Distributing Responsibilities for Exception Handling in JaCaMo

M. Baldoni[1], C. Baroglio[1], O. Boissier[2], R. Micalizio[1], S. Tedeschi[1]

[1] Università degli Studi di Torino, Dipartimento di Informatica, Italy, firstname.lastname@unito.it
[2] Laboratoire Hubert Curien UMR CNRS 5516, Institut Henri Fayol, MINES Saint-Etienne, France, Olivier.Boissier@emse.fr

Overview

We present an extension of the organizational model adopted in **JaCaMo*** that explicitly encompasses the notion of **exception**

We show how **exception handling** can be grafted inside the normative system of a MAS organization to gain **robustness** in execution

The proposed exception handling mechanism relies on abstractions that are seamlessly integrated with organizational concepts, like:

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

Robustness and Exception Handling

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

[ISO/IEC/IEEE 24765:2017 Systems and software engineering — Vocabulary]

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

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

The need of exceptions emerges from the desire of modularizing software, separating concerns into components that interact

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

No mechanisms for **exception handling**, as is for programming languages (e.g. Java), or in the actor model (e.g. **Akka**)

Responsibility in Exception Handling

Two important aspects of exception handling:

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

Since MAS organizations are built upon responsibilities, they are suited to encompass an exception handling mechanism

* http://jacamo.sourceforge.net/

Key features of many organizational models:

- Functional decomposition of the organizational goal
- Normative system

Norms shape the scope of the responsibilities that agents take when joining the organization

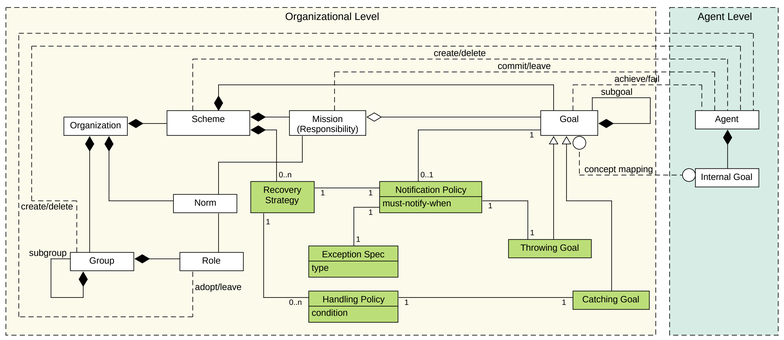
→ What agents should do to contribute to the achievement of the organizational goal

Our Proposal

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

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

Extending JaCaMo



Recovery Strategy encodes when and how a given exception is to be raised and handled within the organization

Notification Policy specifies when the exception must be raised

Throwing Goal denotes the organizational goal of raising the exception

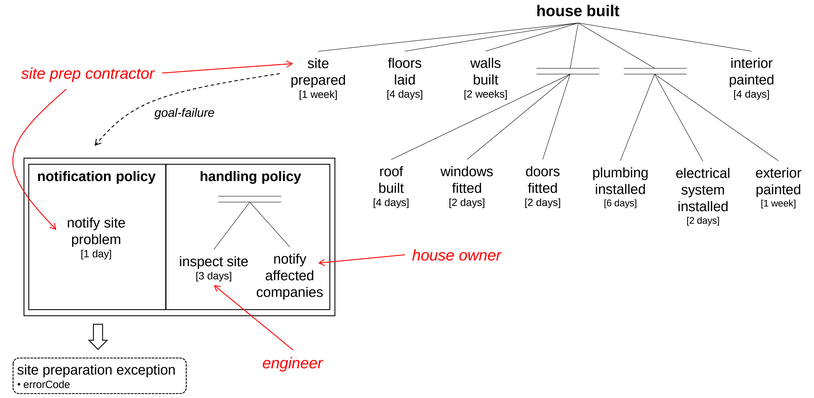
Exception Spec encodes the kind of information to be produced by the agent raising the exception

Handling Policy specifies a way in which the exception must be handled

Catching Goal captures the course of action to follow for handling the exception and possibly remediate

Agents are held to explicitly take responsibility for throwing and catching goals

Example: House Building




```
1 +obligation(Ag,_,done(_,site_prepared,Ag),_)
2     : .my_name(Ag)
3   <- !site_prepared;
4      goalAchieved(site_prepared).
5
6 +!site_prepared
7   <- prepareSite.
8
9 -!site_prepared
10  <- goalFailed(site_prepared);
11     .fail.
12
13 +obligation(Ag,_,done(_,notify_site_problem,Ag),_)
14    : .my_name(Ag) &
15      // the site is flooded
16  <- throwException(site_preparation_exception,
17                    [errorCode(flooding)]);
18     goalAchieved(notify_site_problem).
```

Code of the *site prep contractor* agent

```
1 +obligation(Ag,_,done(_,inspect_site,Ag),_)
2     : .my_name(Ag) &
3       exceptionArgument(_,site_preparation_exception,
4       errorCode(flooding))
5   <- performSiteAnalysis(Result);
6      fixFlooding(Result);
7      goalReleased(site_prepared);
8      goalAchieved(inspect_site).
9
10 +obligation(Ag,_,done(_,inspect_site,Ag),_)
11    : .my_name(Ag) &
12      exceptionArgument(_,site_preparation_exception,
13      errorCode(archaeologicalRemains))
14  <- delimitSite;
15     carefullyRemoveRemains;
16     resetGoal(site_prepared).
```

Code of the *engineer* agent

The source code of the extension together with some examples is available at:
http://di.unito.it/moiseexceptions

A video presentation of the demonstration is available at: http://di.unito.it/aamas2021demo