Università degli Studi di Torino
Computer Science Department
Ph.D. Program in Computer Science
Cycle XXXIII

Exception Handling for Robust Multi-Agent Systems

Stefano Tedeschi

October 15th, 2021

Outline

Introduction

What is this presentation about?

We outline a vision of how **robustness** in Multi-Agent Systems (MAS) can be granted as a design property

We present a model for MAS **organizations** explicitly encompassing the notion of **exception**
→ **Exception handling** is grafted inside the normative system of the organization

The mechanism relies on abstractions that are seamlessly integrated with organizational concepts:

- **Responsibilities**
- **Goals**
- **Norms**

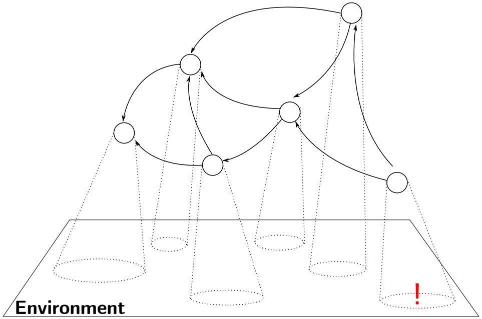
We exemplify our vision on the **JaCaMo**[1] multi-agent platform

[1] Olivier Boissier et al. *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. MIT Press, 2020.

Motivation

In a MAS, the agent detecting a **perturbation**:

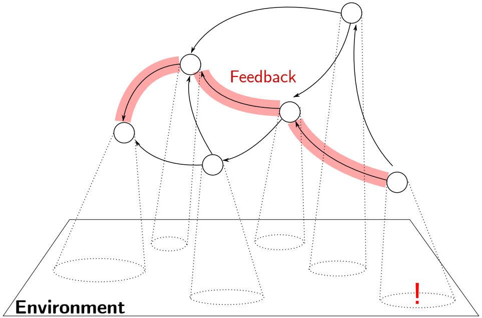
1. May not be equipped with the means to handle it
2. May not be able to determine its impact over the overall distributed execution



Motivation

In a MAS, the agent detecting a **perturbation**:

1. May not be equipped with the means to handle it
2. May not be able to determine its impact over the overall distributed execution

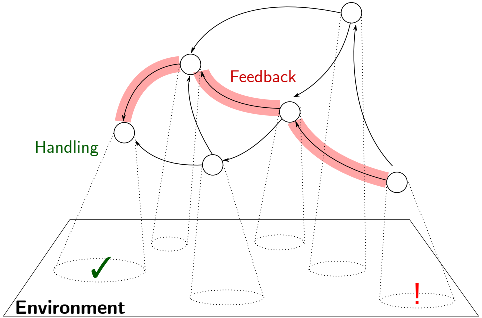


→ No structured way for collecting and propagating **feedback** about encountered situations

Motivation

In a MAS, the agent detecting a **perturbation**:

1. May not be equipped with the means to handle it
2. May not be able to determine its impact over the overall distributed execution



→ No structured way for collecting and propagating **feedback** about encountered situations

→ No clear **distribution of responsibilities** concerning the handling of perturbations

Robustness and Exception Handling

Robustness

The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions[2]

[2] "ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary". In: *ISO/IEC/IEEE 24765:2010(E)* (2010), pp. 1–418.

Robustness and Exception Handling

Robustness

The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions[2]

One mechanism that supports robustness is **exception handling**

→ Equipping the system with the capabilities to tackle classes of abnormal situations

→ Benefits in terms of **modularity** and **decoupling**

[2] "ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary". In: *ISO/IEC/IEEE 24765:2010(E)* (2010), pp. 1–418.

Robustness and Exception Handling

Robustness

The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions[2]

One mechanism that supports robustness is **exception handling**

→ Equipping the system with the capabilities to tackle classes of abnormal situations

→ Benefits in terms of **modularity** and **decoupling**

Current MAS architectures fall short in addressing robustness in a systematic way

→ No mechanisms for exception handling, as for **programming languages** or the **actor model**

[2] "ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary". In: *ISO/IEC/IEEE 24765:2010(E)* (2010), pp. 1–418.

Exception Handling in MAS

Research Objective

To present an **exception handling** mechanism for use in **multi-agent systems**, encompassing exceptions as first-class elements, and based on the notions of **responsibility** and **feedback**

Exception Handling in MAS

Research Objective

To present an **exception handling** mechanism for use in **multi-agent systems**, encompassing exceptions as first-class elements, and based on the notions of **responsibility** and **feedback**

MAS organizations are built upon **responsibilities**

→ Naturally suited to encompass an exception handling mechanism

Exception Handling in MAS

Research Objective

To present an **exception handling** mechanism for use in **multi-agent systems**, encompassing exceptions as first-class elements, and based on the notions of **responsibility** and **feedback**

MAS organizations are built upon **responsibilities**
→ Naturally suited to encompass an exception handling mechanism

Proposal

When joining an organization, agents will be asked to take on the **responsibilities**:

1. For **providing feedback** about the context where exceptions are detected
2. If appointed, for **handling** such exceptions once the needed information is available

Background

Exception Handling in Programming Languages

Among the first to address explicitly
the concern of exception handling[3]

[3] John B. Goodenough. "Exception Handling: Issues and a Proposed Notation". In: *Communications of the ACM* 18.12 (1975), pp. 683–696

Exception Handling in Programming Languages

Among the first to address explicitly the concern of exception handling[3]

Dedicated **language constructs** enable a **systematic treatment** of perturbations

```
public static void m1(...) throws Exception {
    if(...) throw new Exception(...);
}

public static void m2 {
    try {
        m1(...);
    }
    catch(Exception e) {
        ...
    }
}
```

[3] John B. Goodenough. "Exception Handling: Issues and a Proposed Notation". In: *Communications of the ACM* 18.12 (1975), pp. 683–696

Exception Handling in Programming Languages

Among the first to address explicitly the concern of exception handling[3]

Dedicated **language constructs** enable a **systematic treatment** of perturbations

When a perturbation is detected, the exection flow is deviated to a **handler**

```
public static void m1(...) throws Exception {
    if(...) throw new Exception(...);
}

public static void m2 {
    try {
        m1(...);
    }
    catch(Exception e) {
        ...
    }
}
```

[3] John B. Goodenough. "Exception Handling: Issues and a Proposed Notation". In: *Communications of the ACM* 18.12 (1975), pp. 683–696

Exception Handling in Programming Languages

Among the first to address explicitly the concern of exception handling[3]

Dedicated **language constructs** enable a **systematic treatment** of perturbations

When a perturbation is detected, the exection flow is deviated to a **handler**

The search for a suitable handler follows the program **call stack**

```
public static void m1(...) throws Exception {
    if(...) throw new Exception(...);
}

public static void m2 {
    try {
        m1(...);
    }
    catch(Exception e) {
        ...
    }
}
```

[3] John B. Goodenough. "Exception Handling: Issues and a Proposed Notation". In: *Communications of the ACM* 18.12 (1975), pp. 683–696

Exception Handling in the Actor Model

All computational entities are modeled as independent **actors**

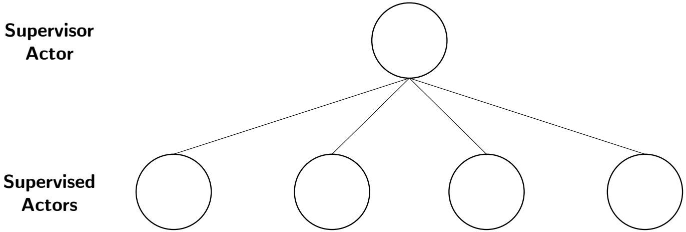
→ Communication through **message passing**

Exception Handling in the Actor Model

All computational entities are modeled as independent **actors**

→ Communication through **message passing**

→ Organized into a **supervision hierarchy**

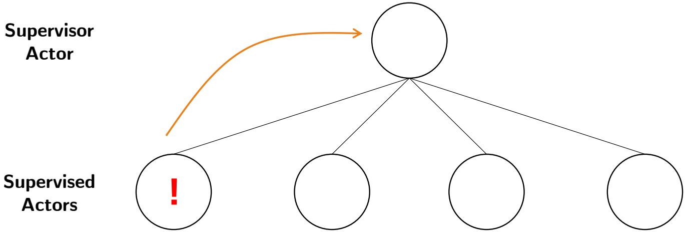


Exception Handling in the Actor Model

All computational entities are modeled as independent **actors**

→ Communication through **message passing**

→ Organized into a **supervision hierarchy**



Actors can notify **exceptions** to their parent actor

The parent should implement a suitable **supervision strategy** (or escalate the exception)

Business Process

A set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal.[4]

[4] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.

[5] Stephen A. White. "Introduction to BPMN". In: *IBM Cooperation* 2.0 (2004).

Business Process

A set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal.[4]

Different graphical notations to express process models → **BPMN**[5] is *de facto* standard

[4] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.

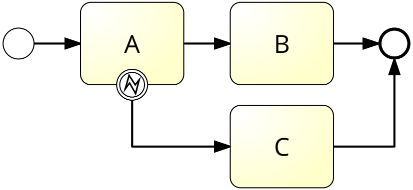
[5] Stephen A. White. "Introduction to BPMN". In: *IBM Cooperation* 2.0 (2004).

Business Process

A set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal.[4]

Different graphical notations to express process models → **BPMN**[5] is *de facto* standard

Error events model the occurrence of errors during the execution of an activity



[4] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.

[5] Stephen A. White. "Introduction to BPMN". In: *IBM Cooperation* 2.0 (2004).

Exception Handling in the MAS research

Multiple approaches have been proposed, but **without clear consensus**

- Guardian[6]
- Sentinels[7]
- Exceptions in the agent execution model[8]
- Failures events in SARL
- Contingency plans in Jason

[6] Anand Tripathi and Robert Miller. "Exception Handling in Agent-Oriented Systems". In: *Advances in Exception Handling Techniques*. Springer, 2001, pp. 128–146.

[7] Staffan Hägg. "A sentinel approach to fault handling in multi-agent systems". In: *Multi-Agent Systems Methodologies and Applications*. Springer, 1997, pp. 181–195.

