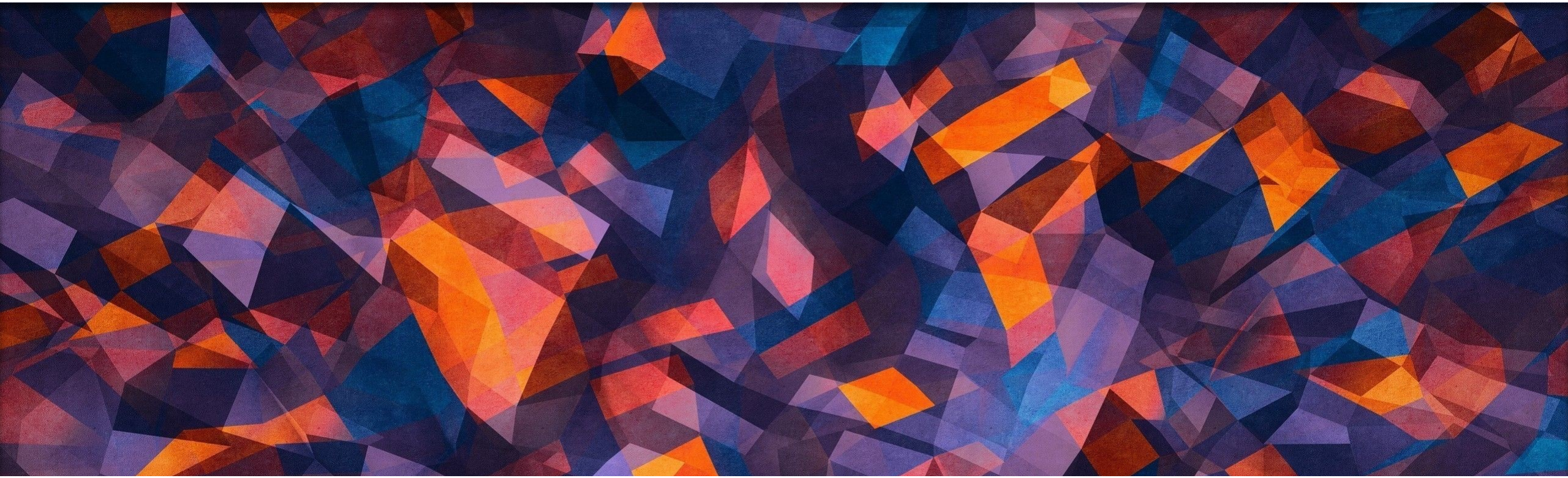
Universität St.Gallen



Web-based Autonomous Systems

Autonomous Agents: Architectures and Programming Paradigms

Chair for Interaction- and Communication-based Systems (ICS-HSG)

Our Journey

Universität St.Gallen

Prerequisites:
- **ASSE**
- **(…)**

Week 1:
Introduction

Week 2:
A Web for Machines

Week 3:
Knowledge Representation and Reasoning for the Web

Week 4:
Linked Data and Distributed Knowledge Graphs

Week 5:
Autonomous Agents (Arch. and Programming)

Week 6 (Coordination I):
Agent Communication and Interaction

Week 7 (Coordination II):
Self-Organization and Stigmergy

Week 8 (Coordination III):
Normative MAS and Organizations

Week 9 (Coordination IV):
Trust & Reputation

Week 10:
Game Theory and Social Choice

Week 11:
Reinforcement Learning and Multi-Agent Learning

Week 12:
An Industry Perspective

Exercises

Ex1: Writing Your First Agent(s)!
Ex2: Automated Planning
Ex3: Web Ontologies
Ex4: Operating on Linked Data
Ex5: BDI Agents
Ex6: Interacting Agents on the Web
Ex7: Ant Colony Optimization
Ex8: Organized Agent
Ex9: Trustworthy Agents
Ex10: Axelrod's Agents
Ex11: Reinforcement Learning Agents
Course Review and Q&A

Our learning objectives so far…

Learning objectives

- Students understand and can explain the concept of autonomy, how it applies to software systems, and what are the main technological drivers and limitations behind the engineering of autonomous systems.
- Students understand and can explain in depth the design of the Web Architecture, the latest developments, and the tradeoffs and opportunities they bring to the engineering of Web-based autonomous systems.
- Students understand and gain experience with the formal representation and usage of knowledge on the Web.
- Students can design open and large scale Web based autonomous systems - and can explain the implications of their

abilities to prototype autonomous systems on the Web and off

Universität St.Gallen

Autonomy as a Relational Notion

A multifaceted view drawing from cognitive science [Castelfranchi, 1993; Castelfranchi and Falcone, 2003]

An entity X is autonomous from Y about P, where:

- X: the main entity whose autonomy is considered/evaluated
- P: a function/action/goal that must be realized or maintained by the main entity and on which the autonomy is evaluated
- Y: the secondary entity (human, artificial agent, environment, organization, etc.) with respect to whom X should be considered autonomous given the specified function/action/goal P

Concrete examples:

- autonomy ***from other agents***
- autonomy ***from organizations***
- autonomy ***from the environment***

C. Castelfranchi. *Guarantees for Autonomy in Cognitive Agent Architec*
C. Castelfranchi, R. Falcone. *From Automaticity to Autonomy: The Frontier of Artificial Agents*. In: Hexmoor H., Castelfranchi C., Falcon
Autonomy. Multiagent Systems, Artificial Societies, and Simulated Organizations (International Book Se

Web-based Autonomous Systems — Institute of Computer Science (ICS-HSG)

With **limitations** & state-of-the-art **tooling**

Technological drivers discussed so far:

- automated planning (“AI planning”)
- knowledge representation and reasoning (knowledge graphs)
- Linked Data
- Re-decentralizing the Web (Solid)

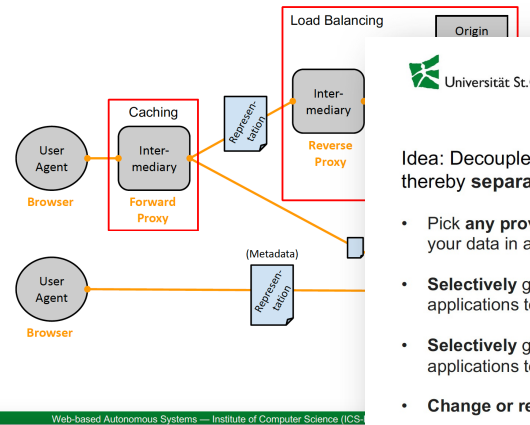
Our learning objectives so far…

Learning objectives

- Students understand and can explain the concept of autonomy, how it applies to software systems, and what are the main technological drivers and limitations behind the engineering of autonomous systems.
- Students understand and can explain in depth the design of the Web Architecture, the latest developments, and the tradeoffs and opportunities they bring to the engineering of Web-based autonomous systems.
- Students understand and gain experience with the formal representation and usage of knowledge on the Web.
- Students can design open and large-scale Web-based a… ons of their

Universität St.Gallen

Essence of the World Wide Web



Universität St.Gallen

httpRange-14

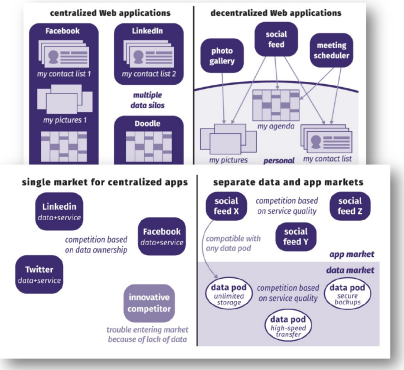
All design decisions underlying the Web Architecture are documented and available online…

Universität St.Gallen

World-Wide Linked Data: Personal Data Pods

Idea: Decouple data storage from applications and thereby **separate markets for data and applications**

- Pick **any provider** (or your own Web server) to store your data in a personal data store, or in multiple stores
- **Selectively** give (fine-granular) permission to applications to **access** your data (but not store it)
- **Selectively** give (fine-granular) permission to applications to **write** to your personal data stores
- **Change or revoke** these permissions at any time



Solid (Social Linked Data)
https://solidproject.org/

Simon Mayer, Advanced Software and Systems Engineering, University of St. Gallen, 2021
https://www.societybyte.swiss/en/2021/01/14/taking-back-control-of-our-data-re-decentralizing-the-web-for-good-this-time-part-2/

Web-based Autonomous Systems — Institute of Computer Science (ICS-HSG) 31

Universität St.Gallen

Operating on Linked Data: An Overview

Several **W3C Recommendations** and **open standards** allow software clients to ***browse***, ***query***, ***observe***, and ***act on*** the hypermedia fabric of the Web.

Query | Act | Observe

SPARQL (querying a single endpoint)

Federated SPARQL (querying across endpoints) https://www.w3.org/TR/sparql11-federated-query/

Linked Data Fragments
- beyond the scope of our course — but **seriously cool**! (HATEOAS meets SPARQL)

https://linkeddatafragments.org/

Recommended (not required) research talk from ISWC* 2014:
http://videolectures.net/iswc2014_verborgh_querying_datasets/

* International Semantic Web Conference (ISWC)

Web-based Autonomous Systems — Institute of Computer Science (ICS-HSG) 34

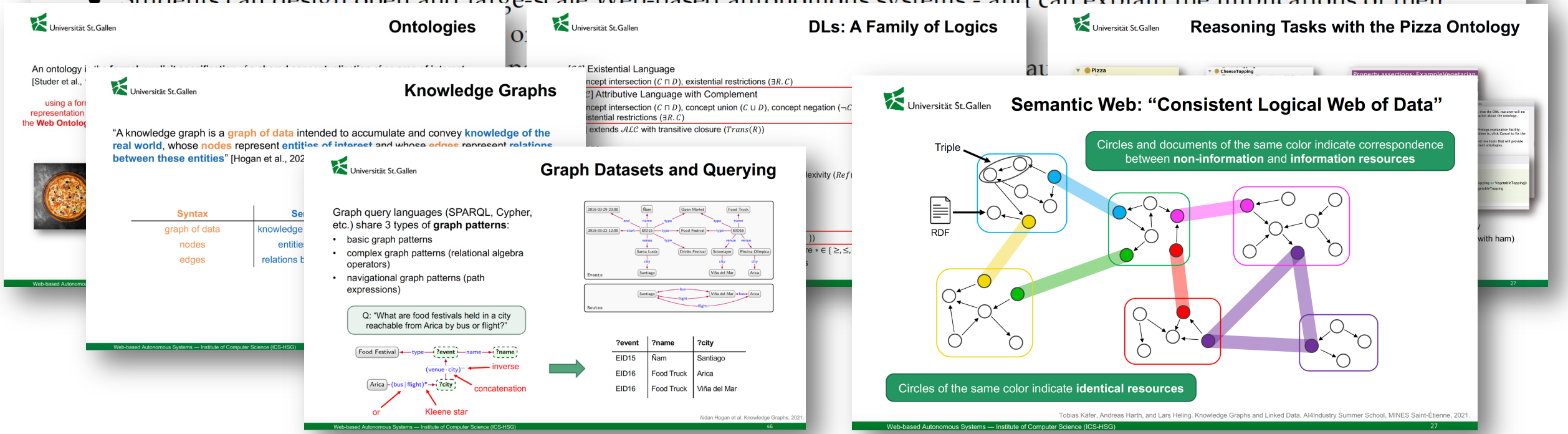
Our learning objectives so far…

URIs, etc.

"Centralized resources on a decentralized web"

Learning objectives

- Students understand and can explain the concept of autonomy, how it applies to software systems, and what are the main technological drivers and limitations behind the engineering of autonomous systems.
- Students understand and can explain in depth the design of the Web Architecture, the latest developments, and the tradeoffs and opportunities they bring to the engineering of Web-based autonomous systems.
- Students understand and gain experience with the formal representation and usage of knowledge on the Web.
- Students can design open and large-scale Web-based autonomous systems - and can explain the implications of their



Dereference URI GET 200 : information resource
304 : representation

Our learning objectives so far…

Learning objectives

- Students understand and can explain the concept of autonomy, how it applies to software systems, and what are the main technological drivers and limitations behind the engineering of autonomous systems.
- Students understand and can explain in depth the design of the Web Architecture, the latest developments, and the tradeoffs and opportunities they bring to the engineering of Web-based autonomous systems.
- Students understand and gain experience with the formal representation and usage of knowledge on the Web.
- Students can design open and large-scale Web-based autonomous systems - and can explain the implications of their design choices for the engineering of such systems.
- Students possess the relevant engineering knowledge and abilities to prototype autonomous systems on the Web and off the Web.

broader than interoperability

Openness: The ability of a system to accommodate new components, which may or may not have been foreseen at design time.

Scalability: The ability of a system to meet new requirements of size and scope.

broader than accommodate many conc. users

> TDs (Ontologies, semantics)
> Coordination: Join multi-agt sys., how to interact, trust
> Regulation

Let's discuss!

> Link-traversal: Architecture of the web! SPARQL
↳ design multi-agt. sys on web
inherit prop.

