Context-Aware Annotations for Distributed Mobile Applications

Daniel Graff[1] Jan Richling[1] Tammo M. Stupp[1]
Matthias Werner[2]

[1]Communication and Operating Systems Group, Berlin University of Technology, Germany

[2]Operating Systems Group, Chemnitz University of Technology, Germany

23.02.2011

Overview

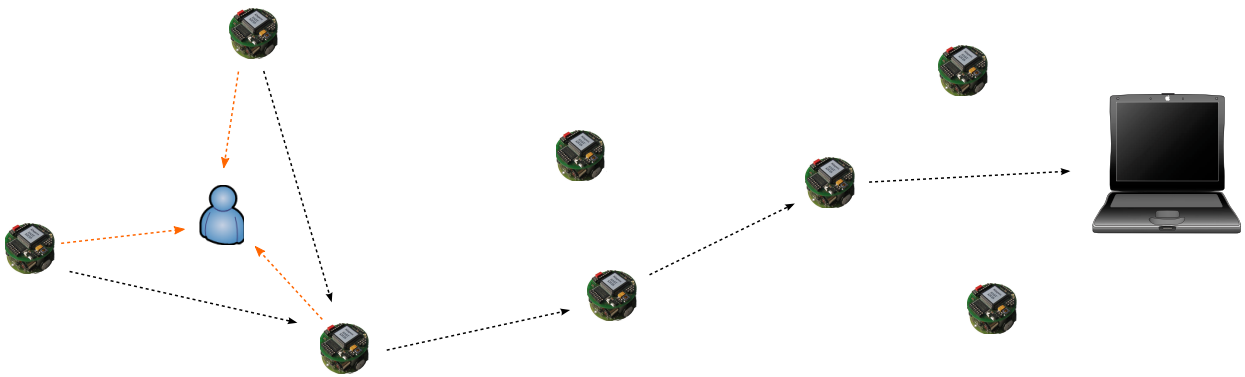
1. Introduction
2. Model
3. Context-Aware Annotations
4. Proof-Of-Concept
5. Conclusion

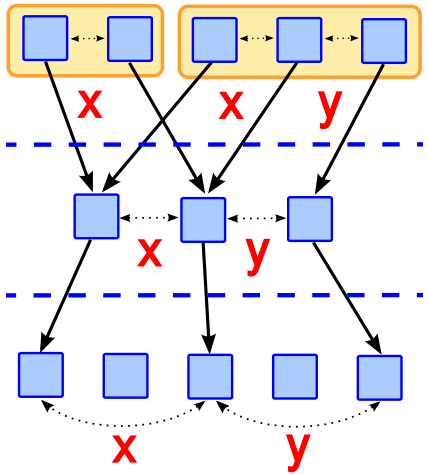
Introduction

- Transparency
  - Major design aspect
  - Distribution is often error-prone
    $\rightarrow$ race conditions
  - Desired transparencies
    - Access transparency
    - Location transparency
- Awareness
  - Concept emerged out of ubiquitous computing
  - Applications behavior depends on context
  - Location and motion required by navigation system

Approach

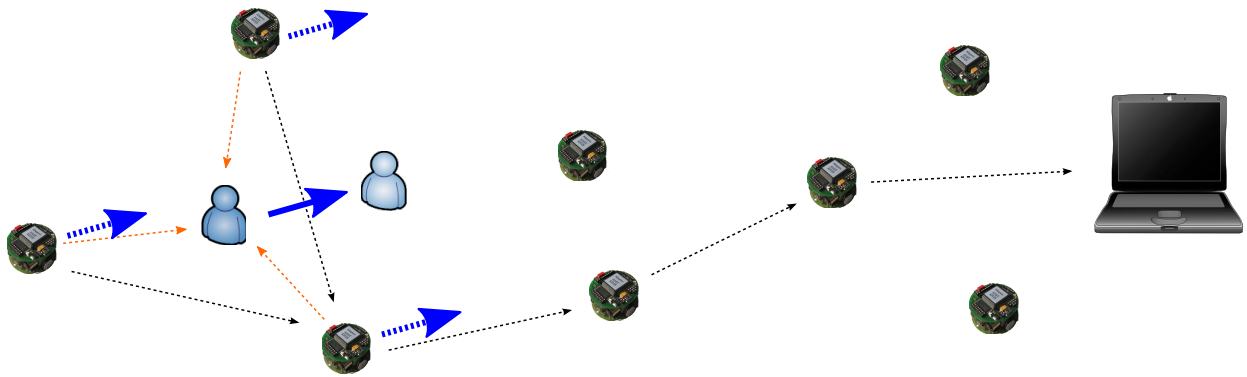
- Target domain: cyber-physical systems (CPS)
- Concept: The “object“ is the application
- Systemic description
  - Actions (impact) on the ”object“
  - E.g., *observe*, *move*
  - Solution will emerge implicit
- Abstraction
  - Location/ distribution transparencies
  - Virtualization of Sensors/ actuators