[8] Eric Platon, Nicolas Sabouret, and Shinichi Honiden. "An architecture for exception management in multiagent systems". In: *IJAOSE* 2.3 (2008), pp. 267–289.

Exception Handling in the MAS research

Multiple approaches have been proposed, but **without clear consensus**

- Guardian[6]
- Sentinels[7]
- Exceptions in the agent execution model[8]
- Failures events in SARL
- Contingency plans in Jason

Main difficulty → Conjugate exception handling with the peculiarities of the agent paradigm

- **Autonomy**
- **Heterogeneity**
- **Openness**
- **Distribution**
- **Situatedness**

[6] Anand Tripathi and Robert Miller. “Exception Handling in Agent-Oriented Systems”. In: *Advances in Exception Handling Techniques*. Springer, 2001, pp. 128–146.

[7] Staffan Hägg. “A sentinel approach to fault handling in multi-agent systems”. In: *Multi-Agent Systems Methodologies and Applications*. Springer, 1997, pp. 181–195.

[8] Eric Platon, Nicolas Sabouret, and Shinichi Honiden. “An architecture for exception management in multiagent systems”. In: *IJAOSE* 2.3 (2008), pp. 267–289.

A Proposal for Exception Handling in Multi-Agent Systems

Responsibility and Feedback in Exception Handling

Two important aspects of exception handling:

Responsibility and Feedback in Exception Handling

Two important aspects of exception handling:

1. Two parties
    - The former is **responsible** for raising an exception
    - The latter is **responsible** for handling it

Responsibility and Feedback in Exception Handling

Two important aspects of exception handling:

1. Two parties
   - The former is **responsible** for raising an exception
   - The latter is **responsible** for handling it
2. It captures the need for some **feedback** from the former to the latter that allows coping with the exception

Responsibility and Feedback in Exception Handling

Two important aspects of exception handling:

1. Two parties
   - The former is **responsible** for raising an exception
   - The latter is **responsible** for handling it
2. It captures the need for some **feedback** from the former to the latter that allows coping with the exception

Exception handling is in essence a matter of **responsibility distribution**

Responsibility and Feedback in Exception Handling

Two important aspects of exception handling:

1. Two parties
   - The former is **responsible** for raising an exception
   - The latter is **responsible** for handling it
2. It captures the need for some **feedback** from the former to the latter that allows coping with the exception

Exception handling is in essence a matter of **responsibility distribution**

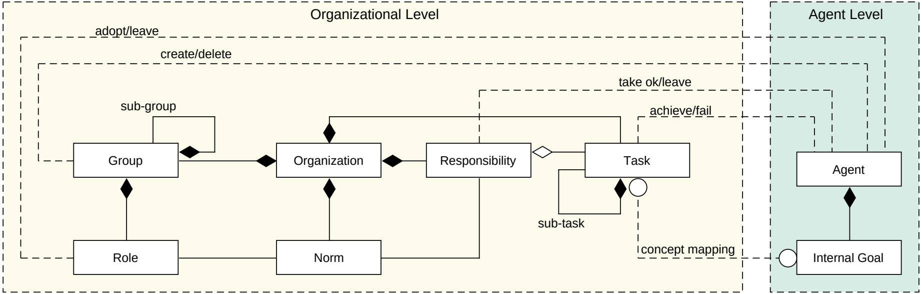
Multi-agent organizations are built upon responsibility

MAS Organizations

Key features of many organizational models:

- Decomposition of the **organizational task**
- **Normative system**

Norms shape the scope of the responsibilities that agents take when joining the organization
→ What agents should do to contribute to the achievement of the organizational task



Introducing Exceptions

Main idea → Review the basic mechanism of exception handling in terms of responsibilities

Introducing Exceptions

Main idea $\rightarrow$ Review the basic mechanism of exception handling in terms of responsibilities

When joining an organization, agents are asked to take on the **responsibilities** not only for organizational tasks, but also:

1. For **rising** exceptions when they encounter problems in fulfilling such responsibilities
2. For **handling** some of the exceptions raised from others

Introducing Exceptions

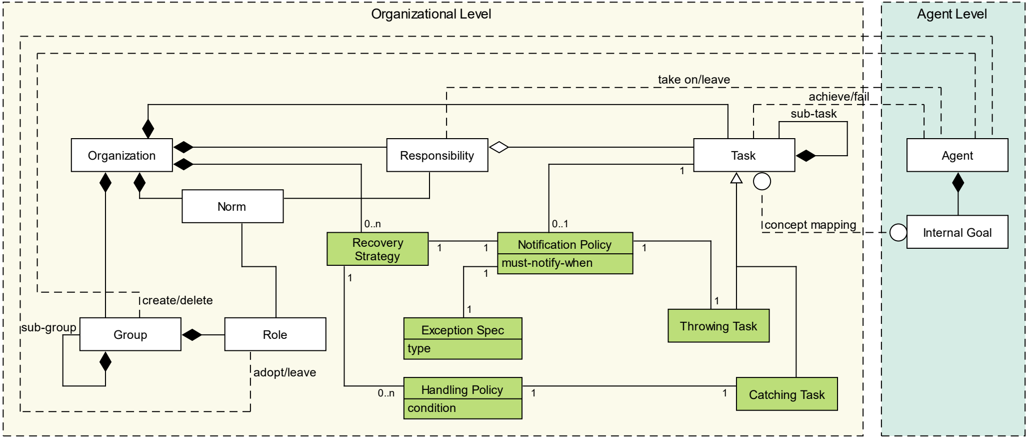
Main idea → Review the basic mechanism of exception handling in terms of responsibilities

When joining an organization, agents are asked to take on the **responsibilities** not only for organizational tasks, but also:

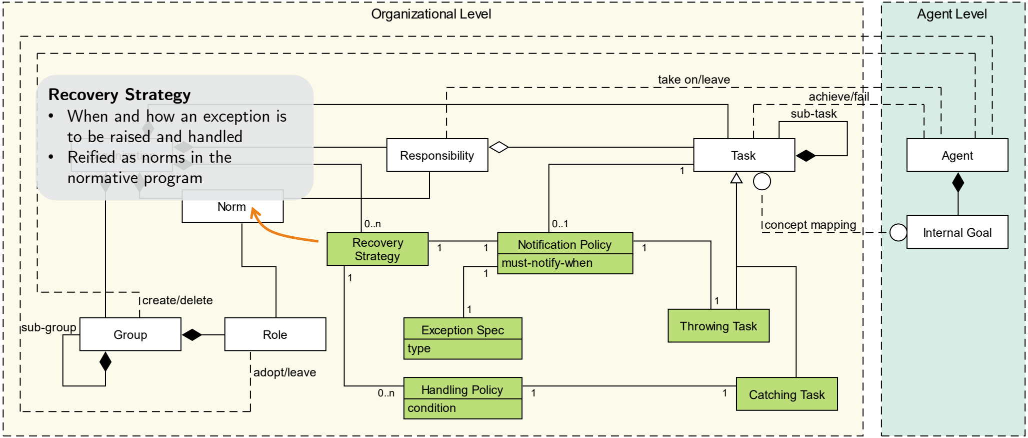
1. For **rising** exceptions when they encounter problems in fulfilling such responsibilities
2. For **handling** some of the exceptions raised from others

Norms govern these additional **responsibilities**, as well
→ **Obligations** issued when necessary

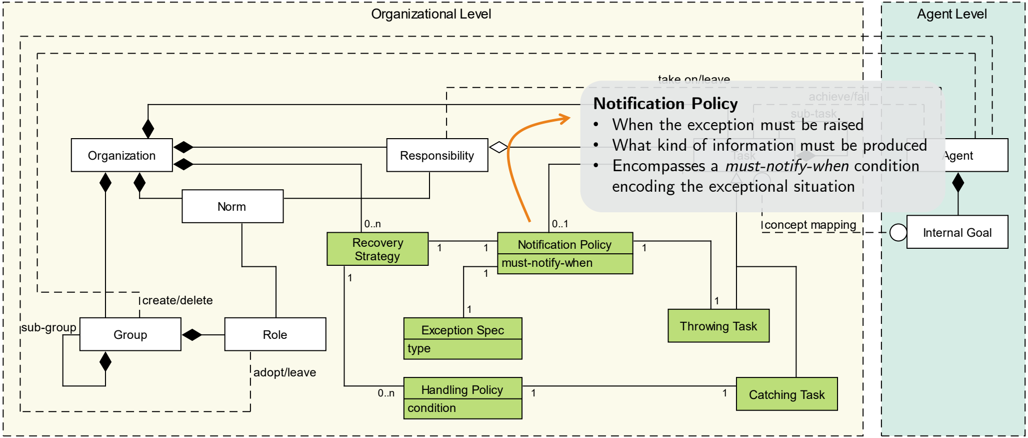
Exception Handling Model



Exception Handling Model



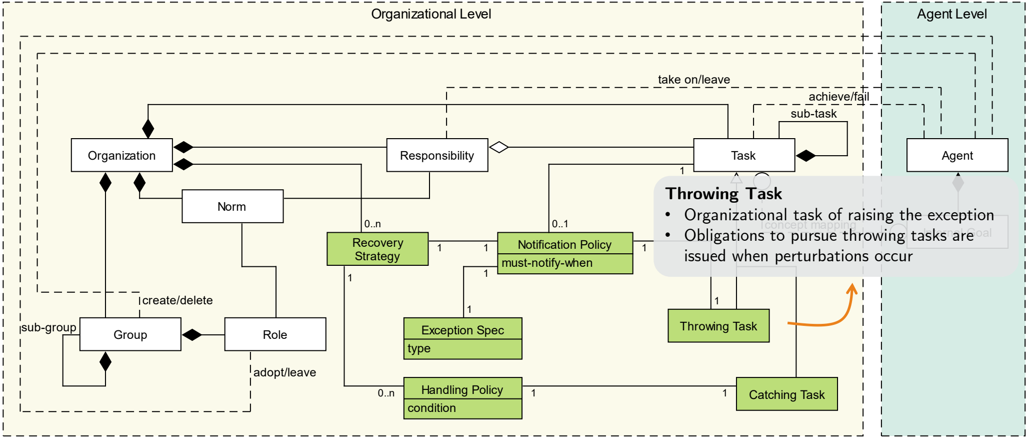
Exception Handling Model



Notification Policy

- When the exception must be raised
- What kind of information must be produced
- Encompasses a *must-notify-when* condition encoding the exceptional situation

