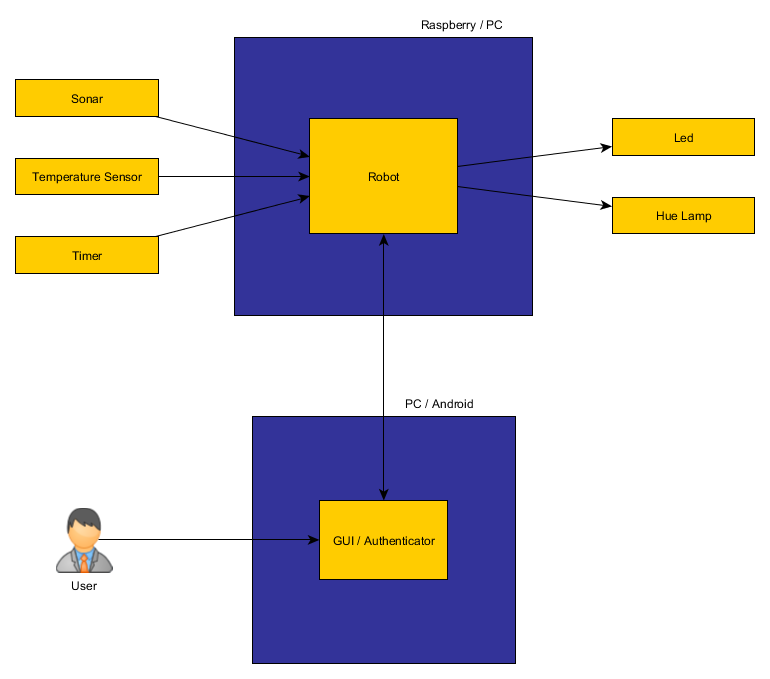
**Analisi dei Requisiti**

Il sistema da modellare sarà, come esplicitato dai requisiti, eterogeneo e distribuito, in particolare composto da almeno due nodi: il nodo “Robot” e il nodo “PC/Android”.

Per la modellazione si utilizza il linguaggio QActor in quanto adatto alla modellazione di sistemi distribuiti.



Prima di formalizzare il contenuto del diagramma, ci soffermiamo sul significato di ciascun elemento.

Il primo dei due nodi che abbiamo modellato è il nodo “PC / Android” che si occupa di mostrare la GUI e interagire direttamente con un utente umano, richiedendone l’autenticazione. Come da requisito **R-Start** l’interfaccia utente deve poter essere utilizzabile sia su PC che su un dispositivo Android, tuttavia, essendo le funzioni che essa deve svolgere identiche in entrambi i casi, abbiamo rappresentato entrambi i nodi PC e Android come uno unico. Su questo nodo esegue l’attore “GUI/Authenticator”, che consente all’utente di autenticarsi e inviare i comandi di “Start” e “Stop” al robot (**R-Start** e **R-Stop**).

Il secondo nodo “Raspberry / PC” controlla il robot, esso può essere in esecuzione su un PC, nel caso del robot virtuale, oppure su un Raspberry Pi nel caso del robot reale. L’attore “Robot” si pone in attesa dei comandi inviati da “GUI/Authenticator” e riceve informazioni sull’ambiente esterno da un sensore di temperatura e un timer (**R-TempOk**, **R-TimeOk**, **R-TempKo**, **R-TimeKo**). Durante l’esecuzione, in caso di movimento, l’attore “Robot” invia a “Led” e a “Hue Lamp” comandi per l’accensione e lo spegnimento (**R-BlinkLed**, **R-BlinkHue**).

L’attore “Robot” si occupa inoltre di gestire la logica applicativa, ovvero, in seguito alla ricezione del comando “Start” da parte dell’utente, prendere decisioni circa il movimento del robot all’interno della stanza – per il robot reale – e all’interno dell’ambiente simulato – per il robot virtuale, tentando di evitare gli ostacoli fissi e mobili (**R-AvoidFix**, **R-AvoidMobile**), costruendo una mappa (**R-Map**) dell’ambiente.

Se l’attore “Robot” trova un ostacolo che non riesce ad evitare si deve fermare (**R-Obstacle**).

Questa situazione si verifica quando il robot trova uno o più ostacoli che gli impediscono di avanzare in una direzione diversa da quella da cui è arrivato.

\* Snippet analisi dei requisiti \*

**Analisi Problema**

Il primo problema che sorge è quello di stabilire quale nodo si occuperà dell’autenticazione dell’utente. Una possibilità è che sia sul nodo del robot: in questo caso il robot potrebbe non disporre delle adeguate risorse computazionali per gestire il processo di autenticazione, tuttavia questo garantirebbe maggiore sicurezza. Un’altra possibilità è che l’autenticazione sia su un nodo diverso rispetto a quello del robot: ciò consente di non utilizzare le risorse computazionali del robot richiedendo però maggiori accortezze sulla sicurezza. In quest’ultimo caso l’autenticazione potrebbe essere gestita dal nodo dell’utente oppure da un nodo distinto, il quale comporterebbe costi maggiori.

Un altro problema è quello dell’interfaccia GUI, che deve poter eseguire su dispositivi eterogenei. A tal proposito, una possibilità sarebbe creare client nativi per ogni piattaforma con costi elevati oppure più semplicemente utilizzare una pagina web.

La comunicazione tra utente e robot tramite GUI può avvenire via messaggi o via eventi. La comunicazione ad eventi permette di disaccoppiare GUI e robot, consentendo di utilizzare un’unica GUI per comunicare con diversi robot. Utilizzando gli eventi può essere adottato un approccio event-based o un approccio event-driven. Nell’approccio event-based il robot non sarebbe sempre sensibile agli eventi, potendone perdere alcuni. Al contrario, nell’approccio event-driven il robot sarebbe sempre sensibile agli eventi perdendo tuttavia reattività.