Analyzing Behavioral Refactoring of Class Models

Wuliang Sun, Robert B. France, Indrakshi Ray

Computer Science Department
Colorado State University
Fort Collins, Colorado

The Models and Evolution (ME) workshop
colocated with the 16th MODELS, Miami, USA

Colorado State University
Computer Science Department

Introduction

- Evolving class models in the Model-Driven Development (MDD) projects
- Software refactoring
  - Goal: achieve maintainability, extensibility, etc.
- Existing class model refactoring techniques
  - Lack support for refactorings that involve making changes on operation specifications

Introduction

- Behavioral refactoring
  - Changes on operation specifications
    - E.g., add, remove, modify specifications
  - Source model
  - Refactored model

- Net effect of an operation
  - Essential behavior
  - Specified using the pre-/post-conditions expressed in the Object Constraint Language (OCL)

Motivation

- Source model
  - FlightManager::bookFlight()
- Refactored model
  - Airline::getAvailableFlights()
  - Flight::getAvailableSeats()
  - Flight::reserveSeat()
  - FlightManager::bookFlightTicket()
- Checking
  - If the net effect of FlightManager::bookFlight() is preserved by the operations in the refactored model

Contribution

- Checking behavioral refactoring that involves making changes on OCL operation specifications
- Tool support based on the Alloy Analyzer
  - UML-to-Alloy transformation
- Lightweight analysis
  - Shield the modeler from the back-end use of the Alloy Analyzer
  - The net effect preservation analysis is checked within a bounded domain

Approach Overview

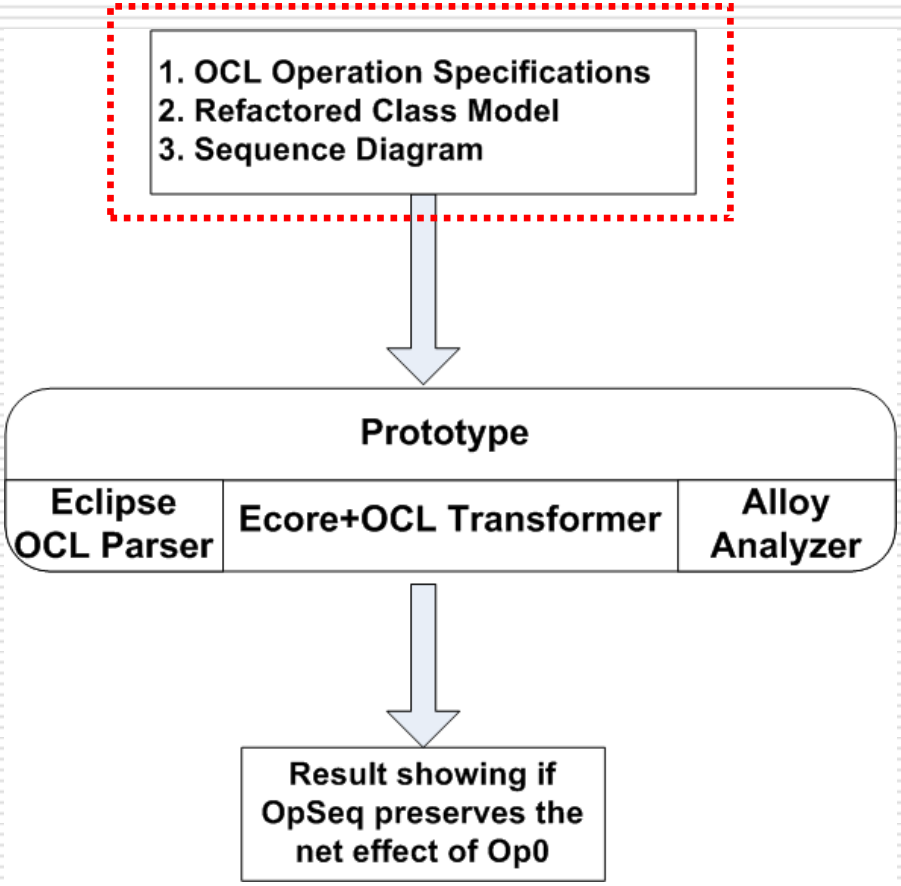
- Net effect preservation
  - OpSeq = [Op1; Op2; ...; OpN] in the refactored model
  - Op0: an operation in the source model