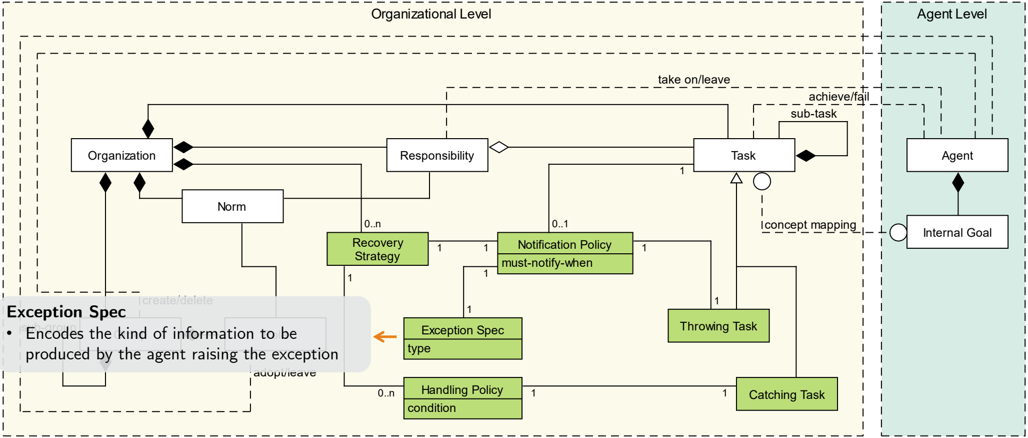
Exception Handling Model



Throwing Task

- Organizational task of raising the exception
- Obligations to pursue throwing tasks are issued when perturbations occur

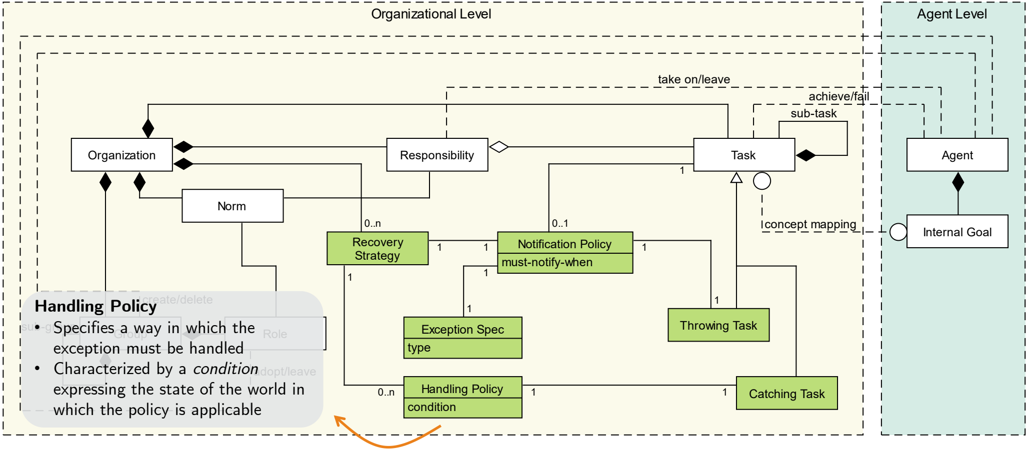
Exception Handling Model



Exception Spec

- Encodes the kind of information to be produced by the agent raising the exception

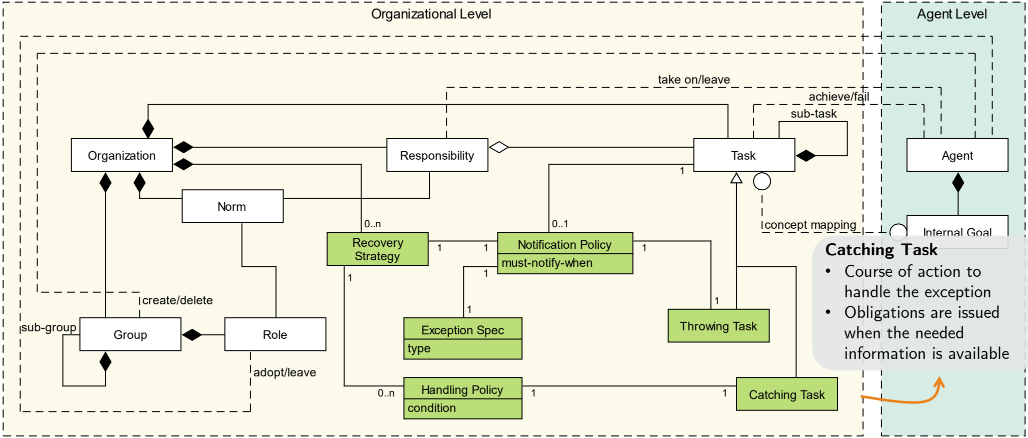
Exception Handling Model



Handling Policy

- Specifies a way in which the exception must be handled
- Characterized by a *condition* expressing the state of the world in which the policy is applicable

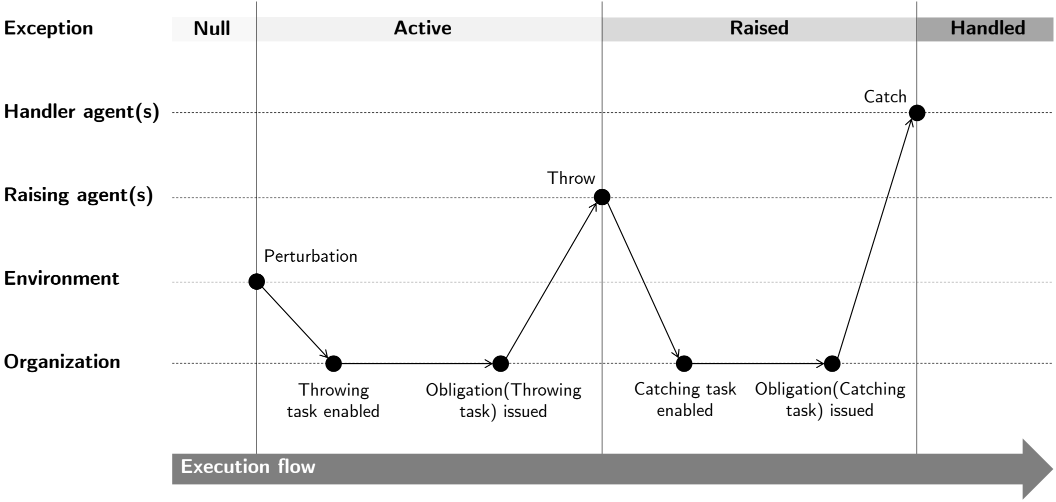
Exception Handling Model



Catching Task

- Course of action to handle the exception
- Obligations are issued when the needed information is available

Exception Lifecycle



Exception Handling in JaCaMo

JaCaMo Basics

JaCaMo is a well-known framework for the development of multi-agent organizations

[9] Rafael H. Bordini, Jomi F. Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

[10] Alessandro Ricci et al. "Environment Programming in CArtAgO". In: *Multi-Agent Programming: Languages, Tools and Applications*. Springer, 2009, pp. 259–288.

[11] Jomi F. Hübner, Jaime S. Sichman, and Olivier Boissier. "Developing Organised Multiagent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels". In: *International Journal of Agent-Oriented Software Engineering* 1.3/4 (2007), pp. 370–395.

JaCaMo Basics

JaCaMo is a well-known framework for the development of multi-agent organizations

It integrates three multi-agent dimensions:

- Agents → **Jason**[9]
- Environments → **CArtAgO**[10]
- Organizations → **Moise**[11]

[9] Rafael H. Bordini, Jomi F. Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

[10] Alessandro Ricci et al. "Environment Programming in CArtAgO". In: *Multi-Agent Programming: Languages, Tools and Applications*. Springer, 2009, pp. 259–288.

[11] Jomi F. Hübner, Jaime S. Sichman, and Olivier Boissier. "Developing Organised Multiagent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels". In: *International Journal of Agent-Oriented Software Engineering* 1.3/4 (2007), pp. 370–395.

JaCaMo Basics

JaCaMo is a well-known framework for the development of multi-agent organizations

It integrates three multi-agent dimensions:

- Agents $\rightarrow$ **Jason**[9]
- Environments $\rightarrow$ **CArtAgO**[10]
- Organizations $\rightarrow$ **Moise**[11]

Its organizational model does not include any mechanism for exception handling

[9] Rafael H. Bordini, Jomi F. Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

[10] Alessandro Ricci et al. "Environment Programming in CArtAgO". In: *Multi-Agent Programming: Languages, Tools and Applications*. Springer, 2009, pp. 259–288.

[11] Jomi F. Hübner, Jaime S. Sichman, and Olivier Boissier. "Developing Organised Multiagent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels". In: *International Journal of Agent-Oriented Software Engineering* 1.3/4 (2007), pp. 370–395.