Universität St.Gallen

Our learning objectives so far…

Learning objectives

- Students understand and can explain the concept of autonomy, how it applies to software systems, and what are the main technological drivers and limitations behind the engineering of autonomous systems.
- Students understand and can explain in depth the design of the Web Architecture, the latest developments, and the tradeoffs and opportunities they bring to the engineering of Web-based autonomous systems.
- Students understand and gain experience with the formal representation and usage of knowledge on the Web.
- Students can design open and large-scale Web-based autonomous systems - and can explain the implications of their design choices for the engineering of such systems.
- Students possess the relevant engineering knowledge and abilities to prototype autonomous systems on the Web and off the Web.

So far, focus on **specific methods** for enabling autonomy

– knowledge representation, reasoning, automated planning, etc.

Today, we'll start prototyping autonomous agents

Universität St.Gallen

Our Journey

Prerequisites:
- **ASSE**
- **(…)**

Week 1:
Introduction

Week 2:
A Web for Machines

Week 3:
Knowledge Representation and Reasoning for the Web

Week 4:
Linked Data and Distributed Knowledge Graphs

Week 5:
Autonomous Agents (Arch. and Programming)

Week 6 (Coordination I):
Agent Communication and Interaction

Week 7 (Coordination II):
Self-Organization and Stigmergy

Week 8 (Coordination III):
Normative MAS and Organizations

Week 9 (Coordination IV):
Trust & Reputation

Week 10:
Game Theory and Social Choice

Week 11:
Reinforcement Learning and Multi-Agent Learning

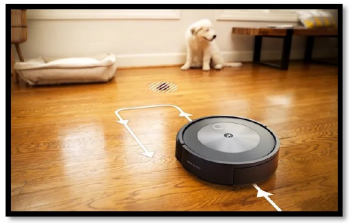
Week 12:
An Industry Perspective

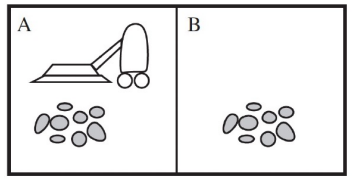
Exercises

Ex1: Writing Your First Agent(s)!
Ex2: Automated Planning
Ex3: Web Ontologies
Ex4: Operating on Linked Data
Ex5: BDI Agents
Ex6: Interacting Agents on the Web
Ex7: Ant Colony Optimization
Ex8: Organized Agent
Ex9: Trustworthy Agents
Ex10: Axelrod's Agents
Ex11: Reinforcement Learning Agents
Course Review and Q&A

Lecture #1: The Vacuum-Cleaner Agent Revisited

Example: A Vacuum-Cleaner Agent





Vacuum-cleaner world with **2 locations** (*A* and *B*)

Percepts: [*location*, *status*], e.g. [*A*, *Clean*]
Actions: *MoveRight*, *MoveLeft*, *Vacuum*, *DoNothing*

Partial tabulation of the **agent function**:

Example: A Vacuum-Cleaner Agent

The **agent function** is implemented by an **agent program** running on an **agent architecture**

- **agent function**: abstract mathematical description of the agent's behavior
- **agent program**: concrete implementation of the agent's behavior
- **agent architecture**: more of that Lecture #5!

```
1  function Reflex-Vacuum-Agent([location,status]) returns an action
2    if status == Dirty then return Vacuum
3    else if location == A then return MoveRight
4    else if location == B then return MoveLeft
```

Agent program for a simple vacuum-cleaner agent that implements our agent function.

Reactive Agent

We can say a lot about an agent by **observing** its behavior (without looking into the agent's internals)
– in practice, this is often necessary (different agent architectures, programming languages, etc.)

Stuart Russell and Peter Norvig (2020) Artificial Intelligence: A Modern Approach (Fourth Edition).

Universität St.Gallen

Types of Agents (for today's lecture)

Can not perceive Situation, only do steps of action

Automated Planning

Situation → **Action**

Action is **coupled** to perception

- agents react to percepts (**reactive behavior**)
- agents are usually programmed in terms of **condition-action rules** "if X then Y"
- agents **may or may not** have a model of their environment
- agent communication is usually **indirect** through the environment (stigmergy)

Reactive Agents

Situation → **Deliberation & Means-End** → **Action**

Action is **decoupled** from perception through deliberation & means-end reasoning

- agents react to percepts and pursue long-term goals (**reactive** and **proactive behavior**)
- agents are usually programmed in terms of **mental states** (in the ***agent-oriented programming*** paradigm)
- agents usually **have a model** of their environment (current state of e., goal state of e.)
- agent communication is both **direct** through messages and **indirect** through the environment

Deliberative Agents

≙ Cognitive Agt.

but w/o mental states

Both types of agents are useful depending on the design requirements at hand

In this course, we are taking a software engineering viewpoint (not an AI viewpoint)

Lecture #11: **reinforcement learning agents**

From Software Architecture…

ASSE Review

Universität St.Gallen

Software Architecture

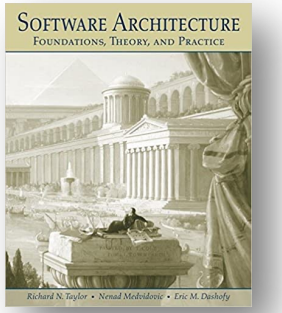
Software architecture consists of the *structure* of the system, combined with *architecture characteristics* ("-ilities") the system must support, *architecture decisions*, and finally *design principles*.

Source: Fundamentals of Software Architecture

REST design decisions

↓

properties

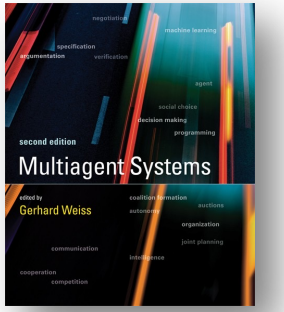


"A software system's architecture is **the set of principal design decisions** made about the system."

— Richard N. Taylor et al., 2010: Software Architecture: Foundations, Theory, and Practice

Universität St.Gallen

…to Agent Architecture

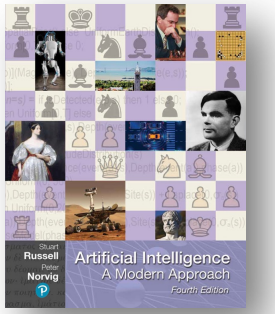


deliberate, reason

"Agent architectures are **software architectures for decision-making systems** that are **embedded in an environment**."

— Michael Wooldridge, 2013: Intelligent Agents. In Multiagent Systems (Second Edition).

perceive & act



"The job of AI is to design an agent program that implements the agent function—the mapping from percepts to actions. We assume this program will run on **some sort of computing device with physical sensors and actuators**—we call this the **architecture**: *agent* = *architecture* + *program*."

—Russell and Norvig, 2020: Artificial Intelligence: A Modern Approach (Fourth Edition).

In **software agents**, we think of the **agent architecture** as a **Process Virtual Machine**

create / kill

Process VM: Agt. is process on a VM

Universität St.Gallen

Today's Agenda

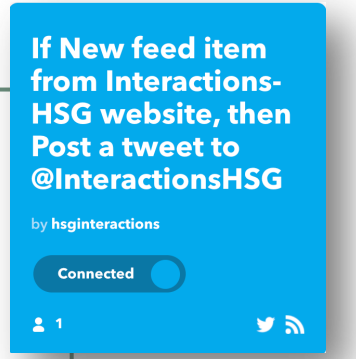
- Introduction
- Reactive Agents
  - The Subsumption Architecture
- Deliberative Agents
  - Agent-Oriented Programming
  - The Belief-Desire-Intention (BDI) Architecture
  - Programming BDI Agents in Jason

Reactive Web (of Things) Systems

Rule-Based Programming

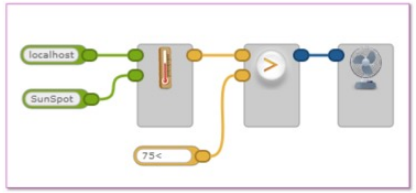
IFTTT
http://ifttt.com

zapier
http://zapier.com

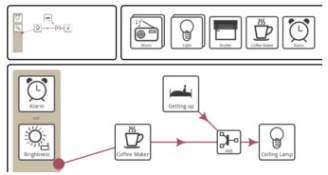


We can use **rule-based programming for Linked Data as well**
(e.g., see [Käfer & Harth, 2018])

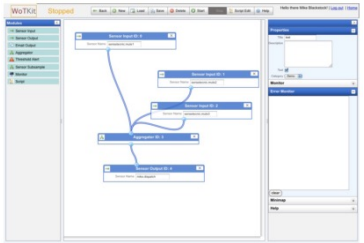
Dataflow programming for the Web of Things



[Guinard et al., 2009]



[Rietzler et al., 2013]

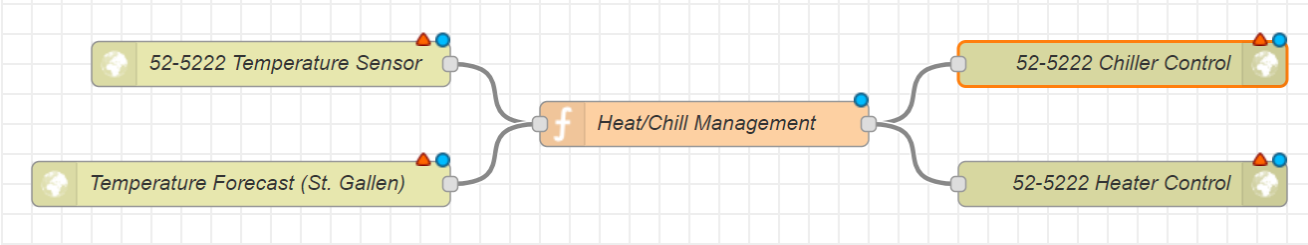


[Blackstock & Lea, 2012]

Node-RED

"Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways.

http://nodered.org



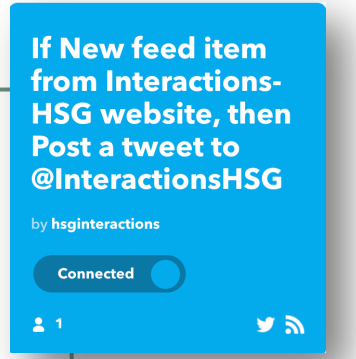
T. Käfer and A. Harth. Rule-based Programming of User Agents for Linked Data. LDOW 2018.
D. Guinard, V. Trifa, T. Pham, and O. Liechti. Towards physical mashups in the web of things. INSS 2009.
M. Blackstock and R. Lea. 2012. WoTKit: A lightweight toolkit for the web of Things. WoT 2012.
M Rietzler, J. Greim, M. Walch, F. Schaub, B. Wiedersheim, and M. Weber. HomeBLOX: Introducing process-driven home automation. HomeSys 2013.

Reactive Web (of Things) Systems

Rule-Based Programming

IFTTT
http://ifttt.com

zapier
http://zapier.com

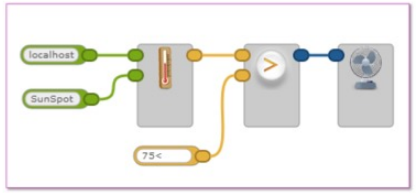


Lecture #1: Ericsson's "**Social Web of Things**"
https://youtu.be/i5AuzQXBsG4

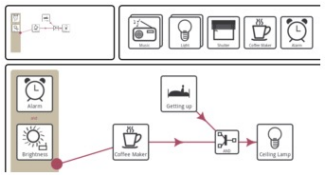
What was the **problem**?