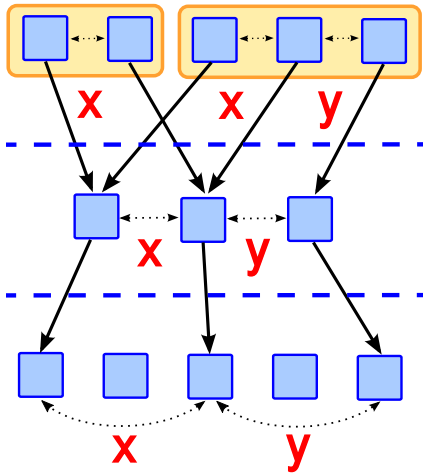




Approach

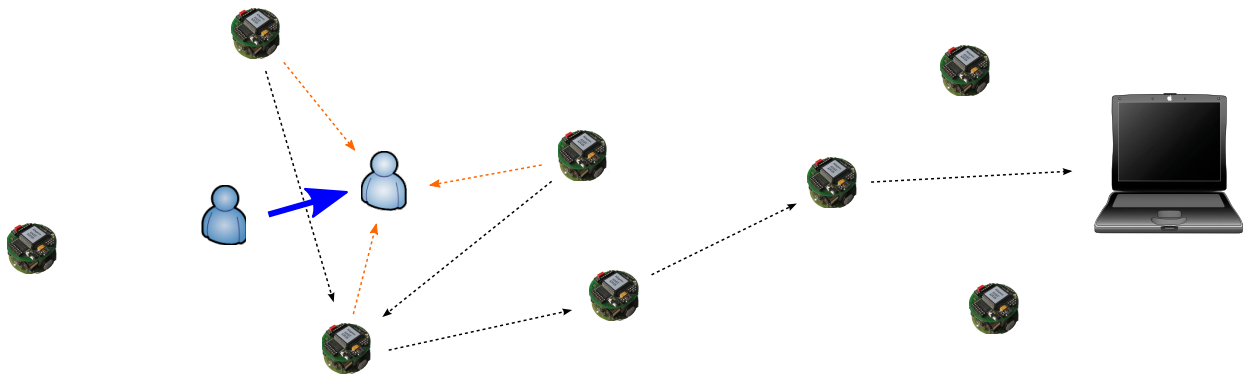
- Target domain: cyber-physical systems (CPS)
- Concept: The “object“ is the application
- Systemic description
  - Actions (impact) on the ”object“
  - E.g., *observe*, *move*
  - Solution will emerge implicit
- Abstraction
  - Location/ distribution transparencies
  - Virtualization of Sensors/ actuators