Jason, CArtAgO and Moise

```
happy(bob).

!say(hello).

+!say(X)
     : happy(bob)
    <- .print(X).

// ...
```

Sample Jason agent

```
public class Counter extends Artifact {
    void init(int initialValue) {
        defineObsProperty("count",
                          initialValue);
    }

    @OPERATION
    void inc() {
        ObsProperty prop =
                getObsProperty("count");
        prop.updateValue(prop.intValue()+1);
    }
}
```

Sample artifact in CArtAgO

```
<organisational-specification ...>
    <structural-specification>
        <role-definitions>
            ...
        </role-definitions>
        <group-specification id="...">
            ...
        </group-specification>
    </structural-specification>

    <functional-specification>
        <scheme id="...">
            <goal id="...">
                ...
            </goal>
            <mission id="...">
                ...
            </mission>
        </scheme>
    </functional-specification>

    <normative-specification>
        <norm ... />
        ...
    </normative-specification>
</organisational-specification>
```

Sample org. spec. in Moise

Extending JaCaMo for Exception Handling

- The extension mostly concerns the **organizational dimension**
  → The Moise component

Extending JaCaMo for Exception Handling

- The extension mostly concerns the **organizational dimension**
  → The Moise component

- Changes are **as conservative as possible**
  → When no recovery strategy is specified, we fall back to standard JaCaMo

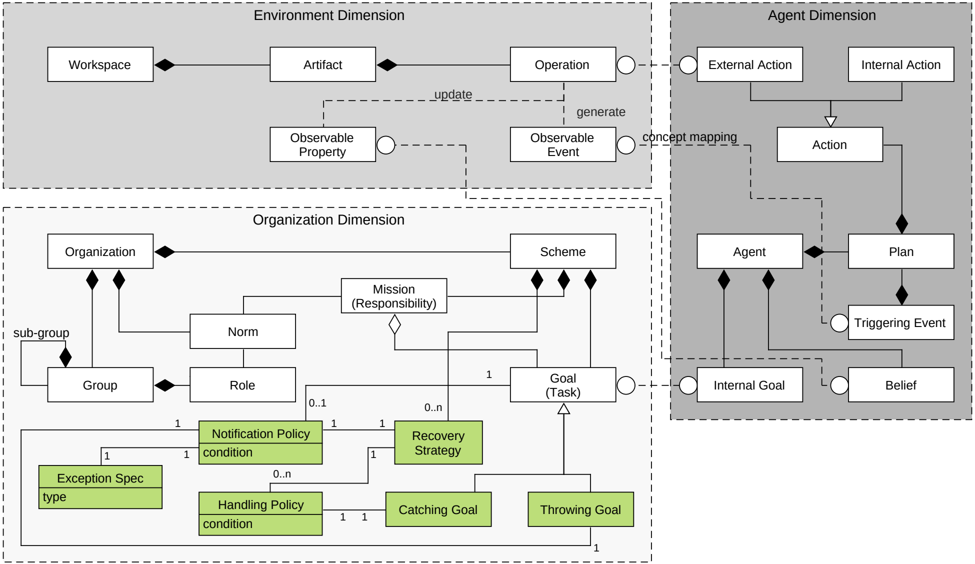
Extending JaCaMo for Exception Handling

- The extension mostly concerns the **organizational dimension**
  → The Moise component

- Changes are **as conservative as possible**
  → When no recovery strategy is specified, we fall back to standard JaCaMo

- We satisfy three needs
  → **Specify recovery strategies** within an organization
  → Translate recovery strategies into a **corresponding body of norms**
  → Give agents the capability of **throwing exceptions** and **marking goals** not only as achieved, but also as **failed** or **released**

JaCaMo Metamodel Extended



Generating norms from recovery strategies

Recovery are strategies reified as norms in JaCaMo's normative program

Exception Handling Infrastructure

Generating norms from recovery strategies

Recovery are strategies reified as norms in JaCaMo's normative program

New **normative facts**, such as:

- notificationPolicy(NP,Condition)
- handlingPolicy(HP,Condition)
- policy_goal(P,G)
- exceptionArgument(E,ArgFunctor,ArgArity)
- failed(S,G)
- thrown(S,E,Ag,Args)

Exception Handling Infrastructure

Generating norms from recovery strategies

Recovery are strategies reified as norms in JaCaMo's normative program

New **normative facts**, such as:

- notificationPolicy(NP,Condition)
- handlingPolicy(HP,Condition)
- policy_goal(P,G)
- exceptionArgument(E,ArgFunctor,ArgArity)
- failed(S,G)
- thrown(S,E,Ag,Args)

New **rules** and **norms**, like:

```
enabled(S,TG) :-
        policy_goal(P,TG) & notificationPolicy(P,Condition) & Condition &
        goal(_, TG, Dep, _, NP, _) & NP \== 0 &
        ((Dep = dep(or,PCG) & (any_satisfied(S,PCG) | all_released(S,PCG))) |
        (Dep = dep(and,PCG) & all_satisfied_released(S,PCG))).
```

Exception Handling Infrastructure

Extending the organizational artifacts

Agents commit to missions encompassing standard goals, as well as throwing and catching ones
→ Obligations for **throwing goals** issued when perturbations occur
→ Obligations for **catching goals** issued when an **exception** has been thrown

Exception Handling Infrastructure

Extending the organizational artifacts

Agents commit to missions encompassing standard goals, as well as throwing and catching ones
→ Obligations for **throwing goals** issued when perturbations occur
→ Obligations for **catching goals** issued when an **exception** has been thrown

New operations are made available to the agents:

- goalFailed(G)
- throwException(S,E,Ag,Args)
- goalReleased(G)

Exception Handling Infrastructure

Extending the organizational artifacts

Agents commit to missions encompassing standard goals, as well as throwing and catching ones
→ Obligations for **throwing goals** issued when perturbations occur
→ Obligations for **catching goals** issued when an **exception** has been thrown

New operations are made available to the agents:

- goalFailed(G)
- throwException(S,E,Ag,Args)
- goalReleased(G)

Exceptions are made available as artifact's **observable properties**

Agent Programming with Exceptions

```
+obligation(Ag,_,done(_,SOME_GOAL,Ag),_)
    : my_name(Ag)
   <- !SOME_GOAL;
      goalAchieved(SOME_GOAL).

+!SOME_GOAL
   <- // do something to achieve the goal

-!SOME_GOAL
   <- goalFailed(SOME_GOAL);
      .fail.

+obligation(Ag,_,done(_,THROWING_GOAL,Ag),_)
    : my_name(Ag)
   <- throwException(E,[arg1(A1),...,argN(AN)]);
      goalAchieved(THROWING_GOAL).
```

Raising agent

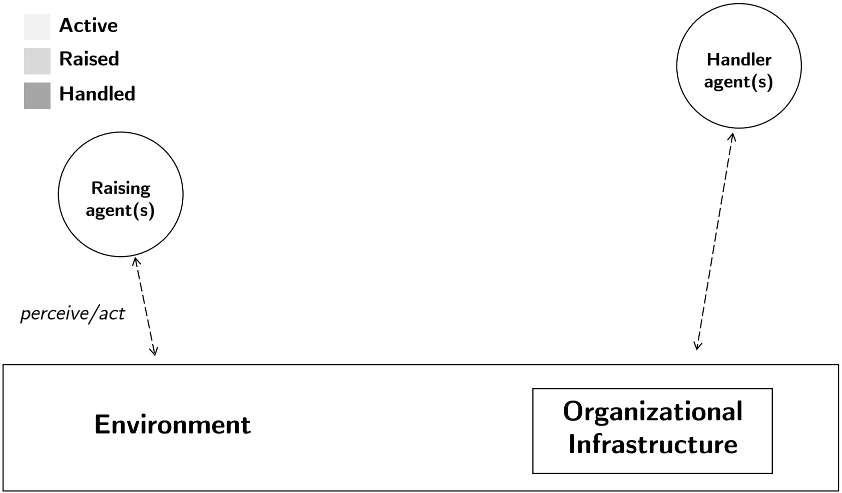
```
+obligation(Ag,_,done(_,CATCHING_GOAL,Ag),_)
    : my_name(Ag) &
      exceptionArgument(_,E,arg1(A1)) & ... &
      exceptionArgument(_,E,arg1(AN))
   <- // do something to handle the exception
      goalReleased(SOME_GOAL);
      // or resetGoal(SOME_GOAL);
      goalAchieved(CATCHING_GOAL).
```

Handler agent

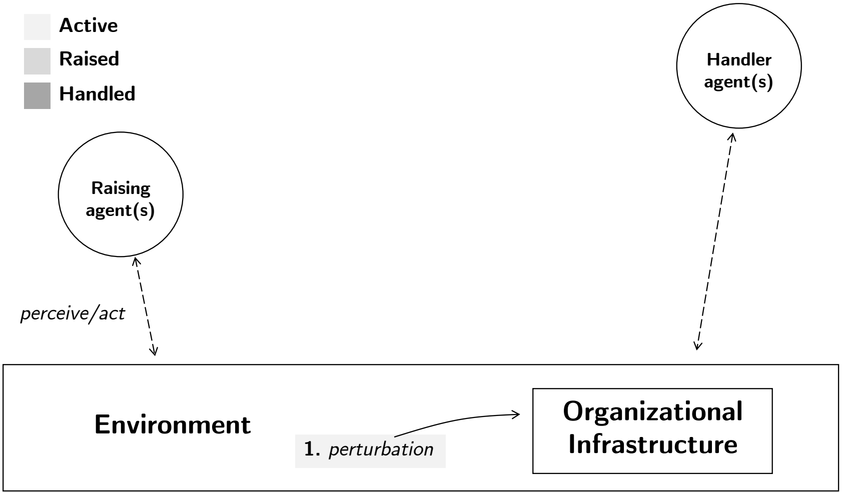
```
<recovery-strategy id="...">
  <notification-policy id="...">
    <condition type="goal-failure">
      <condition-argument id="target" value="SOME_GOAL" />
    </condition>
    <exception-spec id="E">
      <exception-argument id="arg1" arity="..." />
      ...
      <exception-argument id="argN" arity="..." />
    </exception-spec>
    <goal id="THROWING_GOAL" />
  </notification-policy>
  <handling-policy id="...">
    <condition type="always" />
    <goal id="CATCHING_GOAL" />
  </handling-policy>
</recovery-strategy>
```

