Fawkes at a Glance

Introduction to the Fawkes Robot Software Framework

Tim Niemueller <tim@niemueller.de>

February 1st 2007

1. Retrospective
2. Design Goals
3. Components
4. Development
5. Loose Ends

1 Retrospective
- RCSoft Drawbacks
- RCSoft Benefits

2 Design Goals

3 Components

4 Development

5 Loose Ends

Code Management Flaws

- CVS hackery took place
- Many dependencies
- External libs inside the source tree
- Dependencies not documented and even somewhat hidden
- Inefficient build system with long build times and huge Makefiles

Code Management Flaws

- CVS hackery took place
- Many dependencies
- External libs inside the source tree
- Dependencies not documented and even somewhat hidden
- Inefficient build system with long build times and huge Makefiles

All of these problems can be fixed inside RCSoft itself with some effort.

Error Inducing Problems

- Hard to debug with gdb (signal pollution)
- Code polluted (i.e. two approaches in BB)
- No detection of multiple module instances or BB writers
- Overhead due to process switches
- Insufficient config system (no long-time storage)
- Parameters unknown or even “working against each other”
- No central logging, just logging transferred data that made it through the network

Error Inducing Problems

- Hard to debug with gdb (signal pollution)
- Code polluted (i.e. two approaches in BB)
- No detection of multiple module instances or BB writers
- Overhead due to process switches
- Insufficient config system (no long-time storage)
- Parameters unknown or even “working against each other”
- No central logging, just logging transferred data that made it through the network

Besides GDB problem this can be fixed inside RCSoft with a re-factoring, cleanup and fixing development cycle.

Performance Issues

- Long time from power-on to working robot
- Global storage locking
- Polling everywhere, no events
- Overhead due to process switches
- All processes run continuously (even if no other process can make use of the result)

Performance Issues

- Long time from power-on to working robot
- Global storage locking
- Polling everywhere, no events
- Overhead due to process switches
- All processes run continuously (even if no other process can make use of the result)

Influenced by hardware performance. No easy way of fixing these problems inside RCSoft.

No Guarantees - no Timing

- No timing guarantees, not even information
- No synchronization
- No information about data age
- No in-cycle-guarantee

No Guarantees - no Timing

- No timing guarantees, not even information
- No synchronization
- No information about data age
- No in-cycle-guarantee

RCSoft probably wastes a lot of time calculating data twice before it is actually used. No information available about program flow. No way to fix this without major changes.

RCSoft’s Good Stuff

- Modular structure

RCSoft’s Good Stuff

- Modular structure
- Unified information exchange (BlackBoard)

RCSoft's Good Stuff

- Modular structure
- Unified information exchange (BlackBoard)
- Similarities in basic program structure

RCSoft’s Good Stuff

- Modular structure
- Unified information exchange (BlackBoard)
- Similarities in basic program structure
- Network transparency for information exchange

RCSoft’s Good Stuff

- Modular structure
- Unified information exchange (BlackBoard)
- Similarities in basic program structure
- Network transparency for information exchange
- Many tools, although many sub-optimal

RCSoft's Good Stuff

- Modular structure
- Unified information exchange (BlackBoard)
- Similarities in basic program structure
- Network transparency for information exchange
- Many tools, although many sub-optimal

Start over from scratch. Keep the good ideas, improve or replace the not-so-good ones.

1 Retrospective

2 Design Goals
- Masterplan
- Summary

3 Components

4 Development

5 Loose Ends

What to Keep

- Keep modular structure
- Keep unified information storage (similar to BlackBoard)
- Provide simple templates for new modules
- Provide the well-known tools (RCCC, vis_bb, FireStation)

What to Avoid

- Avoid polling, events and blocked waiting instead
- Avoid dependencies where possible, have few and document them well
- Avoid code duplication and more approaches where useless (different visions, localizations etc. may be OK, having two thread implementations is useless)
- Avoid everything that makes debugging hard or impossible

What to Add and Improve

- Source code management
- Enforce documentation
- Sleek and fast build system
- Easy modular software structure
- Mutual exclusions on information storage
- Throw away stuff not needed any more, it's still in SVN!
- Make it debuggable and do it! (gdb, valgrind, QA apps, unit tests)
- Use exceptions for good error handling and better readability

Guarantees

- Guarantees ensure certain conditions, behavior and functionality of the software stack

Guarantees

- Guarantees ensure certain conditions, behavior and functionality of the software stack
- Have guarantees!
- Guarantees minimize error handling in upper levels
- Failures in guaranteed components are a bug. No more discussion about that.
- Not met guarantees are crash points by design

Guarantees

- Guarantees ensure certain conditions, behavior and functionality of the software stack
- Have guarantees!
- Guarantees minimize error handling in upper levels
- Failures in guaranteed components are a bug. No more discussion about that.
- Not met guarantees are crash points by design

Guarantees are needed to keep the code simple, to have well defined software interfaces and to minimize the risk of errors.

