*Abstract*

The third part agent(s) (a piece of software that represents an actual device) will be

written in dedicated class and will implement appropriate interfaces.

Also the .so (shared object) file representing the agents library of a particular vendor(s)

will contain a special class AgentsFactory (the name name vary) that will actually

create the agents by demand. But, above all, this file will contain a special

“C” style function , which name and signature must be preserved , since it’s

an entry point for the system and that function’s details are hardcoded.

*Description*

Lets look on the fig.1 for the UML class diagram of the Concrete Class Agent, that will be implemented, and all relevant interfaces (for implementation) and concrete classes (structs),

that are in relation of “association” or “dependency” with it.

On upper side are interfaces : “Runnable” , that enables the system to run it (in thread) and

“Stopable” , that enables the system to gently stop it.

The interface “Agent” represents how the the system will interact with the agent.

The interface “Sensor” for an agent willing to send events to the system adds such functionality.

Symmetrically, the interface “Controller” for an agent willing to receive events from the system adds such functionality.

Also every agent (independently of it’s nature) will receive upon creation references (pointers)

to the interface “Registrator” for registering to particular (types of) events and the interface

“EventsReceiver” which is an addressee for sending events, and a reference to an object of the type

“Configration” which contains all necessary data for creation and configuration of the agent (like ID, location, configuration sting etc.).

Upon it’s creation an agent (sensor) will generate example event with the fields “topic” and

“location” of the desired types of events . The agent will be registered to events according to

to the content of these fields. One or more of these fields( “topic” and the fields of “location”) can be empty. In such case the agent will be registered to a broader range of events’ types.

E.g. , when only the “topic” field is filled, the agent will be registered to all events of that topic.

If all mentioned above fields are empty, the agent will be registered to all events received by

the system (can be useful for loggers).

An agent that desires to send event to the system will invoke an appropriate function of the “EventsReceiver” object with the newly generated event.

The system makes use of smart pointers for handling the events , so every event sent and/or

received will be of such pointer type.

Also the following should be noticed : the system provides an interface only to treat

the events (the “Event” interface) . So each agent (or a group of agents) should be accompanied with an appropriated event type (class), which should implement the system provided interface.

Each .so file may contain one or more “Agent” implementing classes. But it also will contain

concrete implementation of the interface “ModelsFactory” (see fig.2) , designed to facilitate

the agents creation. The system will invoke the entry function, which will return a (an instance of)

ModelsFactory’s implementation object, so the system could create and destroy the agents by

employing the facilities of that class.