- OpSeq preserves Op0 if
  - The start states associated with the pre-condition of Op0 is included by that of OpSeq
  - The ending states associated with post-condition of Op0 is included by that of OpSeq

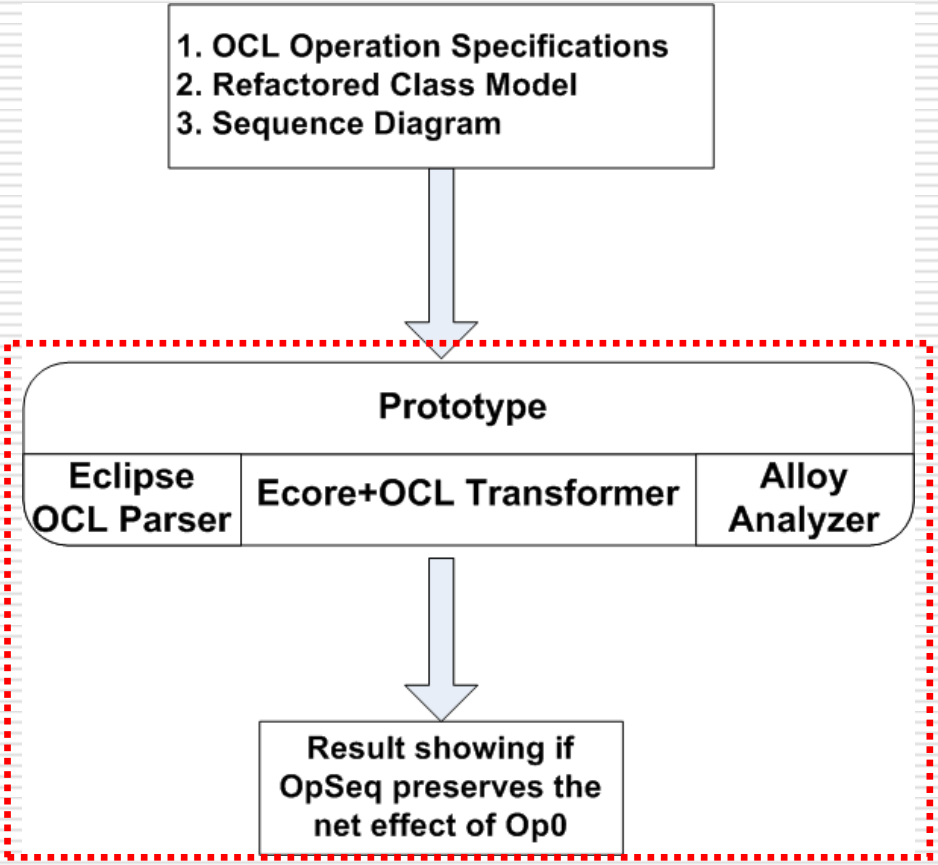
Approach Overview

- Op0 is preserved by the refactoring if there exists OpSeq
  - OpSeq starts in all the states that satisfy the pre-condition of Op0
  - OpSeq
    - Starting in a state satisfying the Op0 pre-condition
    - Ending in a state satisfying the Op0 post-condition