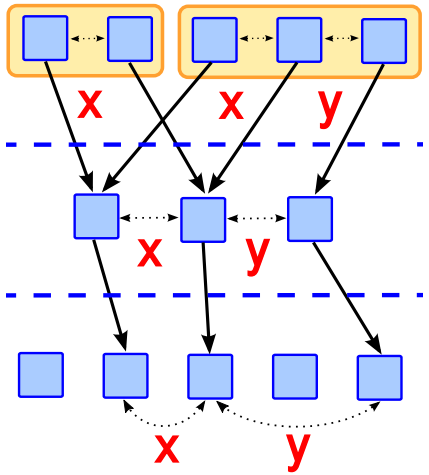




Approach

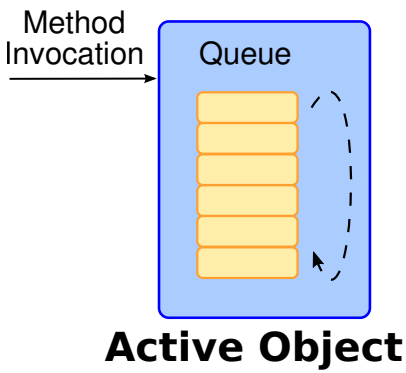
- Target domain: cyber-physical systems (CPS)
- Concept: The “object“ is the application
- Systemic description
  - Actions (impact) on the ”object“
  - E.g., *observe*, *move*
  - Solution will emerge implicit
- Abstraction
  - Location/ distribution transparencies
  - Virtualization of Sensors/ actuators

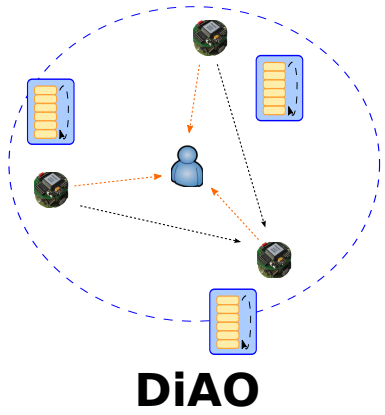




Model: Distributed Active Objects (DiAOs)

- Solution: Distributed Active Objects (DiAOs)
- DiAO represents *controlled object*
  - Virtual hull
- Extend the concept of *active objects*
  - distribution
  - context-aware
- Objective: Perform actions on *controlled object*
  - Systemic view
  - Translated for distributed execution
    $\rightarrow$ in time and space





Context-Aware Annotations

- Actions (of DiAOs) can be annotated by context conditions
- General: Constraint programming is a well known paradigm
  $\rightarrow a < 10 \ \wedge \ b > 3$
- We use: Spatio-temporal constraints (for context awareness)
  - Restriction in time and space
  - $d(X1, X2) == 5$
  - $now() - last >= 500$
- Different types of annotations
  - Goal
  - Action
  - Invariant

Context-Aware Annotations - Actions

- Actions are annotated methods
- Different execution semantic
  - Methods are invoked (synchronous)
  - Actions are activated (asynchronous)
- Run as long as conditions are valid
- Need to be deactivated explicitly

```
@Constraint(a == true)
A() {
   b = true;
   deactivate();
}
```

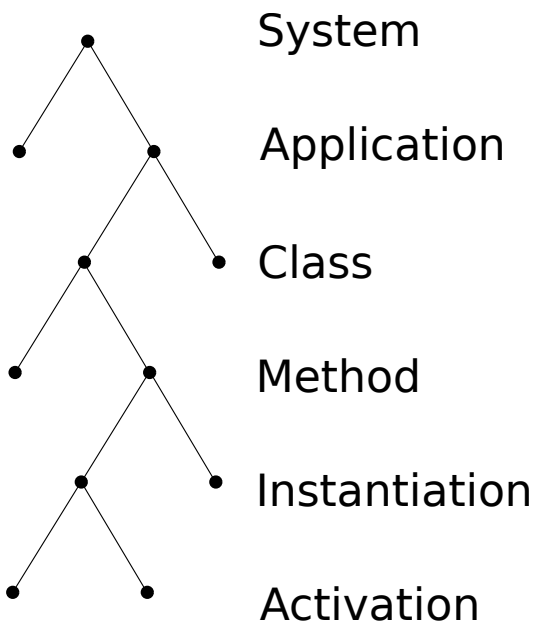
```
@Constraint(b == true)
B() {
   c = true
   deactivate();
}
```

```
@Constraint(c == true)
C() {
   ..
   deactivate();
}
```

Context-Aware Annotations - Semantics

- Execution semantic depends on place of annotation
- Hierarchical organized constraint sets
- Logically conjuncted
- 2 constraint trees
  - Goal + Invariant
  - Action + Invariant