Needed Guarantees (known so far)

- *Initialization:* either a component/thread/aspect is successfully initialized or never started

Needed Guarantees (known so far)

- *Initialization:* either a component/thread/aspect is successfully initialized or never started
- *Dependencies:* Either all requirements are met or a component cannot run (and the problem is detected!)

Needed Guarantees (known so far)

- *Initialization:* either a component/thread/aspect is successfully initialized or never started
- *Dependencies:* Either all requirements are met or a component cannot run (and the problem is detected!)
- *Concurrency:* There must be mutually exclusive access to critical components like data storage (single writer)

Needed Guarantees (known so far)

- *Initialization:* either a component/thread/aspect is successfully initialized or never started
- *Dependencies:* Either all requirements are met or a component cannot run (and the problem is detected!)
- *Concurrency:* There must be mutually exclusive access to critical components like data storage (single writer)
- *Timing:* Guarantee a defined and documented call chain per loop

Needed Guarantees (known so far)

- *Initialization:* either a component/thread/aspect is successfully initialized or never started
- *Dependencies:* Either all requirements are met or a component cannot run (and the problem is detected!)
- *Concurrency:* There must be mutually exclusive access to critical components like data storage (single writer)
- *Timing:* Guarantee a defined and documented call chain per loop
- *Time Source:* Guarantee that all components use the exact same time source

Summary

- Modular

Summary

- Modular
- Unified information storage

Summary

- Modular
- Unified information storage
- No polling, events

Summary

- Modular
- Unified information storage
- No polling, events
- Central timing and time source

Summary

- Modular
- Unified information storage
- No polling, events
- Central timing and time source
- Debuggable

Summary

- Modular
- Unified information storage
- No polling, events
- Central timing and time source
- Debuggable
- Guarantees

Summary

- Modular
- Unified information storage
- No polling, events
- Central timing and time source
- Debuggable
- Guarantees
- Minimize dependencies

Summary

- Modular
- Unified information storage
- No polling, events
- Central timing and time source
- Debuggable
- Guarantees
- Minimize dependencies

One to rule them all: only one dynamic application

1 Retrospective

2 Design Goals

3 Components

- At a Glance
- Libraries
- Infrastructure Components
- Main Application and Plugins
- Aspects

4 Development

5 Loose Ends

Basic Components

Libraries

Basic set of libraries that can be used in Fawkes.

Main Application

Application able to load, init and run Fawkes plugins.

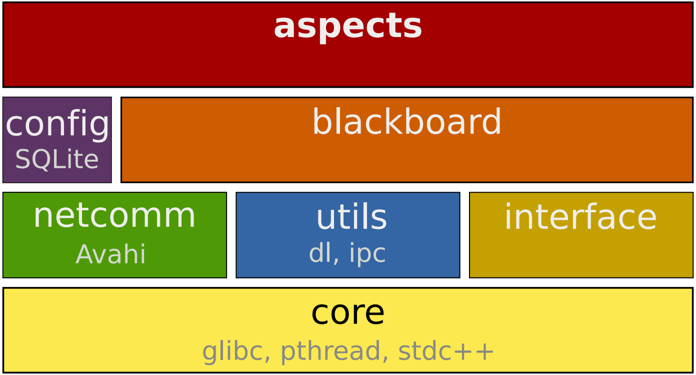
Threads

Everything is a thread. All operations are carried out in a thread.

Plugins

Units containing the real functionality of the robot.

