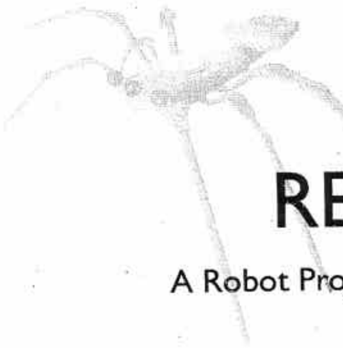


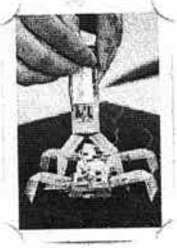
REACT

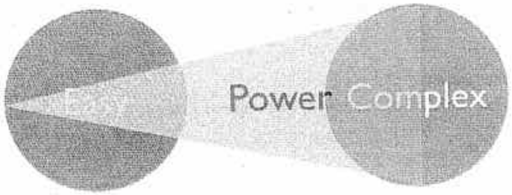
A Robot Programming Language

Sergio Benitez

Challenges

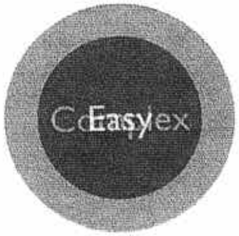
- Diverse Audience
- Many Possible Applications
- Complexity
- Fragile Robots & Conditions





Possible Solutions

• Diverse Audience	=> Layered Power
• Many Possible Applications	=> Powerful Primitives
• Complexity	=> Abstraction, Layering
• Fragile Robots/Conditions	=> Enforceable Constraints



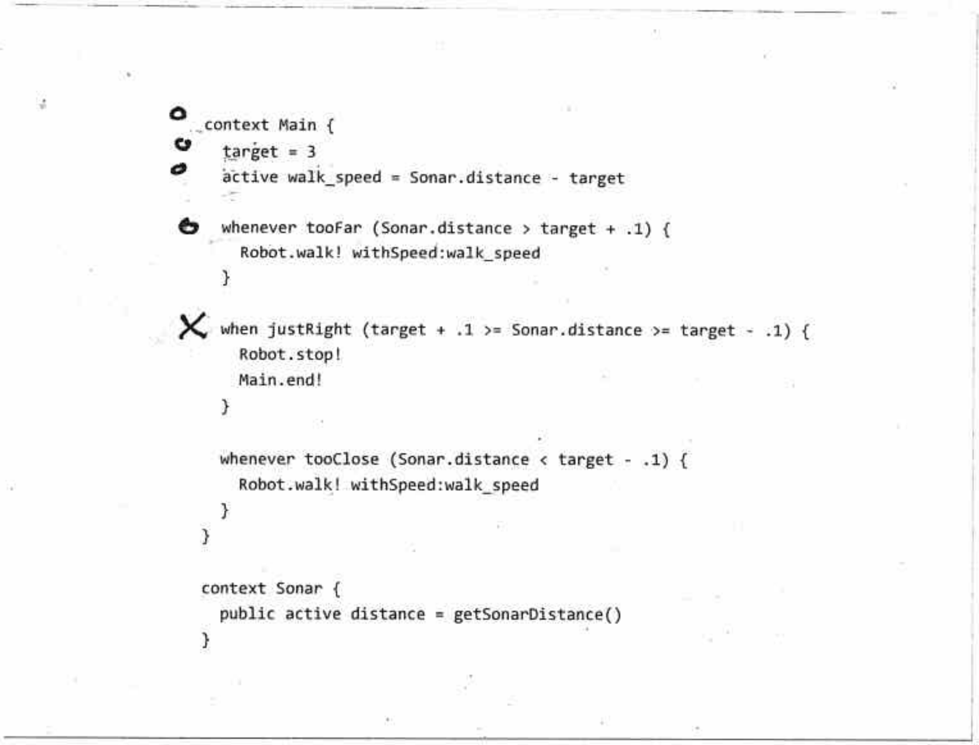
REACT:

