Cross-Language Program Understanding, Code Analysis and Refactoring

Philip Mayer and Andreas Schroeder

Programming & Software Engineering Group

Ludwig-Maximilians-Universität München, Germany

SCAM 2012
Riva del Garda, Italy

XLL Cross Language Links

Context, Situation & Remedy

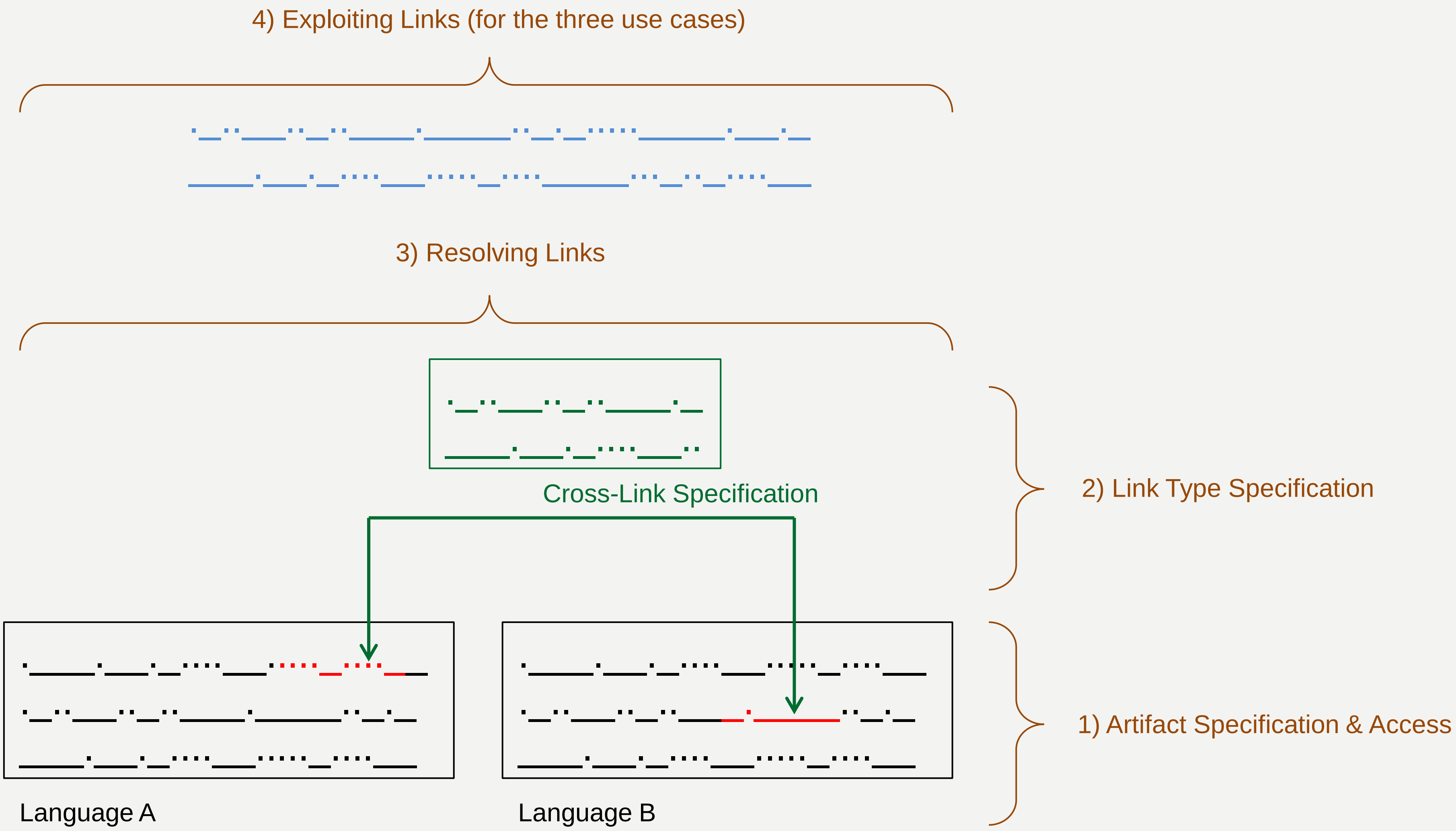
- **Context**: MLSAs (Multi-Language Software Applications)
  - ...are systems written using different programming languages and
  - …involve **artifacts** in different languages which are **linked** together
  - …only work (properly) if the links are intact
- **Situation**: MLSAs are badly supported by tools leading to productivity loss
  - No compiler help / error marking => might forget links while coding
  - No refactoring support => might break links => more bugs
  - No code navigation / visualization => program understanding is harder
- **Remedy**: Explicit description of links & tools

