# Utilizing JPF for Multi-Agent Verification

Berndt Farwer

Department of Computer Science
Durham University

*berndt.farwer@durham.ac.uk*

Joint Project with

the University of Liverpool

Rafael H. Bordini[1], Michael Fisher[2],
Berndt Farwer[1] , Louise Dennis[2]

[1] Durham University, UK
[2] University of Liverpool, UK

Supported by EPSRC grants EP/D054788 (Durham) and EP/D052548 (Liverpool).

Durham University

# Outline

- Multi-Agent Programming
  - BDI Paradigm
  - Agent Infrastructure Layer (AIL)
  - AIL-based Interpreters
- Property Specification
- Agent JPF (AJPF)
- MCAPL Interface: Adding Further Languages
- Outlook

Durham University

# Multi-Agent Programming

- BDI (beliefs, desires, and intentions) Agents
  - Mental State
    - Beliefs ('knowledge')
    - Goals (states that the agent wants to bring about)
    - Plans (recipes for achieving plans)
    - Intentions (stacks of plans for adopted goals)
- Environment
  - Events (external)
  - Agents  interact with Environment through *perception*

Durham University

# Multi-Agent Programming

- **Reasoning Cycle**
  - ***Typical stages:***
    - Selection of an *Intention*
    - Selection of an *Event*
    - Determining *applicable plans*
    - Selection of a *plan* → addition to the current intention stack
    - Execution of the *head of the intention stack*
    - *Perception* of the environment
    - *Message* handling
    - *Cleanup*

Durham University

# Executing BDI Programs: The Basic Idea

- Given an *achievement goal* !g
- Find a plan in the plan library for achieving that goal
  - Plan body = sequence of deeds
  - Deeds = actions or (sub-)goals
- Add plan to intention stack
- Cycle through the stages repeatedly
  - executing actions
  - adding plans for new sub-goals
  - perceiving changes in the environment
  - handling messages

Durham University

# Agent Infrastructure Layer (AIL)

- General model-checking framework for agent programming languages
- Not a new programming language
  - Does *not* have its own reasoning cycle
- AIL is a Java library with clear semantics
  - *Data structures* for Beliefs, Goals, Intentions, and Plans
  - *Rules* and *operations* to build own reasoning cycle
  - default functions
  - extensible (new rules and operations, overridden defaults)
- Integrated *property specification language*
  - Properties are specified at the AIL level

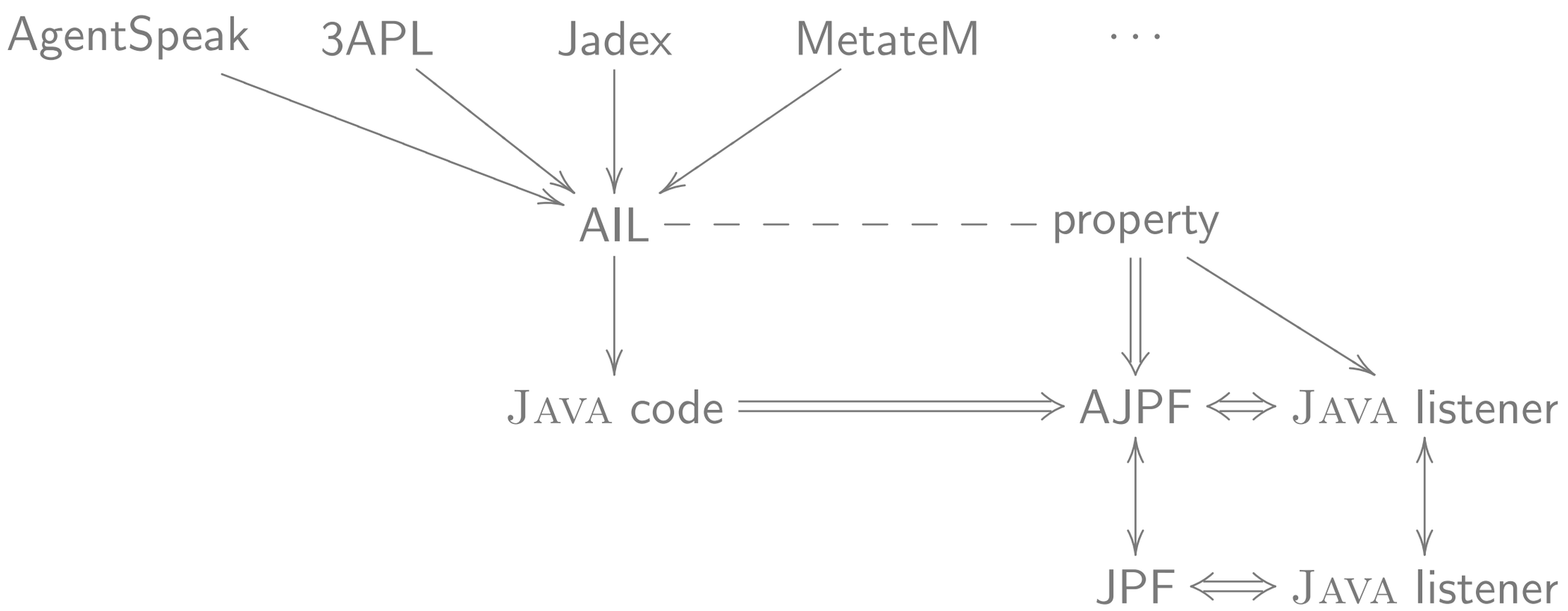
representing the agents' metal states based on stacks of deeds