Library Overview



aspects

config
SQLite

blackboard

netcomm
Avahi

utils
dl, ipc

interface

core
glibc, pthread, stdc++

Fawkes Core Library (FCL, core)

- Contains core components, may not depend on other non-system libraries
- Threading API
- Synchronization constructs (mutex, wait condition, read-write lock etc.)
- Exception API
- Plugin API
- Base utils needed in FCL

Fawkes Utilities (utils)

- Resembles old utilities
- May not depend on other non-system libs besides core
- Currently:
  - Logging
  - IPC
  - Plugin loading
  - System
  - Text
  - Time

Network Communication Library (netcomm)

- Socket API
- Fawkes Network Protocol implementation
- Multicast WorldInfo Transceiver
- Robot and service discovery using DNS-based service discovery over multicast DNS (mDNS-SD)

BlackBoard (BB, blackboard)

- Unified and central information storage
- Completely new implementation with similar goals
- Read/write lock per interface
- No more searching, simple pointers, small data blocks (few bytes)
- No multi-process access
- Central logging instance integrated (tbd)
- Possibility to get notified of changes (tbd)
- Guarantees writer singleton

BlackBoard - Interfaces

- Access to BB data via so-called interfaces
- C++ wrapper class with `read()` and `write()` operations
- Interfaces are instantiated by the InterfaceManager
- There may be only one writer instance at any one time per Interface
- Protected with Read/Write locks
- Interface generator to transform XML descriptions into code

Configuration Subsystem

- C++ interface defines access independently of implementation
- Currently SQLite implementation exists
- Configurations may be tagged, used for instance for different locations and backups
- Handlers can be registered that are immediately notified of any configuration modification
- Network protocol and tool implemented to modify configuration
- Default and host configuration (overlay to default)

Network Communication

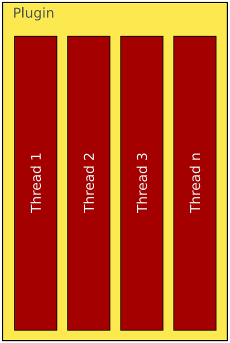
- Fawkes Network Protocol implemented
- TCP connection, announced and found via mDNS-SD
- Arbitrary communication can happen over the Fawkes connection
- Currently: PluginManager, ConfigurationManager
- BlackBoard communication inside Fawkes (tbd)
- Multicast inter-robot communication about world information (wip)

Main Application

- Implements the infrastructure using previously shown components
- Uses managers to handle configuration, plugins etc.
- Network communication handled by managers or plugins
- Fawkes main thread handles timing and basic synchronization
- Is the only application run on the robot
- Plugins are loaded dynamically

Plugins

- Module for a specific task
- Consists of one or more threads
- Threads are initialized
- It is guaranteed that either all threads are successfully initialized or none is started at all



Aspects and the Tulip Principle

- Aspects are wrapped to a thread like the leaves of a tulip's flower

Aspects and the Tulip Principle

- Aspects are wrapped to a thread like the leaves of a tulip's flower
- An Aspect adds a specific functionality to a thread
- A thread may have any number of aspects

Aspects and the Tulip Principle

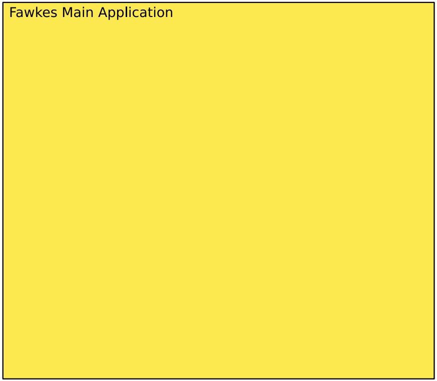
- Aspects are wrapped to a thread like the leaves of a tulip's flower
- An Aspect adds a specific functionality to a thread
- A thread may have any number of aspects

- *BlockedTimingAspect:* thread integrates into the main loop
- *BlackBoardAspect:* thread needs access to the BlackBoard
- *ConfigurableAspect:* thread uses a configuration
- *LoggingAspect:* thread writes output to a logger

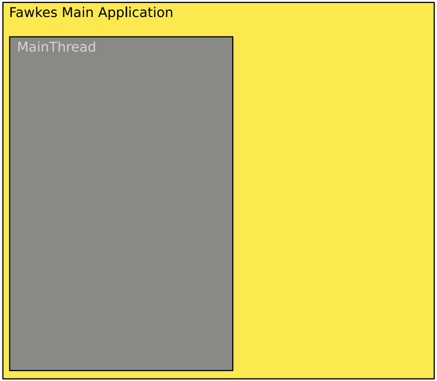
Aspect Initialisation

- Aspects are initialised by the AspectInitializer
- If any error occurs during the initialisation of an aspect the thread is never started
- Aspects are a clean way to add functionality with minimum overhead
- No further initialisation inside the thread needed, just deriving the aspect base class is sufficient
- Knowledge on how to handle a thread can be derived from the aspects

