Dependence Analysis for Distributed Event-Based Systems

Daniel Popescu, Joshua Garcia, and Nenad Medvidović
March 8, 2011
Software Evolution

- Software systems are typically changed constantly
- Changing complex software systems
  - Time-Intensive
  - Error Prone
  - Costs vs. added value
I’m thinking about refactoring this piece of code. What do you think I could break?

good point

everything?
Dependence-Based Impact Analysis

• How does a change to a code element propagate through a system?

• What other source code elements are affected by the change?

Existing techniques lack adequate support for DEB systems
Distributed Event-Based Systems

- Implicit Invocations
- Typically based on message-oriented middleware (MOM)
public class HelloWorld extends Component {

    void explicit() {
        PrinterComponent.println("Hello World");
    }

    void implicit() {
        Message m = new Message("Hello World");
        publish(m);
    }
}
Implementation of DEB Components

• What do we want to analyze?
  • Programming languages that are most widely supported by existing MOM platforms
  • Message interfaces

    public interface MessageListener {
      void onMessage(Message message);
    }

    public interface MessageProducer {
      void send(Message message);
      ...
    }
void onMessage(Message msg)
{
    String type = msg.getJMSType();
    if (type.equals("On/Off")) {
        ...  
    } else if (type.equals("PicRequest")) {
        ...
    }
}

Ambiguity of Message Interfaces
Research Questions

• RQ: How can we recover more precise dependences from existing DEB systems (compared to existing approaches)?
Impact Analysis Process
Iterative Impact Analysis Example
Iterative Impact Analysis

- Input: Dependence graph, code element to be changed
- Output: Code elements that require changes
- Iteratively inspect potentially affected elements
  - Changing element
  - Propagating element
  - Unaffected element
Impact Analysis Process

1. Source Code
2. Dependence Analysis
3. Dependence Graph
4. Iterative Impact Analysis
5. Affected Code Elements

Code Element To Be Changed
Classifying Message Dependences

- Inter-Component Message Dependence

- Viewer
  - PicReq
  - PicUpdate

- Camera
  - PicReq
  - PicUpdate
  - On/Off

- Controller
  - PwrReq
  - Update
  - On/Off

- Power Sensor
  - PwrReq
  - PwrUpdate
Classifying Message Dependences

- Inter-Component Message Dependence
- Intra-Component Dependence
  - Control-Flow-Based
Classifying Message Dependences

- Inter-Component Message Dependence
- Intra-Component Dependence
  - Control-Flow-Based
  - Data-Flow-Based
Classifying Message Dependences

- Inter-Component Message Dependence
- Intra-Component Dependence
  - Control-Flow-Based
  - Data-Flow-Based
- Dependence Paths

![Diagram showing message dependencies between components](image-url)
Impact Analysis Process

Source Code → Dependence Analysis → Dependence Graph → Iterative Impact Analysis → Affected Code Elements

- Code Element To Be Changed
Hyperion

- **Input:** Source code of DEB system
  - Uses mainstream OO language
  - Uses MOM platform
- **Output:** Message dependences
- **Challenge:** Ambiguous message interfaces
  - Insight: Use of messages reveals types
  - Insight: Type discriminations requires flow-sensitive analysis
Hyperion’s Approach

• Propagate data from message sink throughout component

• Collect information generated by each statement
  • Message types
  • Accessed state variables
public class Camera extends Component {

    private boolean isOn = false;

    public void onMessage(Message m) {
        ...
    }

    private void takePicture(PicReq r) {
        ...
    }

    private void triggerPower() {
        ...
    }
}
public class Camera extends Component {
    @State("powerStatus")
    private boolean isOn = false;

    @Modifies("powerStatus")
    public void onMessage(Message m) {
        ...}

    private void takePicture(PicReq r) {
        ...}

    @Modifies("powerStatus")
    private void triggerPower() {...}
public class Camera extends Component {
    @State("powerStatus")
    private boolean isOn = false;

    @Modifies("powerStatus")
    public void onMessage(Message m) {
        if (m instanceof OnOff) {
            triggerPower();
        }
        if (m instanceof PicReq) {
            PicReq request = (PicReq) m;
            if (isOn) takePicture(request);}
    }

    private void takePicture(PicReq r) {
        int resolution = r.resolution;
        Picture p = CameraDriver.get(resolution);
        PicUpdate update = new PicUpdate(p);
        send(update);
    }

    @Modifies("powerStatus")
    private void triggerPower() { isOn = !isOn; }
}
public class Camera extends Component {
  @State("powerStatus")
  private boolean isOn = false;

  @Modifies("powerStatus")
  public void onMessage(Message m) {
    if (m instanceof OnOff) {
      triggerPower();
    }
    if (m instanceof PicReq) {
      PicReq request = (PicReq) m;
      if (isOn) takePicture(request);
    }
  }

  private void takePicture(PicReq r) {
    int resolution = r.resolution;
    Picture p = CameraDriver.get(resolution);
    PicUpdate update = new PicUpdate(p);
    send(update);
  }

  @Modifies("powerStatus")
  private void triggerPower() {
    isOn = !isOn;
  }
}
Inter-Component Dependencies

- Viewer
  - PicReq
  - PicUpdate

- Controller
  - PwrReq
  - PwrUpdate
  - On/Off

- Camera
  - PicReq
  - PicUpdate
  - On/Off

- Power Sensor
  - PwrReq
  - PwrUpdate
Preliminary Evaluation

- Evaluation dimension: precision
- Empirical evaluation using five existing DEB systems
- RQ: To what extend does Hyperion recover more precise intra-component dependences than LSME
Preliminary Evaluation Results

Hyperion recovers x% fewer false positive intra-component dependences than LSME
Conclusion

- Hyperion recovers more precise sound dependences from DEB systems than existing techniques

→ Fewer dependences help to reduce impact analysis effort

Thank you!