→ Heterogenous systems, complexity

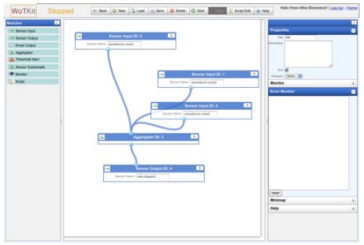
Dataflow programming for the Web of Things



[Guinard et al., 2009]



[Rietzler et al., 2013]

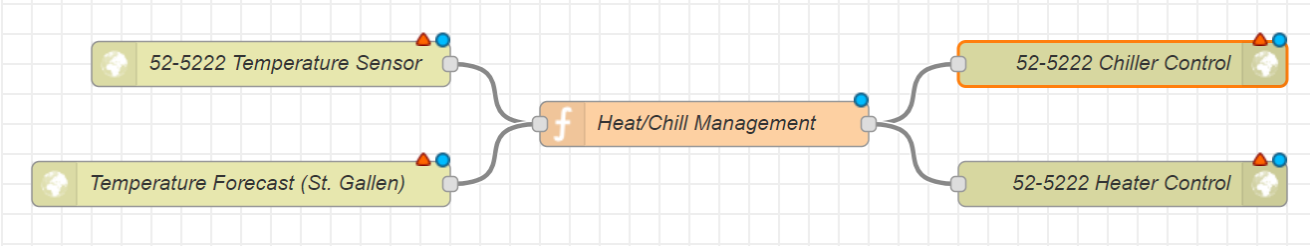


[Blackstock & Lea, 2012]

Node-RED

"Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways.

http://nodered.org

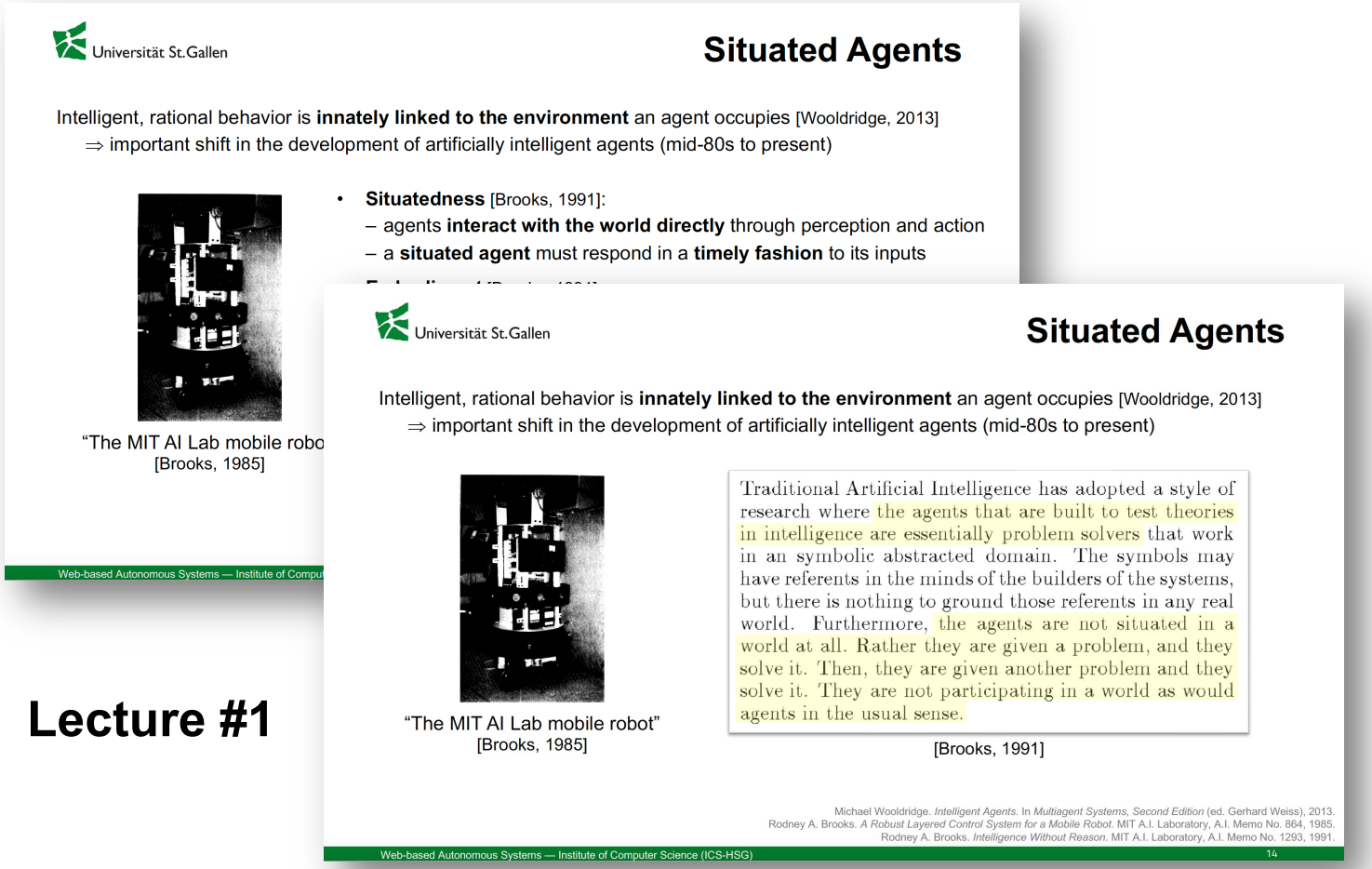


T. Käfer and A. Harth. Rule-based Programming of User Agents for Linked Data. LDOW 2018.
D. Guinard, V. Trifa, T. Pham, and O. Liechti. Towards physical mashups in the web of things. INSS 2009.
M. Blackstock and R. Lea. 2012. WoTKit: A lightweight toolkit for the web of Things. WoT 2012.
M Rietzler, J. Greim, M. Walch, F. Schaub, B. Wiedersheim, and M. Weber. HomeBLOX: Introducing process-driven home automation. HomeSys 2013.

The Subsumption Architecture

> Back in the days robots took more time to plan → env. o/r. changed

> situatedness: Agt. needs to be in env. And needs to react shortly



Lecture #1

Rodney Brooks, "Robots Rising"
Discover Channel, 1998 (from 16:43 to 17:37)
https://youtu.be/7-Na7nkAH1k?t=1003

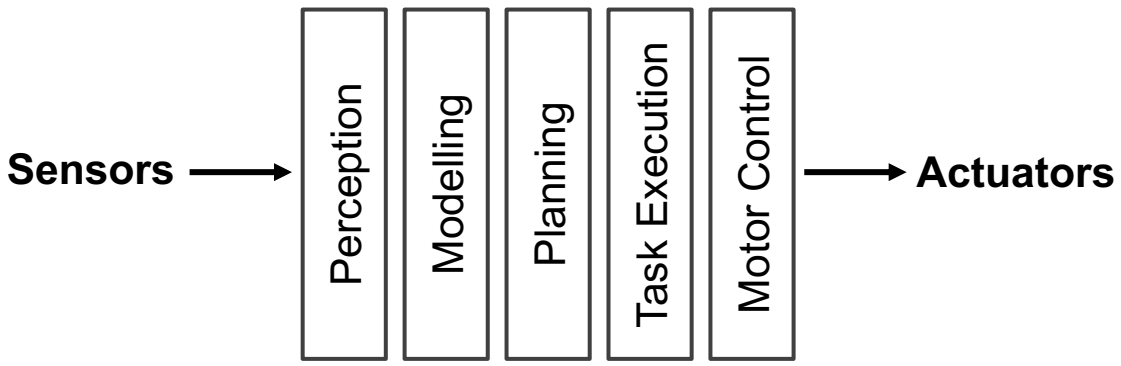
Rodney A. Brooks. *A Robust Layered Control System for a Mobile Robot*. MIT A.I. Laboratory, A.I. Memo No. 864, 1985.
Rodney A. Brooks. *Intelligence Without Reason*. MIT A.I. Laboratory, A.I. Memo No. 1293, 1991.

Universität St.Gallen

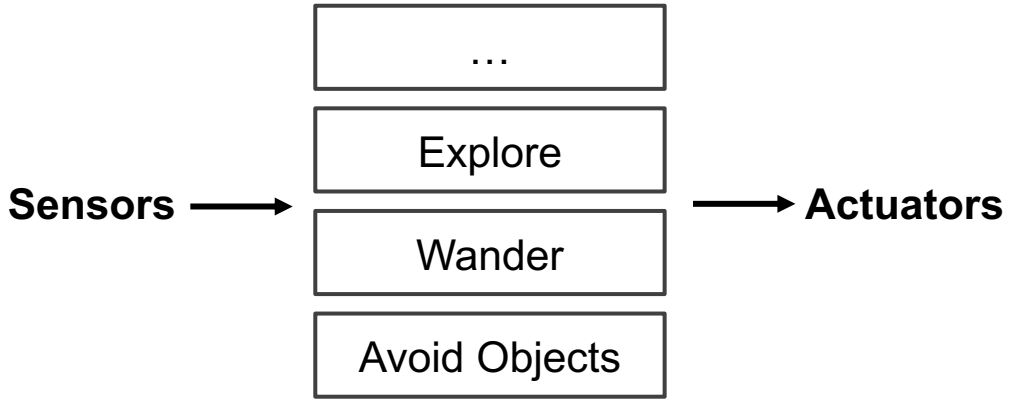
The Subsumption Architecture

Radical new idea: **concurrent behaviors**

- **levels of competence**: each behavioral level runs independently, concurrently, and asynchronously
  - ⇒ each behavioral level continually takes perceptual input and maps it to an action (can fire simultaneously)



Decomposition of a mobile robot control system into **functional modules** (horizontal decomposition)



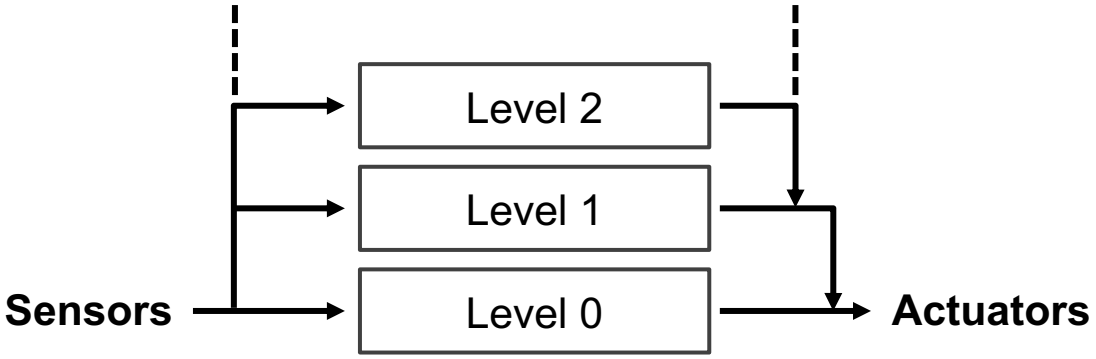
Decomposition of a mobile robot control system into **task achieving behaviors** (vertical decomposition)

Rodney A. Brooks. *A Robust Layered Control System for a Mobile Robot*. MIT A.I. Laboratory, A.I. Memo No. 864, 1985.

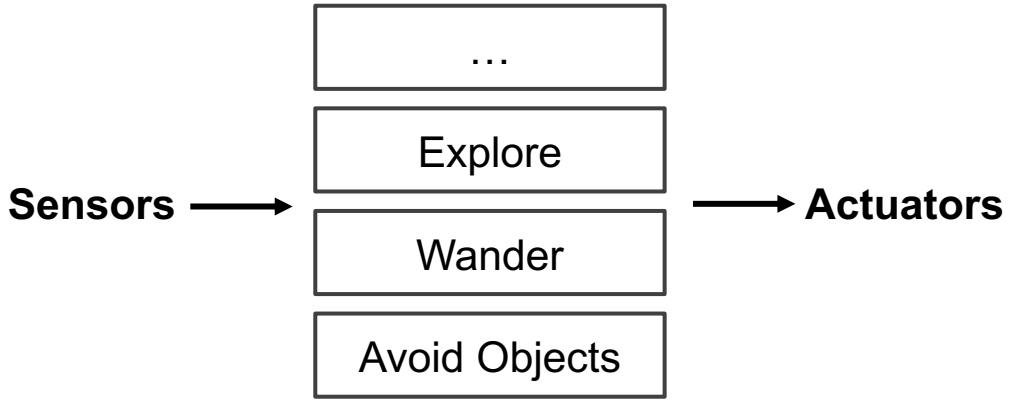
The Subsumption Architecture

Radical new idea: **concurrent behaviors**

- **levels of competence**: each behavioral level runs independently, concurrently, and asynchronously
  - ⇒ each behavioral level continually takes perceptual input and maps it to an action (can fire simultaneously)
- **layers of control**: higher behavioral levels subsume the roles of lower levels in order to take control
  - the system can be partitioned at any level and the lower layers form a complete operational control system



Control is layered: higher behavioral levels can **suppress** the input and **inhibit** the output of lower levels



Decomposition of a mobile robot control system into **task achieving behaviors** (vertical decomposition)

Rodney A. Brooks. *A Robust Layered Control System for a Mobile Robot*. MIT A.I. Laboratory, A.I. Memo No. 864, 1985.

The Subsumption Architecture

(founded in 1990 by Rodney Brooks and two MIT graduate students)

A Can-Collecting Robot [Connell, 1989]:
https://youtu.be/YtNKuwiVYm0

- No built-in model of the environment
- No symbolic representation of knowledge
- Behavioral modules as finite state machines

Remember **situation calculus**? (Lecture #2)

Proposed in **strong reaction** against the "traditional" logic-based agents

Highly influential in robotics, but also for the engineering of software agents

Remember **situatedness, embodiement**? (Lecture #1)

Jonathan Connell. *A Colony Architecture for an Artificial Creature*. MIT A.I. Laboratory, Technical Report No. 1151, 1989.