XLL

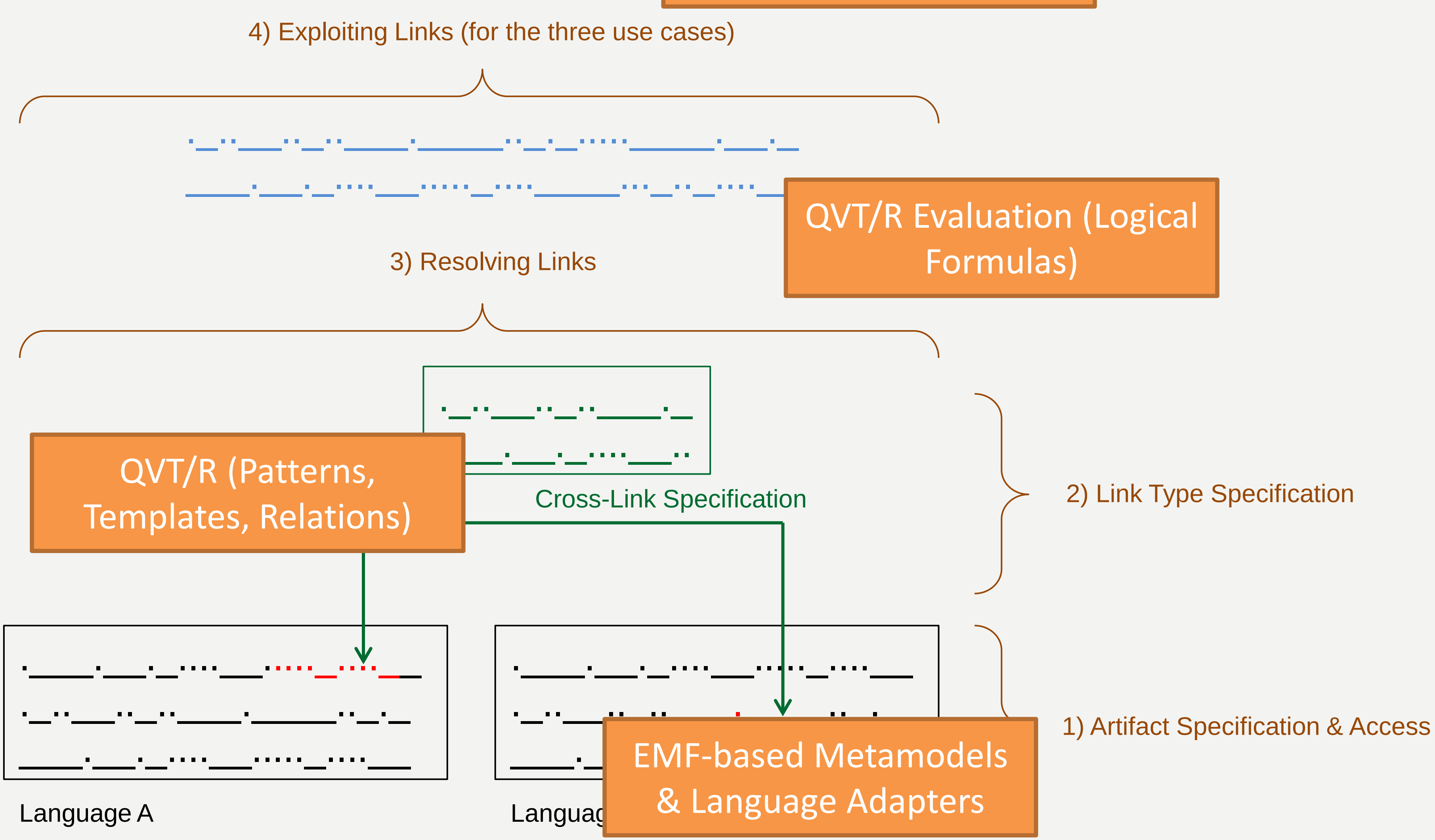
- Our approach: A framework (XLL) for handling cross-language links
  - Allows **explicitly** declaring link types
  - Performs **live link monitoring** (for established and broken links)
  - Plugs into **refactorings** (to keep links intact)

- Support three use cases
  - **Program Understanding**: Code Navigation & Code Visualization
  - **Code Analysis**: Indicate Errors or Possible Problems / Perform Complexity Analysis
  - **Refactoring & Code Generation**: Propagate Changes (with additional refactorings) / Generate Code

What do we need?