Event-Driven

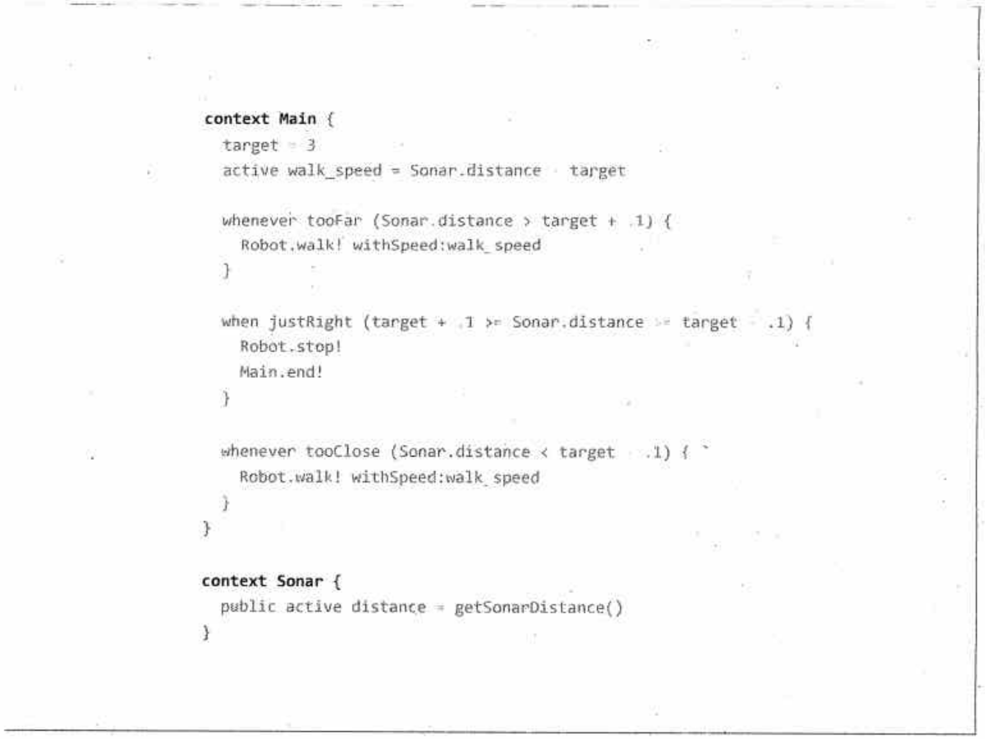
Asynchronous

Concurrent

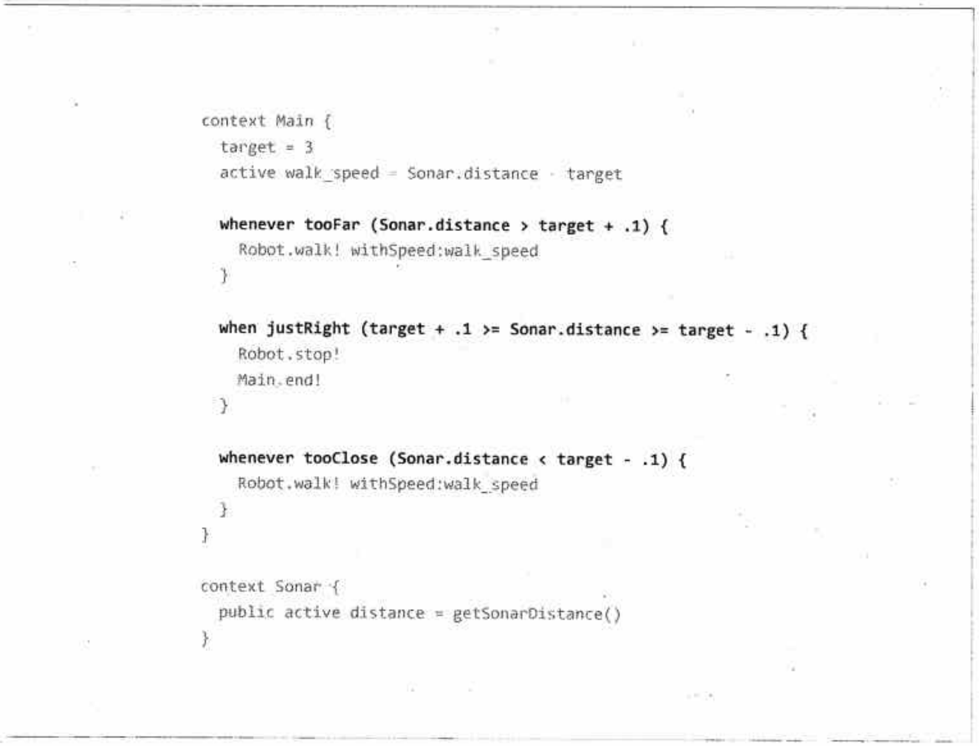
Turing-complete



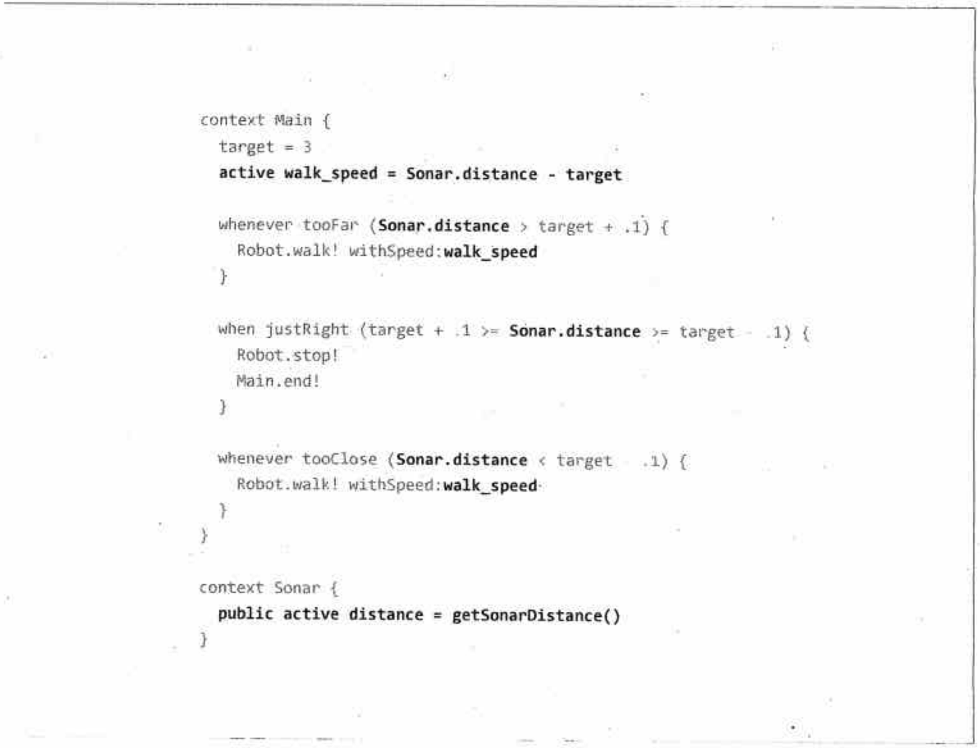
```
o  context Main {
c    target = 3
o    active walk_speed = Sonar.distance - target

6    whenever tooFar (Sonar.distance > target + .1) {
       Robot.walk! withSpeed:walk_speed
     }

X    when justRight (target + .1 >= Sonar.distance >= target - .1) {
       Robot.stop!
       Main.end!
     }

     whenever tooClose (Sonar.distance < target - .1) {
       Robot.walk! withSpeed:walk_speed
     }
   }

   context Sonar {
     public active distance = getSonarDistance()
   }
```



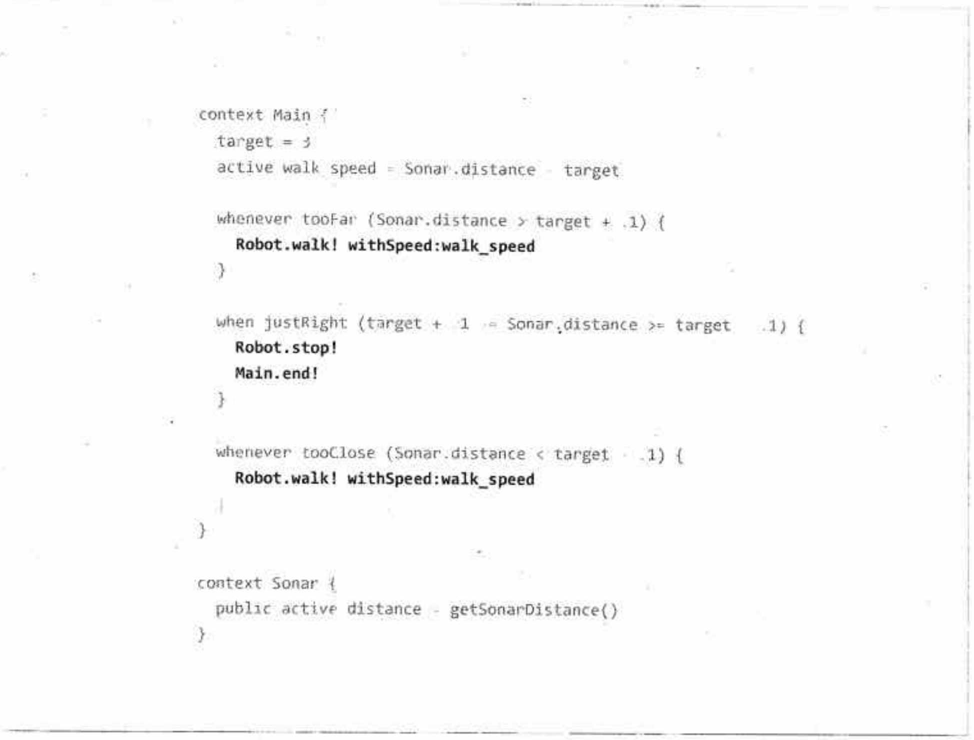
```
context Main {
  target = 3
  active walk_speed = Sonar.distance - target

  whenever tooFar (Sonar.distance > target + .1) {
    Robot.walk! withSpeed:walk_speed
  }

  when justRight (target + .1 >= Sonar.distance >= target - .1) {
    Robot.stop!
    Main.end!
  }

  whenever tooClose (Sonar.distance < target - .1) {
    Robot.walk! withSpeed:walk_speed
  }
}

context Sonar {
  public active distance = getSonarDistance()
}
```