	Goal	Invariant	Action
System		x	
Application	x	x	
Class	x	x	
Method			x
Instantiation	x	x	
Activation			x



Action-tree

Language Extension (1)

```
@Constraint(G1)(foreach(Pos Xi = cameras[i].pos),
  distance (X1, X2) == D,
  distance (X2, X3) == D,
  distance (X1, X3) == D,
  distance (X1, observationObject.pos) == D1,
  distance (X2, observationObject.pos) == D1,
  distance (X3, observationObject.pos) == D1,
  D1 in [..], long last = now())
@Invariant(D > d)
public class ProducerDiAO extends ObservationDiAO {

  @PhysicalEntity(uid=4711)
  private PhysicalEntity observationObject;

  @Capability(
   {Camera.Spectrum in [..], ..},
   {Camera.Spectrum in [..], ..},
   {Camera.Spectrum in [..], ..})
  private Camera[] cameras;

  private int snapshotId = 1;
  ...
```

Language Extension (1)

```
@Constraint(G1)(foreach(Pos Xi = cameras[i].pos),
  distance (X1, X2) == D,
  distance (X2, X3) == D,
  distance (X1, X3) == D,
  distance (X1, observationObject.pos) == D1,
  distance (X2, observationObject.pos) == D1,
  distance (X3, observationObject.pos) == D1,
  D1 in [..], long last = now())
@Invariant(D > d)
public class ProducerDiAO extends ObservationDiAO {

  @PhysicalEntity(uid=4711)
  private PhysicalEntity observationObject;

  @Capability(
   {Camera.Spectrum in [..], ..},
   {Camera.Spectrum in [..], ..},
   {Camera.Spectrum in [..], ..})
  private Camera[] cameras;

  private int snapshotId = 1;
  ...
```

Goal

Language Extension (1)

```
@Constraint(G1)(foreach(Pos Xi = cameras[i].pos),
  distance (X1, X2) == D,
  distance (X2, X3) == D,
  distance (X1, X3) == D,
  distance (X1, observationObject.pos) == D1,
  distance (X2, observationObject.pos) == D1,
  distance (X3, observationObject.pos) == D1,
  D1 in [..], long last = now())
@Invariant(D > d)                                    Invariant
public class ProducerDiAO extends ObservationDiAO {

  @PhysicalEntity(uid=4711)
  private PhysicalEntity observationObject;

  @Capability(
   {Camera.Spectrum in [..], ..},
   {Camera.Spectrum in [..], ..},
   {Camera.Spectrum in [..], ..})
  private Camera[] cameras;

  private int snapshotId = 1;
  ...
```

Language Extension (1)

```
@Constraint(G1)(foreach(Pos Xi = cameras[i].pos),
  distance (X1, X2) == D,
  distance (X2, X3) == D,
  distance (X1, X3) == D,
  distance (X1, observationObject.pos) == D1,
  distance (X2, observationObject.pos) == D1,
  distance (X3, observationObject.pos) == D1,
  D1 in [..], long last = now())
@Invariant(D > d)
public class ProducerDiAO extends ObservationDiAO {

  @PhysicalEntity(uid=4711)
  private PhysicalEntity observationObject;

  @Capability(
   {Camera.Spectrum in [..], ..},
   {Camera.Spectrum in [..], ..},
   {Camera.Spectrum in [..], ..})
  private Camera[] cameras;

  private int snapshotId = 1;
  ...
```

Physical Entity

Language Extension (1)

```
@Constraint(G1)(foreach(Pos Xi = cameras[i].pos),
  distance (X1, X2) == D,
  distance (X2, X3) == D,
  distance (X1, X3) == D,
  distance (X1, observationObject.pos) == D1,
  distance (X2, observationObject.pos) == D1,
  distance (X3, observationObject.pos) == D1,
  D1 in [..], long last = now())
@Invariant(D > d)
public class ProducerDiAO extends ObservationDiAO {

  @PhysicalEntity(uid=4711)
  private PhysicalEntity observationObject;

  @Capability(
   {Camera.Spectrum in [..], ..},
   {Camera.Spectrum in [..], ..},
   {Camera.Spectrum in [..], ..})
  private Camera[] cameras;

  private int snapshotId = 1;
  ...
```

