


Introduction

This case-study deals with the design and development of proactive/reactive software systems based on the concept of Actor, as introduced in [LabIss2021 | wshttp support with ActorBasicJava](#) observers.

Requirements

- Design and build a software system that allow the robot described in [VirtualRobot2021.html](#) to exhibit the following behaviour:
- the robot lives in a rectangular room, delimited by walls that includes one or more devices (e.g. sonar) able to detect the presence of obstacles, including the robot itself;
  - the robot has a **den** for refuge, located in the position shown in the picture
- 
- the robot works as an *explorer of the room*. Starting from its **den**, the goal of the robot is to create a map of the room that records the position of the fixed obstacles. The presence of mobile obstacles in the room is (at the moment) excluded;
  - since the robot is '*cautious*', it returns immediately to the **den** as soon as it finds an obstacle. It also stops for a while (e.g. 2 seconds), when it 'feels' that the sonar has detected its presence.

Delivery

The customer requires to receive at **12 noon on April 6** a file named

cognome\_nome\_cea.pdf

including a (synthetic) description of the project (preceded by a proper analysis of the problem) based on components of type [ActorBasicJava](#) and a reference to a *first working prototype* (even with limited capabilities) of the system.

Meeting

A SPRINT-review phase with the customer is planned (via Teams) at **5.15 pm on April 6**.

Requirement analysis

- In seguito alla nostra interazione con il cliente abbiamo ottenuto informazioni precise riguardo alle seguenti componenti:
- robot**: dispositivo in grado di muoversi nella stanza, tutte le specifiche date dal cliente sono presenti nel file [VirtualRobot2021.html](#);
  - move**: sempre con riferimento al documento [VirtualRobot2021.html](#) il robot è in grado di muoversi in avanti e indietro, oltre a poter ruotare di novanta gradi a destra e a sinistra;
  - sonar**: dispositivo in grado di rilevare la presenza del **robot**, non abbiamo informazioni sulla posizione ne sul numero esatto di sonar presenti;
  - den**: angolo nord-ovest della stanza, come intuibile anche dall'immagine fornita dal cliente.utilizzato dal **robot** come punto di partenza per ogni sua esplorazione e come punto di ritorno quando incontra un ostacolo;
  - obstacle**: elemento della stanza diverso dalle pareti, scopo del **robot** è quello di trovare questi elementi nella stanza. al momento della richiesta l’ostacolo è considerato immobile;
  - robot map**: insieme di dati di cui non è stata specificata una struttura ma che devono essere in grado di rappresentare la posizione degli ostacoli nell'ambiente.

User Story 1(no rilevamento sonar)

il committente prevede di porre il **robot** nella **den** all'interno della stanza rettangolare. attraverso il software, deve essere possibile inviare dei comandi al **robot** in modo tale che esso proceda all'esplorazione della stanza. In seguito alla collisione con un ostacolo, il robot dovrà tornare alla sua **den**. Eseguendo esplorazioni multiple, il robot sarà in grado di popolare la mappa dell'ambiente contenente la posizione degli ostacoli incontrati.

