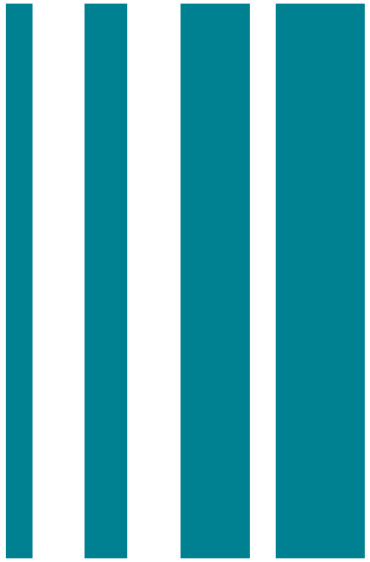


A Short Introduction to JIAC

Version 5.2.4

November 2019

DAI-Labor
Distributed Artificial Intelligence Laboratory
berlin

Useful Resources

Everything you need for getting started with JIAC

- These Slides
  - Short overview: What is JIAC? What can it do?
  - Basic JIAC concepts, tools, etc.
- JIAC Website: http://www.jiac.de/agent-frameworks/jiac-v/
  - Manual with detailed examples and tutorials
  - Developer Documentation
  - Papers and Publications
  - Download JIAC and tools
- Questions & Answers: http://www.jiac.de/qa
  - Frequently (and infrequently) asked questions
  - Whatever else you need to know (just ask!)

JIAC Basics

What is this „JIAC“ anyway, and why should I care?

- Easy creation of complex, scalable distributed systems
- Quick to learn for developers familiar with Java
- Integration of agents and service-oriented architecture
- Abstraction from underlying network architecture
- Transparent communication across the network
- Many extensions for: migration, persistence, encryption, reactive behavior, service matching, load balancing, ...

JIAC Basics (cont.)

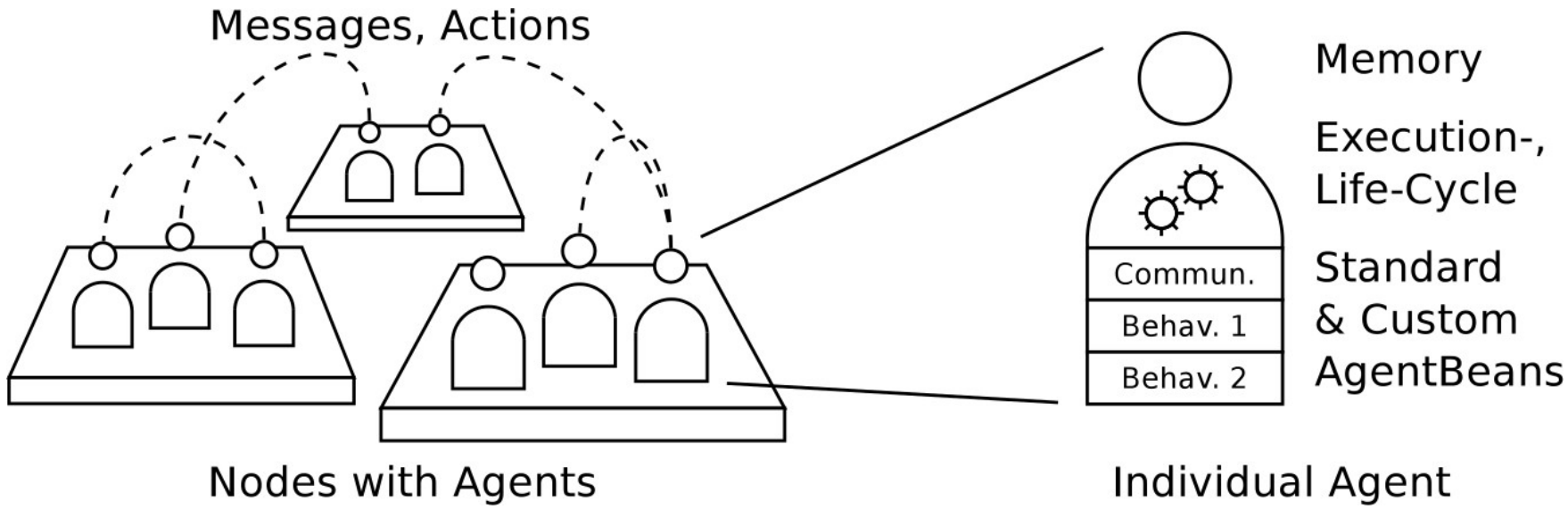
What else?