Recovery strategy

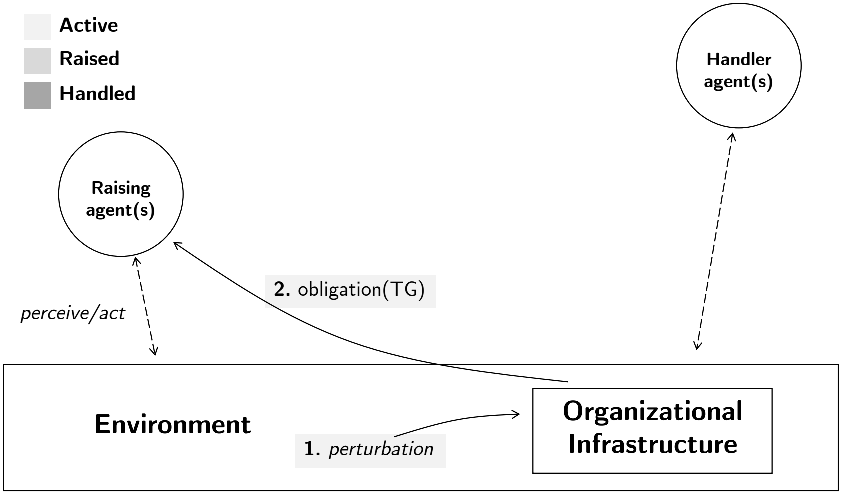
Exception Handling in Operation



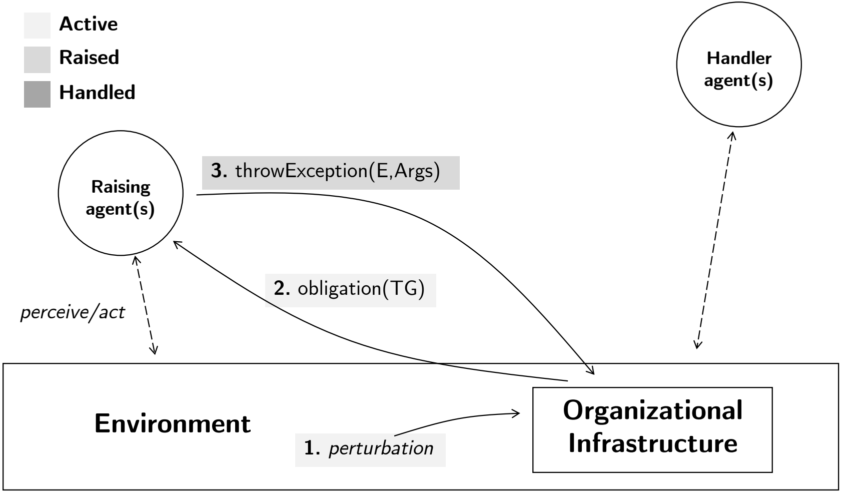
Exception Handling in Operation



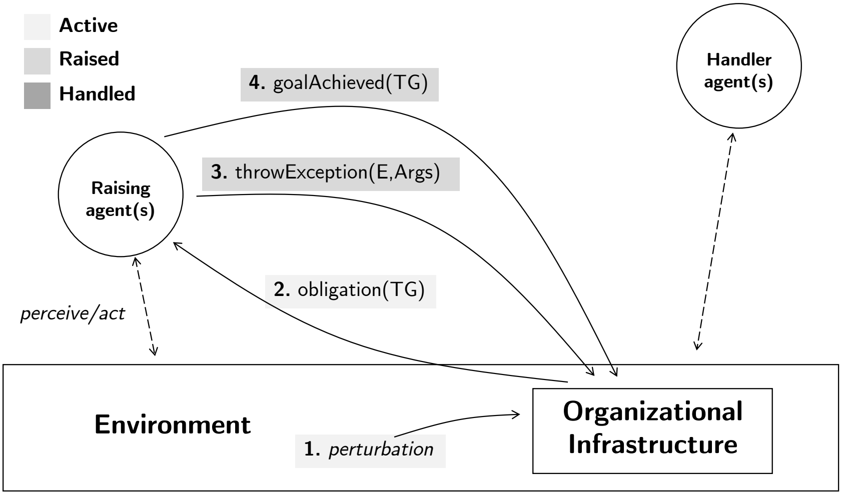
Exception Handling in Operation



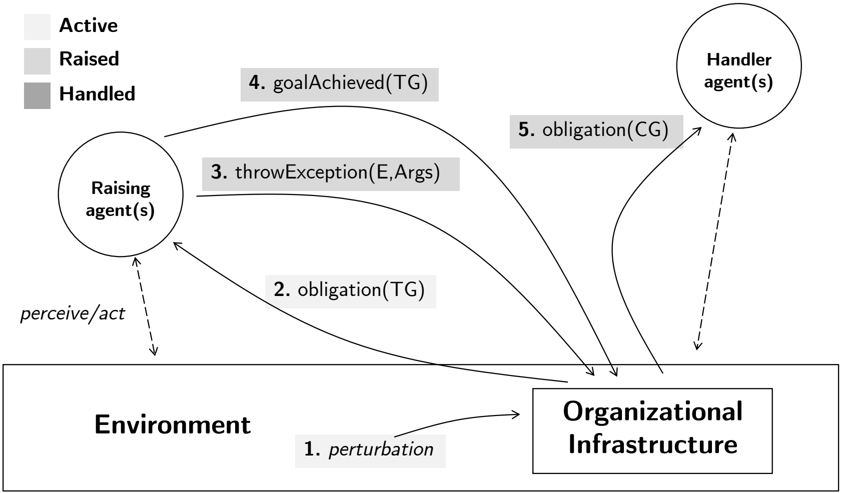
Exception Handling in Operation



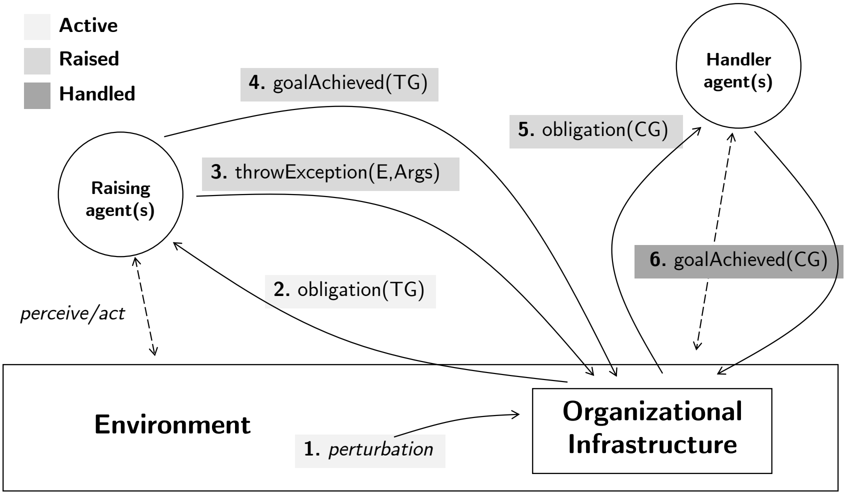
Exception Handling in Operation



Exception Handling in Operation

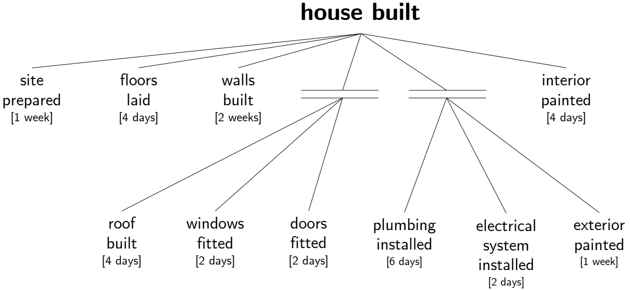


Exception Handling in Operation



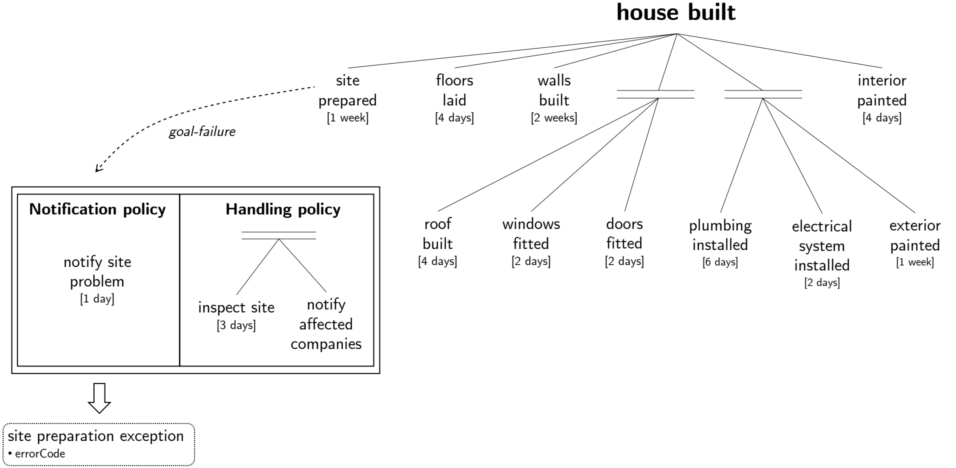
Example: House Building

- The organizational goal is to build a house on a plot
- Site preparation must be completed before any other step; should a failure occur, the whole construction could not proceed



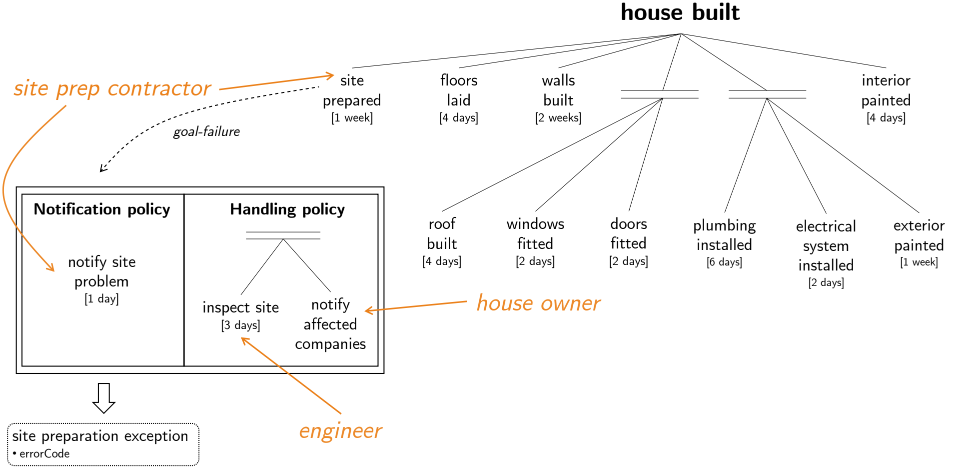
Example: House Building

- The organizational goal is to build a house on a plot
- Site preparation must be completed before any other step; should a failure occur, the whole construction could not proceed



