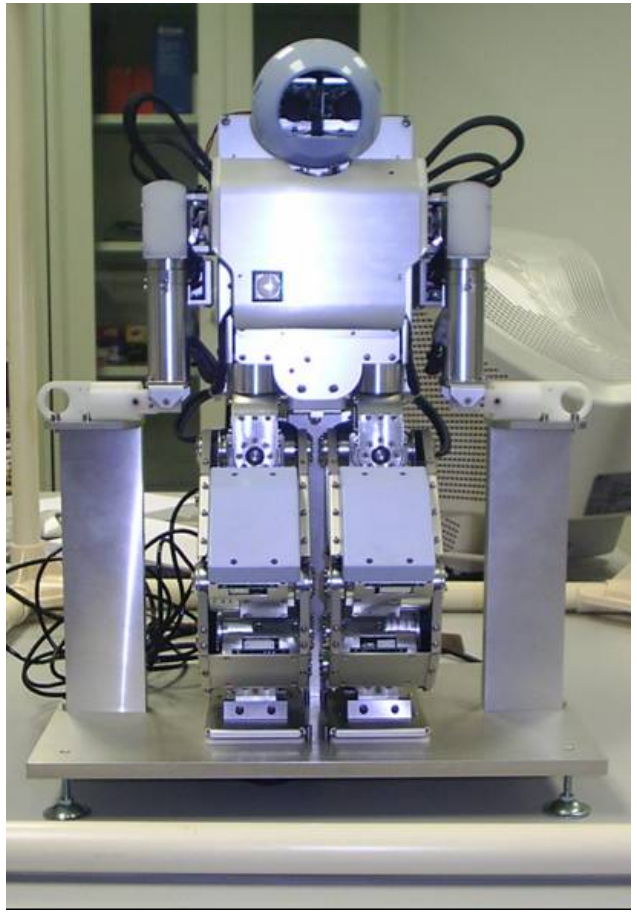
•Pioneer 2 AT: two cameras, PTU, sonar, GPS, compass, two embedded PCs.



Components of the architecture

Robot platform

- HOAP-1:
  - 20 degrees of freedom.
  - 6 kg, 48 cm tall.
  - Four pressure sensors in each foot.
  - Accelerometer, gyro.
  - Stereo cameras.



Components of the architecture

Robot platform

- NOMADA:
  - 9 degrees of freedom (so far).
  - 155 cm tall.
  - Sonar, infrared, bumpers.
  - Wheeled locomotion.
  - Stereo cameras with a ‘human-like’ baseline.

Index

- Introduction
- Related work
- Components of the architecture
- Implementation details
- Conclusions

Implementation details

- Building robust, efficient, & extensible concurrent & distributed applications is hard

- A common solution for the robotics field is the use of a component-based middleware (such as CORBA or DDS)
  - Generic/independent entities
  - Network details are hidden
  - Flexible/Scalable (with *some* effort)
  - Predictible & deterministic
  - *High* performance impact. Use with care! (Real-Time and "fast" are not synonymous)

Implementation details

- Some interesting examples of distributed architectures:

  Client/Server TCP-based (PlayerStage)

  CORBA-based (Robocode)

  Publish/Subscribe (IPC)

They use an object-oriented paradigm, with patterns and frameworks…

The software dimension

Patterns

- *Solutions* to common software *problems* arising within a certain *context*
- *Best practices or "recipes"*