LTL

special modalities for belief, goal, etc.

Durham University

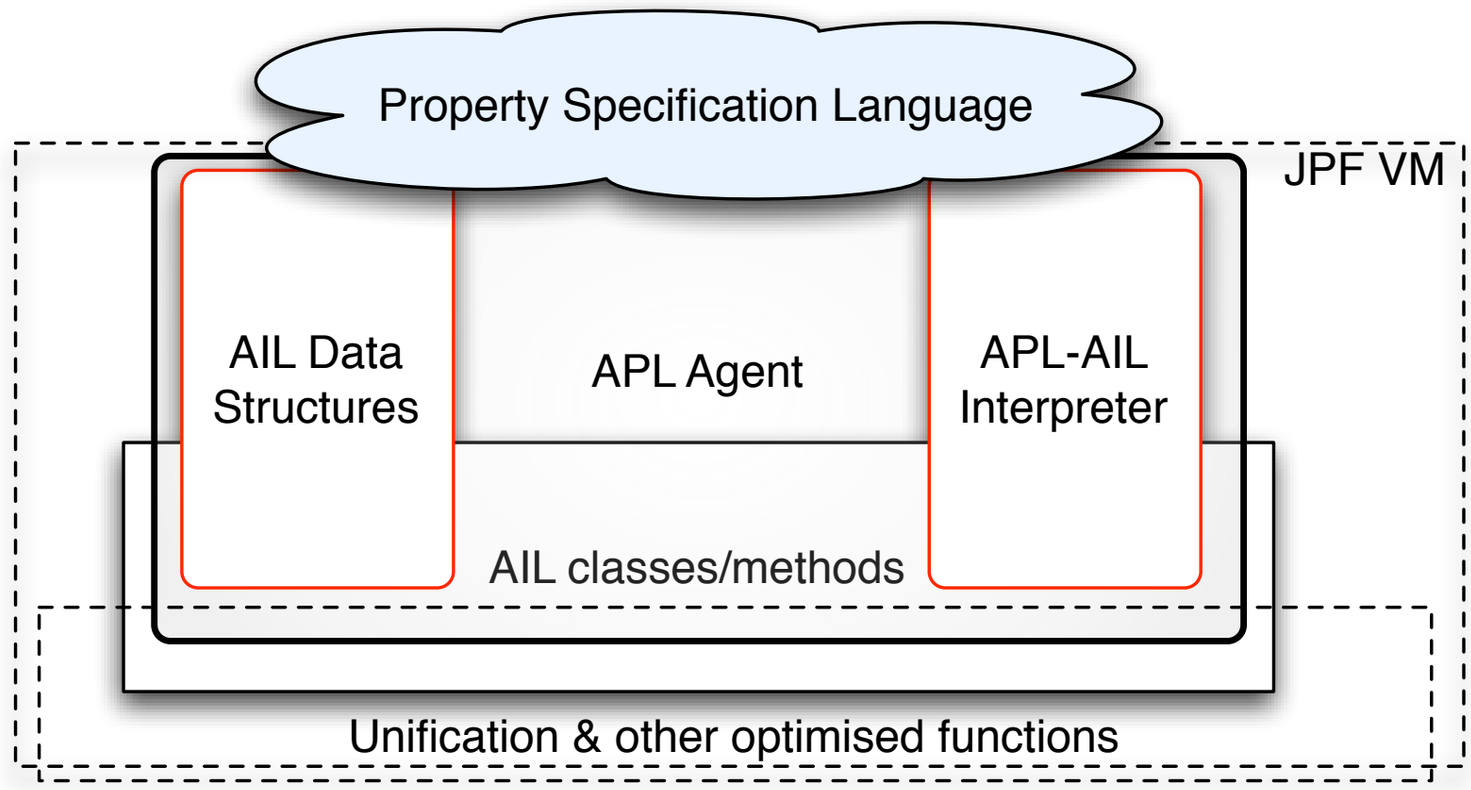
# AIL architecture



# AIL Toolkit

- AIL data structures
- AIL methods used by language interpreters
- Common property specification language

- Extensibility
- Open Source




Durham University

# Languages

- The framework is designed to execute and verify many BDI languages, such as
  - GOAL
  - SAAPL
  - AgentSpeak
  - 3APL/2APL
  - … *your favourite language*

… also allows heterogeneous MAS!

