AP2DX

Awesomizing the P2DX

Jasper Timmer, Maarten Inja, Maarten de Waard, Wadie Assal

UvA

June 30, 2011

Introduction

- The same assignment
- Who we are
- What is special in our case

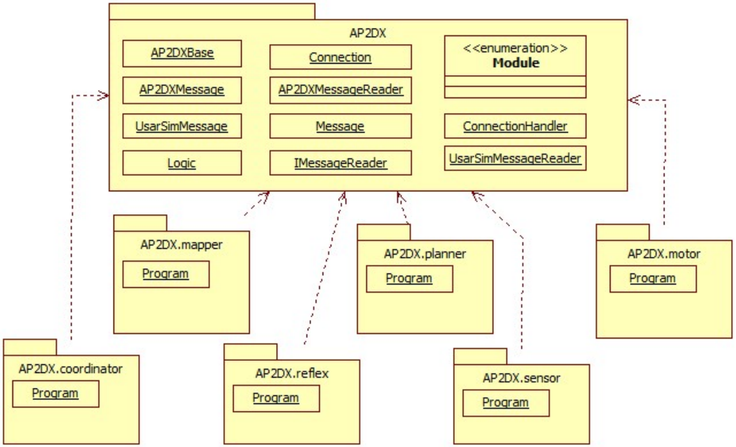
Goals

- Loosely coupled modules based on network communication
- Robot should be safe, i.e. stop for obstacles
- Robot should be able to drive autonomously through the environment
- Robot should be able to create a map of the environment
- No user input will be required

Architecture - the basics

$$\begin{tikzcd}
 & USARSim \arrow[d, bend right] & \\
 & Coordinator \arrow[u, bend right] \arrow[dr] & \\
AbstractMotor \arrow[ur] & Reflex \arrow[l] & Sensor \arrow[l] \arrow[d] \\
 & Planner \arrow[u] & Mapper \arrow[l]
\end{tikzcd}$$

Architecture - into the depths



Abstract base class

We decided to use an abstract class, to base all our classes on.

Advantages

- Very easy to work with
- Only have to make the connection protocol once
- Strict contracts with team mates

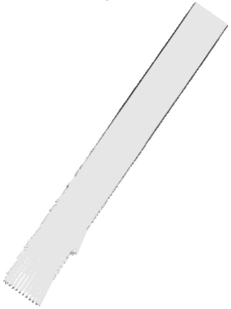
Disadvantages

- Stuck with one language
- Hard debugging
- Hard to make big changes, or add things we had not thought of

boobs

Mapper

We did not make our own mapper. We used DP Slam[1][2].
The program works in C, and creates a map like this one:



[1]http://www.cs.duke.edu/ parr/dpslam/

[2]Algorithm from: Austin Eliazar, Ronald Parr: DP-SLAM: Fast, Robust Simultainous Localization and Mapping Without Predetermined Landmarks

Mapper - What makes it special

- Two ways to use a mapper:
    - While driving
    - After driving (with saved sensordata)
- We make a map, while driving
- Mapper uses Odometry and Laser range scanner data
- Currently only works on linux

Messages

$$\begin{tikzcd}
& Message \arrow[dl] \arrow[dr] & \\
AP2DXMessage \arrow[d] & & UsarSimMessage \arrow[d] \\
SpecializedMessage \arrow[d] & & (individualmessages) \\
(individualmessages) & &
\end{tikzcd}$$

Messages - explained

There are a couple of advantages and disadvantages:

Advantages

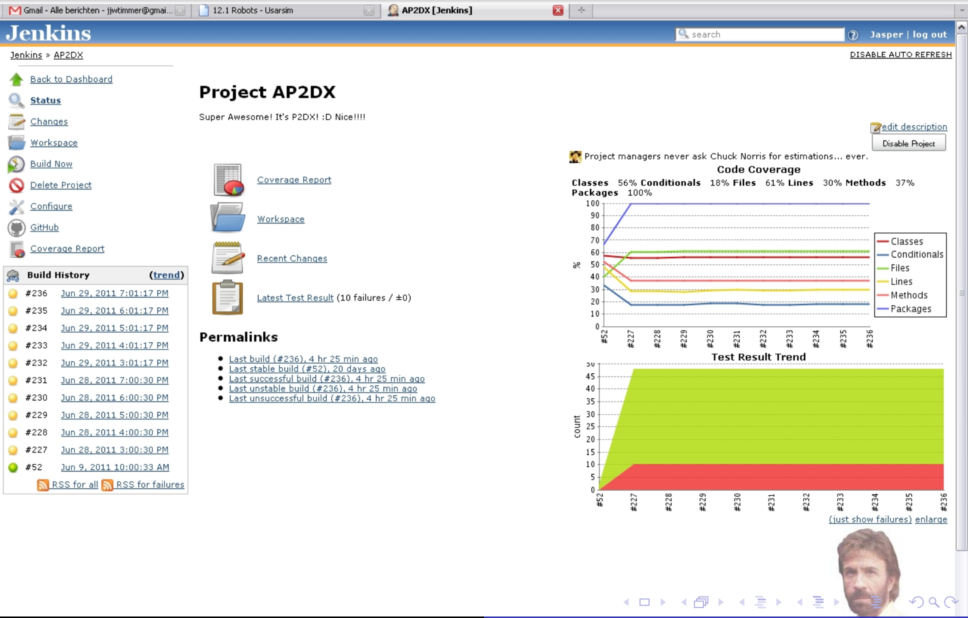
- Very easy to work with
- Easy to add a new kind of message
- Strict restrictions to how a message should look (and thus uniformity)