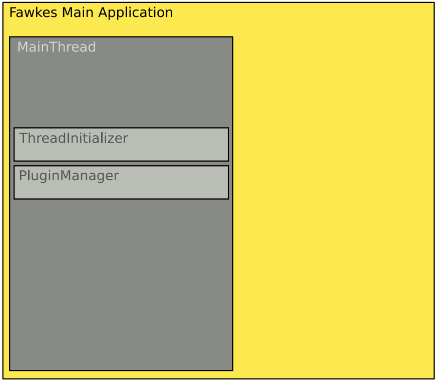
Fawkes Diagram



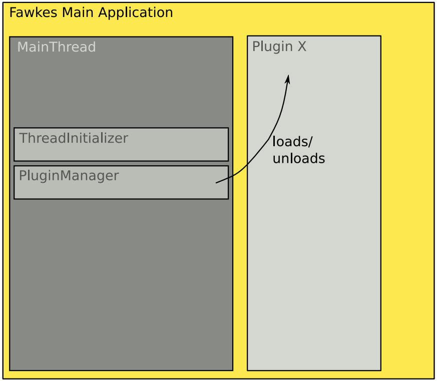
Fawkes Diagram



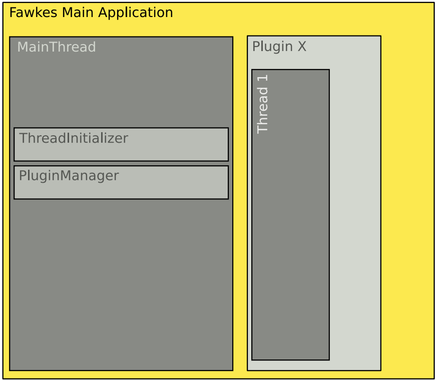
Fawkes Diagram



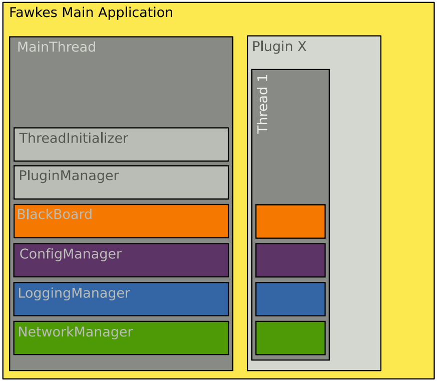
Fawkes Diagram



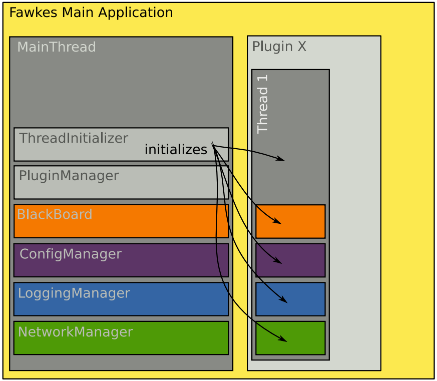
Fawkes Diagram



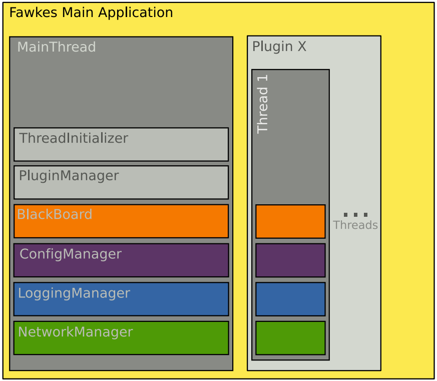
Fawkes Diagram



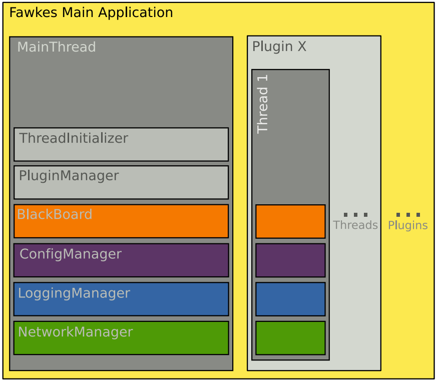
Fawkes Diagram



Fawkes Diagram



Fawkes Diagram



1 Retrospective

2 Design Goals

3 Components

4 Development
- Principles
- Improvements
- Implementing a Plugin
- Adding an Aspect
- Running Fawkes

5 Loose Ends

Fawkes Development Principles

- Document all your code immediately

Fawkes Development Principles

- Document all your code immediately
- Fix bugs before implementing new stuff

Fawkes Development Principles

- Document all your code immediately
- Fix bugs before implementing new stuff
- Exploit all available tools, use gdb and valgrind

Fawkes Development Principles

- Document all your code immediately
- Fix bugs before implementing new stuff
- Exploit all available tools, use gdb and valgrind
- Use what’s there, do not re-invent the wheel

Improvements to Development Tools

- Subversion for version control
  - Finally refactoring is fun (move cmd)
  - Blame command to see originator
  - Offline operations for status, diffs etc.

Improvements to Development Tools

- Subversion for version control
  - Finally refactoring is fun (move cmd)
  - Blame command to see originator
  - Offline operations for status, diffs etc.
- Trac
  - Source code browser
  - Ticket management for bugs and features
  - Timeline and Roadmap
  - Access to documentation, API reference and wiki

Creating the Threads

- Derivative of Thread
- If needed use WAITFORWAKEUP mode (thread will wait after every loop for a wake-up call, needed for BlockedTimingAspect)
- Do all initialisation in the constructor
- Implement `loop()` to do what you need to do
- Add any aspect that you need by deriving its aspect class
- If threads need synchronisation among each other pass the needed constructs to the constructor (consider a synchronized shared data object)

Creating a Plugin

- Derivative of Plugin
- Implement plugin's threads
- Implement `threads()` to return a list of instantiated threads, take care of inter-thread synchronisation details here
- Implement `plugin_factory()` and `plugin_destroy()`

Creating a Plugin

- Derivative of Plugin
- Implement plugin's threads
