GORITE Execution

- **team model**
  - Performer, Team, Team.Task, Team.Role

- **process model**
  - Capability, Plan, Goal
    - SequenceGoal, ParallelGoal, BDIGoal, TeamGoal, ...

- **data model**
  - Data, Data.Element
  - Relation, Query
    - And, Or, Distinct, ...
Three Model Perspectives

Team Model

Data Model

Process Model

Code
public class FriendGreeting extends Capability {

    public Relation friends = new Relation( "friends", String.class );

    public FriendGreeting() {
        addGoal( new Plan( "say hello" , new Goal [] {
            new BDIGoal( "say friendly hello" ),
            new BDIGoal( "review things lent to friend" )
        } ) {
            public Query context(Data d) {
                return friends.get( d.getValue( "who" ) );
            }
        } );
    }
}
public class Friendly extends Performer {

    public Friendly(String name) {
        addCapability(new FriendlyGreeting());
    }

    public Perceptor hello = new Perceptor(this, "say hello") {{
        data_name = "who";
    }};
}

[Friendly]

hello.perceive( ... )
new Thread( Executor.getDefaultExecutor() ).start(); // Start GORITE

Performer kevin = new Friendly( “kevin” );

kevin.hello.perceive( “marge” );
Executor Perspective

[Executor] “[gorite]”

[Performer] “kevin”

[FriendlyGreeting]

friends (String)

[Plan] “say hello”

BDIGoal(“say hello”).Instance

perceive(“marge”)

[Perceptor]
goal = “say hello”
data= “who”

[Data]
who=“marge”

run()
Process Model Execution

- Goal performed by “instantiating” it
  - represents intention
  - Goal.Instance
    - extension according to goal type
    - keeps state and has progress method
  - single thread progressing all intentions of all teams
  - step may fail, pass, stop or block
- Data object
  - named data elements
BDIGoal Execution

- a BDIGoal of given name is performed by:
  - look up the plans of that same name within the current capability;
  - apply their context queries to determine variants;
  - select one and perform it;
    - if it fails:
      - restore data context
      - redo procedure, but exclude failed variants
  - succeed as soon as a plan variant succeeds
    - preserve the plan variant's data context
Current Capability

- data element “agent”
  - the current performer (which is a capability)
- recursive lookup:
  - all plans in current capability and
  - any sub capability
- looking for plans of same name as BDIGoal
Context Query

- a “logical query” possibly involving “logical variables” (Ref objects)
  - Ref $x = new Ref( "$x" )
- each valid binding defines a plan variant
- when selected, the Data is updated with the selected binding
  - data.getValue( "$x" )
  - restored on failure, preserved on success
addGoal( new Plan( "say hello" , new Goal [] { 
new BDIGoal( "say friendly hello" ), 
new BDIGoal( "review things lent to friend" ) 
} ) { 
    public Query context(Data d) { 
        return friends.get( d.getValue( "who" ) ); 
    } 
};
Context Method Return

- Query object
  - treated as predicate $P(x_1,..,x_n)$
  - processed exhaustively to generate plan variants
  - may provide zero variants
- null
  - "true"; one variant of the plan
- Context.EMPTY
  - "false"; the plan is not applicable
Composite Queries

- **And** (Query... conjuncts)
- **Or** (Query... disjuncts)
- **Condition**
  - public boolean condition() { ... }
- **Distinct** (Ref... variables)
- **Snapshot** (Query query)
- interface Query, class QueryBase
Ref Objects

- named
  - new Ref( String name )
  - name used when context established in Data
- void set( Object value )
  - null means un-set
- Object get()
Plan Variants

- when / why would you want multiple plan variants simultaneously applicable?
Data