Disadvantages

- Very hard to debug
- Hard to add a type of message that doesn't fit in

Testing - an introduction

We decided to use jUnit tests, since all our code is in Java.
For this, we used Jenkins, an automatic build and test server



Gmail - Alle berichten - jjwtimmer@gmai... | 12.1 Robots - Usarsim | AP2DX [Jenkins]

Jenkins | search | Jasper | log out

Jenkins » AP2DX | DISABLE AUTO REFRESH

- Back to Dashboard
- Status
- Changes
- Workspace
- Build Now
- Delete Project
- Configure
- GitHub
- Coverage Report

Build History (trend)

#236	Jun 29, 2011 7:01:17 PM
#235	Jun 29, 2011 6:01:17 PM
#234	Jun 29, 2011 5:01:17 PM
#233	Jun 29, 2011 4:01:17 PM
#232	Jun 29, 2011 3:01:17 PM
#231	Jun 28, 2011 7:00:30 PM
#230	Jun 28, 2011 6:00:30 PM
#229	Jun 28, 2011 5:00:30 PM
#228	Jun 28, 2011 4:00:30 PM
#227	Jun 28, 2011 3:00:30 PM
#52	Jun 9, 2011 10:00:33 AM

RSS for all RSS for failures

Project AP2DX

Super Awesome! It's P2DX! :D Nice!!!!

edit description | Disable Project

- Coverage Report
- Workspace
- Recent Changes
- Latest Test Result (10 failures / ±0)

Permalinks

- Last build (#236), 4 hr 25 min ago
- Last stable build (#52), 20 days ago
- Last successful build (#236), 4 hr 25 min ago
- Last unstable build (#236), 4 hr 25 min ago
- Last unsuccessful build (#236), 4 hr 25 min ago

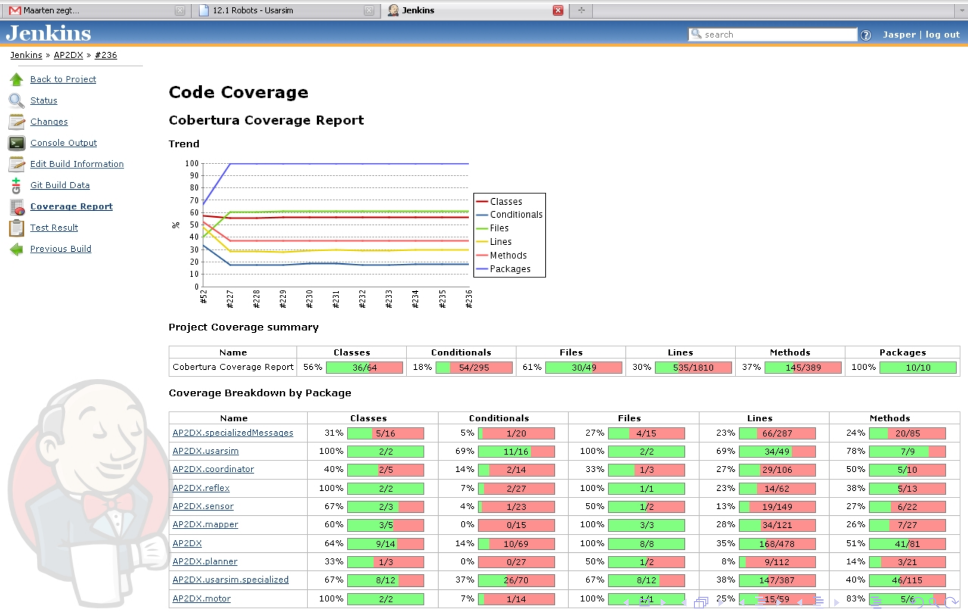
Project managers never ask Chuck Norris for estimations... ever.