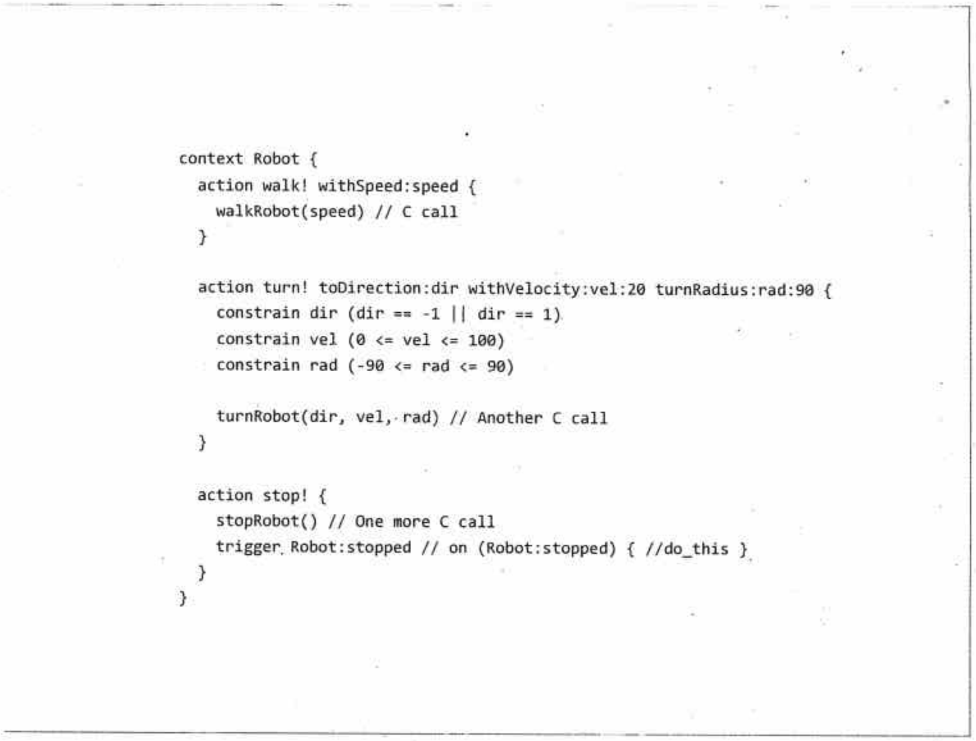
```
context Main {
  target = 3
  active walk_speed = Sonar.distance - target

  whenever tooFar (Sonar.distance > target + .1) {
    Robot.walk! withSpeed:walk_speed
  }

  when justRight (target + .1 >= Sonar.distance >= target - .1) {
    Robot.stop!
    Main.end!
  }

  whenever tooClose (Sonar.distance < target - .1) {
    Robot.walk! withSpeed:walk_speed
  }
}

context Sonar {
  public active distance = getSonarDistance()
}
```



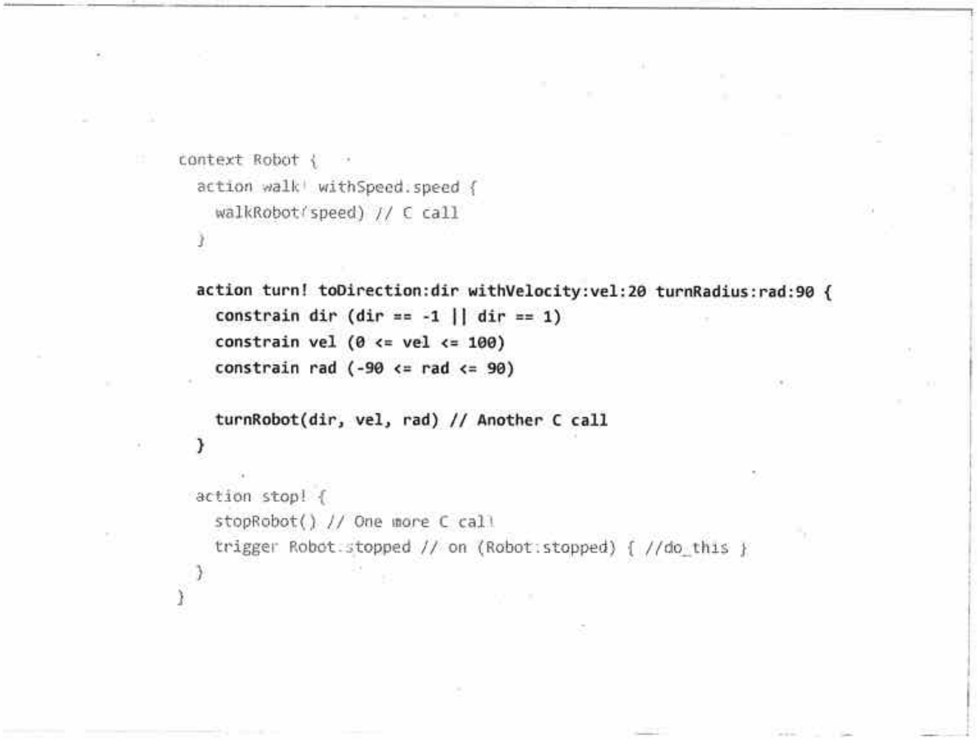
```
context Main {
  target = 3
  active walk_speed = Sonar.distance - target

  whenever tooFar (Sonar.distance > target + .1) {
    Robot.walk! withSpeed:walk_speed
  }

  when justRight (target + .1 >= Sonar.distance >= target - .1) {
    Robot.stop!
    Main.end!
  }

  whenever tooClose (Sonar.distance < target - .1) {
    Robot.walk! withSpeed:walk_speed
  }
}

context Sonar {
  public active distance = getSonarDistance()
}
```



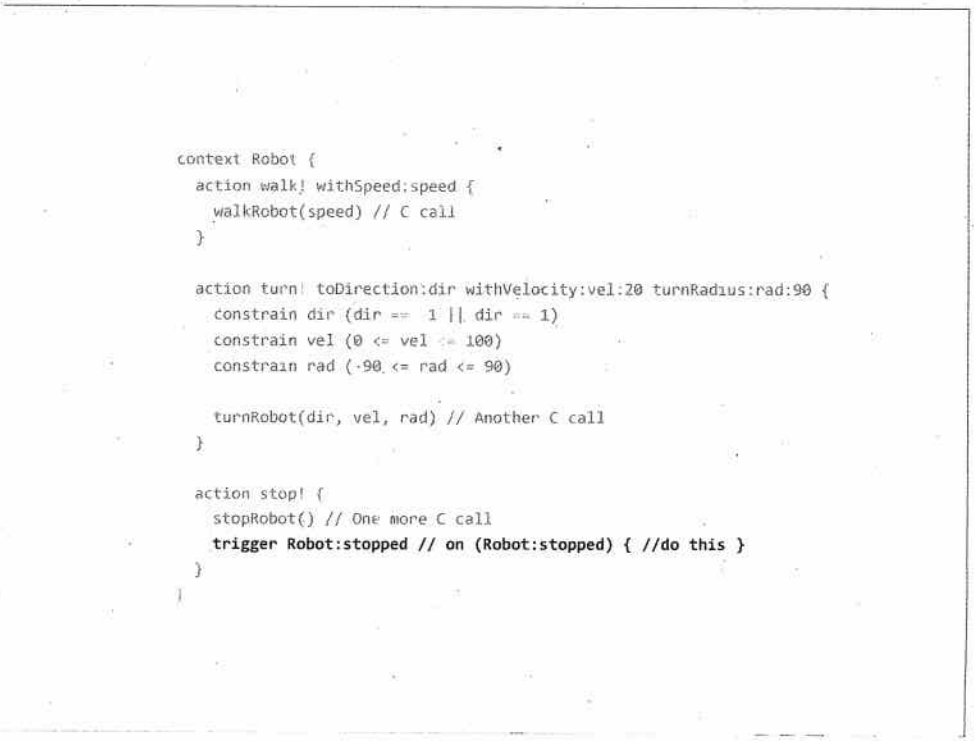
```
context Main {
  target = 3
  active walk_speed = Sonar.distance - target

  whenever tooFar (Sonar.distance > target + .1) {
    Robot.walk! withSpeed:walk_speed
  }

  when justRight (target + .1 >= Sonar.distance >= target - .1) {
    Robot.stop!
    Main.end!
  }

  whenever tooClose (Sonar.distance < target - .1) {
    Robot.walk! withSpeed:walk_speed
  }
}

context Sonar {
  public active distance = getSonarDistance()
}
```