- Built on and around proven standards
- Core configuration: Java 8+, Spring, Maven
- Communication: ActiveMQ, JMS
- Semantics: OWL, OWL-S, SWRL
- Interfaces: JMX, REST, Webservices

JIAC Architecture

Architecture of a distributed JIAC application

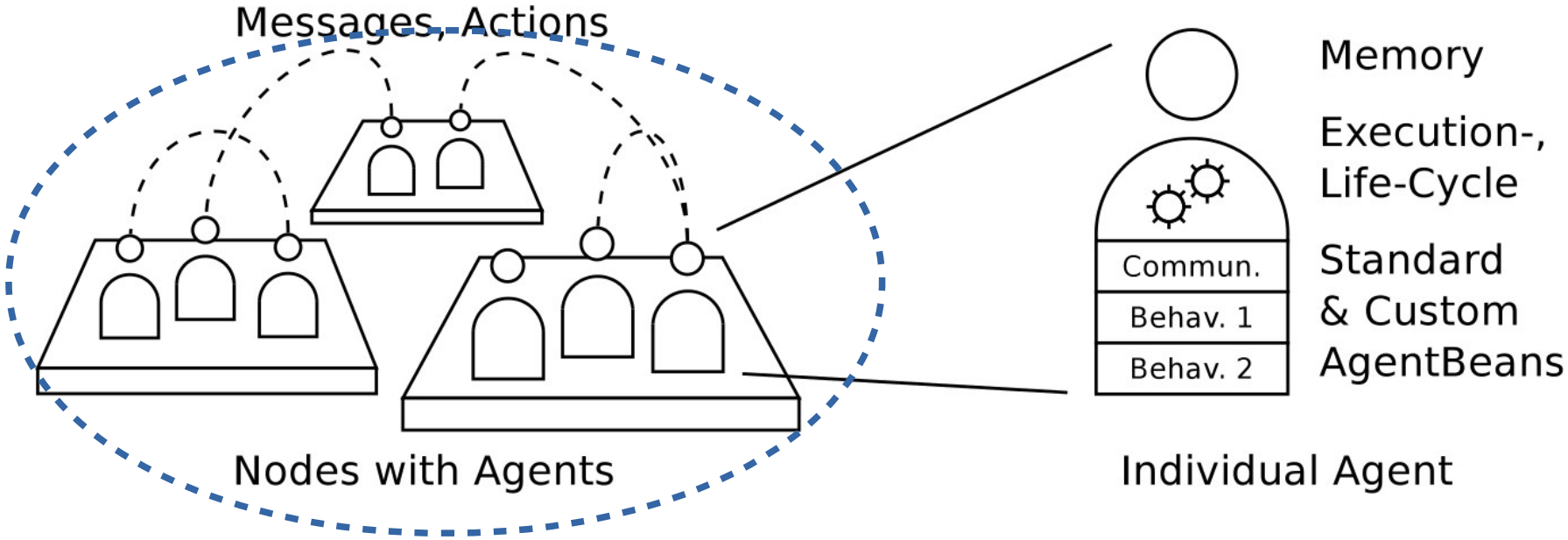
- **Agent platform** composed of several **nodes** (one node per JVM)
- Communication via messages and actions / services
- **Agents** composed of different **agent beans**
- Agents have lifecycle, execution-cycle, memory
- Beans provide actions, lifecycle-callback methods, listeners



JIAC Architecture: Platform

What is a „Platform“?