Approach Overview



Approach Overview

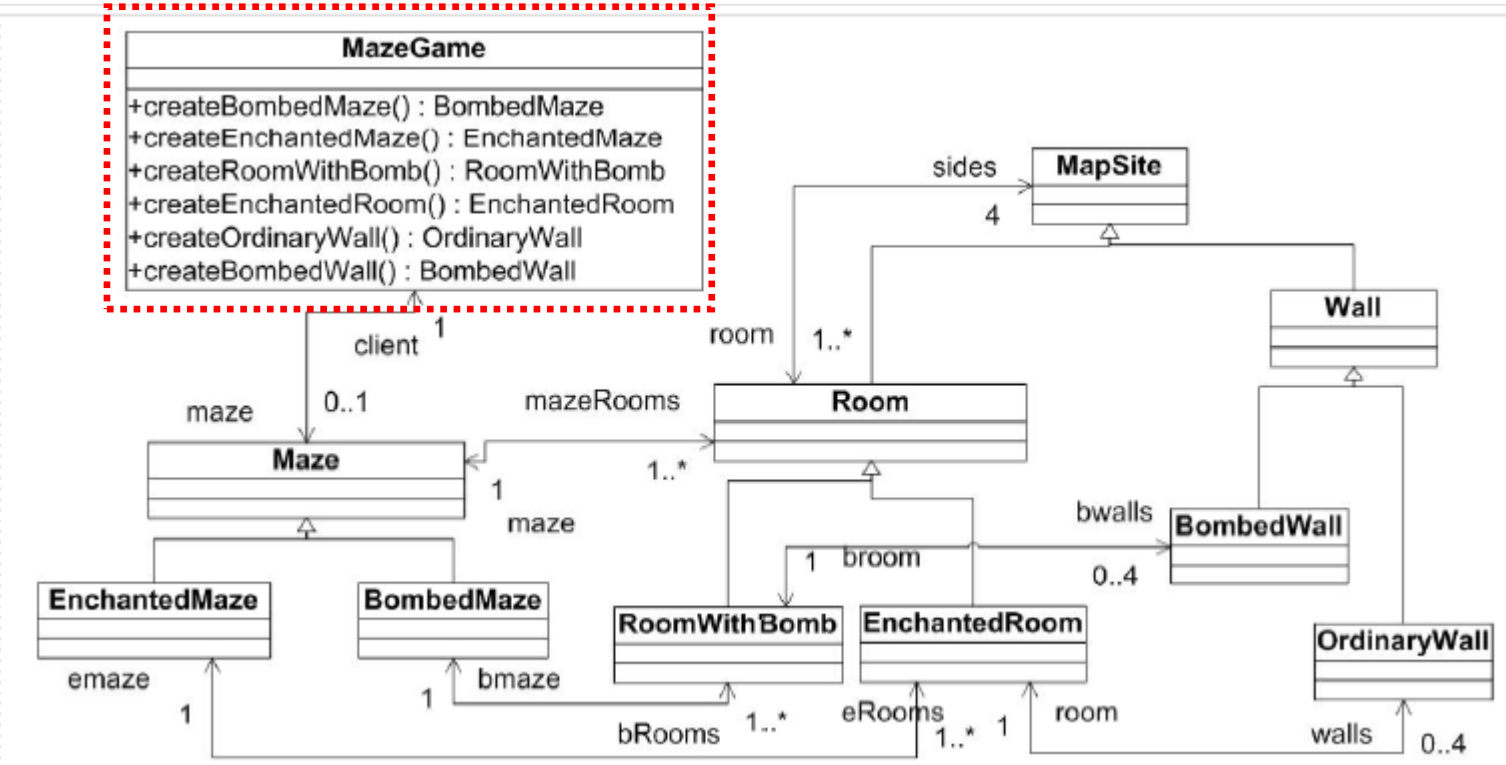


Approach

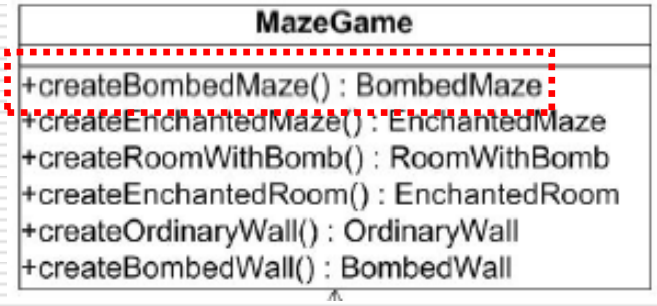
- First step: check that the elements referenced in the Op0 operation specification also appear in the refactored model
- Second step: generate an Alloy model using class model-to-Alloy and OCL-to-Alloy model transformation
- Third step: produce an Alloy predicate from Op0 and sequence diagram