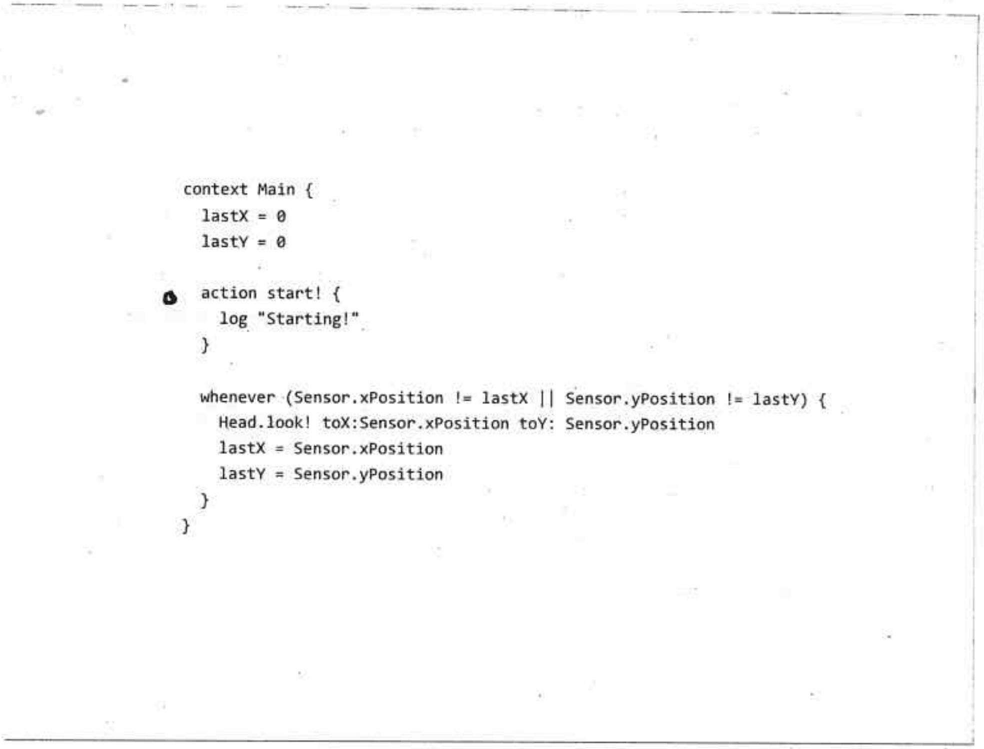
```
context Robot {
  action walk! withSpeed:speed {
    walkRobot(speed) // C call
  }

  action turn! toDirection:dir withVelocity:vel:20 turnRadius:rad:90 {
    constrain dir (dir == -1 || dir == 1)
    constrain vel (0 <= vel <= 100)
    constrain rad (-90 <= rad <= 90)

    turnRobot(dir, vel, rad) // Another C call
  }

  action stop! {
    stopRobot() // One more C call
    trigger Robot:stopped // on (Robot:stopped) { //do_this }
  }
}
```



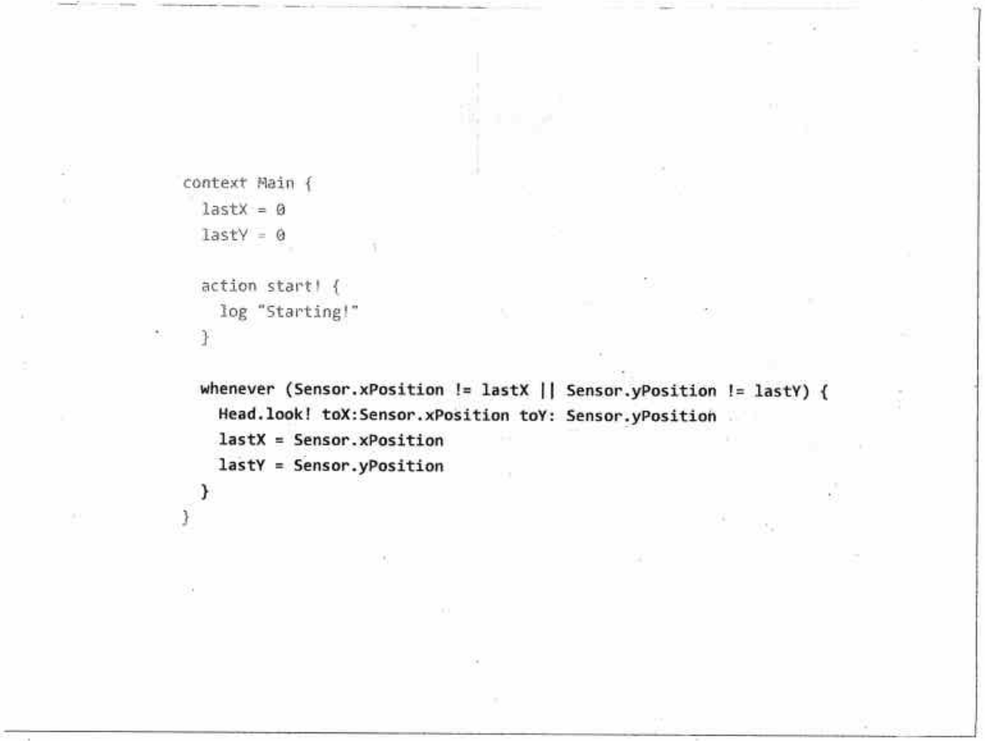
```
context Robot {
  action walk! withSpeed:speed {
    walkRobot(speed) // C call
  }

  action turn! toDirection:dir withVelocity:vel:20 turnRadius:rad:90 {
    constrain dir (dir == -1 || dir == 1)
    constrain vel (0 <= vel <= 100)
    constrain rad (-90 <= rad <= 90)

    turnRobot(dir, vel, rad) // Another C call
  }

  action stop! {
    stopRobot() // One more C call
    trigger Robot:stopped // on (Robot:stopped) { //do_this }
  }
}
```