- goal execution is done with respect to a Data container object
  - container of named data elements
    - multi-valued ("temporal")
    - gold-fish mode
  - transient, dynamic bindings
    - name lookup relative intention
      - i.e. name "foo" may differ on parallel paths
Recall Example Plan Again

```java
addGoal( new Plan( "say hello" , new Goal [] { 
    new BDIGoal( "say friendly hello" ),
    new BDIGoal( "review things lent to friend" )
} ) { 
    public Query context(Data d) {
        return friends.get( d.getValue( "who" ) );
    }
} );
```

plan body = sub goals

Plan extends SequenceGoal
Example Goal Hierarchy

[Plan] “say hello”
{ friends(“who”) }

1

[BDIGoal]
“say friendly hello”

2

[BDIGoal]
“review things lent to friend”
SequenceGoal Execution

- performs each sub-goal in sequence
  - if sub goal fails, then the SequenceGoal fails
  - if sub goal blocks, then the SequenceGoal blocks
    - re-enters same sub goal on re-entry
  - if sub goal stops, then the SequenceGoal stops
    - re-enters same sub goal on re-entry
  - if sub goal passes, then the next sub goal is instantiated and performed
    - if last sub goal, then the SequenceGoal passes
  - if sub goal throws exception, the SequenceGoal throws
ParallelGoal Execution

- performs all sub goals “in parallel”
- instantiates all as “branches”, and performs one
  - if sub goal fails, all other branches are cancelled, and the ParallelGoal fails
  - if sub goal blocks or stops, the ParallelGoal shifts focus to the next branch
    - if all sub blocks or stops, the ParallelGoal blocks or stops
  - if sub goal passes, the ParallelGoal shifts focus to next branch
    - if all have passed, the ParallelGoal passes
  - if sub goal throws exception, all branches are cancelled and the exception is re-thrown
ConditionGoal Execution

- tries sub goals in sequence until one passes
  - on sub goal fail: instantiate and try next
  - on sub goal pass: pass
  - on sub goal block: block
  - on sub goal stop: stop
  - on sub goal exception: cancel and re-throw
... and so forth

- SequenceGoal, ParallelGoal, ConditionGoal
- FailGoal, ControlGoal, RepeatGoal,
- LoopGoal, EndGoal
- BDIGoal, TeamGoal
- Plan, Goal
TeamGoal Execution

- find fillers as data element named by role
  - Performer.RoleFilling objects
  - splits into a parallel branch for each filler
- BDI style goal processing
  - using the Performer.RoleFilling capability
    - instantiated to combine Role and Performer
    - Role goals performed by team, focussed on filler
    - otherwise execution is “transferred into performer”
      - by manipulating the “agent” data element
Transient Shared Data

[Team]

[Data]

“foo” = \{v1, v2, ..\}
“bar” = \{a, b, c, ..\}
“fum” = \{..\}
“agent” = \{current\}
“…” = \{..\}

[Performer]
The Team Picture

[Capability]

Beliefs

[Performer]

(intention)

[Team]

[TaskTeam]

[Role]  [Role]  [Role]
an Executor object that all performers are added to

for executing TodoGroup intentions

default

- single Executor, automatic, lazily

- if a Performer lacks an Executor when one is needed, then the default one is use (created first)
Executor Perspective

Executor

[Performer] Team

[Performer] Team

[Performer] Team

Performer

Performer

Performer
Two Execution Entry Points

- Perceptor.perceive(Object)
  - asynchronous events model
  - Executor.run()
- Performer.performGoal(Goal, String, Data)
  - application main thread model
    - e.g. closed loop simulation
  - traverses intention tree
    - down to TodoGroup intentions
    - then, later, into those via Executor
Perceptor = Percept Stream

new Perceptor( Performer, String )

- handler goal .. new BDIGoal( “handle percept” )
- data name .. “percept”
- todo group name .. “percepts”
- performer

perceptor.perceive( perception_data );
Thread Control Picture (In)

run()

EITHER!

performGoal()

NOT BOTH!

perceive(Object)

percept

[Executor]

[Performer]

[Performer]
The Executor

- runs through all (its) performers
  - performing their todo groups
    - each intention until it passes, fails, stops or blocks
  - performGoal(..) uses Executor
- run() method
  - implements Runnable
- usually single Executor
  - each Executor claims its own “threading sandbox”
  - No Shared Performers !!!
PeformerTodoGroup