*** More details of the approach can be found in a technical report. See the link below**
http://www.cs.colostate.edu/TechReports/Reports/2013/tr13-104.pdf

Case Study – Maze Game

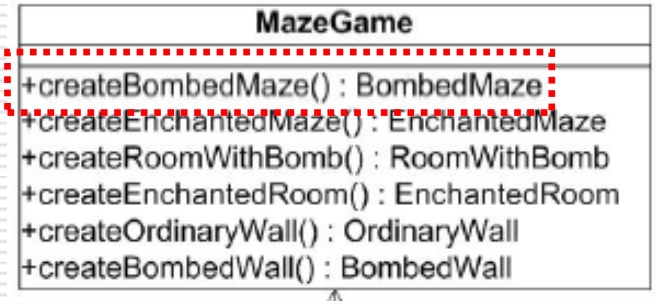


Case Study – Maze Game



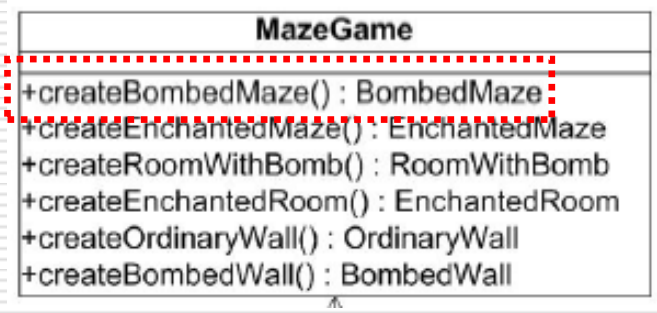
Context *MazeGame::createBombedMaze() : BombedMaze*
// Pre-condition: no maze has been created
Pre: *self.maze→isEmpty()*
// Post-condition: a bombed maze has been created, and it includes a room
// with four walls
Post: *result.oclIsNew() and self.maze.bRooms→size() = 1 and*
self.maze.bRooms→forAll(r : RoomWithBomb | r.bwalls→size() = 4)