Code Coverage

Classes 56% **Conditionals** 18% **Files** 61% **Lines** 30% **Methods** 37% **Packages** 100%

Test Result Trend

(just show failures) enlarge



Code Coverage

Cobertura Coverage Report

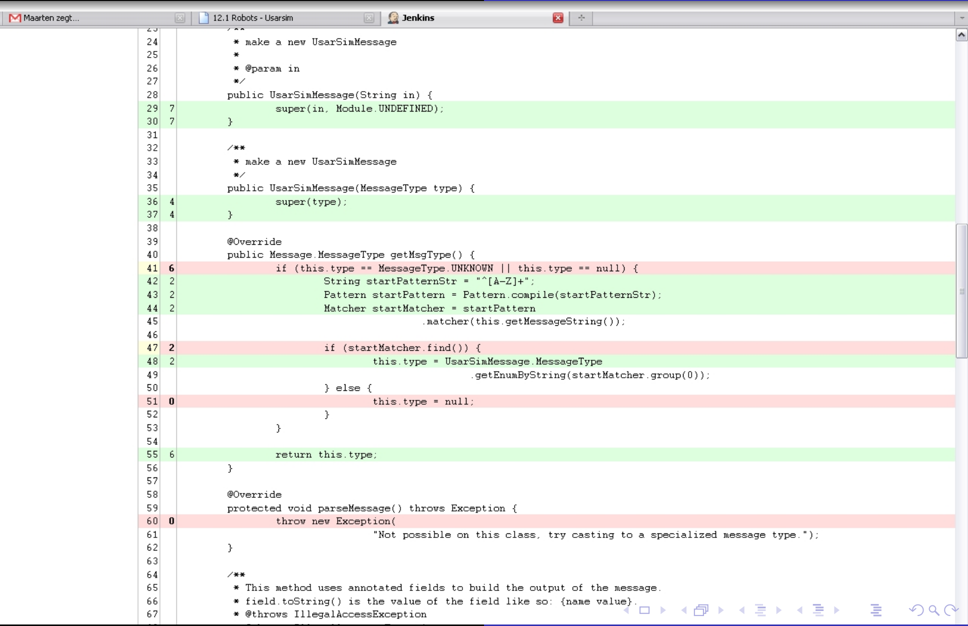
Trend

Project Coverage summary

Name	Classes		Conditionals		Files		Lines		Methods		Packages	
Cobertura Coverage Report	56%	36/64	18%	54/295	61%	30/49	30%	535/1810	37%	145/389	100%	10/10

Coverage Breakdown by Package

Name	Classes		Conditionals		Files		Lines		Methods	
AP2DX.specializedMessages	31%	5/16	5%	1/20	27%	4/15	23%	66/287	24%	20/85
AP2DX.usarsim	100%	2/2	69%	11/16	100%	2/2	69%	34/49	78%	7/9
AP2DX.coordinator	40%	2/5	14%	2/14	33%	1/3	27%	29/106	50%	5/10
AP2DX.reflex	100%	2/2	7%	2/27	100%	1/1	23%	14/62	38%	5/13
AP2DX.sensor	67%	2/3	4%	1/23	50%	1/2	13%	19/149	27%	6/22
AP2DX.mapper	60%	3/5	0%	0/15	100%	3/3	28%	34/121	26%	7/27
AP2DX	64%	9/14	14%	10/69	100%	8/8	35%	168/478	51%	41/81
AP2DX.planner	33%	1/3	0%	0/27	50%	1/2	8%	9/112	14%	3/21
AP2DX.usarsim.specialized	67%	8/12	37%	26/70	67%	8/12	38%	147/387	40%	46/115
AP2DX.motor	100%	2/2	7%	1/14	100%	1/1	25%	15/59	83%	5/6



```
      * make a new UsarSimMessage
      *
      * @param in
      */
     public UsarSimMessage(String in) {
7            super(in, Module.UNDEFINED);
7    }

     /**
      * make a new UsarSimMessage
      */
     public UsarSimMessage(MessageType type) {
4            super(type);
4    }

     @Override
     public Message.MessageType getMsgType() {
6            if (this.type == MessageType.UNKNOWN || this.type == null) {
2                    String startPatternStr = "^[A-Z]+";
2                    Pattern startPattern = Pattern.compile(startPatternStr);
2                    Matcher startMatcher = startPattern
                                     .matcher(this.getMessageString());

2                    if (startMatcher.find()) {
2                            this.type = UsarSimMessage.MessageType
                                             .getEnumByString(startMatcher.group(0));
                     } else {
0                            this.type = null;
                     }
             }

6            return this.type;
     }

     @Override
     protected void parseMessage() throws Exception {
0            throw new Exception(
                             "Not possible on this class, try casting to a specialized message type.");
     }

     /**
      * This method uses annotated fields to build the output of the message.
      * field.toString() is the value of the field like so: {name value}.
      * @throws IllegalAccessException
```