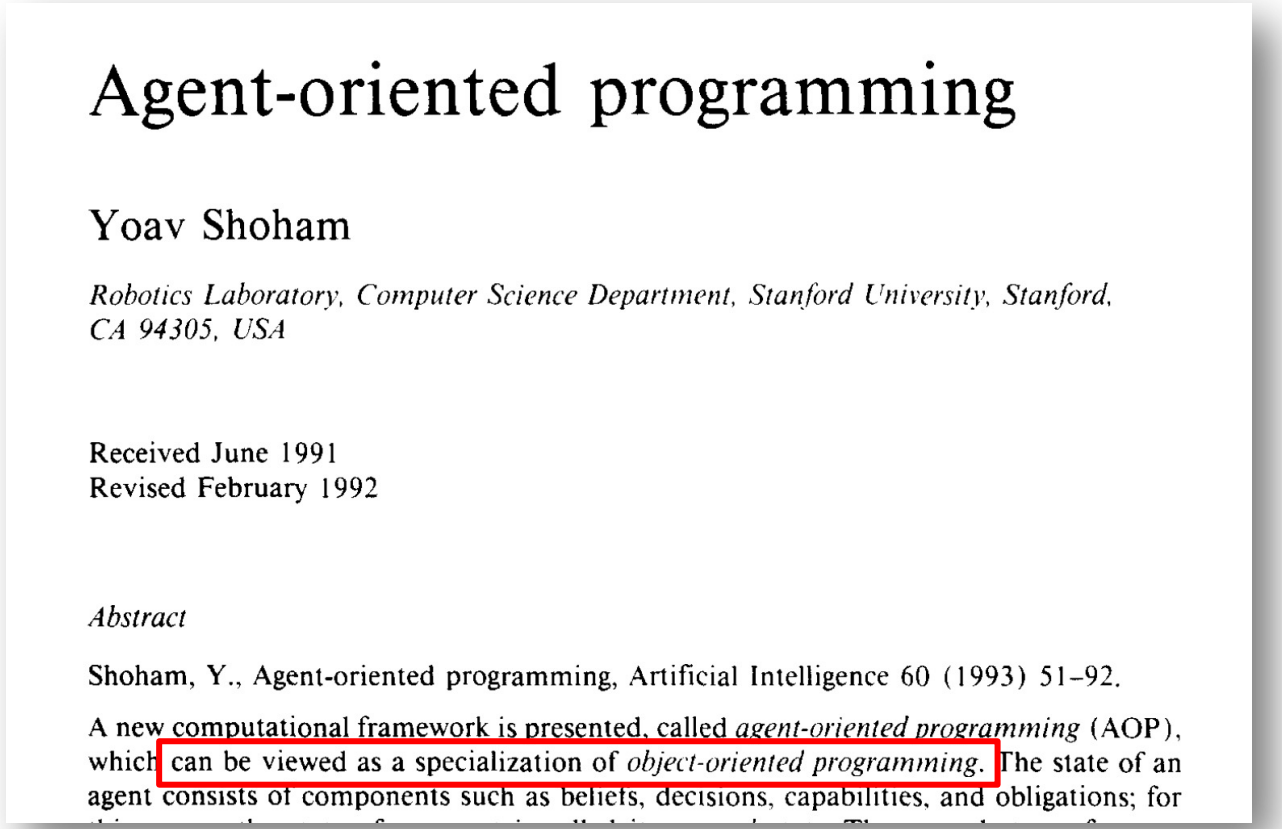
Universität St.Gallen

Today's Agenda

- Introduction
- Reactive Agents
  - The Subsumption Architecture
- Deliberative Agents
  - Agent-Oriented Programming
  - The Belief-Desire-Intention (BDI) Architecture
  - Programming BDI Agents in Jason

Universität St.Gallen

Agent-Oriented Programming (AOP)



Agent-oriented programming

Yoav Shoham

Robotics Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305, USA

Received June 1991
Revised February 1992

Abstract

Shoham, Y., Agent-oriented programming, Artificial Intelligence 60 (1993) 51–92.

A new computational framework is presented, called *agent-oriented programming* (AOP), which can be viewed as a specialization of *object-oriented programming*. The state of an agent consists of components such as beliefs, decisions, capabilities, and obligations; for

Programming Paradigms & Metaphors

imperative	=>	machines
functional	=>	mathematics
object-oriented	=>	“world of objects”
(multi-)agent-oriented	=>	“world of humans”

Agents as **first-class design** and **programming abstractions**

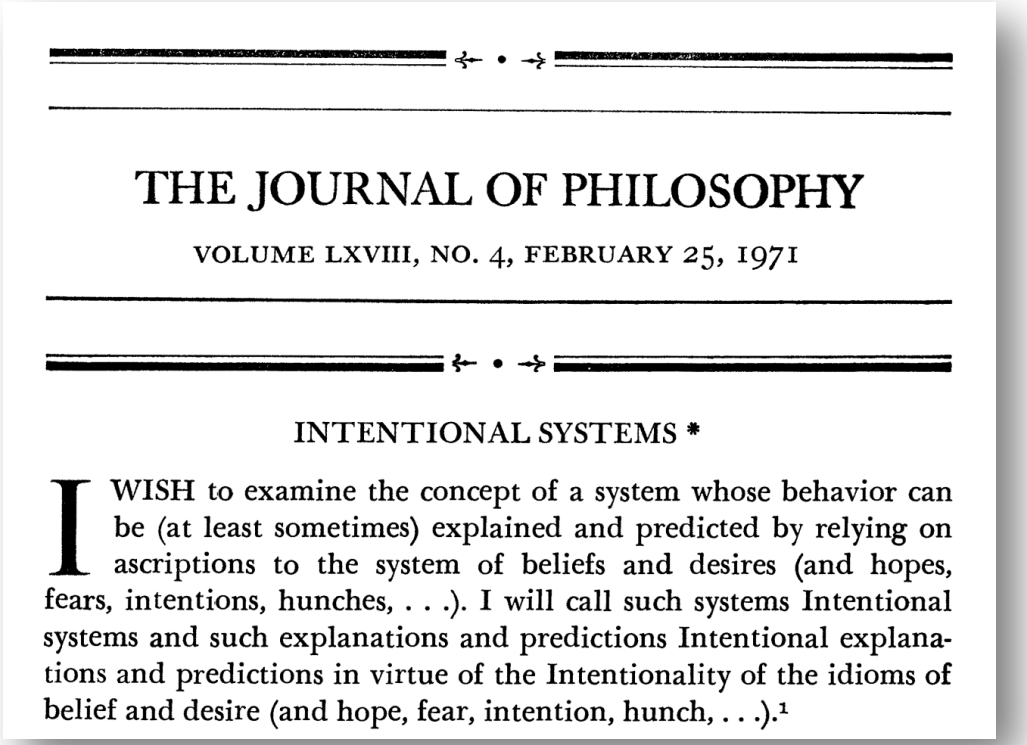
Assumption: **ascribing mental qualities** to artificial agents simplifies development

Alessandro Ricci, *Levels of Abstraction in Designing and Programming Systems of Cognitive Agents*, HyperAgents 2019: http://www2019.hyperagents.org/
Yoav Shoham, *Agent-oriented programming*, Artificial Intelligence, Volume 60, Issue 1, 1993.

Ascribing Mental Qualities to Artificial Agents

Theoretical underpinning:

- Philosophy: the **intentional stance** [Dennett, 1971]



THE JOURNAL OF PHILOSOPHY

VOLUME LXVIII, NO. 4, FEBRUARY 25, 1971

INTENTIONAL SYSTEMS *

I WISH to examine the concept of a system whose behavior can be (at least sometimes) explained and predicted by relying on ascriptions to the system of beliefs and desires (and hopes, fears, intentions, hunches, . . .). I will call such systems Intentional systems and such explanations and predictions Intentional explanations and predictions in virtue of the Intentionality of the idioms of belief and desire (and hope, fear, intention, hunch, . . .).[1]

Playing against a **chess program:** do you treat it differently from a human player when trying to **explain** or **predict** its moves?

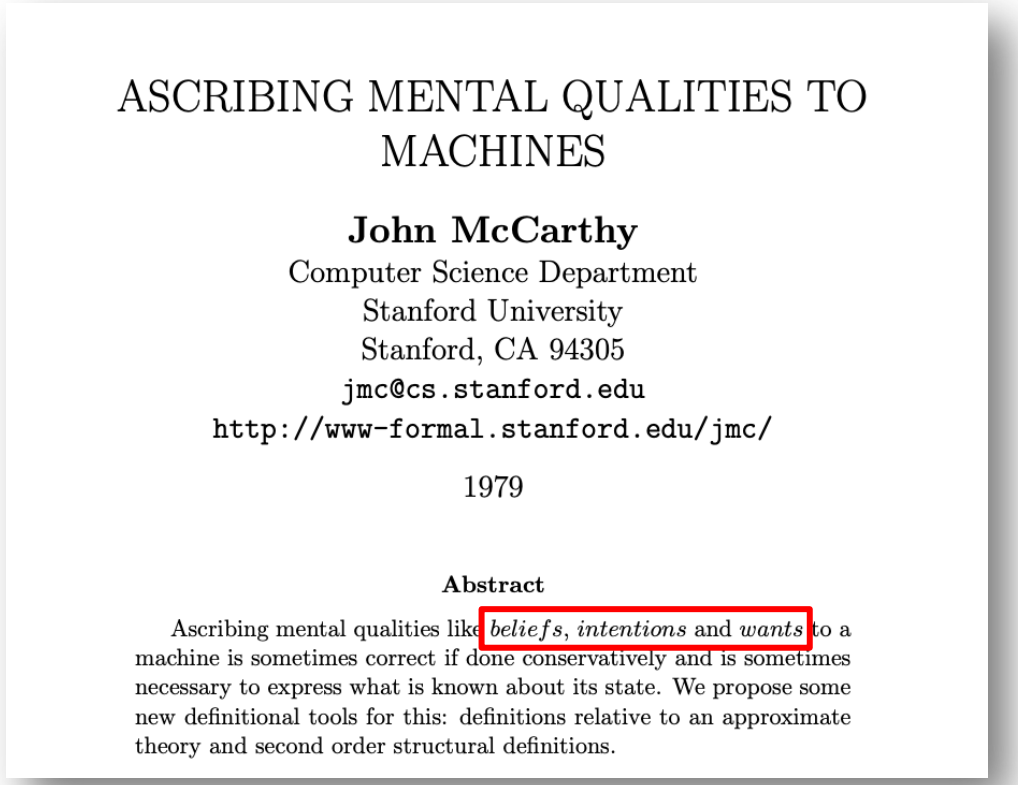
Focus is on the **usefulness** of the intentional stance (not if the system has mental qualities legitimately)

Daniel C. Dennett, Intentional Systems, The Journal of Philosophy, Volume 68, Issue 4, 1971.

Ascribing Mental Qualities to Artificial Agents

Theoretical underpinning:

- Philosophy: the **intentional stance** [Dennett, 1971]
- Artificial Intelligence: **ascribing mental qualities** to machines [McCarthy, 1979]



ASCRIBING MENTAL QUALITIES TO MACHINES

John McCarthy
Computer Science Department
Stanford University
Stanford, CA 94305
jmc@cs.stanford.edu
http://www-formal.stanford.edu/jmc/

1979

Abstract

Ascribing mental qualities like *beliefs*, *intentions* and *wants* to a machine is sometimes correct if done conservatively and is sometimes necessary to express what is known about its state. We propose some new definitional tools for this: definitions relative to an approximate theory and second order structural definitions.

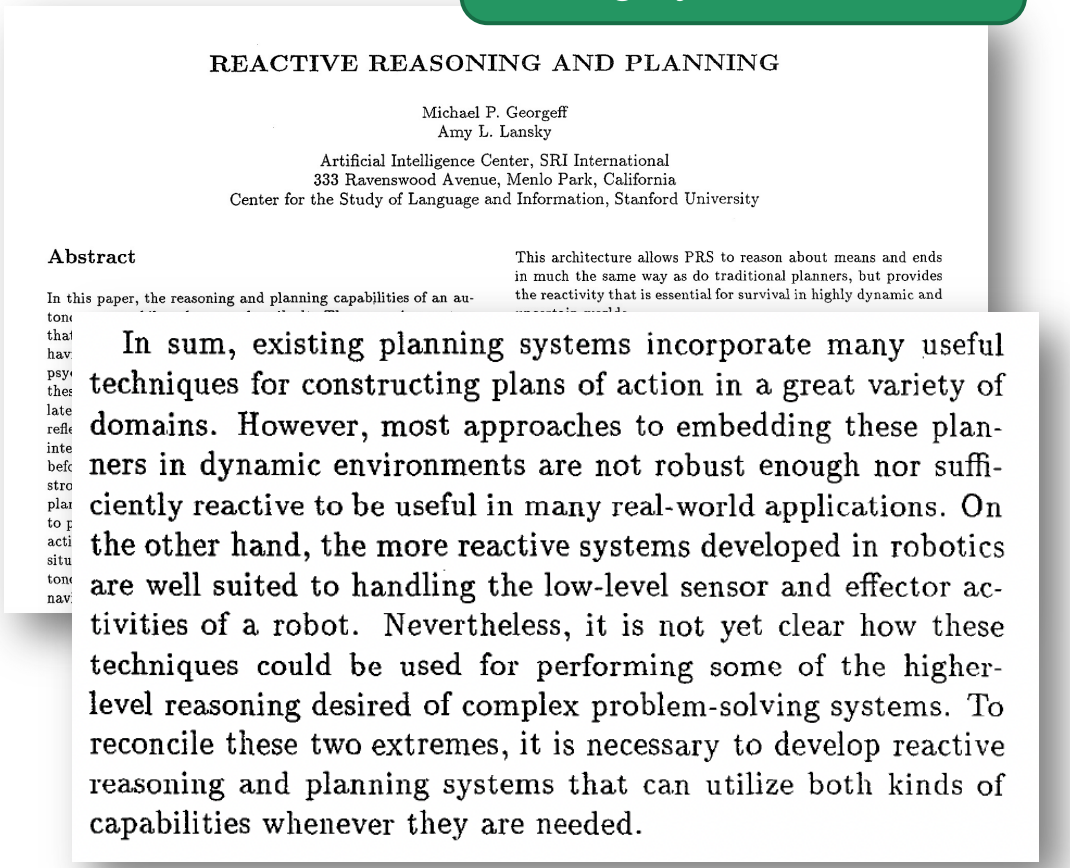
Focus is again on the usefulness of ascribing such mental qualities to machines, not on the legitimacy of the ascription:

> "It is useful when the ascription helps us understand the structure of the machine, its past or future behavior, or how to repair or improve it."