Case Study – Maze Game



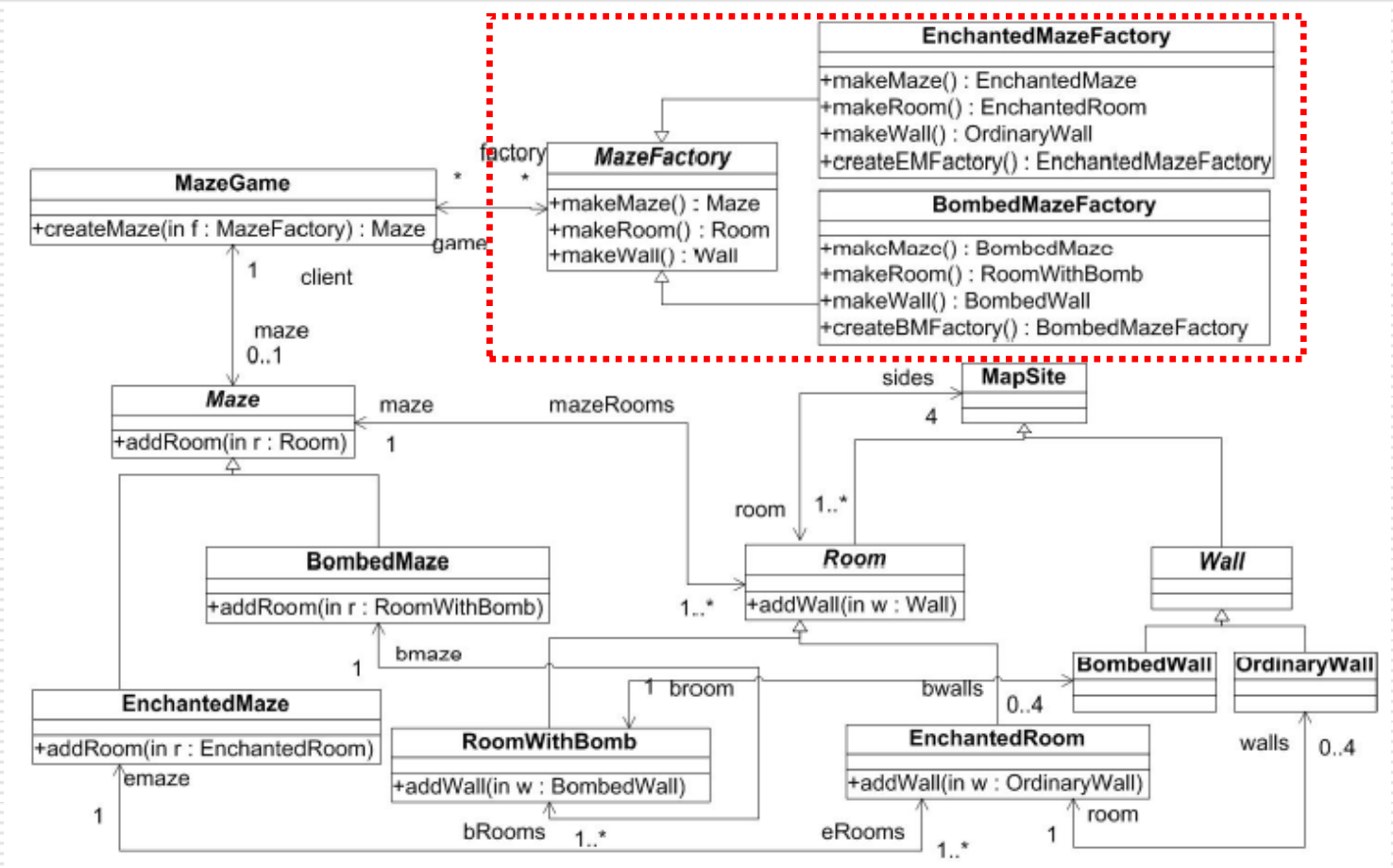
Context *MazeGame::createBombedMaze() : BombedMaze*
// Pre-condition: no maze has been created
Pre: *self.maze→isEmpty()*
// Post-condition: a bombed maze has been created, and it includes a room
// with four walls
Post: *result.oclIsNew() and self.maze.bRooms→size() = 1 and*
self.maze.bRooms→forAll(r : RoomWithBomb | r.bwalls→size() = 4)