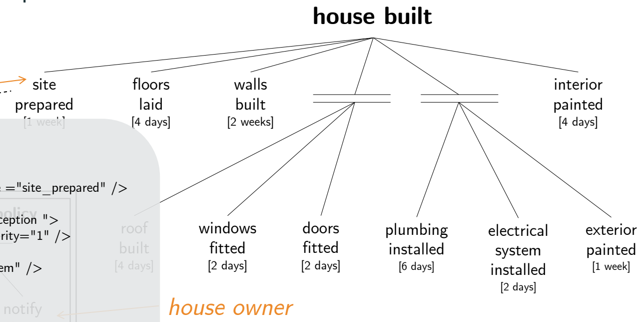
Example: House Building

- The organizational goal is to build a house on a plot
- Site preparation must be completed before any other step; should a failure occur, the whole construction could not proceed



Example: House Building

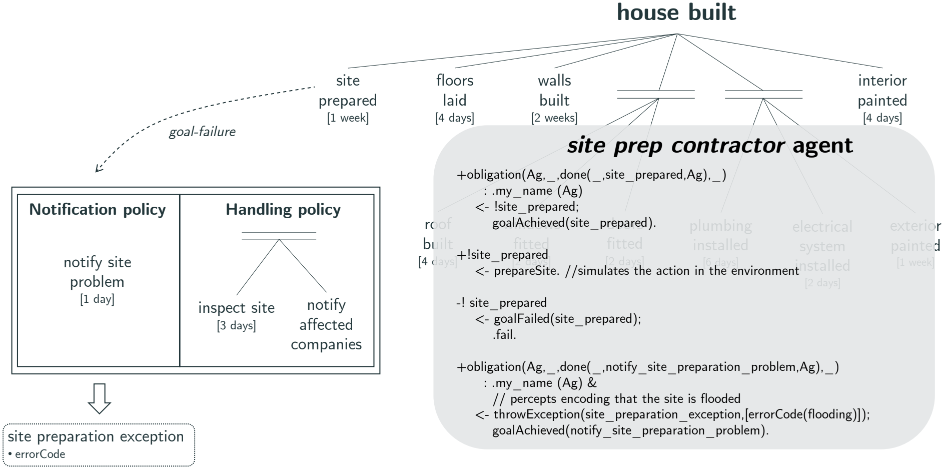
- The organizational goal is to build a house on a plot
- Site preparation must be completed before any other step; should a failure occur, the whole construction could not proceed




```xml
<recovery-strategy id="rsSitePreparation ">
  <notification-policy id="np1">
    <condition type="goal-failure">
      <condition-argument id="target" value ="site_prepared" />
    </condition>
    <exception-spec id="site_preparation_exception ">
      <exception-argument id="errorCode" arity="1" />
    </exception-spec>
    <goal id="notify_site_preparation_problem" />
  </notification-policy>
  <handling-policy id="hp1">
    <condition type="always" />
    <goal id="handle_site_problem">
      <plan operator="parallel">
        <goal id="inspect_site" />
        <goal id="notify_affected_companies" />
      </plan>
    </goa >
  </handling-policy>
</recovery-strategy>
```

Example: House Building

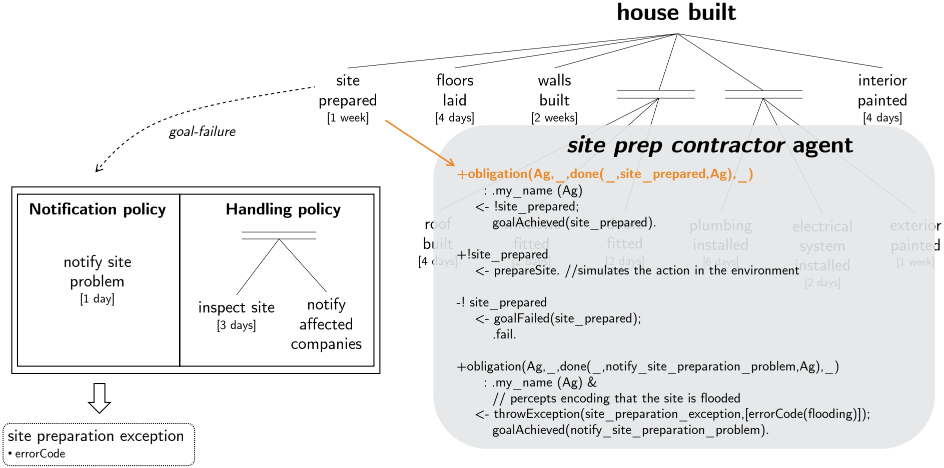
- The organizational goal is to build a house on a plot
- Site preparation must be completed before any other step; should a failure occur, the whole construction could not proceed




```
+obligation(Ag,_,done(_,site_prepared,Ag),_)
    : .my_name (Ag)
   <- !site_prepared;
      goalAchieved(site_prepared).

+!site_prepared
   <- prepareSite. //simulates the action in the environment

-! site_prepared
   <- goalFailed(site_prepared);
      .fail.

+obligation(Ag,_,done(_,notify_site_preparation_problem,Ag),_)
    : .my_name (Ag) &
      // percepts encoding that the site is flooded
   <- throwException(site_preparation_exception,[errorCode(flooding)]);
      goalAchieved(notify_site_preparation_problem).
```

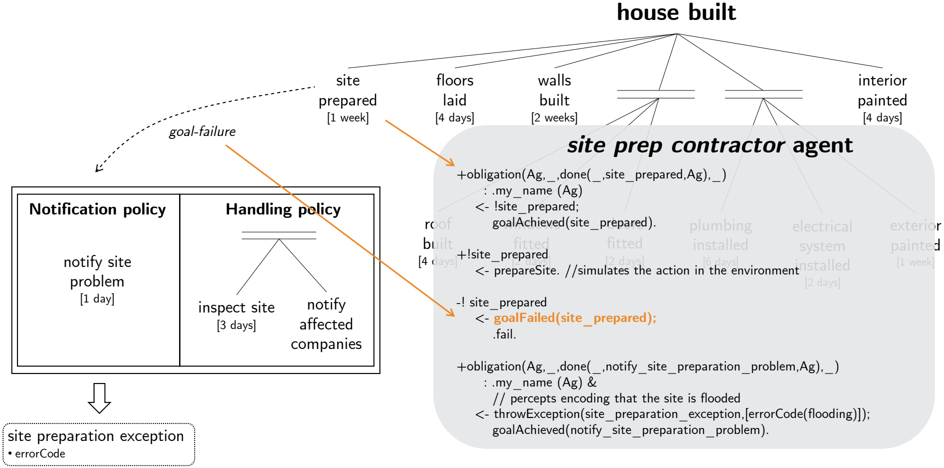
Example: House Building

- The organizational goal is to build a house on a plot
- Site preparation must be completed before any other step; should a failure occur, the whole construction could not proceed



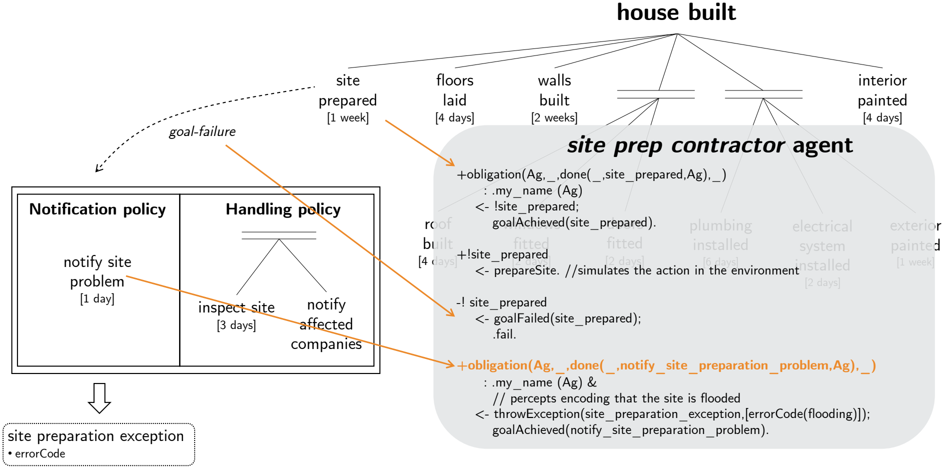
Example: House Building

- The organizational goal is to build a house on a plot
- Site preparation must be completed before any other step; should a failure occur, the whole construction could not proceed



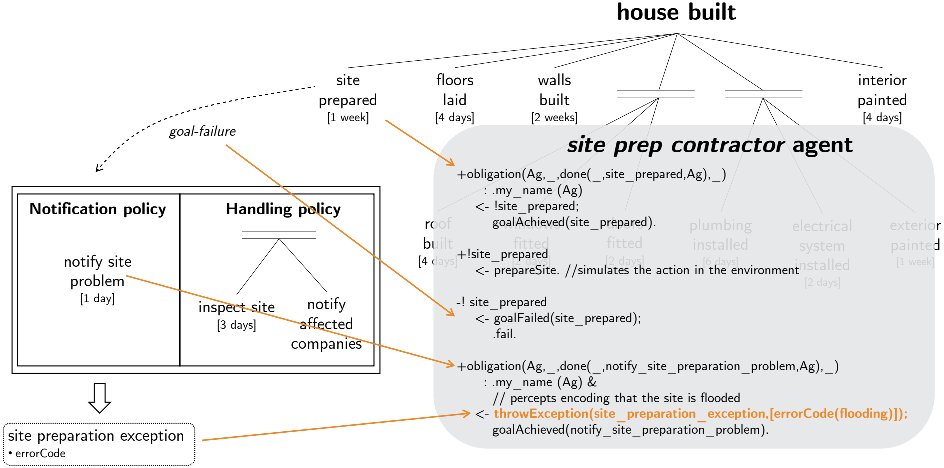
Example: House Building

- The organizational goal is to build a house on a plot
- Site preparation must be completed before any other step; should a failure occur, the whole construction could not proceed



