Asynchronous Operation Invocation in TuCSoN

In this brief "how-to", you will learn TuCSoN main API available to developers of MAS willing to exploit *asynchronous invocation mode* of TuCSoN coordination operations.

Assumptions are you are familiar with TuCSoN core API.

Suggested readings complementing the content of this "how-to" are:

- Mariani, S., Omicini, A., Sangiorgi, L.: "*Models of Autonomy and Coordination: Integrating Subjective & Objective Approaches in Agent Development Frameworks*". In Intelligent Distributed Computing VIII, Studies in Computational Intelligence 570, pages 69-79, 2015. Available here.

---

1. Asynchronous Operation Invocation in TuCSoN

1.1 API Overview

Coordination operations in TuCSoN may be invoked in two modes:

- **synchronous** - the most common invocation mode, enforces coordination by blocking the caller agent whenever the invoked operation gets suspended--e.g., until a `in` operation does not find a matching tuple, the caller agent is **suspended together** with the operation until successful completion or timeout
- **asynchronous** - an alternative invocation mode, enforcing coordination without hindering agent autonomy, by decoupling the agent control flow from the coordination operation control flow--e.g., if a `in` operation does not find a matching tuple, the caller agent is **NOT suspended together** with the operation (which is suspended, still); instead, it may undergo concurrent activities while being asynchronously notified upon successful completion or timeout

The asynchronous invocation mode is firstly supported by TuCSoN through dedicated ACCs, in package `alice.tucson.api.*`, providing methods with a `TucsonOperationCompletionListener` object as parameter--instead of the `Long` object

provided by ACCs supporting the synchronous mode. The `TucsonOperationCompletionListener` object exposes overridable "hook" methods to manage asynchronous operations completion notifications.

The asynchronous invocation mode is also supported through a novel dedicated TuCSoN component (actually, a TuCSoN agent): the **`AsynchOpsHelper`** in package **`alice.tucson.asynchSupport`**. Application agents may delegate asynchronous invocation of TuCSoN coordination operations to the `AsynchOpsHelper`, which performs invocation on their behalf (that is, using their IDs), which then keeps track of pending and completed operations on their behalf. This way, application agents may query the `AsynchOpsHelper` about the completion state of an operation whenever they want, in complete **autonomy**.

>>> `AsynchOpsHelper`. The API exposed by the `AsynchOpsHelper` consists of the following methods:

```
public final boolean enqueue(final AbstractTucsonAction action, final
TucsonOperationCompletionListener listener)
```

Adds an operation to the queue of **pending operations**--given one of the *shutdown* operations below haven't been called yet. `action` is an object wrapping the TuCSoN operation to execute (see below), `listener` is the listener object in charge of asynchronously handling each operation completion.

```
public final SearchableOpsQueue getPendingOps()
```

Gets the queue of pending operations. Such queue is a `SearchableOpsQueue` object (see below), wrapping a thread-safe queue storing pending TuCSoN operations and providing a `getMatchingOps` method to filter on operations type--e.g., to retrieve only `in` pending operations.

```
public final CompletedOpsQueue getCompletedOps()
```

Gets the queue of **completed operations**. Such queue is a `CompletedOpsQueue` object (see below), wrapping a thread-safe queue storing completed TuCSoN operations (both successful and failed ones) and providing a methods to filter on operations features (type, outcome)--e.g., to retrieve only `in` completed operations, only successful operations, only failed operations.

```
public final void shutdownGracefully()
```

Requests **soft shutdown** of the helper, that is, shutdown *waits* for pending operations to

complete.

```
public final void shutdownNow()
```

Requests **hard shutdown** of the helper, that is, shutdown happens as soon as the current operation in execution completes: *pending operations are discarded* instead.