Case Study – Maze Game

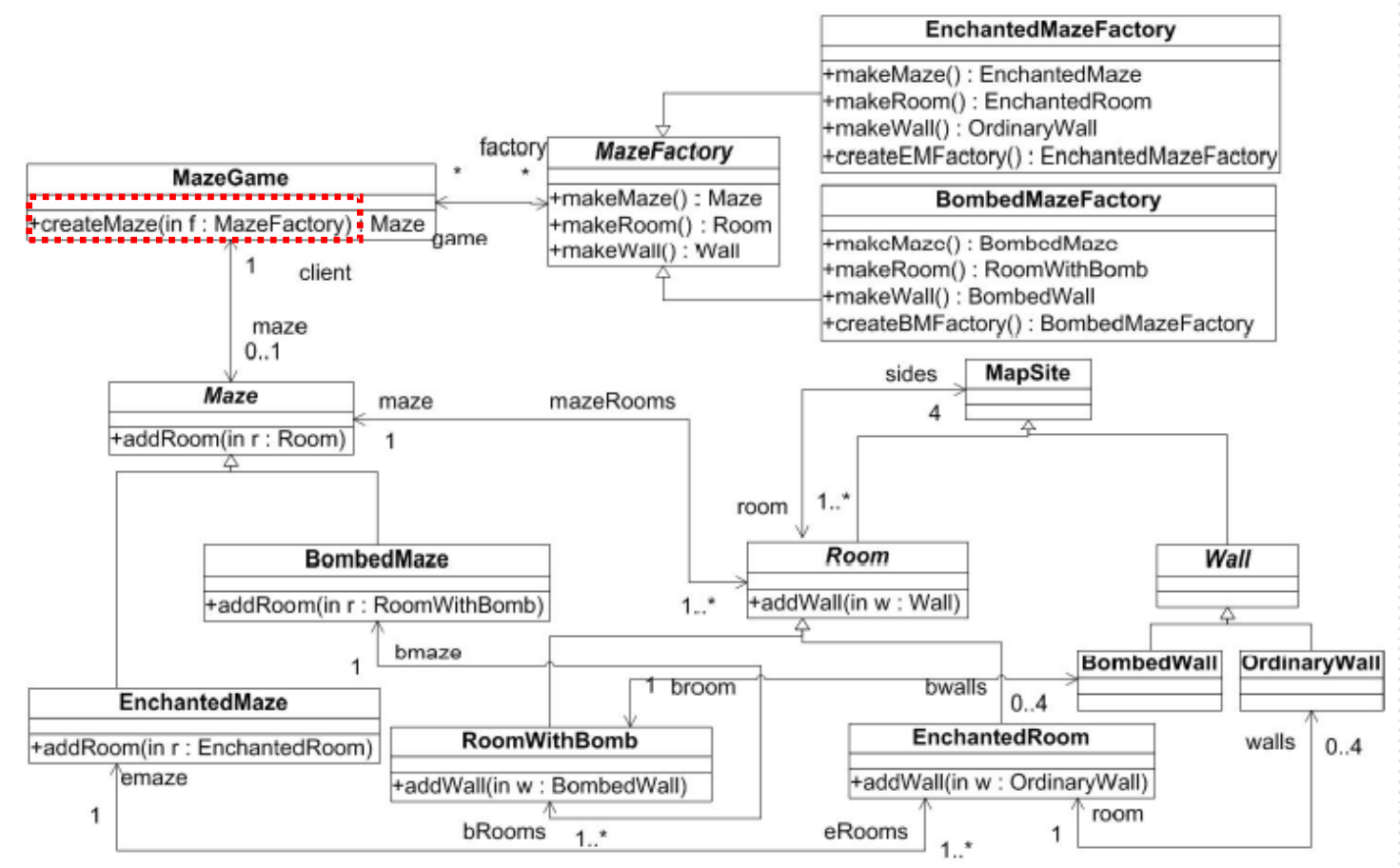


Context *MazeGame::createBombedMaze() : BombedMaze*
// Pre-condition: no maze has been created
Pre: *self.maze→isEmpty()*
// Post-condition: a bombed maze has been created, and it includes a room
// with four walls
Post: *result.oclIsNew() and self.maze.bRooms→size() = 1 and*
self.maze.bRooms→forAll(r : RoomWithBomb | r.bwalls→size() = 4)