Frameworks

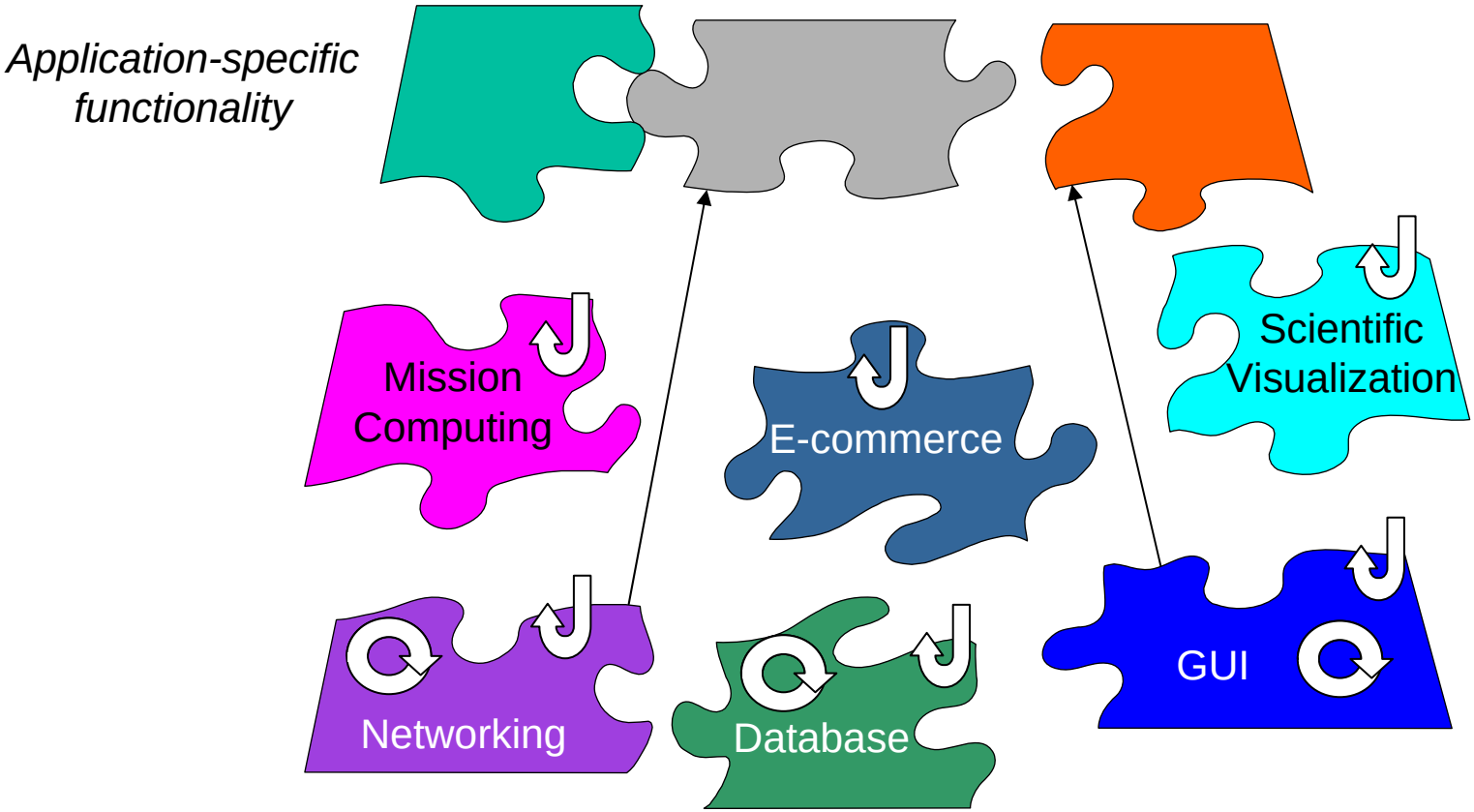
- *A framework is an integrated set of classes that collaborate to produce a reusable architecture for a family of applications*
- *Frameworks implement pattern languages*

The software dimension: patterns

Type	Description	Examples
Idioms	**Restricted to a particular language, system, or tool**	**Scoped locking**
Design patterns	**Capture the static & dynamic roles & relationships in solutions that occur repeatedly**	**Active Object, Bridge, Proxy, Wrapper Façade, & Visitor**
Architectural patterns	**Express a fundamental structural organization for software systems that provide a set of predefined subsystems, specify their relationships, & include the rules and guidelines for organizing the relationships between them**	**Half-Sync/Half-Async, Layers, Proactor, Publisher-Subscriber, & Reactor**
Optimization principle patterns	**Document rules for avoiding common design & implementation mistakes that degrade performance**	**Optimize for common case, pass information between layers**