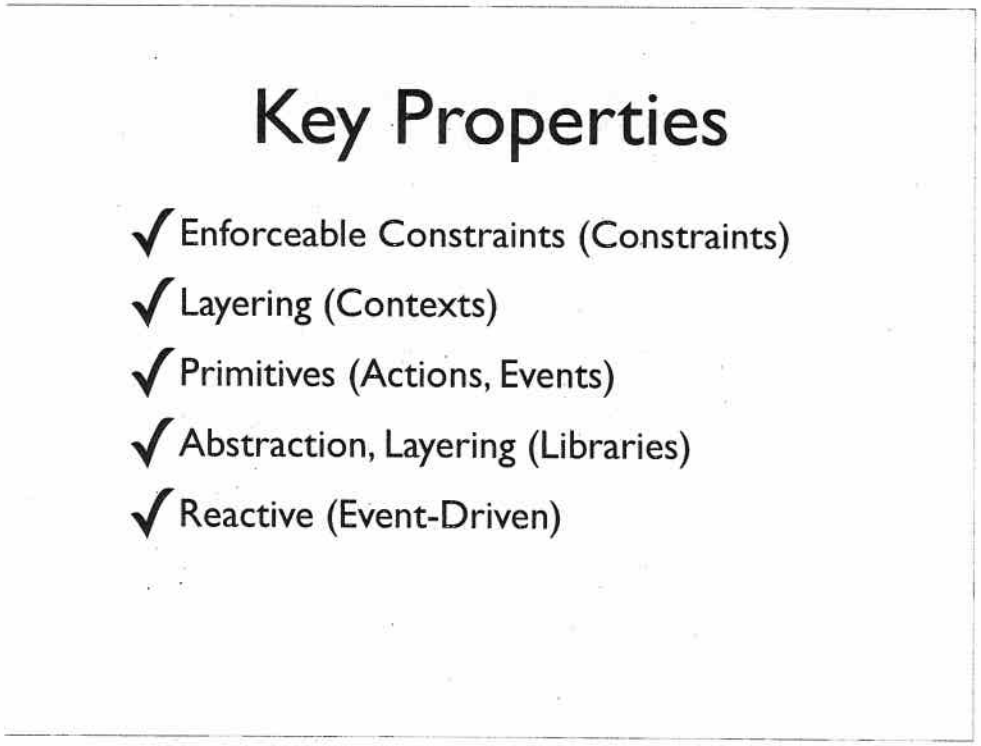
```
context Robot {
  action walk! withSpeed:speed {
    walkRobot(speed) // C call
  }

  action turn! toDirection:dir withVelocity:vel:20 turnRadius:rad:90 {
    constrain dir (dir == -1 || dir == 1)
    constrain vel (0 <= vel <= 100)
    constrain rad (-90 <= rad <= 90)

    turnRobot(dir, vel, rad) // Another C call
  }

  action stop! {
    stopRobot() // One more C call
    trigger Robot:stopped // on (Robot:stopped) { //do this }
  }
}
```



```
context Main {
  lastX = 0
  lastY = 0

  action start! {
    log "Starting!"
  }

  whenever (Sensor.xPosition != lastX || Sensor.yPosition != lastY) {
    Head.look! toX:Sensor.xPosition toY: Sensor.yPosition
    lastX = Sensor.xPosition
    lastY = Sensor.yPosition
  }
}
```

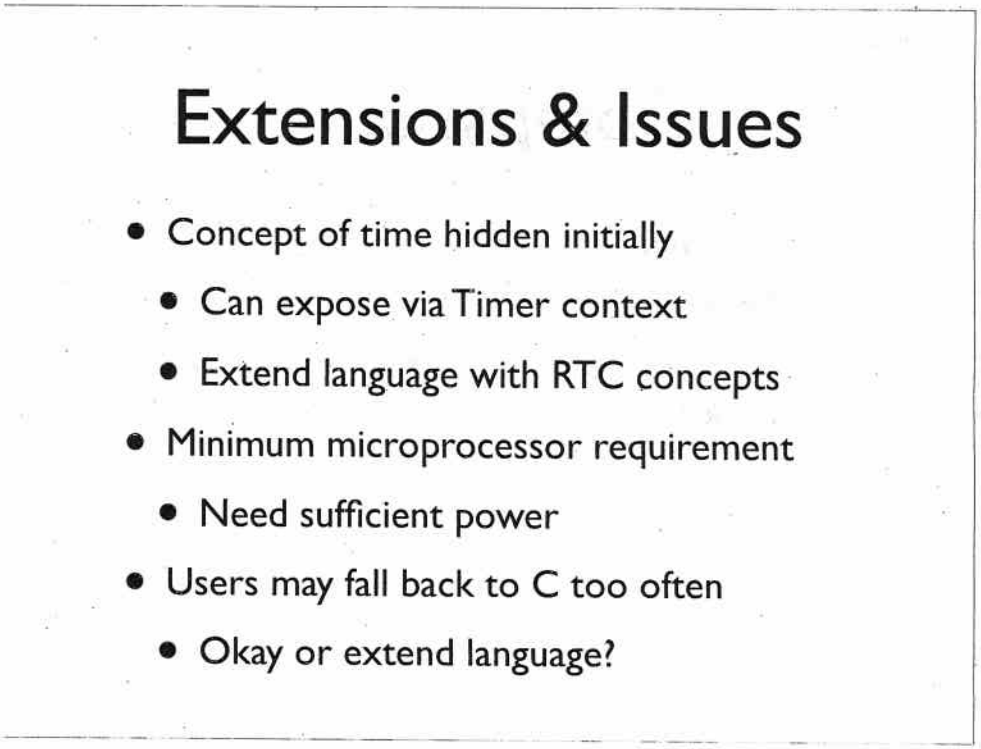


```
context Main {
  lastX = 0
  lastY = 0

  action start! {
    log "Starting!"
  }

  whenever (Sensor.xPosition != lastX || Sensor.yPosition != lastY) {
    Head.look! toX:Sensor.xPosition toY: Sensor.yPosition
    lastX = Sensor.xPosition
    lastY = Sensor.yPosition
  }
}
```



Key Properties

- ✓ Enforceable Constraints (Constraints)
- ✓ Layering (Contexts)
- ✓ Primitives (Actions, Events)
- ✓ Abstraction, Layering (Libraries)
- ✓ Reactive (Event-Driven)



Extensions & Issues

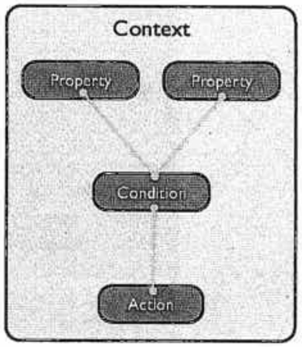
- Concept of time hidden initially
  - Can expose via Timer context
  - Extend language with RTC concepts
- Minimum microprocessor requirement
  - Need sufficient power
- Users may fall back to C too often
  - Okay or extend language?