Daniel C. Dennett, Intentional Systems, The Journal of Philosophy, Volume 68, Issue 4, 1971.
John McCarthy, Ascribing Mental Qualities to Machines, Technical Report ADA071423, Stanford, 1979.

Belief-Desire-Intention (BDI) Agents

Radical new approach, highly influential



REACTIVE REASONING AND PLANNING

Michael P. Georgeff
Amy L. Lansky

Artificial Intelligence Center, SRI International
333 Ravenswood Avenue, Menlo Park, California
Center for the Study of Language and Information, Stanford University

Abstract

In this paper, the reasoning and planning capabilities of an au-

This architecture allows PRS to reason about means and ends in much the same way as do traditional planners, but provides the reactivity that is essential for survival in highly dynamic and

In sum, existing planning systems incorporate many useful techniques for constructing plans of action in a great variety of domains. However, most approaches to embedding these planners in dynamic environments are not robust enough nor sufficiently reactive to be useful in many real-world applications. On the other hand, the more reactive systems developed in robotics are well suited to handling the low-level sensor and effector activities of a robot. Nevertheless, it is not yet clear how these techniques could be used for performing some of the higher-level reasoning desired of complex problem-solving systems. To reconcile these two extremes, it is necessary to develop reactive reasoning and planning systems that can utilize both kinds of capabilities whenever they are needed.

Summary of the related work section
(the state-of-the-art back in 1987)

Rooted in the BDI model of **human practical reasoning** [Bratman, 1987]

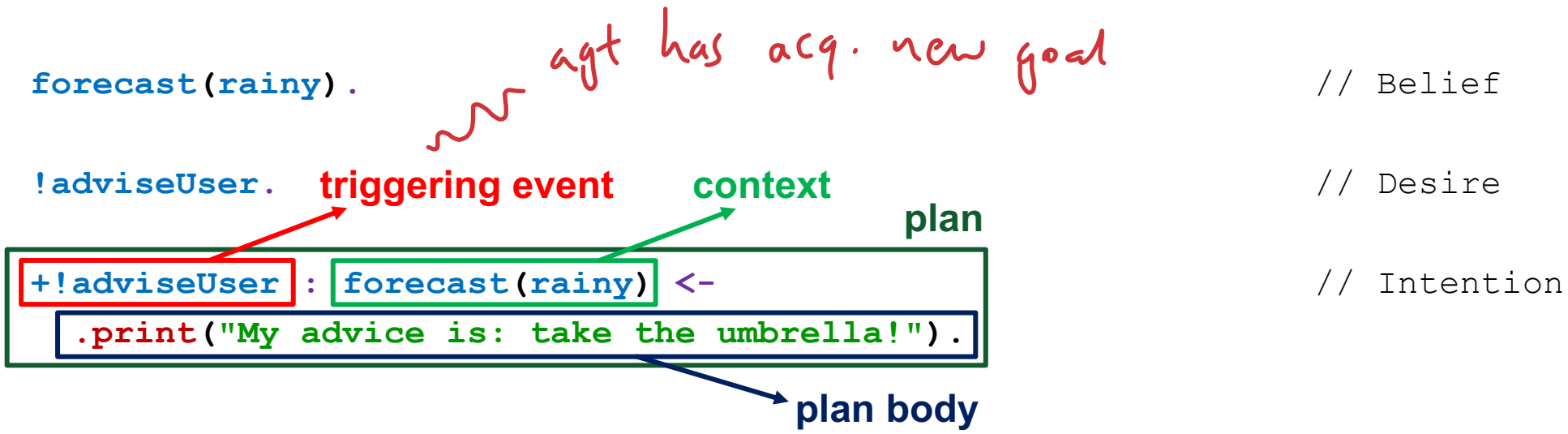
Response to the **diverging views** at the time: logic-based agents vs. reactive agents *à la* subsumption architecture

Novelty: an agent architecture that can balance proactive and reactive behavior (the **Procedural Reasoning System**)

- the internal state of agents is represented in terms of **beliefs**, **desires**, and **intentions** (the agents' **mental states**)
- developers can program agents to **deliberate** about their own beliefs, desires, and intentions — and to modify them as needed:
  - an agent can **decide** to suspend the execution of an intention in order to react to an important event;
  - an agent can **drop an intention** if it becomes unachievable;
  - etc.

M. Georgeff and A. Lansky. *Reactive Reasoning and Planning*. AAAI, 1987.
Michael Bratman, *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

Example (in Jason): Weatherman Bob



- **Beliefs**: information the agent holds about the world; beliefs can be out of date or inaccurate
- **Desires**: states of affairs the agent wishes to bring to the world (broadly, the agent's **goals**)
- **Intentions**: the states of affairs the agent has decided to work towards (goals that the agent is **committed to achieve**)

Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

Universität St.Gallen

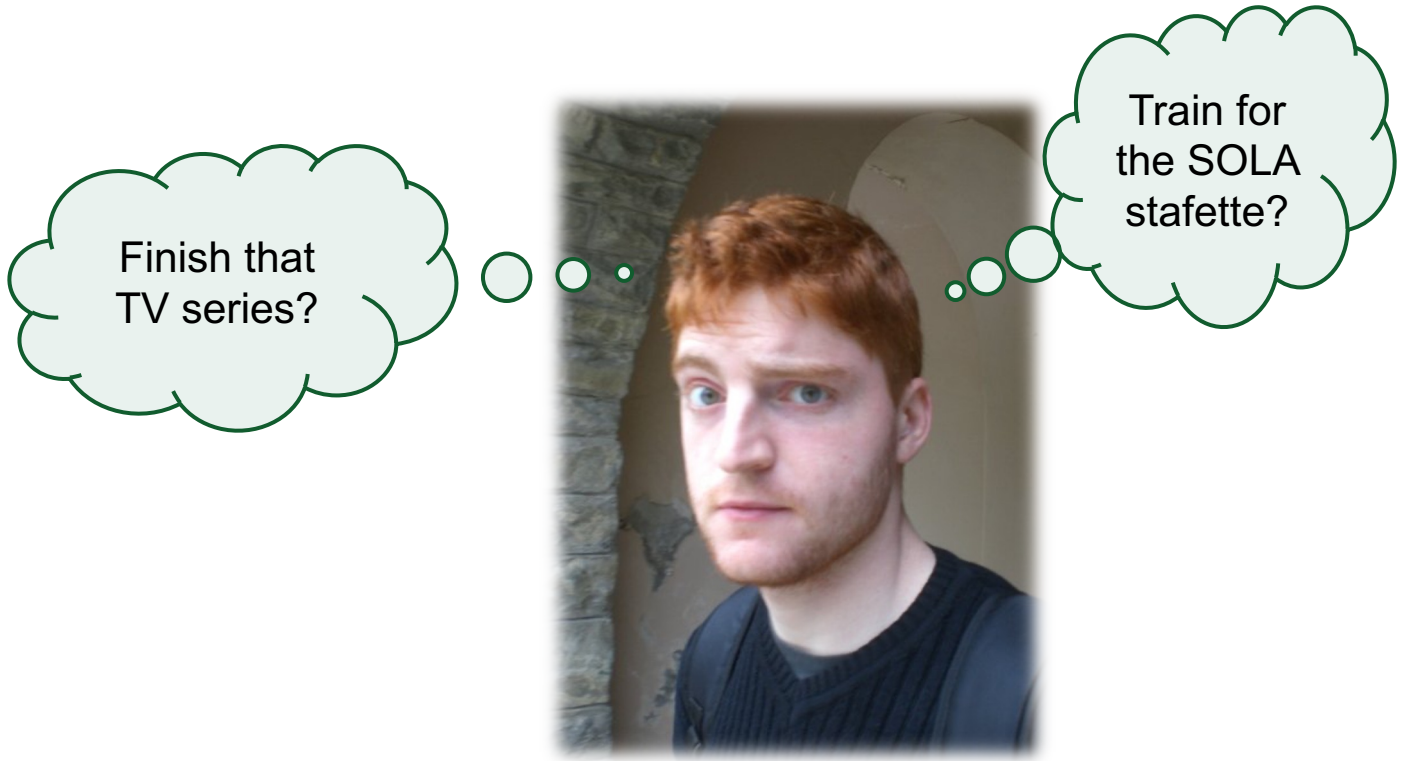
Today's Agenda

- Introduction
- Reactive Agents
  - The Subsumption Architecture
- Deliberative Agents
  - Agent-Oriented Programming
  - The Belief-Desire-Intention (BDI) Architecture
  - Programming BDI Agents in Jason

The BDI Architecture

Human practical reasoning involves two main processes [Bratman, 1987]:

- **deliberation**: deciding *what* goals to achieve
- **means-end reasoning**: deciding *how* to achieve those goals using the available means

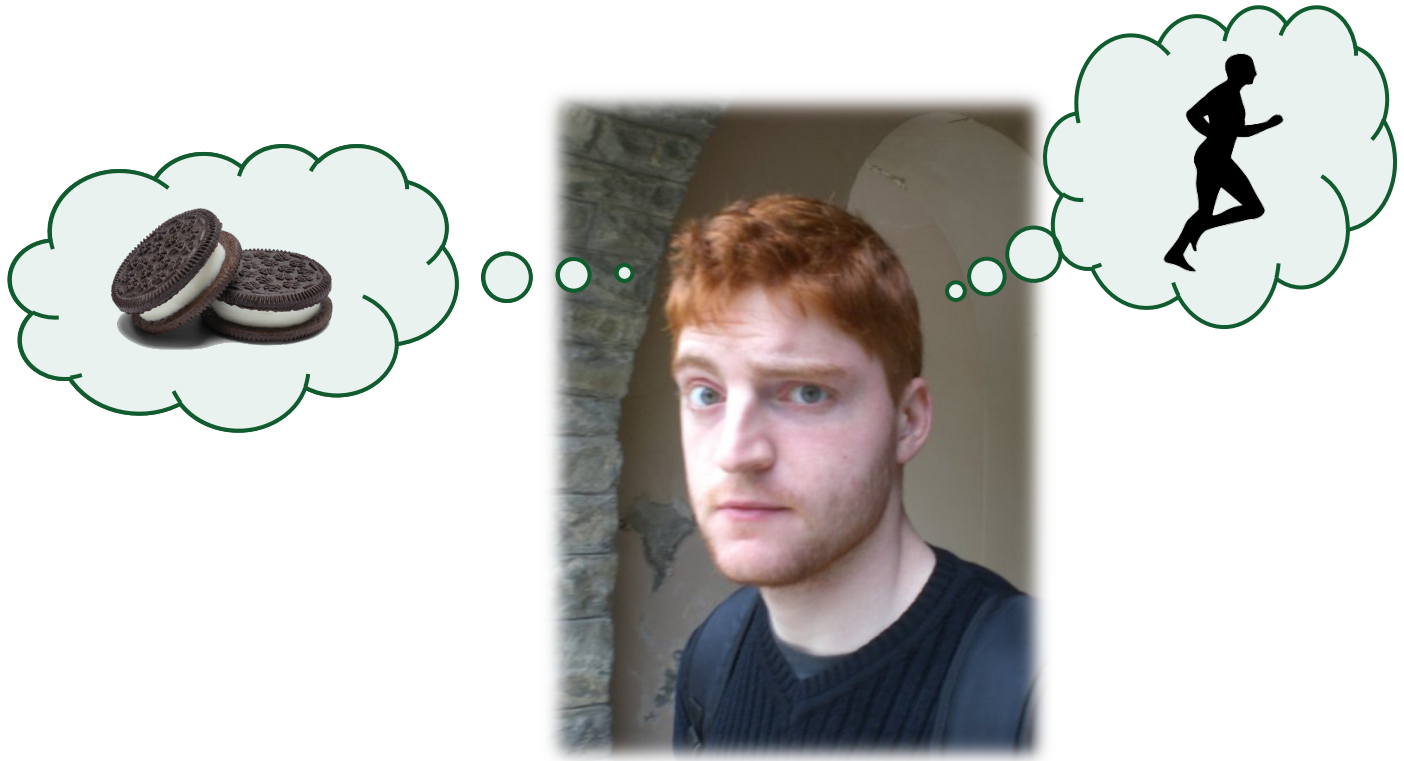


Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Michael Bratman, *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