Capability

Language Extension (2)

```
  @Constraint(G1, (now()-last) >= 500)
  public void actionObserve() {
    foreach(Camera camera :  cameras) {
      activate(camera.takePicture(
        snapshotId, observationObject));
    }
    snapshotId++;
    set(last, now());
  }

  @Constraint(
   foreach(int IDi = cameras[i].getPictureId),
   ID1 == ID2, ID2 == ID3)
  public void actionComposeSnapshot() {
    Snapshot snapshot = new Snapshot();
    foreach(Camera camera :  cameras) {
      snapshot.add(camera.getPictureWithId(get(ID1)))
    }
    snapshot.setId(ID1);
    queue.add(snapshot);
  }
}
```

Language Extension (2)

Action

```
  @Constraint(G1, (now()-last) >= 500)
  public void actionObserve() {
    foreach(Camera camera :  cameras) {
      activate(camera.takePicture(
        snapshotId, observationObject));
    }
    snapshotId++;
    set(last, now());
  }

  @Constraint(
   foreach(int IDi = cameras[i].getPictureId),
   ID1 == ID2, ID2 == ID3)
  public void actionComposeSnapshot() {
    Snapshot snapshot = new Snapshot();
    foreach(Camera camera :  cameras) {
      snapshot.add(camera.getPictureWithId(get(ID1)))
    }
    snapshot.setId(ID1);
    queue.add(snapshot);
  }
}
```

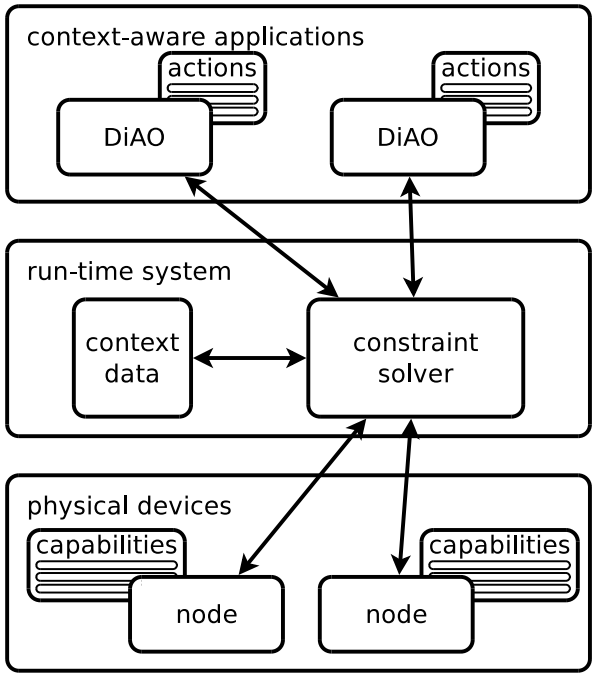
Language Extension (2)

```
  @Constraint(G1, (now()-last) >= 500)
  public void actionObserve() {
    foreach(Camera camera :  cameras) {
      activate(camera.takePicture(
        snapshotId, observationObject));
    }
    snapshotId++;
    set(last, now());
  }

  @Constraint(
   foreach(int IDi = cameras[i].getPictureId),          Action
   ID1 == ID2, ID2 == ID3)
  public void actionComposeSnapshot() {
    Snapshot snapshot = new Snapshot();
    foreach(Camera camera :  cameras) {
      snapshot.add(camera.getPictureWithId(get(ID1)))
    }
    snapshot.setId(ID1);
    queue.add(snapshot);
  }
}
```