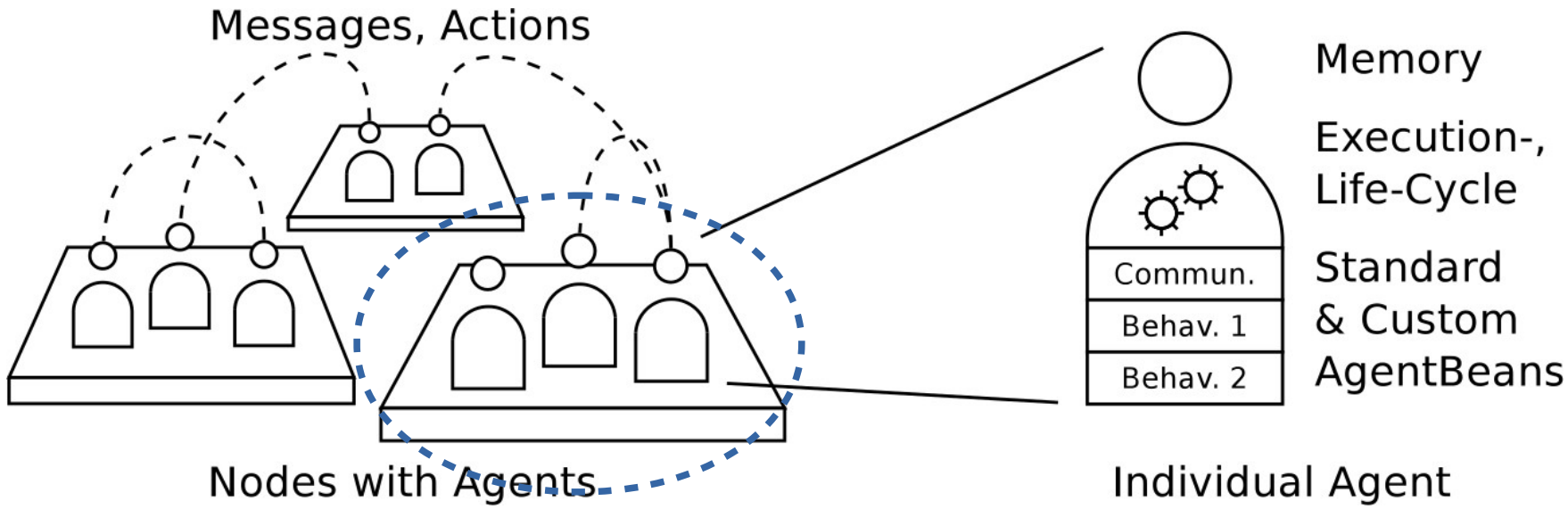
- Implicitly formed by all nodes that can communicate with each other
- By default: all nodes in the local network; can be restricted and/or extended using discovery URIs or dedicated Gateway nodes
- Using ActiveMQ for inter-node communication
- Available agents, actions are broadcasted to directories on all nodes in the platform
- If one node fails, others can still continue to work (except for the Gateway, if any)



JIAC Architecture: Nodes

What is a „Node“?