The BDI Architecture

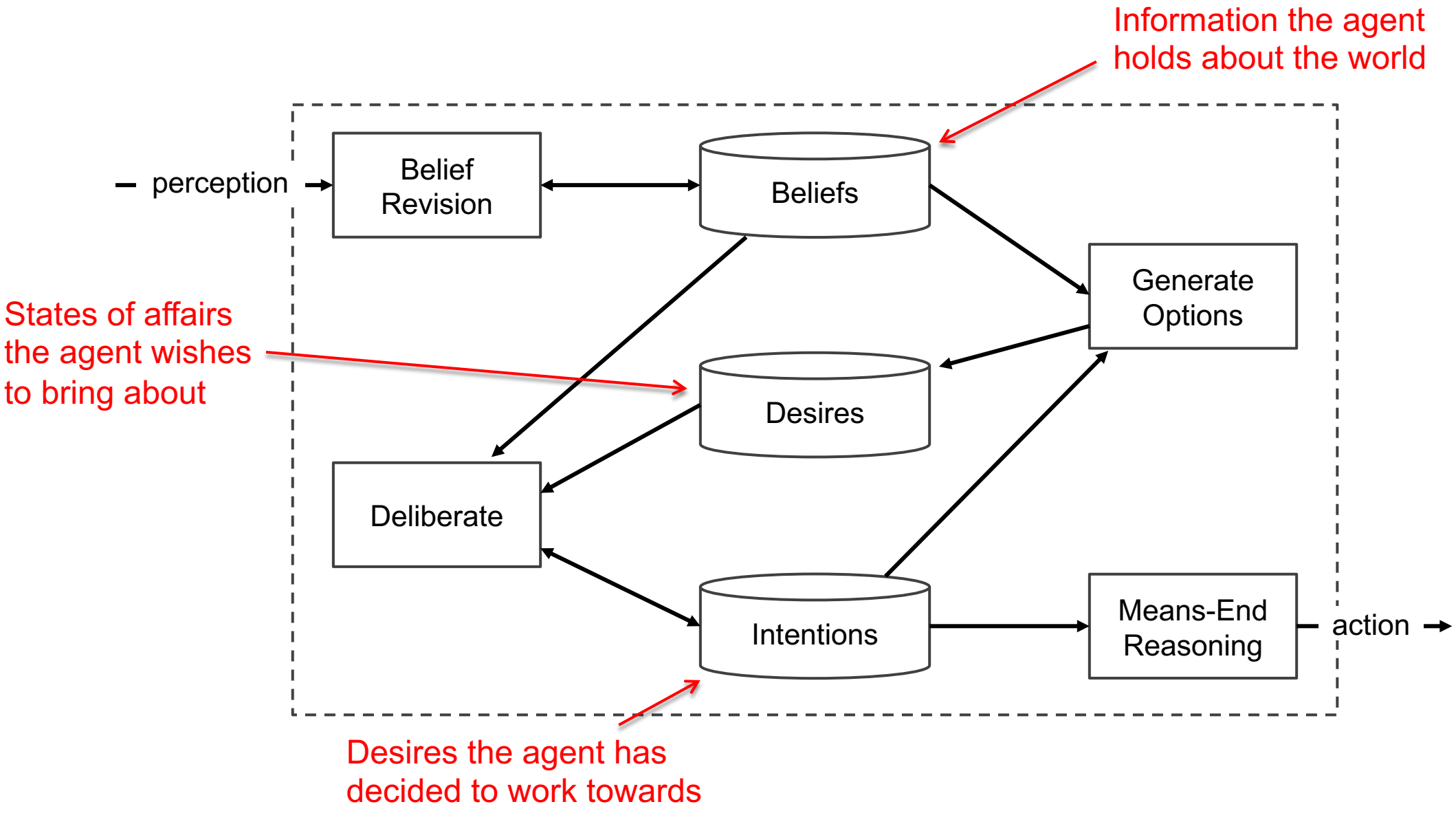
Human practical reasoning involves two main processes [Bratman, 1987]:

- **deliberation**: deciding *what* goals to achieve
- **means-end reasoning**: deciding *how* to achieve those goals using the available means



Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Michael Bratman, *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

The BDI Architecture: The Mentalistic View



Universität St.Gallen

The BDI Architecture: Reasoning Cycle

```
1 while true do
2   B ← brf(B,perception()) ;          // belief revision
3   D ← options(B,I) ;                 // desire revision
4   I ← deliberate(B,D,I) ;            // deliberation
5   π ← meansend(B,I,A) ;              // means-end reasoning
6   while π != Ø do                    // (A is the set of
7     execute( head(π) )               // available actions)
8     π ← tail(π)
```

"Back to the Future", 1985
(from 02:05 to 02:55)

https://youtu.be/3isQI0nXQRE?t=125

What's wrong here?

The BDI Architecture: Reasoning Cycle

```
1 while true do
2   B ← brf(B,perception()) ;          // belief revision
3   D ← options(B,I) ;                 // desire revision
4   I ← deliberate(B,D,I) ;            // deliberation
5   π ← meansend(B,I,A) ;              // means-end reasoning
6   while π != Ø do                    // (A is the set of
7     execute( head(π) )               // available actions)
8     π ← tail(π)
```

Works for proactivity, but not for reactivity (**blind commitment** to the selected plan)

The BDI Architecture: Reasoning Cycle

```
1 while true do
2   B ← brf(B,perception()) ;          // belief revision
3   D ← options(B,I) ;                 // desire revision
4   I ← deliberate(B,D,I) ;            // deliberation
5   π ← meansend(B,I,A) ;              // means-end reasoning
6   while π != ∅ do                    // (A is the set of
7     execute( head(π) )               // available actions)
8     π ← tail(π)
9     B ← brf (B,perception())
10    if ¬sound(π,I,B) then
11      π ← meansend(B,I,A)
```

Revises commitment to the selected plan (re-planning for context adaption)

… but **blind commitment** to the selected intention
(never reconsiders the selected intention)

Universität St.Gallen

The BDI Architecture: Reasoning Cycle

```
1 while true do
2   B ← brf(B,perception()) ;                                  // belief revision
3   D ← options(B,I) ;                                         // desire revision
4   I ← deliberate(B,D,I) ;                                    // deliberation
5   π ← meansend(B,I,A) ;                                      // means-end reasoning
6   while π != ∅ and ¬succeeded(I,B) and ¬impossible(I,B) do   // (A is the set of
7     execute( head(π) )                                       // available actions)
8     π ← tail(π)
9     B ← brf (B,perception())
10    if ¬sound(π,I,B) then
11      π ← meansend(B,I,A)
```

Checks if the selected intention was already achieved or became impossible

… but **single-minded commitment** to the selected intention
(will not drop the selected intention while it is achievable)

Universität St.Gallen

The BDI Architecture: Reasoning Cycle

```
1 while true do
2   B ← brf(B,perception()) ;                                  // belief revision
3   D ← options(B,I) ;                                         // desire revision
4   I ← deliberate(B,D,I) ;                                    // deliberation
5   π ← meansend(B,I,A) ;                                      // means-end reasoning
6   while π != ∅ and ¬succeeded(I,B) and ¬impossible(I,B) do   // (A is the set of
7     execute( head(π) )                                       // available actions)
8     π ← tail(π)
9     B ← brf (B,perception())
10    if reconsider(I,B) then
11      D ← options(B,I)
12      I ← deliberate(B,D,I)
13    if ¬sound(π,I,B) then
14      π ← meansend(B,I,A)
```

An **open-minded** BDI agent can reconsider its intentions (not necessarily at each iteration)
(only maintains its intentions if they are still its desires)

The BDI Architecture: Reasoning Cycle

What's the **trade-off**?

efficiency ← → adaptivity

blind commitment
(to selected plan/intention)

e.g. clean room although already clean

vs.

single-minded commitment
(to selected intention)

e.g. clean room only if not clean

vs.

open-minded commitment
(to selected intention)

e.g. go to coop first, bc. closes in 5m.

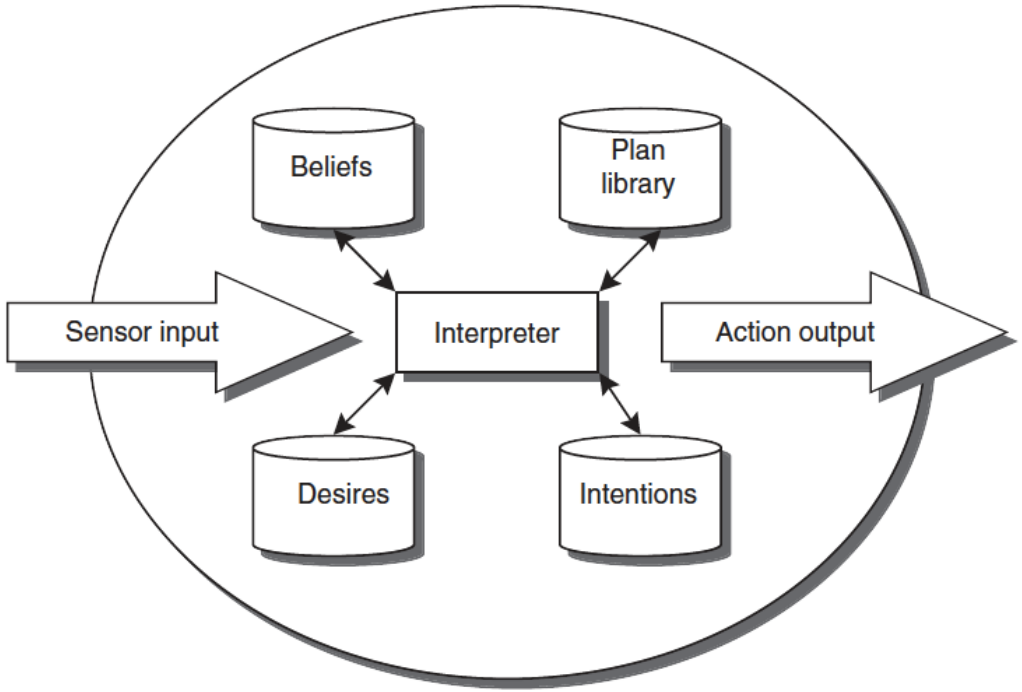
adaptivity vs. **efficiency**

A **central feature** of the BDI Architecture is that it balances **proactive** and **reactive behavior**

The BDI Architecture: Means-End Reasoning

Means-end reasoning: the process of deciding how to achieve an end (i.e., an intention) using the available means (i.e., the actions that can be performed in the environment)

- **automated planning**: yields proactive behavior, but low reactiveness
- **Procedural Reasoning System (PRS)** [Georgeff & Lansky, 1987]: balances proactiveness and reactiveness



Radical new idea: **procedural reasoning**

- no automated planning
- agents are equipped with libraries of plans (**procedural knowledge**)
- agents assemble plans at run time (**procedural reasoning**)

To program such agents, we need a **practical programming language**

M. Georgeff and A. Lansky. *Reactive Reasoning and Planning*. AAAI, 1987.
Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

Universität St.Gallen

Today's Agenda

- Introduction
- Reactive Agents
  - The Subsumption Architecture
- Deliberative Agents
  - Agent-Oriented Programming
  - The Belief-Desire-Intention (BDI) Architecture
  - Programming BDI Agents in Jason

Programming BDI Agents in Jason

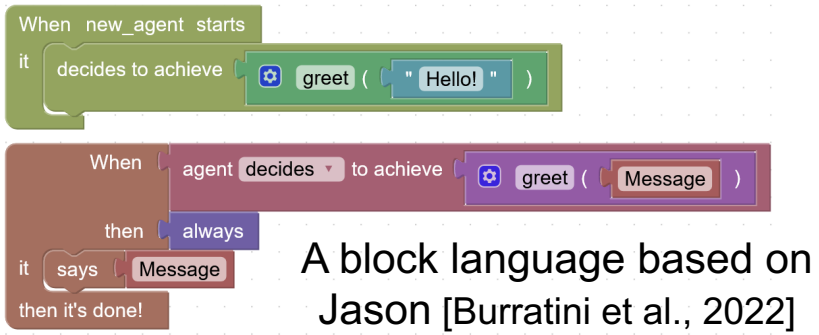
Jason is an interpreter for an extended version of AgentSpeak(L)

- AgentSpeak(L) [Rao, 1996] is a programming language for BDI agents inspired by the PRS and grounded in first-order logic
- Jason brings several practical extensions such as communication

Jason is part of the JaCaMo platform [Boissier et al., 2020]

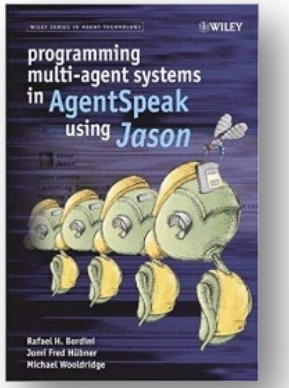
- Jason commonly used to refer to the programming language it implements

Main language constructs: beliefs, goals, and plans

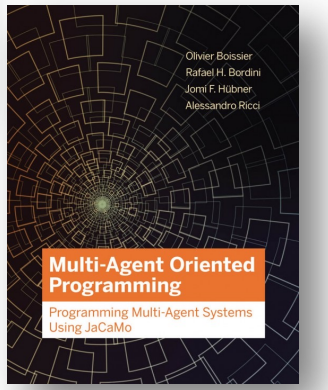


A block language based on Jason [Burratini et al., 2022]

Abstractions so intuitive they could work for non-technical users



Chapters 1-3



Chapters 3-4

Anand S. Rao, *AgentSpeak(L): BDI agents speak out in a logical computable language*, MAAMAW 1996.
Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
O. Boissier, R. H. Bordini, J.F. Hubner, A. Ricci. *Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo*, The MIT Press, 2020.
Samuele Burratini et al., *Agent-Oriented Visual Programming for the Web of Things*, EMAS 2022 (in submission).