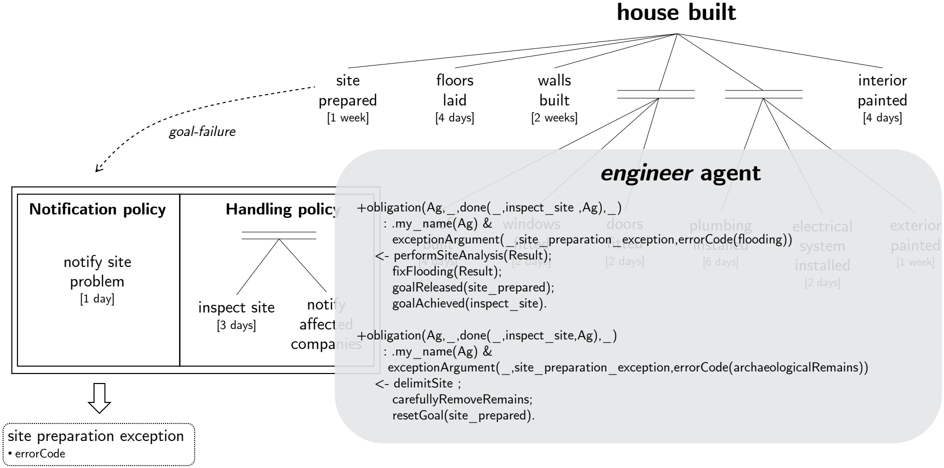
Example: House Building

- The organizational goal is to build a house on a plot
- Site preparation must be completed before any other step; should a failure occur, the whole construction could not proceed



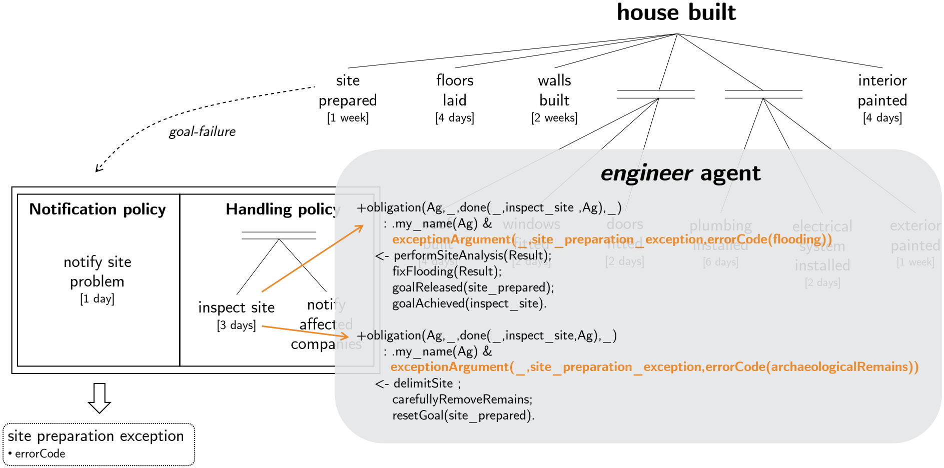
Example: House Building

- The organizational goal is to build a house on a plot
- Site preparation must be completed before any other step; should a failure occur, the whole construction could not proceed



Example: House Building

- The organizational goal is to build a house on a plot
- Site preparation must be completed before any other step; should a failure occur, the whole construction could not proceed



Discussion and Conclusions

Benefits

Five important features of the approach:

- **Autonomy preservation**
- **Decentralization**
- **Responsibility distribution**
- **Reliable feedback**
- **Platform integration**

Benefits

Five important features of the approach:

Autonomy preservation

Decentralization

Responsibility distribution

Reliable feedback

Platform integration

We can capture a wide range of situations:

- Goal **failures**
- Goal **delays**
- **Custom** perturbations
- Exceptions **raised collectively**
- Exceptions **handled collectively**
- **Recurrent** exception handling
- **Concerted** exceptions
- **BPMN** error events

Exception Handling and Message Passing

Robustness could be achieved by relying on **inter-agent messages**

Exception Handling and Message Passing

Robustness could be achieved by relying on **inter-agent messages**

But...

Exception Handling and Message Passing

Robustness could be achieved by relying on **inter-agent messages**

But...

→ The agent detecting a perturbation may not know to whom a message must be sent
→ No **expectation** on agents' behavior

Exception Handling and Message Passing

Robustness could be achieved by relying on **inter-agent messages**

But...

→ The agent detecting a perturbation may not know to whom a message must be sent
→ No **expectation** on agents' behavior

Responsibilities concerning the handling of perturbations are not clear
→ No uniform approach for agent programming
→ Bad impact in terms of **modularity** and **decoupling**

Exception Handling and Message Passing

Robustness could be achieved by relying on **inter-agent messages**

But...

→ The agent detecting a perturbation may not know to whom a message must be sent
→ No **expectation** on agents' behavior

Responsibilities concerning the handling of perturbations are not clear
→ No uniform approach for agent programming
→ Bad impact in terms of **modularity** and **decoupling**

We rely on the **normative layer** of the organization to encode such responsibilities
→ **No significant increase of computational cost** when exception handling is in place

Modularity through Exception Handling

Exceptions and exception handling are **not only needed to deal with errors**

[12] John B. Goodenough. "Exception Handling: Issues and a Proposed Notation". In: *Communications of the ACM* 18.12 (1975), pp. 683–696.

Modularity through Exception Handling

Exceptions and exception handling are **not only needed to deal with errors**

Means for enabling **robust software composition**[12]
→ Allow the invoker of an operation to **extend the operation domain or its range**
→ **Increase in generality**: the "fixup" will depend on the exception receiver's objective

[12] John B. Goodenough. "Exception Handling: Issues and a Proposed Notation". In: *Communications of the ACM* 18.12 (1975), pp. 683–696.

Modularity through Exception Handling

Exceptions and exception handling are **not only needed to deal with errors**

Means for enabling **robust software composition**[12]
→ Allow the invoker of an operation to **extend the operation domain or its range**
→ **Increase in generality**: the "fixup" will depend on the exception receiver's objective

MAS bring software modularity and separation of concerns to an extreme

[12] John B. Goodenough. "Exception Handling: Issues and a Proposed Notation". In: *Communications of the ACM* 18.12 (1975), pp. 683–696.

Exception Handling as Accountability

Exception handling can be effectively conceived, more generally, in terms of **accountability relationships** among agents

Exception Handling as Accountability

Exception handling can be effectively conceived, more generally, in terms of **accountability relationships** among agents

Accountability (Cambridge Dictionary)

The fact of being responsible for what you do and able to give a satisfactory reason for it, or the degree to which this happens

Exception Handling as Accountability

Exception handling can be effectively conceived, more generally, in terms of **accountability relationships** among agents

Accountability (Cambridge Dictionary)

The fact of being responsible for what you do and able to give a satisfactory reason for it, or the degree to which this happens

Channels through which relevant local information (**accounts**) flow from informed sources (**a-givers**) to the agents competent to understand the answer (**a-takers**)

Exception Handling as Accountability

Exception handling can be effectively conceived, more generally, in terms of **accountability relationships** among agents

Accountability (Cambridge Dictionary)

The fact of being responsible for what you do and able to give a satisfactory reason for it, or the degree to which this happens

Channels through which relevant local information (**accounts**) flow from informed sources (**a-givers**) to the agents competent to understand the answer (**a-takers**)

Accountability supports robustness when the account about a perturbation is reported to the agent who is responsible for treating that perturbation

Future Directions

1. **Unexpected** (or unanticipated) **exceptions** → Self-Adaptive Systems

Future Directions

1. **Unexpected** (or unanticipated) **exceptions** $\rightarrow$ Self-Adaptive Systems
2. **Robustness through accountability** $\rightarrow$ Many system properties can be seen as kinds of robustness
   - **Reliability**
   - **Efficency**
   - **Scalability**
   - **Modularity**
   - **Evolvability**

Future Directions

1. **Unexpected** (or unanticipated) **exceptions** → Self-Adaptive Systems

2. **Robustness through accountability** → Many system properties can be seen as kinds of robustness
   - **Reliability**
   - **Efficency**
   - **Scalability**
   - **Modularity**
   - **Evolvability**

Accountability as a **framework to exchange reliable feedback** among distributed components in a structured way
→ Support for a wide range of **non-functional requirements**

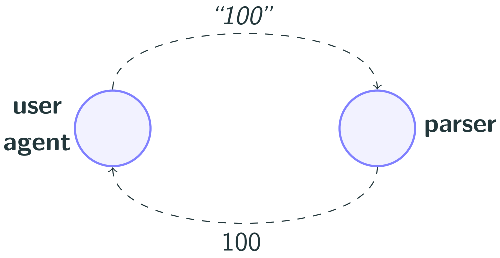
Thank you for your attention!

Questions?

ATM Example

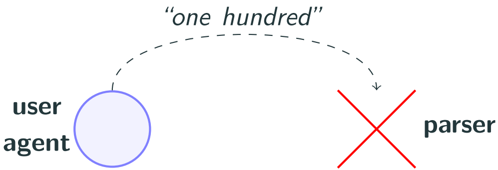
Money withdrawal at an ATM involves two steps:

1. The desired amount is collected from the user by a **user agent** as a string
2. The string is parsed by a **parser** and the amount is given back to the user agent to provide the money



ATM Example

If the string is not a number in digits parsing fails



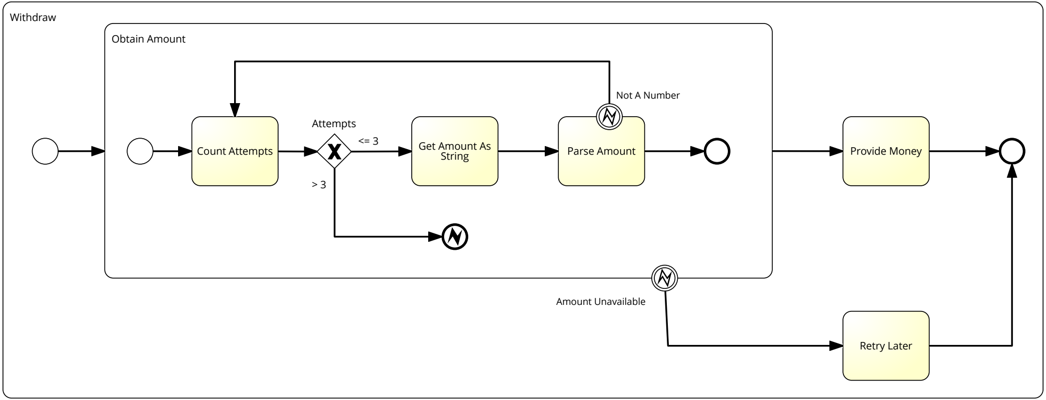
The parser agent has only a **partial view**