>>> `SearchableOpsQueue`. Atm, a single filtering method is available to application agents to query the pending operations queue:

```
public SearchableOpsQueue getMatchingOps(final Class<? extends
AbstractTucsonAction> optype)
```

which gets all the operations whose *type matches the given type*, that is, whose Java class is a subclass of `AbstractTucsonAction` (see below). The method returns another `SearchableOpsQueue` to support incremental search refinements--in case new filtering criteria are added in the future.

Other public methods should rarely be used, since they are automatically called by `AsynchOpsHelper` when needed.

>>> `CompletedOpsQueue`. Atm, three filtering criteria are provided through dedicated methods to query the completed operations queue:

```
public CompletedOpsQueue getMatchingOps(final Class<? extends
AbstractTucsonAction> optype)
```

which is similar to the homonym method in `SearchableOpsQueue` class except for the return type, which is another `CompletedOpsQueue`, still to support incremental search refinements--e.g., first collect completed `in`, then filter only those successfully completed.

```
public CompletedOpsQueue getSuccessfulOps()
```

which gets all successfully completed operations.

```
public CompletedOpsQueue getFailedOps()
```

which gets all failed operations.

Other methods allow removal of successful/failed operations from the queue--see TuCSoN javadoc for further info [1].

>>> `AbstractTucsonAction`. The root class of the hierarchy of TuCSoN actions, split in

packages

- `alice.tucson.asynchSupport.actions.ordinary` for ordinary TuCSoN operations
- `alice.tucson.asynchSupport.actions.specification` for specification operations
- `alice.tucson.asynchSupport.actions.ordinary.bulk` for bulk operations
- `alice.tucson.asynchSupport.actions.ordinary.uniform` for uniform operations

Public methods, inherited by subclasses representing all the TuCSoN operations (e.g., `in`, `out_s`, `no_all`, `urd`), are automatically called by the `AsynchOpsHelper` to asynchronously execute delegated TuCSoN operations [2]. Constructors are provided depending on whether the TuCSoN action represents an *ordinary operation* or a *specification one*:

- for ordinary operations, the ID of the tuple centre target of the operation and its logic tuple argument should be specified--except for `Get` action (representing `get` TuCSoN operation), which only requires the tuple centre ID
- for specification operations, the ID of the tuple centre target of the operation, the logic tuple representing the triggering event, the one representing the guard predicates, and that representing the ReSpecT reaction body should be specified--except for `GetS` action (representing `get_s` TuCSoN operation), which only requires the tuple centre ID

>>> `TucsonOpWrapper`. Is the utility class wrapping together a TuCSoN action and the corresponding TuCSoN operation objects. `TucsonOpWrapper` objects are those tracked by both `SearchableOpsQueue` and `CompletedOpsQueue`, which are iterable queues. The following methods are provided:

```
public final AbstractTucsonAction getAction()
```

Gets the `AbstractTucsonAction` subclassing object wrapped.

```
public AbstractTupleCentreOperation getOp()
```

Gets the `AbstractTupleCentreOperation` object wrapped.

```
public boolean hasBeenRemoved()
```

Checks whether the wrapped action has been removed from the list of pending operations.

Others are intended for usage by the `AsynchOpsHelper`.

1.2 "Hands-on" step-by-step tour

TBD.

In the meanwhile, you can read through the comments in TuCSoN "asynchAPI" example source files, here.

Contact Information

Author of this "how-to":

- *Stefano Mariani*, DISI - Università di Bologna (s.mariani@unibo.it)

Authors of TuCSoN in TuCSoN "People" section of its main site, here > http://apice.unibo.it/xwiki/bin/view/TuCSoN/People.

---

[1] Available within TuCSoN distribution in folder `TuCSoN-${version}/docs/javadoc/`. Latest TuCSoN distribution is available from TuCSoN repository on Bitbucket.

[2] Actually, only method

```
public abstract ITucsonOperation executeAsynch(EnhancedAsynchACC acc,
TucsonOperationCompletionListener listener)
```

Method

```
public abstract ITucsonOperation executeSynch(EnhancedSynchACC acc, Long
timeout)
```

is intended for usage within TuCSoN4JADE extension.