Universität St.Gallen

Beliefs: Syntax

Beliefs are represented by annotated predicates of first-order logic:

```
predicate(term1,…,term2)[annot1,…,annot2]
```

Beliefs can be inaccurate, outdated, contradictory, may have different degrees of certainty, etc.

Example: Weatherman bob's **belief base**

```
forecast(rainy)[source(percept),certainty(0.6)].      // beliefs about the environment
forecast(sunny)[source(jane),certainty(0.8)].

knows(bob, jane)[source(self)].                       // beliefs about the agent itself
knows(bob, joe)[source(self)].

sincere(jane)[source(self)].                          // beliefs about other agents
~sincere(joe)[source(self),source(jane)].
```

Strong negation is used to represent what is believed to be false (necessary under the open-world assumption)

Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Jomi F. Hübner, Multi-Agent Oriented Programming Using JaCaMo, AI4Industry Summer School, MINES Saint-Étienne, 2020.

Beliefs: Dynamics

A Jason agent starts with an initial set of beliefs, i.e. its **initial belief base** (can be empty)

- the agent can acquire new beliefs throughout its lifetime

Through perception

An agent can perceive its environment to acquire new beliefs (annotated with `source(percept)`)

- beliefs based on perception are updated automatically during reasoning cycles by the Jason interpreter

```
forecast(rainy)[source(percept),certainty(0.6)].
```

Through intention

An agent can create/delete beliefs (**mental notes**) as part of a plan using the **+** and **–** operators (annotated with `source(self)`)

```
+sincere(jane);        // adds sincere(jane)[source(self)] to the belief base
-sincere(jane);        // removes sincere(jane)[source(self)] from the belief base
```

Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Jomi F. Hübner, Multi-Agent Oriented Programming Using JaCaMo, AI4Industry Summer School, MINES Saint-Étienne, 2020.

Universität St.Gallen

Beliefs: Dynamics

A Jason agent starts with an initial set of beliefs, i.e. its **initial belief base** (can be empty)

– the agent can acquire new beliefs throughout its lifetime

More on **agent communication languages** next week!

Through communication

When an agent receives a **tell** message, the content is added to the belief base as a new belief annotated with the sender of the message as the source

```
.send(bob, tell, forecast(rainy)[certainty(0.6)]); // sent by jane
// adds forecast(rainy)[source(jane),certainty(0.6)] in bob's belief base
```

If an agent receives an **untell** message and the content of the message matches a belief in the agent's belief base with the sender as the source, then the belief is removed

```
.send(bob, untell, forecast(rainy)[certainty(0.6)]); // sent by jane
// removes forecast(rainy)[source(jane),certainty(0.6)] from bob's belief base
```

Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Jomi F. Hübner, Multi-Agent Oriented Programming Using JaCaMo, AI4Industry Summer School, MINES Saint-Étienne, 2020.

Universität St.Gallen

Beliefs: Inference Rules

Rules can be used to infer new beliefs:

```
happy(bob) :- today(saturday) | today(sunday).

advice("take the umbrella") :- forecast(rainy)[source(Ag),certainty(Cert)] &
                               sincere(Ag) & Cert >= 0.1 .
```

Useful Operators

- boolean operators: `&` (and), `|` (or), `not` (not), `=` (unification), relational operators (`>`,`>=`,`<`,`<=`), `==` (equal), `\==` (different)
- arithmetic operators: `+` (sum), `-` (subtraction), `*` (multiply), `/` (divide), `div` (division quotient), `mod` (division remainder), `**` (power)

Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Jomi F. Hübner, Multi-Agent Oriented Programming Using JaCaMo, AI4Industry Summer School, MINES Saint-Étienne, 2020.

Goals: Syntax

Types of goals:

- achievement goals: goals **to do**

```
!giveWeatherAdvivce.          // bob's initial goal
```

- test goals: goals **to know**; can be used inside plan bodies to query the belief base

```
?advice(Advice);              // test goal for retrieving an advice from the belief base
```

Same syntax as beliefs, but goals are preceded by
! (achievement goal) or **?** (test goal)

Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Jomi F. Hübner, Multi-Agent Oriented Programming Using JaCaMo, AI4Industry Summer School, MINES Saint-Étienne, 2020.

Goals: Dynamics

A Jason agent starts with an **initial set of goals** (can be empty), and can acquire new goals throughout its lifetime

Through intention

The **!** and **?** operators can be used inside a plan to create a new goal with `source(self)`

```
// creates a new achievement goal !giveWeatherAdvivce[source(self)]
!giveWeatherAdvivce;
// creates a new test goal ?advice(Advice)[source(self)]
?advice(Advice);
```

Both achievement goals and test goals can also be acquired through communication

More on **agent communication languages** next week!

Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Jomi F. Hübner, Multi-Agent Oriented Programming Using JaCaMo, AI4Industry Summer School, MINES Saint-Étienne, 2020.

Universität St.Gallen

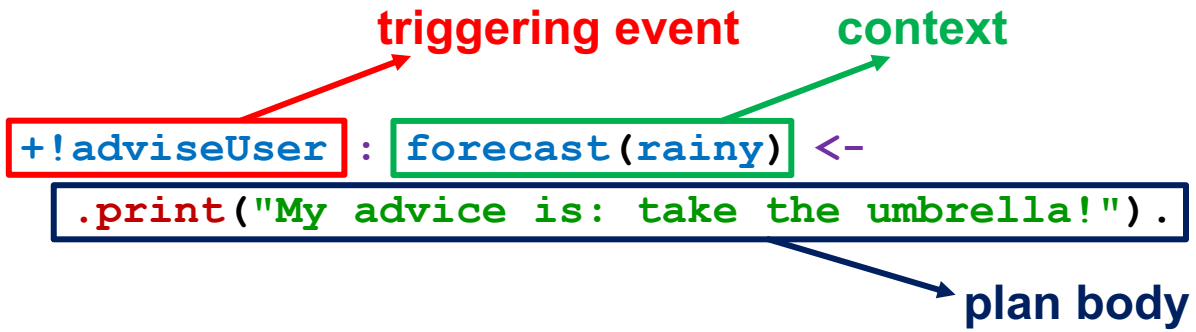
Plans: Syntax

A Jason plan has the following structure:

```
triggering_event : context <- body.
```

where:

- the **triggering event** denotes the event that the plan is meant to handle
- the **context** represents the circumstances in which the plan can be used (**applicability precondition**)
- the plan's **body** is the course of action to be used for handling the event if the context is believed to be true at the time a plan is chosen to handle the event



Triggering Events

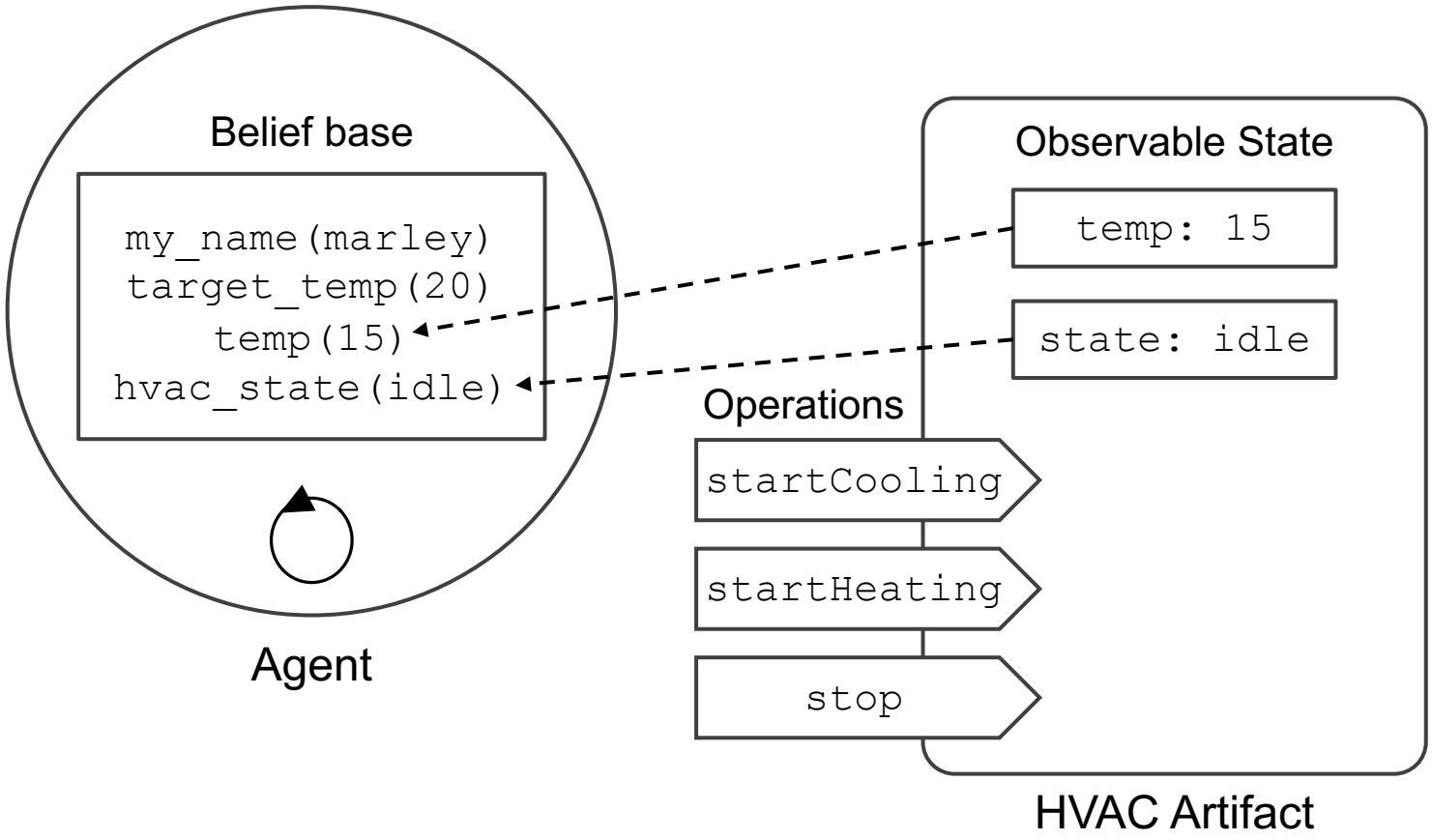
Events are created as consequences to changes in an agent's beliefs or goals:

- `+b` (belief addition)
- `-b` (belief deletion)
- `+!g` (achievement-goal addition)
- `-!g` (achievement-goal deletion)
- `+?g` (test-goal addition)
- `-?g` (test-goal deletion)

An agent reacts to events by executing plans

Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Jomi F. Hübner, Multi-Agent Oriented Programming Using JaCaMo, AI4Industry Summer School, MINES Saint-Étienne, 2020.

Triggering Events: Proactive vs. Reactive Plans



Proactive Plans

```
+!set_temp(Target) : temp(Current)
    & Target > Current
  <- startHeating.

+!set_temp(Target) : temp(Current)
    & Target < Current
  <- startCooling.

+!set_temp(Target) : temp(Current)
    & Target == Current
  <- stop.
```

Reactive Plans

```
// maintenance task
+temp(Current)
  : target_temp(Target)
    & Target \== Current
  <- !set_temp(Target).
```

Proactive plans: pursue (achievement) goals

Reactive plans: react to environmental changes

Alessandro Ricci, *Levels of Abstraction in Designing and Programming Systems of Cognitive Agents*, HyperAgents 2019: http://www2019.hyperagents.org/

Plans: Syntax

Useful operators for the plan's context (same as for inference rules):

- boolean operators: `&` (and), `|` (or), `not` (not), `=` (unification), relational operators (`>`,`>=`,`<`,`<=`), `==` (equal), `\==` (different)
- arithmetic operators: `+` (sum), `-` (subtraction), `*` (multiply), `/` (divide), `div` (division quotient), `mod` (division remainder), `**` (power)

Useful operators for the plan's body:

```
+rain : time_to_leave(T) & time_now(H) & H <= T <-
   !new_goal;                                          // new sub-goal
   ?distance_to_destination(X);                        // new test goal
   X > 10;                                             // constraint to carry on
   X = 100;                                            // assignment
   +note(T - H);                                       // add mental note
   -note(T - H);                                       // remove mental note
   -+note(T * H);                                      // update mental note
   .print("Hello world! Time to leave is: ", T);       // internal action (a print statement)
   open(door).                                         // external action
```

Reference to internal actions provided by Jason: http://jason.sourceforge.net/api/jason/stdlib/package-summary.html

Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Jomi F. Hübner, Multi-Agent Oriented Programming Using JaCaMo, AI4Industry Summer School, MINES Saint-Étienne, 2020.