→ Unaware of the data source and of the aims for which the parsing is requested

Only the user agent has the necessary **contextual information**

→ A new input must be requested to the user

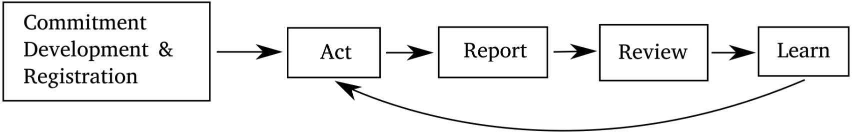
ATM Example in BPMN



Accountability in the Human World

Accountability frameworks describe organization-wide processes for monitoring, analysing, and improving performance in all aspects of an organization[13]

Address recurring and systemic issues to incorporate lessons learned into future activities



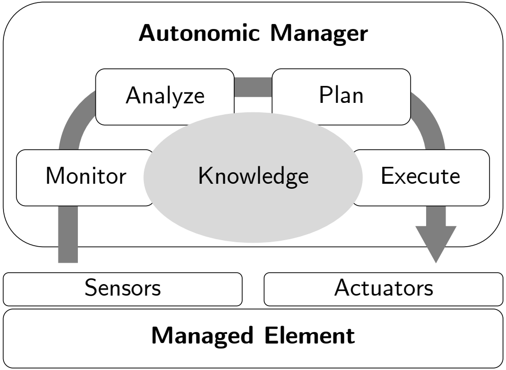
A general schema for accountability frameworks

[13] Executive Board of the United Nations Development Programme and of the United Nations Population Fund. *The UNDP accountability system, Accountability framework and oversight policy*. Tech. rep. DP/2008/16/Rev.1. United Nations, 2008.

Self-Adaptive Systems

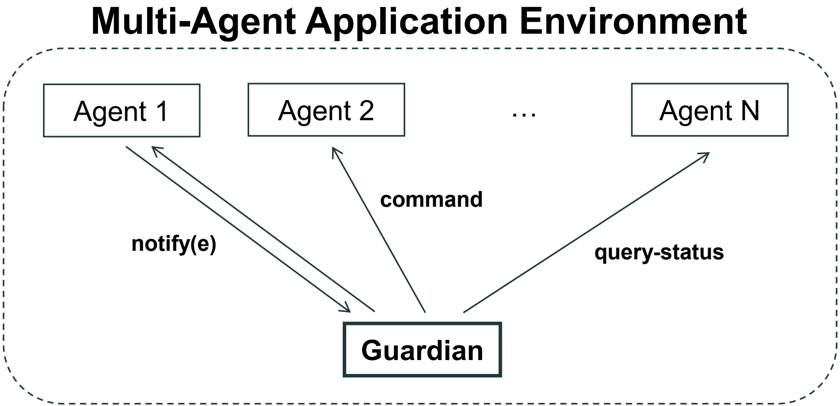
Self-adaptive software aims at **autonomously evaluating and changing its behavior** whenever an evaluation shows that the system is not accomplishing what it was intended to do[14]

Most approaches follow the **MAPE-K loop**[15]



[14] Frank D. Macías-Escrivá et al. "Self-adaptive systems: A survey of current approaches, research challenges and applications". In: *Expert Systems with Applications* 40.18 (2013), pp. 7267–7279.

[15] IBM. *An Architectural Blueprint for Autonomic Computing*. Tech. rep. IBM, 2005.


Multi-Agent Application Environment
Agent 1
Agent 2
…
Agent N
command
notify(e)
query-status
Guardian

Organizational Artifacts in JaCaMo

OrgBoard

- Group Boards
- Scheme Boards
- Org Specification

- createGroup
- removeGroup
- createScheme
- removeScheme

GroupBoard

- Role Players
- Schemes
- Group Specification
- Subgroups
- Parent Group
- Formation Status

- adoptRole
- leaveRole
- addScheme
- removeScheme
- setParentGroup
- destroy

SchemeBoard

- Commitments
- Groups
- Goals States
- Scheme Specification
- Obligations
- Goals Arguments

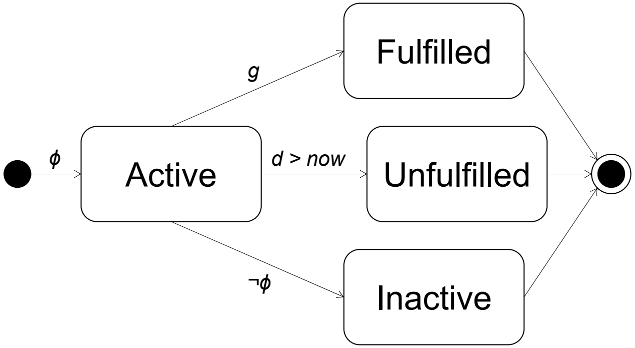
- commitMission
- leaveMission
- goalAchieved
- setArgumentValue
- resetGoal
- destroy

NormativeBoard

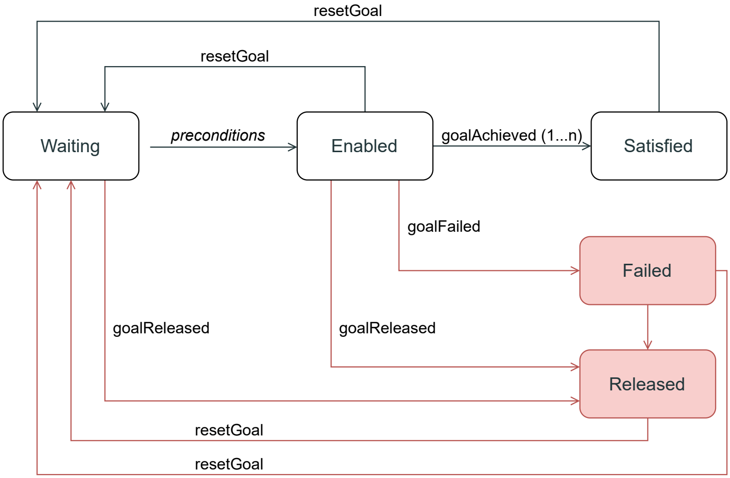
- Obligations

- load
- addFact

State transitions for obligations in JaCaMo



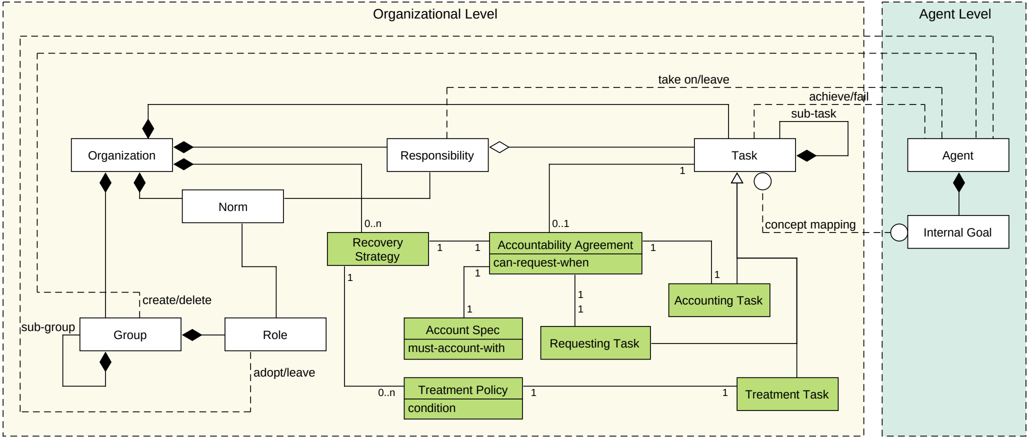
Extended Lifecycle of a JaCaMo Goal



Condition Types for Recovery Strategies

Type	Arguments	Condition formula
always	[]	true
goal-failure	[target]	scheme_id(S) & failed(S,$target)
goal-ttf-expiration	[target]	scheme_id(S) & unfulfilled(obligation(_,_,done(S,$target,_),_))
custom	[formula]	$formula

Accountability Model



- Boissier, Olivier et al. *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. MIT Press, 2020.
- Bordini, Rafael H., Jomi F. Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
- Executive Board of the United Nations Development Programme and of the United Nations Population Fund. *The UNDP accountability system, Accountability framework and oversight policy*. Tech. rep. DP/2008/16/Rev.1. United Nations, 2008.
- Goodenough, John B. "Exception Handling: Issues and a Proposed Notation". In: *Communications of the ACM* 18.12 (1975), pp. 683–696.
- Hägg, Staffan. "A sentinel approach to fault handling in multi-agent systems". In: *Multi-Agent Systems Methodologies and Applications*. Springer, 1997, pp. 181–195.
- Hübner, Jomi F., Jaime S. Sichman, and Olivier Boissier. "Developing Organised Multiagent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels". In: *International Journal of Agent-Oriented Software Engineering* 1.3/4 (2007), pp. 370–395.
- IBM. *An Architectural Blueprint for Autonomic Computing*. Tech. rep. IBM, 2005.

- "ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary". In: *ISO/IEC/IEEE 24765:2010(E)* (2010), pp. 1–418.
- Macías-Escrivá, Frank D. et al. "Self-adaptive systems: A survey of current approaches, research challenges and applications". In: *Expert Systems with Applications* 40.18 (2013), pp. 7267–7279.
- Platon, Eric, Nicolas Sabouret, and Shinichi Honiden. "An architecture for exception management in multiagent systems". In: *IJAOSE* 2.3 (2008), pp. 267–289.
- Ricci, Alessandro et al. "Environment Programming in CArtAgO". In: *Multi-Agent Programming: Languages, Tools and Applications*. Springer, 2009, pp. 259–288.
- Tripathi, Anand and Robert Miller. "Exception Handling in Agent-Oriented Systems". In: *Advances in Exception Handling Techniques*. Springer, 2001, pp. 128–146.
- Weske, Mathias. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.
- White, Stephen A. "Introduction to BPMN". In: *IBM Cooperation* 2.0 (2004).