Durham University

# AIL-Based Interpreter Requirements

- Plug together an AIL-based interpreter for each language
  - Java library of rules and operations on the agent state used to construct a *custom reasoning cycle*
- AIL needs to provide means to do most of what other APLs do, e.g
  - Keep track of open goals, events, (suspended) intentions
  - Maintain multiple sequences of deeds to be performed
    - multiple intentions
  - Allow deed sequences to be related to the goals/events that generated them
  - Allow goals to be dropped
  - Deal with language-specific agent components

Durham University

# AIL Agent State

$$< ag, i, I, Pl, A, B, BR, P, C, In, Out, Cn, Cx, Ann, RC >$$

$ag$ is a unique **identifier** for the agent,

$i$ is the **current intention**, $I$ comprises all **extant intentions**,

$Pl$ the **currently applicable plans**,

$A$ is a set of **actions**,

$B$ the agent's **beliefs**,

$BR$ the agent's **belief rules**,

$P$ the agent's **plan library**,

$C$ the agent's **constraints**,

$In$, $Out$ are the agent's **inbox** and **outbox**,

$Cn$ the agent's **content**, $Cx$ the agent's **context**,

$Ann$ a set of **annotations**,

$RC$ is the **current stage** in the agent's reasoning cycle.

Durham University

# Property Specification Language (PSL)

- LTL
  - $\wedge$, $\vee$, $\neg$, **U**, **R**
- Modalities on ground first order formulae
  - Belief **B**
  - Goal **G**
  - Intention **I**
  - Perception **P**
  - Action **A**

$$
\begin{aligned}
ag &::= \text{constant} \\
f &::= \text{ground first order formula} \\
\phi &::= \mathbf{B}(ag, f) \mid \mathbf{G}(ag, f) \mid \mathbf{A}(ag, f) \mid \mathbf{I}(ag, f) \mid \mathbf{P}(f) \\
&\quad \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \phi \mathbf{U} \phi \mid \phi \mathbf{R} \phi
\end{aligned}
$$

Typical properties:

$\Diamond \mathbf{B}(ag_1, \text{done})$

$\mathbf{G}(ag_1, \text{some\_goal}) \rightarrow \Diamond \mathbf{B}(ag_1, \text{some\_goal})$

$\Diamond \neg \mathbf{G}(ag_1, \text{some\_goal})$

Durham University

# Property Specification Language (PSL)

- $MAS \models_{MC} \mathbf{B}(ag, f)$ iff $f \in ag_{BB}$
  where $ag_{BB}$ is the belief base of agent
- $MAS \models_{MC} \mathbf{G}(ag, f)$ iff $!_a f \in ag_G$
- $MAS \models_{MC} \mathbf{A}(ag, f)$ iff the last action recorded by the environment was $ag$ taking action $f$
- $MAS \models_{MC} \mathbf{I}(ag, f)$ iff $!_a f \in ag_G$
  and $f$ has been added to $ag$'s intentions
- $MAS \models_{MC} \mathbf{P}(f)$ iff $f$ is a percept
  from the environment.

Durham University

# Using AIL/AJPF

- Write program in your favourite language(s)
- Specify properties to check
- Translate program(s) into AIL representation
  - Automated translators under development
- Translate properties into our PSL
  - Again, automated translators will be made available
- Write environment
- Run program(s) in AJPF
  - execution
  - verification