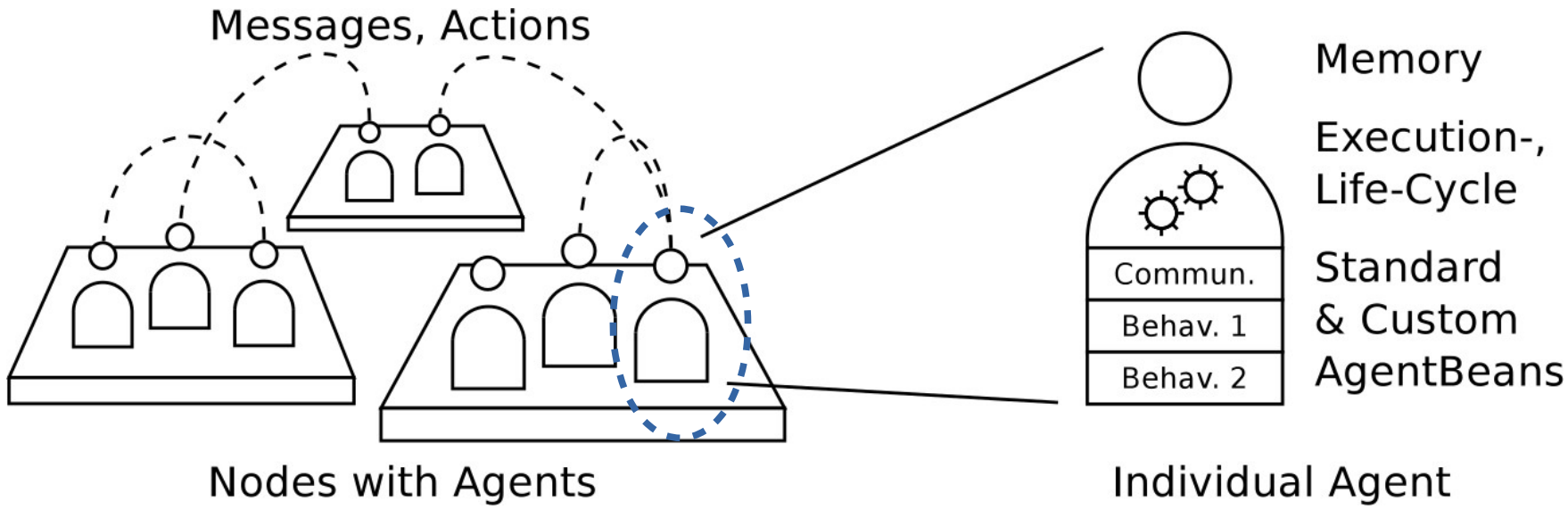
- Each node is running in a separate Java runtime (JVM)
- Configured using Spring, together with all agents on that node, and their beans
- Provides some infrastructure services, e.g. Directory, Transport Security, Webserver, Gateways for interaction with WSDL or REST services, etc.



JIAC Architecture: Agents

What is an „Agent“?

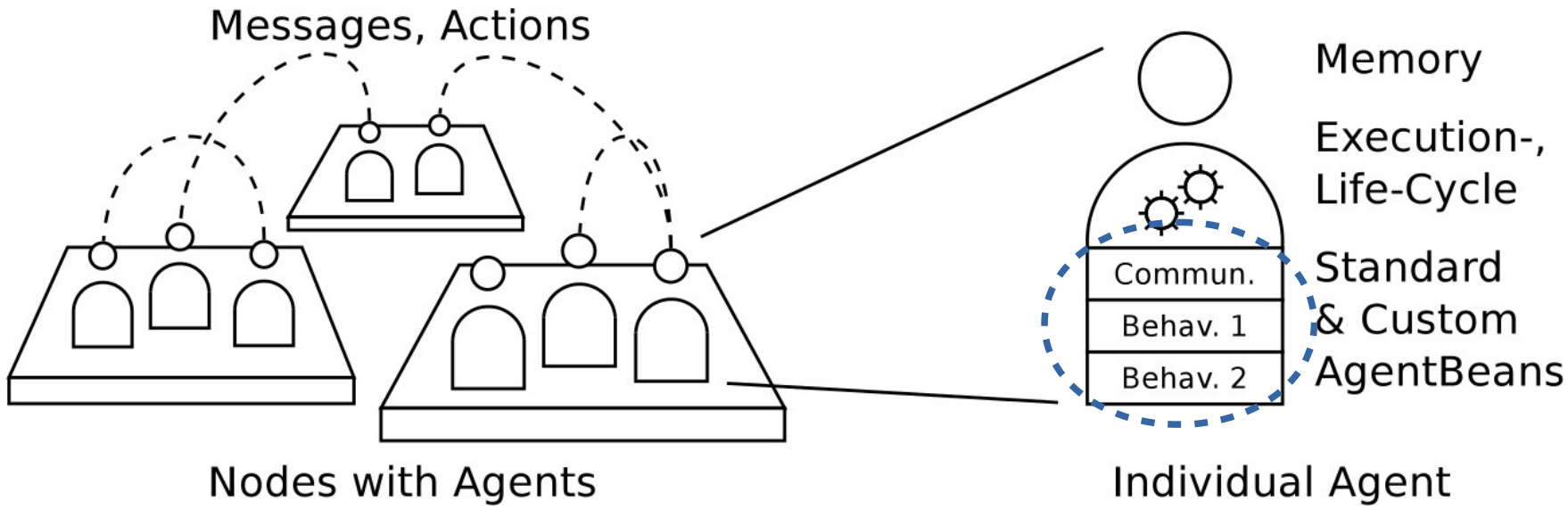
- Each agent is executed in an individual thread
- Container for Agent Beans defining the actual behaviour of the agent
- Controlled by its „life-cycle“ and (when active) „execution-cycle“
- Some infrastructure: Communication Bean for interaction with other agents, tuple-space based memory