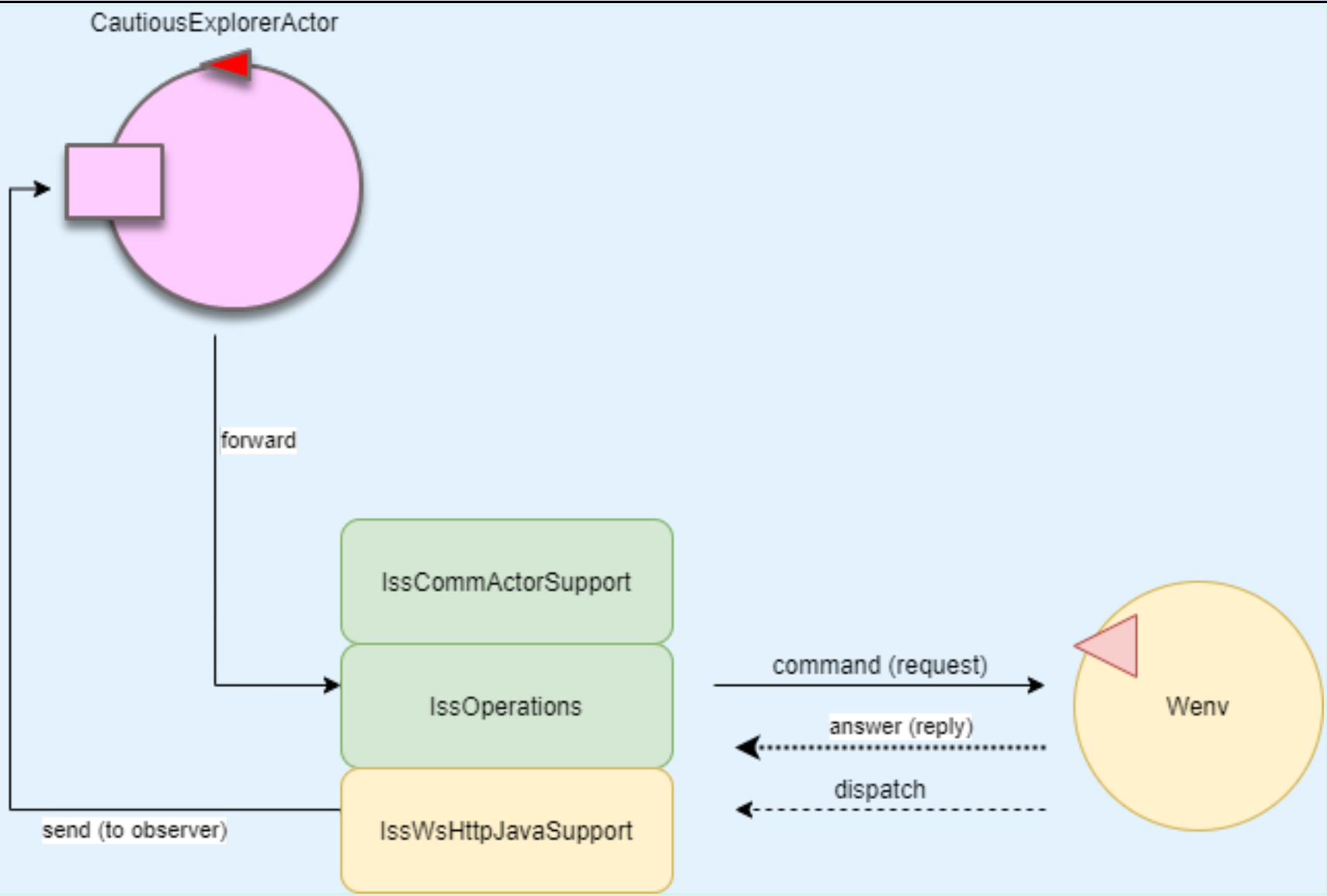
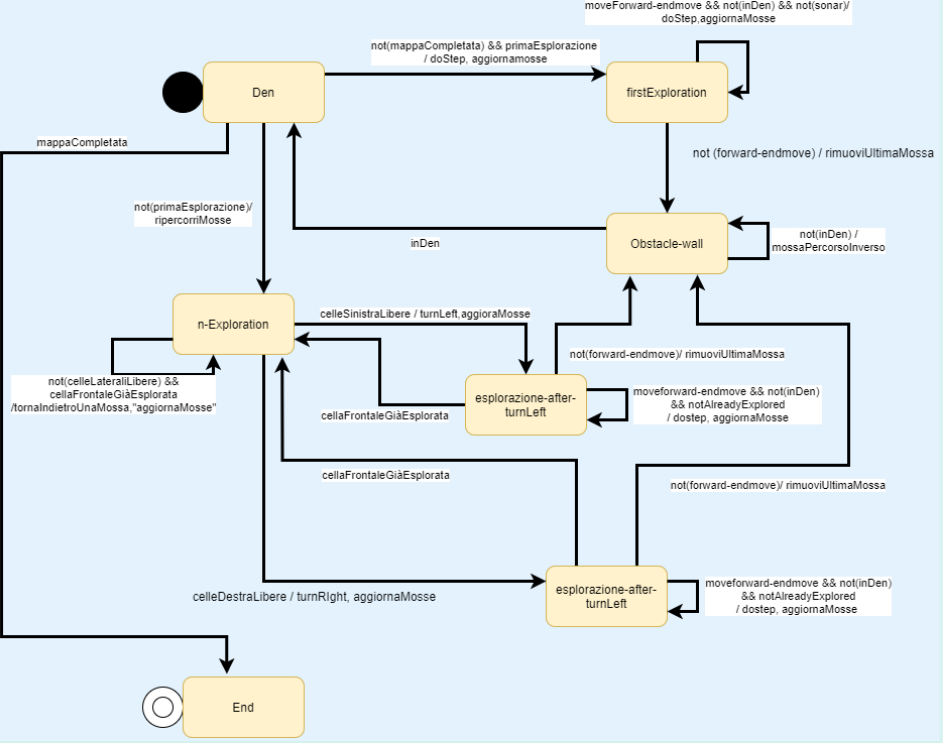
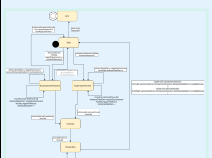
User Story 2(si rilevamento sonar)

il committente prevede di porre il **robot** nella **den** all'interno della stanza rettangolare. attraverso il software, deve essere possibile inviare dei comandi al **robot** in modo tale che esso proceda all'esplorazione della stanza. durante l'esplorazione nel caso in cui il robot venga rilevato dal sonar presente nell'ambiente, questo si dovrà interrompere l'esplorazione per 2 secondi. In seguito alla collisione con un ostacolo, il robot dovrà tornare alla sua **den**. Eseguendo esplorazioni multiple, il robot sarà in grado di popolare la mappa dell'ambiente contenente la posizione degli ostacoli incontrati.