Plans: Dynamics

A Jason agent is equipped with a **library of plans**

The plan library is initialized with the plans defined by the programmer of the agent (in the agent program), but can **evolve** at run time:

- **through intention**: plans can be added or removed dynamically and intentionally by the agent using two internal actions: `.add_plan` and `.remove_plan` (see reference in footnote)
- **through communication**: agents can exchange plans just like they can exchange beliefs or delegate goals

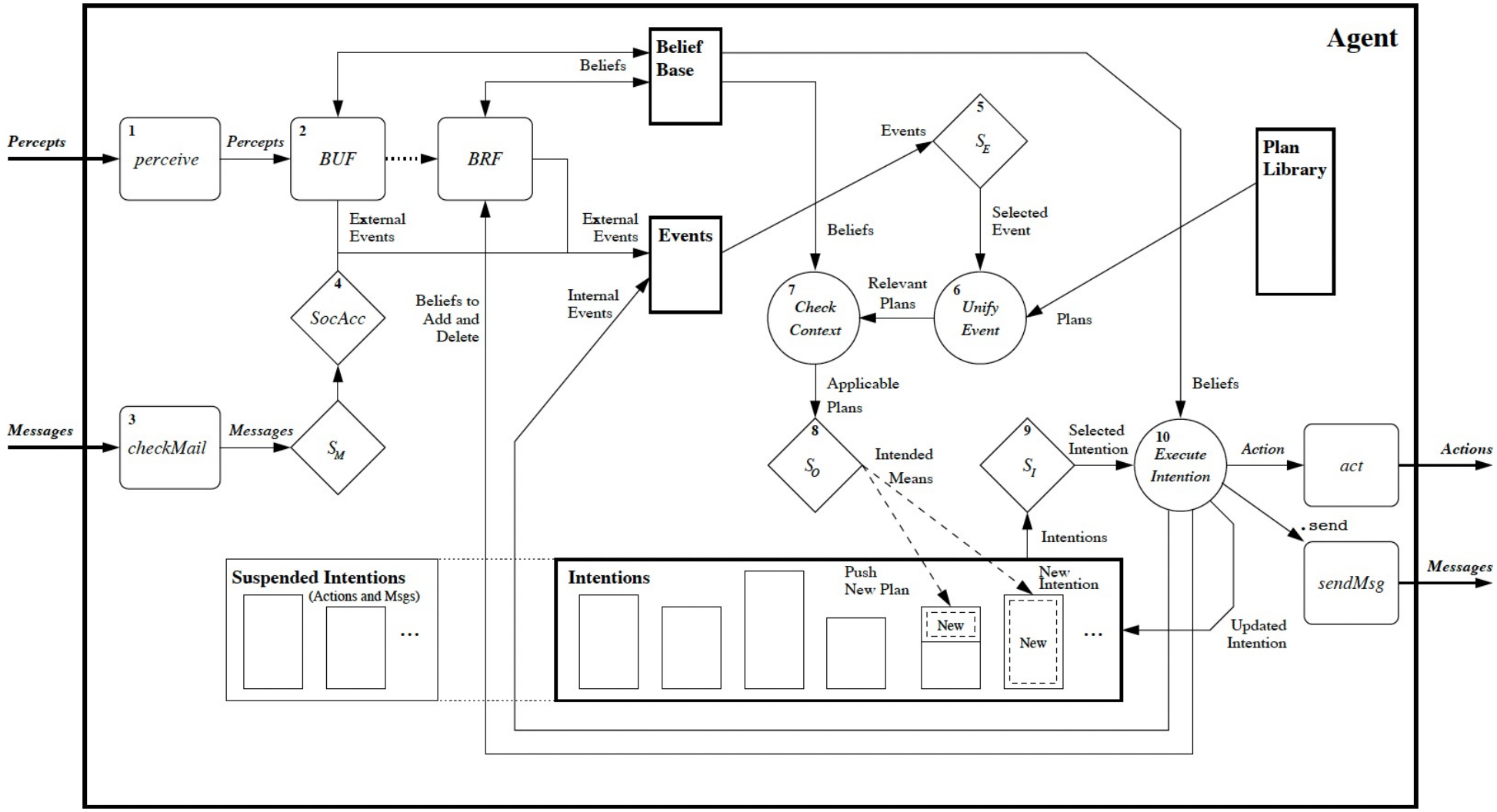
More on **agent communication languages** next week!

Jason agents could also synthesize plans, but this functionality is not trivial — and is not provided out-of-the-box by the JaCaMo platform

- ex: agents could use an external service for PDDL-based planning

Reference to the internal actions supported by Jason: http://jason.sourceforge.net/api/jason/stdlib/package-summary.html
Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Jomi F. Hübner, Multi-Agent Oriented Programming Using JaCaMo, AI4Industry Summer School, MINES Saint-Étienne, 2020.

A Glimpse of the Jason Reasoning Cycle

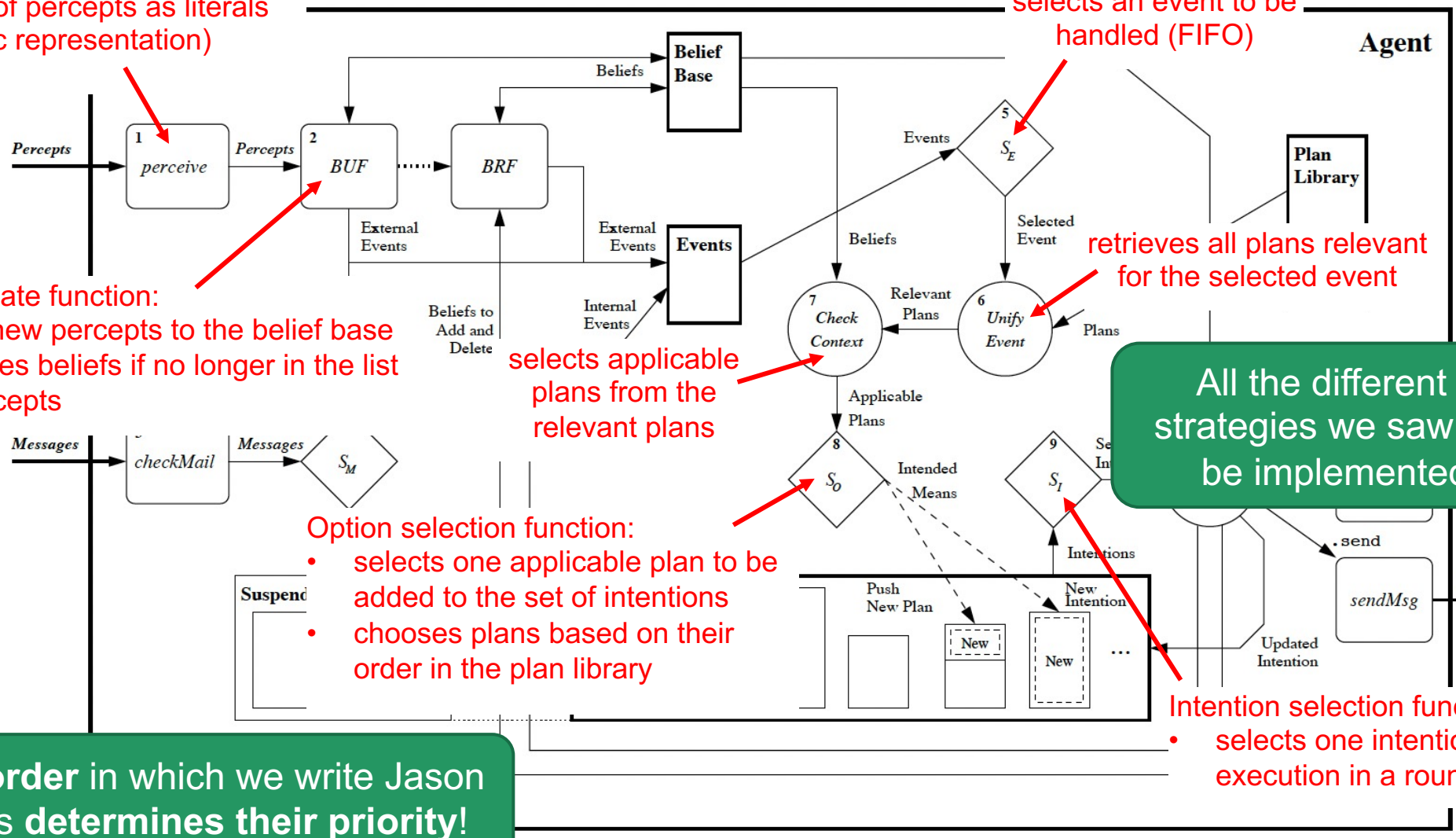


Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Jomi F. Hübner, Multi-Agent Oriented Programming Using JaCaMo, AI4Industry Summer School, MINES Saint-Étienne, 2020.

A Glimpse of the Jason Reasoning Cycle

returns a list of percepts as literals (symbolic representation)

selects an event to be handled (FIFO)



Belief update function:

- adds new percepts to the belief base
- removes beliefs if no longer in the list of percepts

selects applicable plans from the relevant plans

retrieves all plans relevant for the selected event

All the different commitment strategies we saw previously can be implemented with Jason

Option selection function:

- selects one applicable plan to be added to the set of intentions
- chooses plans based on their order in the plan library

Intention selection function:

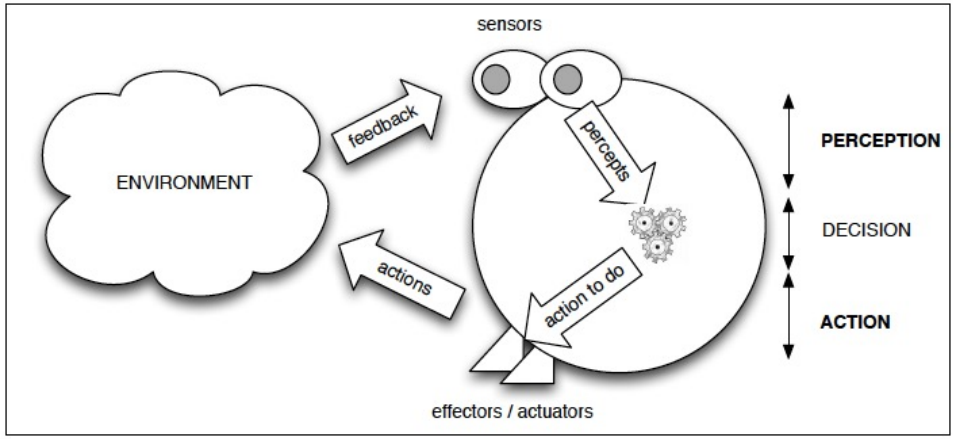
- selects one intention for further execution in a round-robin manner

The **order** in which we write Jason plans **determines their priority**!

Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
Jomi F. Hübner, Multi-Agent Oriented Programming Using JaCaMo, AI4Industry Summer School, MINES Saint-Étienne, 2020.

Concluding Remark: Agents vs. Objects/Actors

Agents as first-class design/programming abstractions:



- task/goal-oriented
- pro-active + reactive
- decision making

Agents vs. Objects

- agents are active, objects are passive (except for active objects)
- agents have stronger encapsulation:
  - state + behavior **+ control of their behavior**
  - decision making

Agents vs. Actors

- actors are reactive, agents are also proactive
- actors operate on an event-loop, agents operate on a reasoning cycle
- actors are message-driven, agents are task/goal-directed

Alessandro Ricci, *Levels of Abstraction in Designing and Programming Systems of Cognitive Agents*, HyperAgents 2019: http://www2019.hyperagents.org/
Rafel Bordini et al., *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

Our Journey

Universität St.Gallen

Prerequisites:
- **ASSE**
- **(…)**

Week 1:
Introduction

Week 2:
A Web for Machines

Week 3:
Knowledge Representation and Reasoning for the Web

Week 4:
Linked Data and Distributed Knowledge Graphs

Week 5:
Autonomous Agents (Arch. and Programming)

Week 6 (Coordination I):
Agent Communication and Interaction

Week 7 (Coordination II):
Self-Organization and Stigmergy

Week 8 (Coordination III):
Normative MAS and Organizations

Week 9 (Coordination IV):
Trust & Reputation

Week 10:
Game Theory and Social Choice

Week 11:
Reinforcement Learning and Multi-Agent Learning

Week 12:
An Industry Perspective

Exercises

Ex1: Writing Your First Agent(s)!
Ex2: Automated Planning
Ex3: Web Ontologies
Ex4: Operating on Linked Data
Ex5: BDI Agents
Ex6: Interacting Agents on the Web
Ex7: Ant Colony Optimization
Ex8: Organized Agent
Ex9: Trustworthy Agents
Ex10: Axelrod's Agents
Ex11: Reinforcement Learning Agents
Course Review and Q&A

Universität St.Gallen

Any Questions / Comments / Doubts / Concerns?

Universität St.Gallen

Images

https://www.istockphoto.com/
https://freepik.com