JIAC Architecture: Beans

What is a „Bean“?

- Two flavours: Agent Bean and Agent Node Bean
- Define the actual behaviours and capabilities of the agents (and the nodes)
- Some standard-beans part of every agent and agent node, e.g. for communication
- Some predefined beans for more complex capabilities, e.g. for rule-based execution, business process interpreters, etc.
- All application-specific behaviour is also defined in agent beans



JIAC Behaviors: AgentBeans and More

The agents‘ behaviours and capabilities are defined in AgentBeans

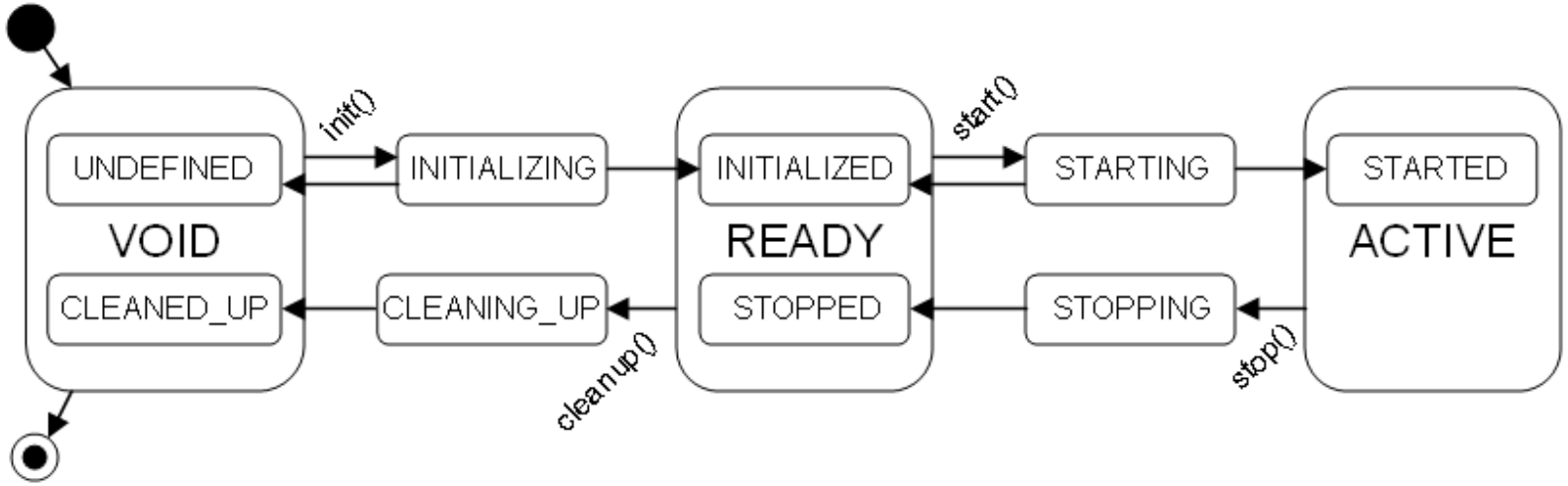
- AgentBeans: application-specific modules for JIAC agents
- Different mechanisms for specifying behaviors
  - Provide lifecycle-methods (executed e.g. when starting/stopping)
  - Implement execute-method (executed in regular intervals)
  - Attach observers to agent’s memory (e.g. for messages)
  - Expose methods as “actions” (invoked by other agents)