The software dimension: frameworks

- Frameworks exhibit "inversion of control" at runtime via callbacks
- Frameworks provide integrated domain-specific structures & functionality
- Frameworks are "semi-complete" applications



www.cs.wustl.edu/~schmidt/frameworks.html

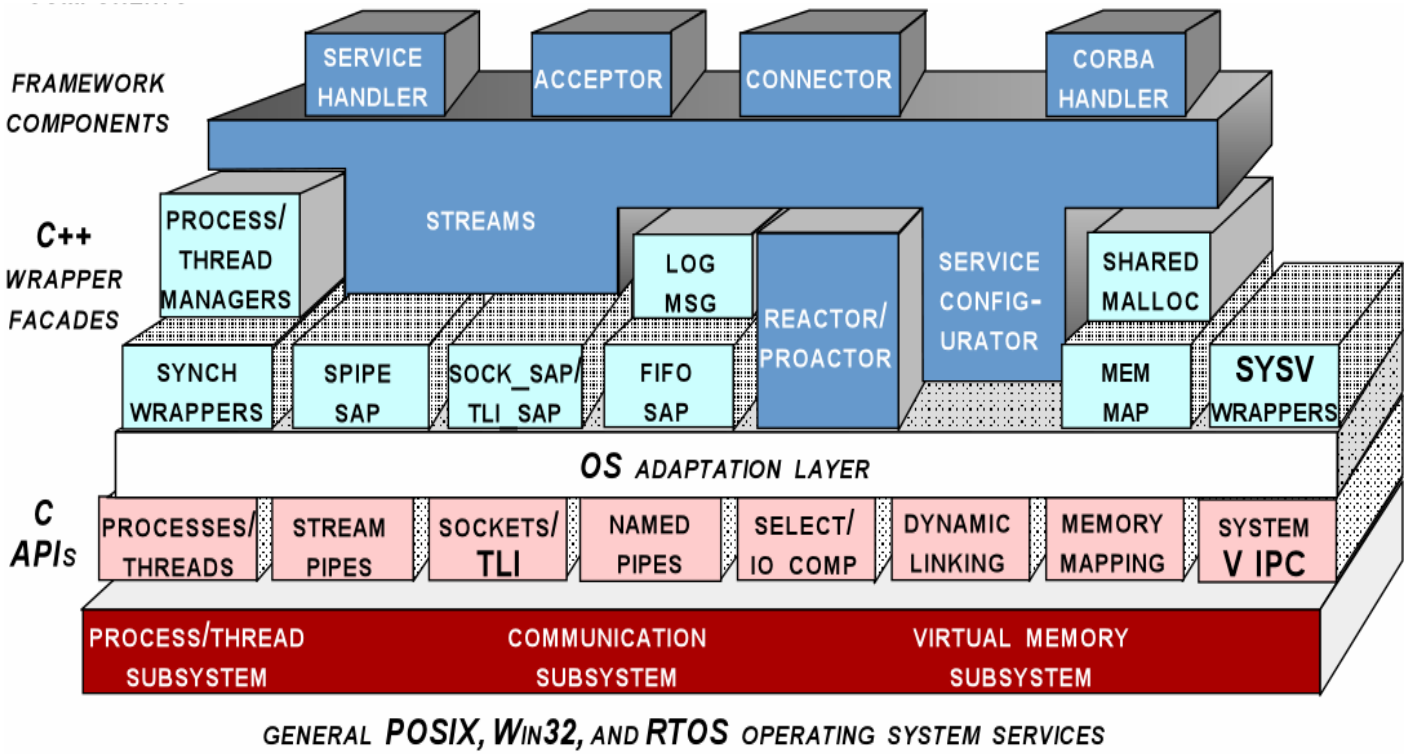
Example: ACE

Domain-Specific Services

Common Middleware Services

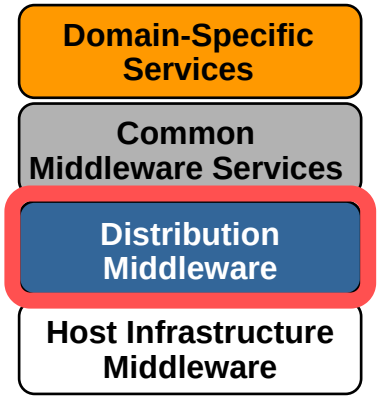
Distribution Middleware

Host Infrastructure Middleware

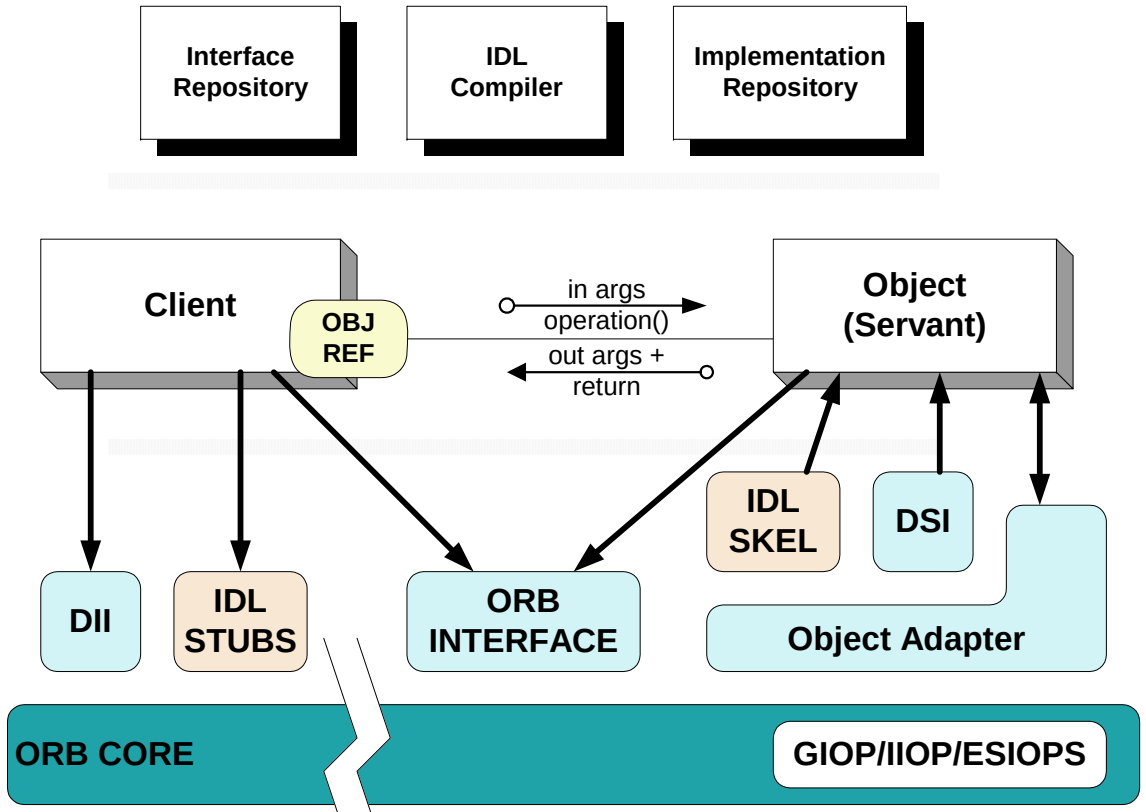


www.cs.wustl.edu/~schmidt/ACE.html

Example: TAO



- Distribution middleware avoids hard-coding client & server application dependencies on object location, language, OS, protocols, & hardware



The *communications* dimension

- Deployment constraints for modules
  - Communications depends on the location

Solution: configuration framework

Flexible enough to allow:

- Different execution modes for modules (threads or processes)
- Different interfaces and protocols (for networked modules)

  - Platform byte-ordering and compiler alignment

Our vision…

- FIRST design, then implement it
  - Start from specific requirements
  - UML with *Profiles* (structure, behaviour, RT, testing)
  - Platform-independent models
- Code reuse IS a must
  - Platform-independent frameworks such as ACE & TAO (available from UML tools through *inverse engineering*)
  - ¿OpenSLICE? for Data Distribution Services
- Apply a distributed paradigm ONLY IF strictly necessary
  - We want to use the best patterns available

Index

Conclusions

- Hybrid architecture that achieve more balanced control between reactive and deliberative layers
- The perception module has a critical role in the robot operation
- Our proposed software is composed of frameworks which provide platform-independent high level abstraction

An Active Perception Control Architecture for Autonomous Robots

- **Thanks for your attention!!**
- **Any questions/advise?**

Instituto de Sistemas é Robótica

Universidade de Coimbra (Portugal)