Risks

- Too Simple
- Too Complex
- Too "Different"
- Incomplete Controller Library

Graphical Version

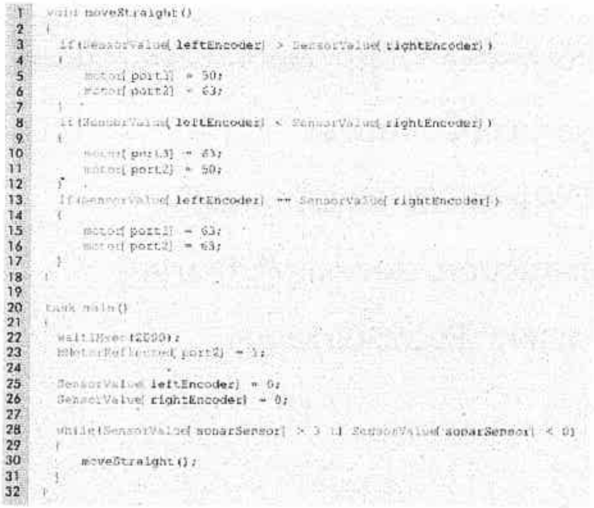
- Unique UI Elements
  - Properties
  - Conditions
  - Actions
- "Linking" mechanism
- Contexts



Comparison

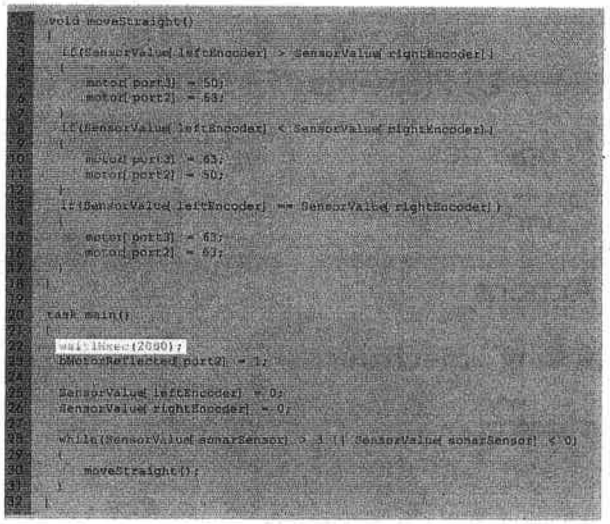
- Text Based:
  - RoboForth
  - RobotC
  - WPI Robotics Library
- Graphical:
  - RoboLab
  - EasyC
  - NXT-G

RobotC



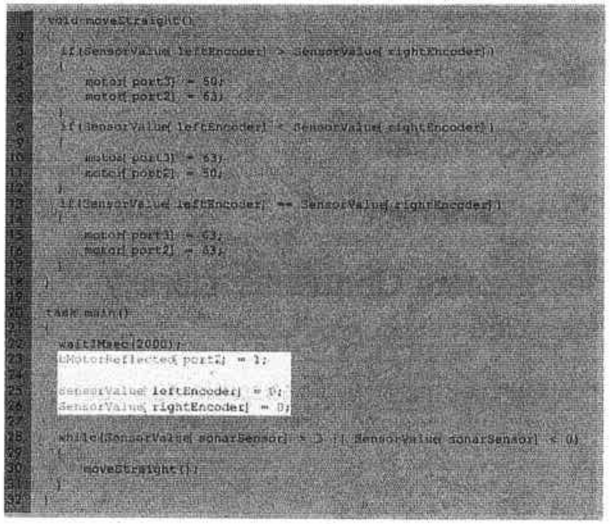
```
void moveStraight()
{
  if(SensorValue[leftEncoder] > SensorValue[rightEncoder])
  {
     motor[port1] = 50;
     motor[port2] = 63;
  }
  if(SensorValue[leftEncoder] < SensorValue[rightEncoder])
  {
     motor[port1] = 63;
     motor[port2] = 50;
  }
  if(SensorValue[leftEncoder] == SensorValue[rightEncoder])
  {
     motor[port1] = 63;
     motor[port2] = 63;
  }
}

task main()
{
  wait1Msec(2000);
  bMotorReflected[port2] = 1;

  SensorValue[leftEncoder] = 0;
  SensorValue[rightEncoder] = 0;

  while(SensorValue[sonarSensor] > 3 || SensorValue[sonarSensor] < 0)
  {
     moveStraight();
  }
}
```

RobotC



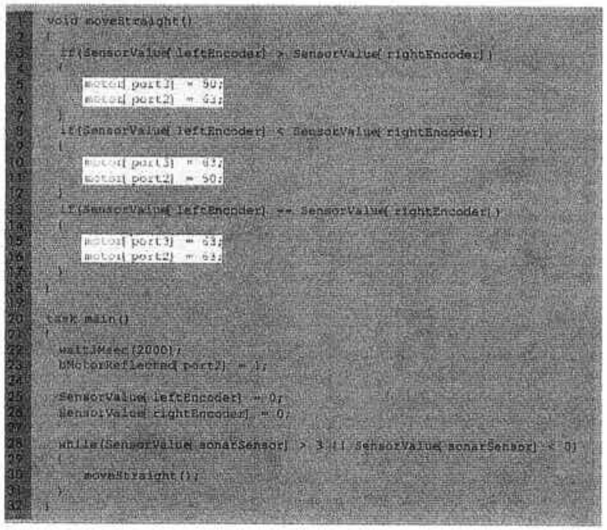
```
void moveStraight()
{
  if(SensorValue[leftEncoder] > SensorValue[rightEncoder])
  {
    motor[port3] = 50;
    motor[port2] = 63;
  }
  if(SensorValue[leftEncoder] < SensorValue[rightEncoder])
  {
    motor[port3] = 63;
    motor[port2] = 50;
  }
  if(SensorValue[leftEncoder] == SensorValue[rightEncoder])
  {
    motor[port3] = 63;
    motor[port2] = 63;
  }
}

task main()
{
  wait1Msec(2000);
  bMotorReflected[port2] = 1;

  SensorValue[leftEncoder] = 0;
  SensorValue[rightEncoder] = 0;

  while(SensorValue[sonarSensor] > 3 || SensorValue[sonarSensor] < 0)
  {
    moveStraight();
  }
}
```

RobotC



```
void moveStraight()
{
  if(SensorValue[leftEncoder] > SensorValue[rightEncoder])
  {
    motor[port3] = 50;
    motor[port2] = 63;
  }
  if(SensorValue[leftEncoder] < SensorValue[rightEncoder])
  {
    motor[port3] = 63;
    motor[port2] = 50;
  }
  if(SensorValue[leftEncoder] == SensorValue[rightEncoder])
  {
    motor[port3] = 63;
    motor[port2] = 63;
  }
}

task main()
{
  wait1Msec(2000);
  bMotorReflected[port2] = 1;

  SensorValue[leftEncoder] = 0;
  SensorValue[rightEncoder] = 0;

  while(SensorValue[sonarSensor] > 3 || SensorValue[sonarSensor] < 0)
  {
    moveStraight();
  }
}
```

RobotC



```
void moveStraight()
{
  if(SensorValue[leftEncoder] > SensorValue[rightEncoder])
  {
    motor[port3] = 50;
    motor[port2] = 63;
  }
  if(SensorValue[leftEncoder] < SensorValue[rightEncoder])
  {
    motor[port3] = 63;
    motor[port2] = 50;
  }
  if(SensorValue[leftEncoder] == SensorValue[rightEncoder])
  {
    motor[port3] = 63;
    motor[port2] = 63;
  }
}

task main()
{
  wait1Msec(2000);
  bMotorReflected[port2] = 1;

  SensorValue[leftEncoder] = 0;
  SensorValue[rightEncoder] = 0;

  while(SensorValue[sonarSensor] > 3 || SensorValue[sonarSensor] < 0)
  {
    moveStraight();
  }
}
```