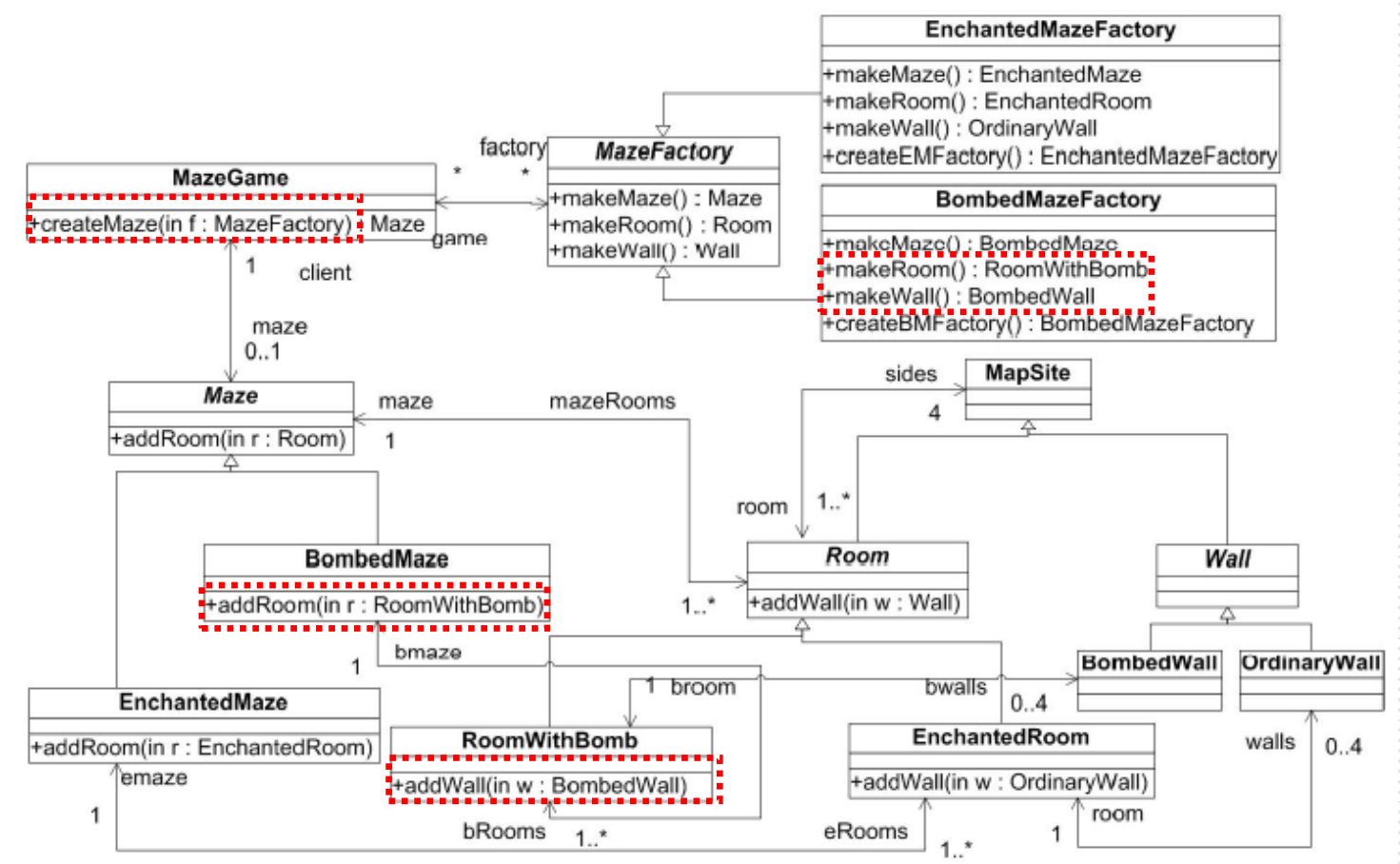
Case Study – Maze Game



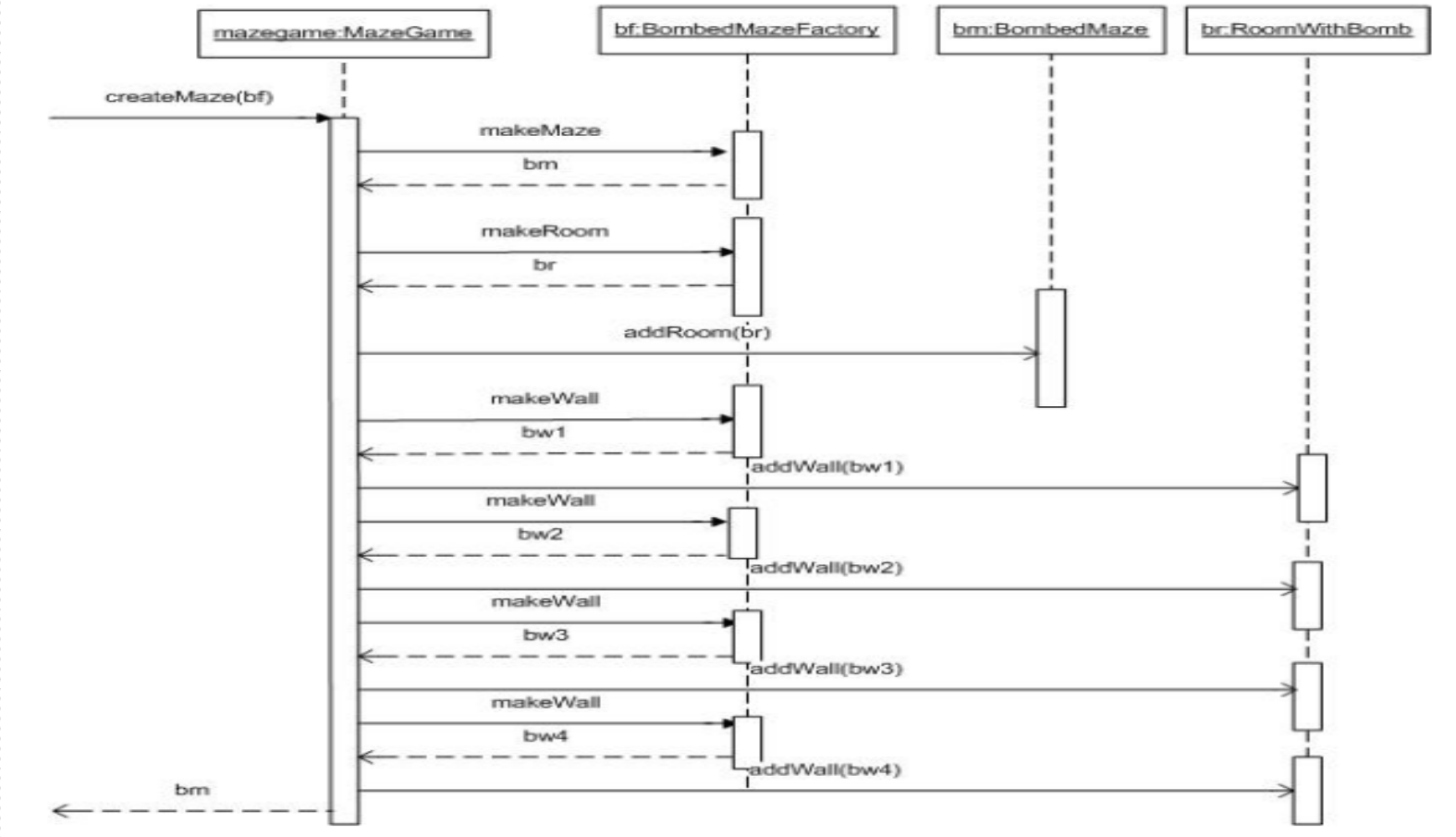
Case Study – Maze Game



Case Study – Maze Game



Case Study – Maze Game



Case Study – Maze Game

- Checking if a given sequence preserves createBombedMaze
  - E.g., an operation invocation sequence =

    [createMaze; makeMaze; makeRoom; addRoom; makeWall; addWall; makeWall; addWall; makeWall; addWall; makeWall; addWall]

- Analysis result: createBombedMaze is preserved by the given sequence

Conclusion and Perspective

- Limitation of the approach
  - OCL-to-Alloy transformation
  - Checking the Op0 referenced elements in the refactored model
- Future work
  - Use SMT solvers (e.g., Microsoft Z3) for the analysis
  - Explore the mappings between equivalent source and refactored models

***Acknowledgment: the work was supported by the National Science Foundation grant CCF-1018711.**

Related Work

- E. Gamma, H. Richard, J. Ralph, and V. John. Design patterns: elements of reusable object-oriented software. Reading: Addison-Wesley Publishing Company, 1995.
- M. Fowler and K. Beck. Refactoring: improving the design of existing code. Addison-Wesley Professional, 1999.
- D. Jackson. Alloy: a lightweight object modelling notation. ACM TOSEM, 2002.
- R. France, S. Chosh, E. Song, and D.K. Kim. A metamodeling approach to pattern-based model refactoring. IEEE Software, 2003.
- P. Van Gorp, H. Stenten, T. Mens, and S. Demeyer. Towards automating source-consistent uml refactorings. UML, 2003.
- W. Sun, R. France, and I. Ray. Rigorous analysis of UML access control policy models. In Proceedings of the POLICY, 2011.

Thanks for your attention!

Any Questions?