- Implement `threads()` to return a list of instantiated threads, take care of inter-thread synchronisation details here
- Implement `plugin_factory()` and `plugin_destroy()`

First steps

Use `src/plugins/example` as a template. It contains a basic plugin that will run a few simple threads.

Creating an Interface

- Write XML template in `src/interfaces`
- `make`
- This will build `.h`/`.cpp` file and compile
- Use it
- Documentation yet to be written

Creating an Aspect

- Plain class, may not derive anything
- May use any library, avoid big fat external dependencies
- May not have pure virtual functions
- May have special constructor
- May have initialization routine, name specific to avoid name clashes (not `init()` but `MyAspect::initMyAspect()`)
- Make AspectInitializer know how to initialize the aspect and to detect any problems to meet guarantees
- Document extensively

Fawkes Tools

- `config`: Configuration editing over the network
- `plugin`: Load and unload plugins
- `interface_generator`: Transform BB interface XML templates into code
- use `-H` argument for a usage message (file a bug if missing!)

Running Fawkes

```
./fawkes
```

1. Retrospective
2. Design Goals
3. Components
4. Development
5. Loose Ends

Done

- Threading and synchronization API
- BlackBoard basics (data storage, messaging)
- Network communication (Fawkes Network Protocol)
- Basic utility library
- Main application able to load plugins, and to initialize and run threads

Done

- Threading and synchronization API
- BlackBoard basics (data storage, messaging)
- Network communication (Fawkes Network Protocol)
- Basic utility library
- Main application able to load plugins, and to initialize and run threads

Enough to implement the first plugins and tune the call dynamics.

To Do

- BlackBoard logging and events
- BlackBoard introspection (+ visual tool)
- Geometry/math library (wip)
- Multicast world info transceiver (wip)
- Porting vision (wip)
- Implementing base applications (world model, agent etc.)
- New control center (probably not for RC2007)
- Graphical config editor

To Do

- BlackBoard logging and events
- BlackBoard introspection (+ visual tool)
- Geometry/math library (wip)
- Multicast world info transceiver (wip)
- Porting vision (wip)
- Implementing base applications (world model, agent etc.)
- New control center (probably not for RC2007)
- Graphical config editor

The foundation has been constructed, now it needs combined efforts to get it fly.

Questions and Discussion