RobotC

```
task main() {
  bMotorReflected[port2] = 1;

  SensorValue[leftEncoder] = 0;
  SensorValue[rightEncoder] = 0;

  while (SensorValue[sonarSensor] > 3 || SensorValue[sonarSensor] < 0) {
    moveStraight(SensorValue[sonarSensor] - 3);
  }
}
```

REACT

```
context Main {
  whenever (Sonar.distance > 3 || Sonar.invalid) {
    Robot.walk! withSpeed: Sonar.distance - 3
  }
}
```

RobotC Pros/Cons

- Pros:
  - Extremely powerful (C backbone)
- Cons:
  - Requires users to understand C, timing
  - Steep learning curve
  - Polling mechanism, impossible to react

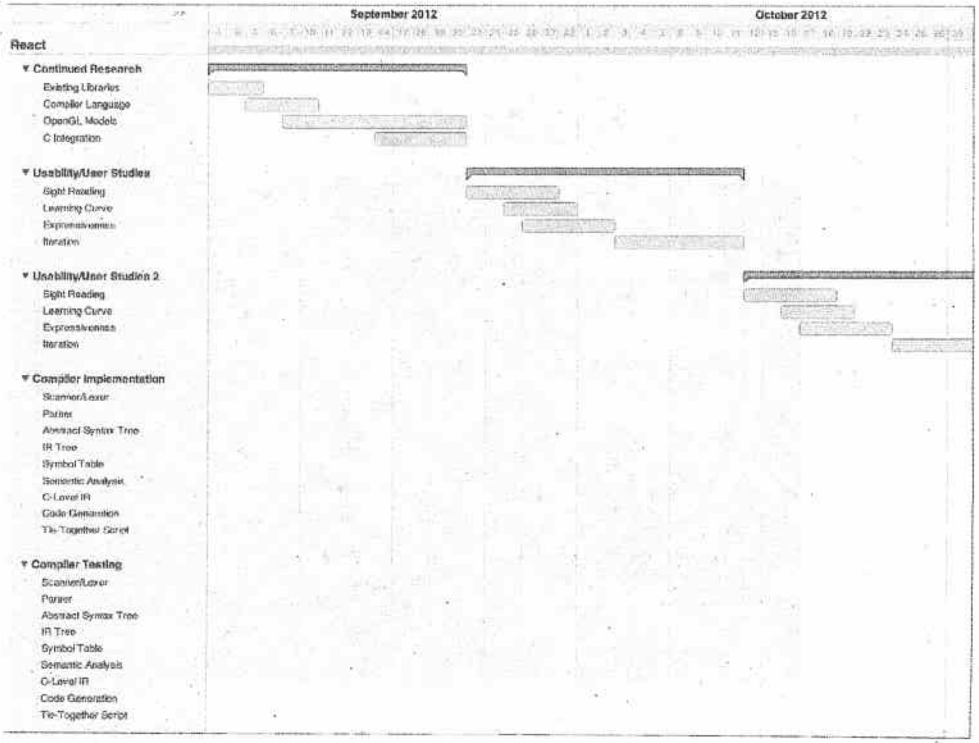
Possible Solutions

Fragile Robots/Conditions	=> Enforceable Constraints
Diverse Audience	=> Layered Power
Many Possible Applications	=> Powerful Primitives
Unthinkable Complexity	=> Abstraction, Layering

Complex
Easy

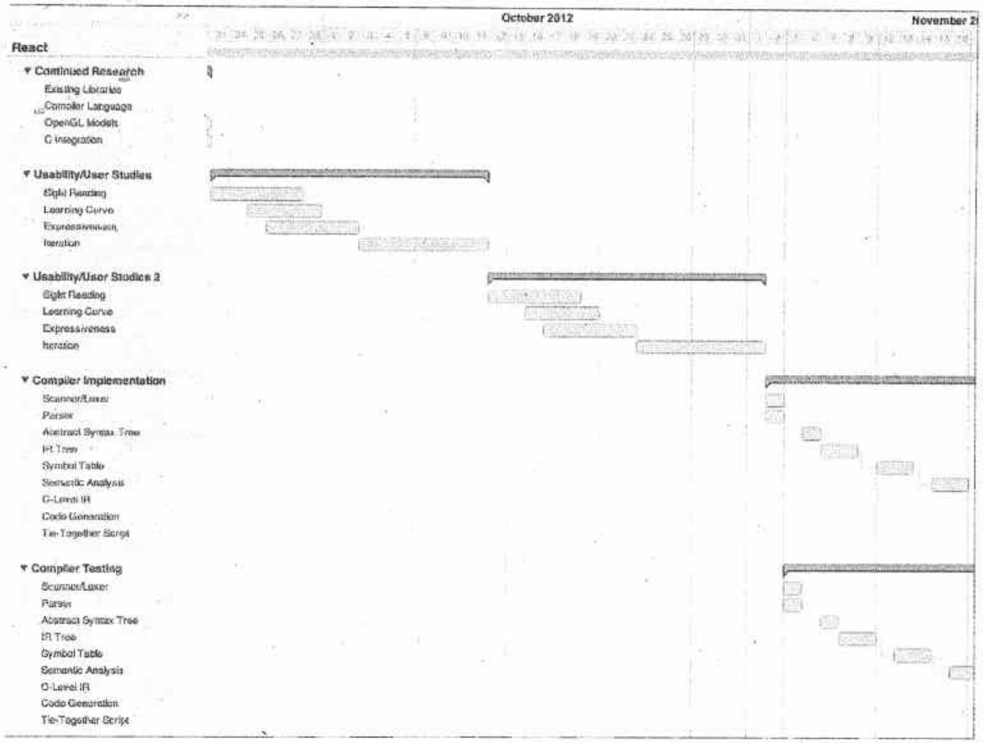
Language Comparison

Language	Easy To Use	Resource Constraints	Temporal Constraints	Event Driven	Async	OO
RobotC	✗	✗	✓	✗	✗	✗
RForth	✓	✗	✓	✗	✗	✗
WPI Lib	✗	✗	✓	✗	✗	✓
REACT	✓	✓	✗	✓	✓	✓