- a Performer's named collection of some on-going “intentions” managed together
  - Goal.Instance objects (sub classed)
  - meta goal invoked to manage todo group
    - decide which one to progress next
      - only one intention is progressed
      - new decision on added, completed or blocked intention
  - goals have “group” attribute
    - progressed by TodoGroup execution
Thread Control Picture (Out)

1. Performer
   - task goal
   - Java code

2. [Equipment]
   - Action. execute

3. RemoteCoach
   - RemoteGoal
   - check status
   - initiate connection

Executor

Performer

Connection

Connector
new Performer("ralph") . performGoal(
    new Goal("say hello") {
        public States execute(Data d) {
            System.out.println("hello world");
            return PASSED;
        }
    }, null, new Data());

- Performer.performGoal( Goal, String, Data )
Capability and Desire

Performer ralph = new Performer( "ralph" ) {{
    addGoal( new Goal( "say hello" ) {
        public States execute(Data d) {
            System.out.println( "hello world" );
            return PASSED;
        }
    } );
}}

ralph.performGoal( new BDIGoal( "say hello" ), "x", new Data() );
Task Goal

- a task goal instance invokes the execute method of the enclosing goal:
  
  ```java
  public States execute(Data d)
  ```

- returns one of
  
  PASSED, FAILED, BLOCKED, STOPPED, CANCEL

- or throws exception
  
  LoopEndException, ParallelEndException
Action

- to represent “equipment interface”
- one equipment - many actions
- action goal
  - a goal that is achieved by performing the action
  - reference to action within goal hierarchy
  - many action goals for the same action (object)
  - input/output data elements named by goal
    - Data.Element [] ins, Data.Element [] outs
public class SteeringSystem {
    Action turn_left = new Action( “turn left” ) {
        public States execute(
            boolean reentry, Data.Element [] ins, Data.Element [] outs ) {
                ......
            }
    }
    Action turn_right = new Action( “turn right” ) {
        public States execute(
            boolean reentry, Data.Element [] ins, Data.Element [] outs ) {
                ......
            }
    }
}
public class Steering extends Capability {

    public Steering(SteeringSystem ss) {
        addGoal( ss.turn_left.create(
            new String [] { “foo”, “bar” }, new String [] { “fum”, “car” } ) );
        addGoal( ss.turn_right.create(
            new String [] { “foo”, “bar” }, new String [] { “fum”, “car” } ) );
    }

    }

}
Remote Connection

- RemoteCoaching( connector )
  - capability (base class)
- RemoteGoal
  - task goal created within a RemoteCoaching capability
  - invokes the connector.perform method
  - input / output data (service request / response)
  - monitoring progress
Connector Model

- Connector
  - manages “channel” to service provider
- Connection
  - instantiated for each service use (session)

```java
public class MultAndCoolyLink implements RemoteCoaching.Connector {
    public RemoteCoaching.Connection perform(String goal, ... ) {
        // ...
    }
}
```
RemoteCoaching.Connection

- cancel()
  - called by RemoteGoal being cancelled
- States check()
  - called by RemoteGoal querying completion
    - fail, pass, stop or block
  - RemoteGoal updates Data on completion
    - via the Ref objects given to Connector.perform
public class Proxy extends Performer {
    public Proxy(String name) {
        super( name );
        addCapability(
            new RemoteCoaching( new MultAndCoolyLink() ) {
                addGoal( new RemoteGoal( "service 1", .... ) );
                addGoal( new RemoteGoal( "service 2", .... ) );
            }
        );
        addCapability( new Capability() {{ .... }} );
    }
}
GORITE Execution Summary

- Perceptor, Executor.run(), performGoal(..)
- goal instances, todo groups
- Goal.States, exceptions
  - pass, fail, stop, block
- Plan, Context, Query, Relation, Ref objects
- Data, named data elements
- equipment modelling, RemoteCoaching
Not Covered

- actions sequenced by data element readiness
- plan precedence
- relation keys
- trace naming of intentions (x.3:4.5*6:2)
- dealing with data at intention fork and join
- blocking and triggering