- Additional/alternative mechanisms (not covered here)
  - Reactive behavior with Drools rules
  - Dynamic services with JADL++ scripts
  - Business process execution with BPMN

JIAC Behaviors: Lifecycle & Execution Cycle

The agent‘s „inner clock“

- Each component in JIAC has a lifecycle – agents, agent (node) beans, nodes
- Defines what the agent (or bean or node) has to do when it is starting, stopping, etc.
- Execution cycle: what to do in each „step“ while running (active), e.g. check messages, execute actions, etc.
- Agent beans can define `execute()` method for additional application-specific behaviour to be regularly executed



JIAC Behaviors: Agent Memory

A place for agent beans to store and share information

- Implemented as a „tuple-space“: data objects are not named, but can be put into the „space“ and be retrieved using an appropriate template
- All beans of the agent have access to the memory
- Can attach observers to be notified when facts are added/updated/removed
- Used for both, application-specific data, and e.g. other known Agents and Actions, incoming messages, etc.
- Objects have to define Getter *and* Setter methods for all relevant attributes

```
memory.write(new Person("Peter", 1992, "m"));
memory.write(new Person("Thomas", 1976, "m"));
memory.write(new Person("Anna", 1995, "w"));
...
res = memory.readAll(new Person(null, null, "m"));
-> [Person("Peter", 1992, "m"), Person("Thomas", 1976, "m")]
```

JIAC Behaviors: Actions

Capabilities provided to other agents

- Actions are advertised to other beans of the same agent, other agents on the node, or agents on other nodes depending on their scope (`AGENT`, `NODE`, `GLOBAL`), or exposed as WSDL or REST services (scope `WEBSERVICE`, with the right node bean)
- Executed in the agent's thread (`SimpleAgent`) or its own threads (`NonBlockingAgent`)
- Can have parameters and zero, one, or multiple return values (all serializable!)
- Defined using `getActions` and `doAction` methods, or (much simpler) by adding the `@Expose` annotation to any of the agent bean's methods

```
@Expose(name="Do Something", scope=GLOBAL)
public Stuff doSomething(Foo x, Bar y) {
    ...
}
```

- Use `thisAgent.searchAction` to search action based on template (similar to memory), then `invoke` (asyncronous) or `invokeAndWaitForResult` to call it (syncronous)

JIAC Behaviors: Actions

Capabilities provided to other agents

- Without `@Expose` : using `getActions` and `doAction` methods

```
public List<Action> getActions() {
    // assemble & return descriptions for all
    // Actions provided by this agent bean
}
public void doAction(DoAction doAction) {
    // decide what to do based on DoAction
    // write ActionResult to agent‘s memory
}
```

- Offers more control, e.g. adding or removing actions at runtime
- Should only be used if you know what you are doing, else just use `@Expose`

JIAC Development Tools

What is needed for developing JIAC agents?

- JIAC is Java-based – just use your usual Java development stack
  - JIAC development in Eclipse or IntelliJ or whatever you like
  - Project and dependency management with Maven
- Optional Eclipse plugins: ‘JIAC Toolipse’
  - JIAC-specific new-project-wizard
  - Integration with running JIAC platforms: service-search and import, testing and deployment
  - Visual editor for agent configurations (AWE)
  - BPMN-based business process modelling (VSDT)
  - Editors and tools for JADL++ scripts
- Runtime monitoring: ASGARD agent viewer

Getting Started

Time to get your hands dirty...

1. Go to www.jiac.de and download the manual
2. Follow the steps to set up a new JIAC project; Maven will automatically load the JIAC libraries and dependencies
3. Create the “Hello World” and “Ping-Pong” examples
4. Experiment with different ways to expose and invoke actions, and different life-cycle methods
5. For the start, try to stick with basic Java and XML tools until you are comfortable with how things work
6. If you have problems or questions, go to `www.jiac.de/qa`