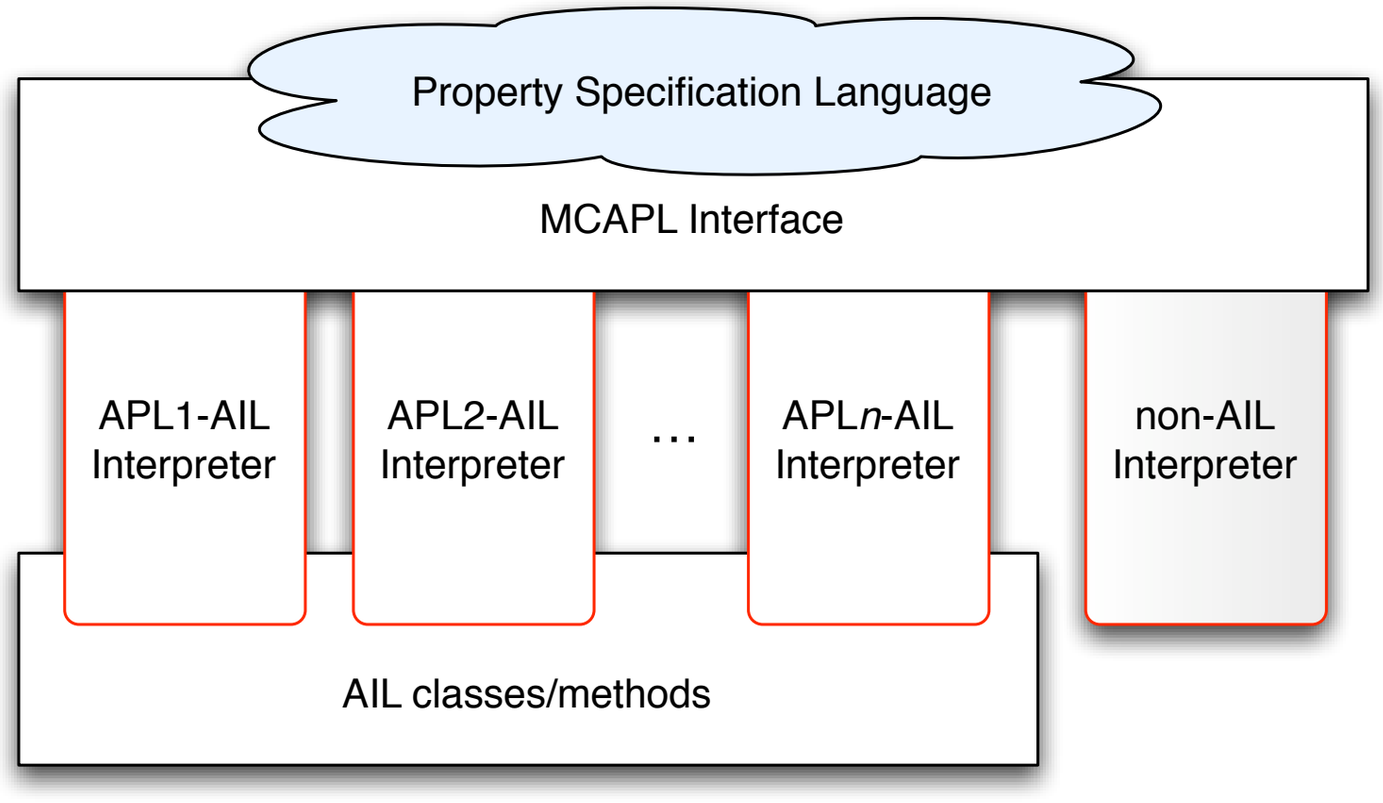
Durham University

# The MCAPL Interface

- What if there is no custom AIL interpreter for a language?
  - Original interpreter can interface with the property specification language and the environment
  - Does not use AIL data structures → has to define modalities
  - allows heterogeneous execution/verification just as with customised AIL-based language interpreters
- However:
  - Is not optimised for model checking
  - Results rely on ‘correct’ implementation of modalities and correct interfacing

Durham University

# MCAPL Interface




- Executes the MAS
- Extends property specification to non-AIL interpreters
- Provides the Listener for JPF model checking

Durham University

# AJPF (Agent Java Pathfinder)

- (Negated) Property translated into Büchi automaton
- On-the-fly construction of product automaton
- Product automaton
  - Agent program
  - Property automaton
- Changes in the product automaton trigger JPF
  - generating a violation or
  - pruning the search space
- Environment is part of the multi-agent program
- Listener
- *Problem*: long execution times

Durham University

# AJPF – Addressing Efficiency

- atomic sections
  - initialisation phase
  - reasoning cycle
- Incorporate AIL, MCAPL, and PSL into JPF.
- Improve Efficiency
  - Cut down on number of objects, …
- Provide adequate UI for
  - agent programs
  - properties
  - other options

Durham University

# Atomics

| $\Diamond_{ag_1}\mathrm{pickup}$ | JPF | AJPF |
|---|---|---|
| elapsed time: | 0:07:03 | 0:00:04 |
| states: | new=11144 | new=24 |
| | visited=11080 | visited=11 |
| | backtracked=22223 | backtracked=34 |
| search: | maxDepth=1860 | maxDepth=12 |
| choice generators: | thread=11145 | thread=25 |
| heap: | gc=28161 | gc=65 |
| | new=3472584 | new=16551 |
| | free=3058118 | free=14741 |
| instructions: | 235599025 | 1051314 |
| max memory: | 59MB | 26MB |
| loaded code: | classes=219 | classes=219 |
| | methods=2807 | methods=2807 |

Durham University

# Outlook

- Make execution in JPF more efficient
  (4 states per second are not enough)

- Automate language and property translations

- Customise state-space visualisation
  - StateSpaceDot listener
    state space visualisation

- Case studies

Durham University