Architecture

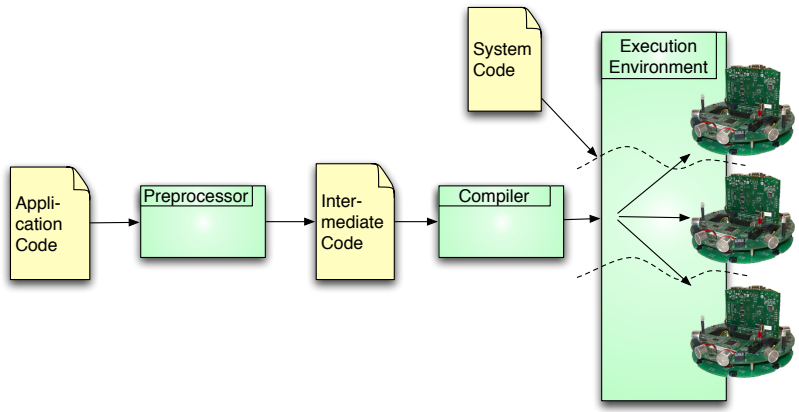
- 3-layer architecture
- Distributed run-time system
  - Global view (about system resources)
  - Interconnections, sensor fusion
- Mobile nodes as executing components (sensors/ actuators)
  - Local view
  - State: location, energy, ..
- Actions are mapped to capabilities
- Capabilities organized in inverted lists
- Constraint Solver computes mapping based on context data
- Communication: space-time events



capability	spec	nodes
camera	res=..	x_1 , x_3
camera	res=..	x_2
temp-sensor	acc=0.1	x_6 , x_7

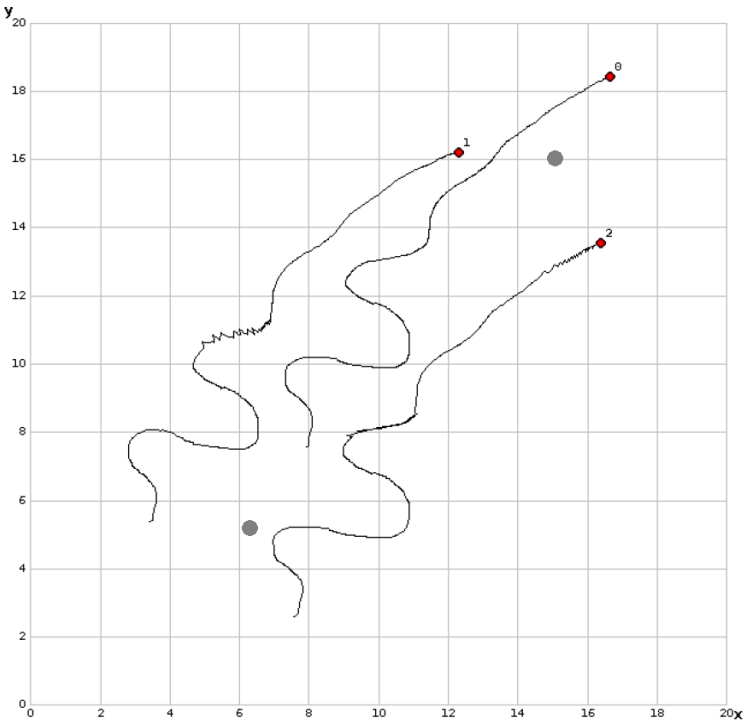
Tool Chain

- Produces explicit based on implicit distribution
- Preprocessing
  $\rightarrow$ intermediate code
  - Actions + conditions are registered (callbacks)
  - Introduces synchronization (shared vars)
  - Introduces concurrency
  - Introduces proxies (RPC)
- Compilation



Proof-Of-Concept

- Simulation (manual)
  - 3 robots (with cameras)
  - Full degree of control of observed object
- Goal: 3-sided observation
  - Equilateral triangle
  - Reactive behavior
- Execution
  - No reassignment
  - Robots had to move
  - $\sim$ 100 calls / sec (solver)



Conclusion

- Task: Context-aware applications for mobile robots (CPS)
- Programming abstraction for distributed mobile applications
  - Hide concurrency and distribution
  - Introduce context awareness
    $\rightarrow$ Context-aware annotations
- DiAOs as programming model
- Target OOP combined with declarative programming

- Future work
  - Improve context language
  - Space-time scheduling
  - Tool chain
  - Hierarchical/ distributed constraint solving