React

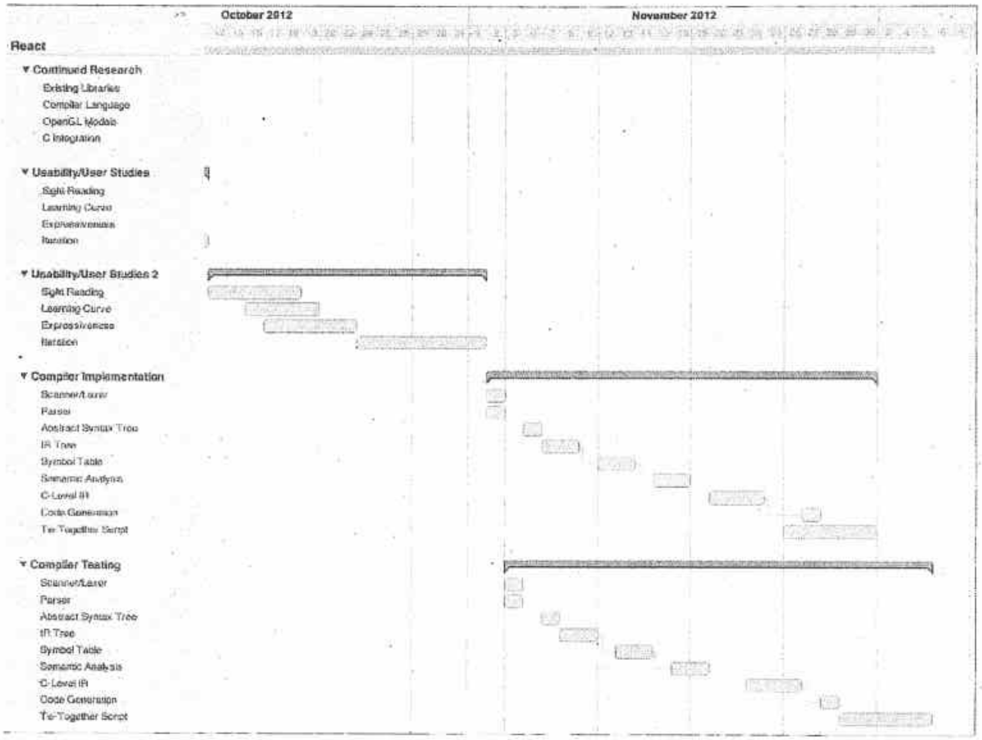
- Continued Research
  - Existing Libraries
  - Compiler Language
  - OpenGL Models
  - C Integration
- Usability/User Studies
  - Sight Reading
  - Learning Curve
  - Expressiveness
  - Iteration
- Usability/User Studies 2
  - Sight Reading
  - Learning Curve
  - Expressiveness
  - Iteration
- Compiler Implementation
  - Scanner/Lexer
  - Parser
  - Abstract Syntax Tree
  - IR Tree
  - Symbol Table
  - Semantic Analysis
  - C-Level IR
  - Code Generation
  - Tie-Together Script
- Compiler Testing
  - Scanner/Lexer
  - Parser
  - Abstract Syntax Tree
  - IR Tree
  - Symbol Table
  - Semantic Analysis
  - C-Level IR
  - Code Generation
  - Tie-Together Script



October 2012 November 2012

React

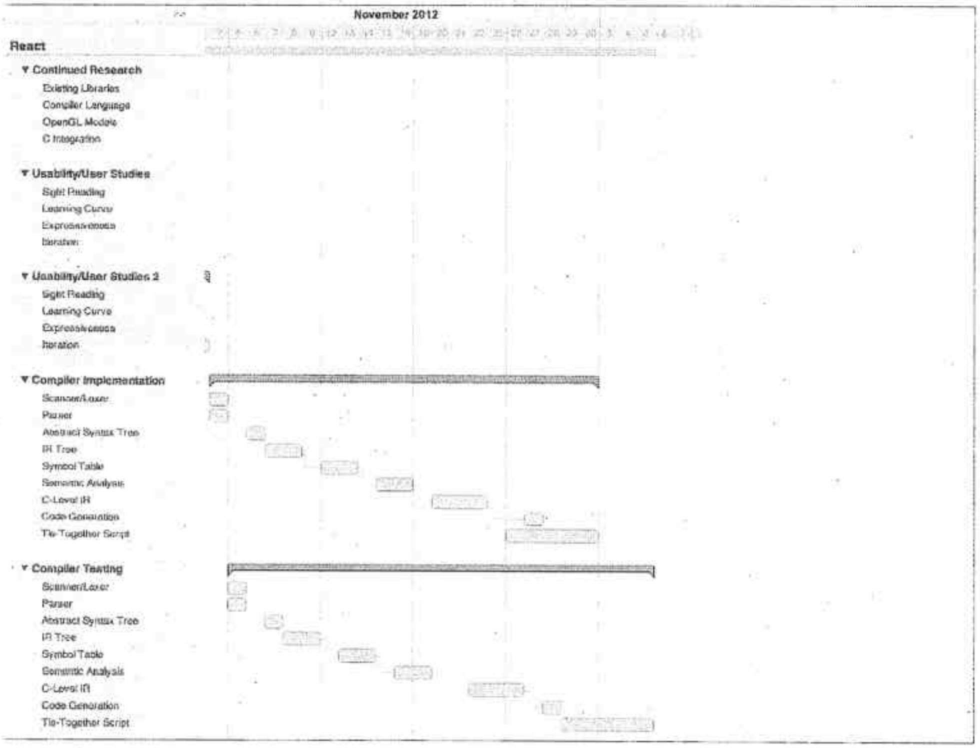
- ▼ Continued Research
  - Existing Libraries
  - Compiler Language
  - OpenGL Models
  - C Integration
- ▼ Usability/User Studies
  - Sight Reading
  - Learning Curve
  - Expressiveness
  - Iteration
- ▼ Usability/User Studies 2
  - Sight Reading
  - Learning Curve
  - Expressiveness
  - Iteration
- ▼ Compiler Implementation
  - Scanner/Lexer
  - Parser
  - Abstract Syntax Tree
  - IR Tree
  - Symbol Table
  - Semantic Analysis
  - C-Level IR
  - Code Generation
  - Tie-Together Script
- ▼ Compiler Testing
  - Scanner/Lexer
  - Parser
  - Abstract Syntax Tree
  - IR Tree
  - Symbol Table
  - Semantic Analysis
  - C-Level IR
  - Code Generation
  - Tie-Together Script



October 2012 November 2012

React

- ▼ Continued Research
  - Existing Libraries
  - Compiler Language
  - OpenGL Models
  - C Integration
- ▼ Usability/User Studies
  - Sight Reading
  - Learning Curve
  - Expressiveness
  - Iteration
- ▼ Usability/User Studies 2
  - Sight Reading
  - Learning Curve
  - Expressiveness
  - Iteration
- ▼ Compiler Implementation
  - Scanner/Lexer
  - Parser
  - Abstract Syntax Tree
  - IR Tree
  - Symbol Table
  - Semantic Analysis
  - C-Level IR
  - Code Generation
  - Tie-Together Script
- ▼ Compiler Testing
  - Scanner/Lexer
  - Parser
  - Abstract Syntax Tree
  - IR Tree
  - Symbol Table
  - Semantic Analysis
  - C-Level IR
  - Code Generation
  - Tie-Together Script



November 2012

React

- ▼ Continued Research
  - Existing Libraries
  - Compiler Language
  - OpenGL Models
  - C Integration
- ▼ Usability/User Studies
  - Sight Reading
  - Learning Curve
  - Expressiveness
  - Iteration
- ▼ Usability/User Studies 2
  - Sight Reading
  - Learning Curve
  - Expressiveness
  - Iteration
- ▼ Compiler Implementation
  - Scanner/Lexer
  - Parser
  - Abstract Syntax Tree
  - IR Tree
  - Symbol Table
  - Semantic Analysis
  - C-Level IR
  - Code Generation
  - Tie-Together Script
- ▼ Compiler Testing
  - Scanner/Lexer
  - Parser
  - Abstract Syntax Tree
  - IR Tree
  - Symbol Table
  - Semantic Analysis
  - C-Level IR
  - Code Generation
  - Tie-Together Script

REACT:

Event-Driven

Asynchronous

Concurrent

Turing-complete