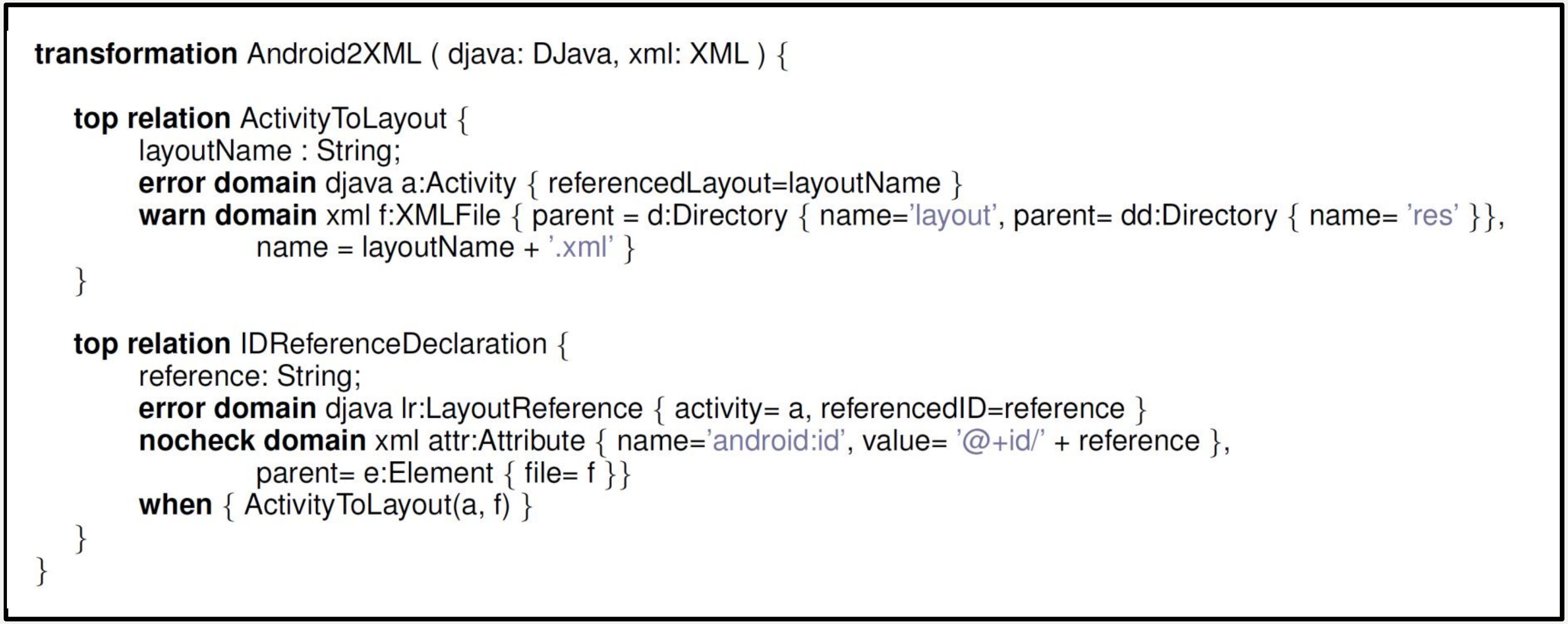

4) Exploiting Links (for the three use cases)
3) Resolving Links
Cross-Link Specification
2) Link Type Specification
1) Artifact Specification & Access
Language A
Language B

XLL: The Solution


Plugging into Eclipse
4) Exploiting Links (for the three use cases)
QVT/R Evaluation (Logical Formulas)
3) Resolving Links
QVT/R (Patterns, Templates, Relations)
Cross-Link Specification
2) Link Type Specification
1) Artifact Specification & Access
Language A
Languag
EMF-based Metamodels & Language Adapters

Example: Android

- Example: Android Java vs. UI XML in QVT/R



```
transformation Android2XML ( djava: DJava, xml: XML ) {

    top relation ActivityToLayout {
        layoutName : String;
        error domain djava a:Activity { referencedLayout=layoutName }
        warn domain xml f:XMLFile { parent = d:Directory { name='layout', parent= dd:Directory { name= 'res' }},
                name = layoutName + '.xml' }
    }

    top relation IDReferenceDeclaration {
        reference: String;
        error domain djava lr:LayoutReference { activity= a, referencedID=reference }
        nocheck domain xml attr:Attribute { name='android:id', value= '@+id/' + reference },
                parent= e:Element { file= f }}
        when { ActivityToLayout(a, f) }
    }
}
```

What we found

- XLL (EMF/QVT/Constraints/Eclipse) has been **implemented** on top of Eclipse and **applied** to three software systems (a few kloc to 100kloc) with a total of five languages

- The **good**:
  - It works ☺ (for simple link types)
  - EMF-based metamodels make sense
  - Eclipse-integration (including refactoring reuse) is relatively painless

- The **bad**:
  - QVT/R is not expressive enough for more complicated links
  - Logic-based evaluation is very hard to debug
  - High coupling between language metamodels and link specifications

Current & Future Work

- Current Work: Working on a better linking language
  - Looking at Query/Addressing Languages
  - Minimize coupling between link specification and metamodels
- Future Work: Evaluation of usefulness claims
  - How does it affect productivity? (i.e. is it worth it?)

Thank You.

XLL Cross Language Links

www.xllsrc.net