Problem analysis

- come riportato nel file [VirtualRobot2021.html](#) il **robot** è in grado di ricevere comandi tramite in due modalità:
- comandi inviati tramite protocollo HTTP usando il metodo POST, presso la porta **8090**
  - comandi inviati utilizzando il costruito websocket utilizzando la porta **8091**
- Facendo riferimento all' introduzione del progetto, si nota che lo scambio di comandi deve essere asincrono, si decide quindi di utilizzare lo scambio di messaggi tramite web-socket attivo sulla porta **8091**. Nonostante il gap di astrazione presente a livello concettuale, l'interazione asincrona sarà poi implementata utilizzando appositi supporti che sono stati già creati per progetti precedenti.
- Per quanto riguarda il linguaggio da utilizzare per comunicare con il **robot**, con riferimento al documento [VirtualRobot2021](#) viene scelto come linguaggio di comunicazione con il **robot** il linguaggio **arill** (abstract robot interaction language) poichè previsto di una temporizzazione tale da poter muover il **robot** in avanti e indietro di un'unità di spazio pari alla sua lunghezza. Viene scelto questo linguaggio a discapito del linguaggio **crill** poichè si interfaccia meglio al requisito di costruire una mappa dell'ambiente esplorato dal robot.
- La **robot map** per la rappresentazione dell'ambiente viene generata sfruttando [mapUtil.kt](#) precedentemente utilizzata per la rappresentazione di altri ambienti rettangolari.
- il sistema dovrà provvedere sia un comportamento di tipo **proattivo**, ovvero l'esplorazione della stanza, sia un comportamento **reattivo**,richiesto dal requisito relativo al sonar
- i due comportamneti saranno realizzati sfruttando il concetto di attori introdotto nell seguente documento [AcotrWithBAsicJava](#)
- non viene rilasciata nessuna specifica dall'utente su come il **robot** debba effettuare l'esplorazione. Da questo fatto comunque si decide che l'esplorazione non avverrà in modo casuale poichè non permetterebbe di effettuare dei test coerenti
- il requisito di interrompere l'esplorazione se il **robot** viene individuato dal **sonar** prevederà non di interrompere la mossa corrente, ma verrà invece bloccato per 2 secondi al termine della suddetta.
- è ora possibile definire cosa debba dare il **robot** al fine di rispettare i requisiti: finchè non è stata completata l'esplorazione della stanza e rilevati tutti gli ostacoli presenti nell'ambiente:
- il **robot** parte dalla sua **den** esplorando in maniera efficiente l'ambiente circostante
  - ad ogni mossa il **robot** aggiorna la mappa dell'ambiente
  - quando rilevato dal **sonar** il **robot** interrompe l'esplorazione per 2 secondi
  - quando incontra un ostacolo il **robot** ritorna alla sua **den**

Architettura logica

<div>Per la rappresentazione della struttura logica si farà utilizzo alcune classi utilizzate in progetti precedenti. Questo permette di ottenre un prototipo funzionante dell'applicazione entro la data di revisione SPRING. le classi in questione sono:</div> <ul style="list-style-type: none"><li><a href="#">IssCommActorSupport</a></li><li><a href="#">IssOperations</a></li><li><a href="#">IssWsHttpJavaSupport</a></li><li><a href="#">ActorBasicJava</a></li></ul> <div>le classi in questione fungono come supporto per la comunicazione con l'ambiente di esecuzione Wenv e per dare un supporto all'utilizzo degli attori per il linguaggio di programmazione Java scelto per il software.</div>	<div></div>	
<div>in base ai requisiti e all'analisi del problema, si decide di utilizzare il concetto di attore Java che è stato introdotto in un progetto precedente (<a href="#">wssupportAsActorJAVA</a>) poichè permette una gestione ottimale dei comportamenti attivi e reattivi del <b>robot</b> viene riportata a fianco una prima rappresentazione dell'architettura citata</div>	<div><div>Prima Vesrione</div></div>	<div><div>Seconda Versione</div></div>

Test plans

Con lo scopo di verificare che il **robot** soddisfi i requisiti si dovrà replicare il seguente testplan, facendo riferimento alle user Story che sono state apporvate:

Project

Testing

Deployment

Maintenance