Project Report Template

Enrico Benini, Nicola Casadei, and Marco Benedetti

Alma Mater Studiorum – University of Bologna
via Venezia 52, 47023 Cesena, Italy
{name1.surename1, name2.surename2, name3.surename3}@mail.com

Table of Contents

1 Introduzione

Questo é il template di progetto del corso di smart city dell'universitá di Bologna. Di seguito sará consultabile tutto il processo di analisi del progetto: modelli, problemi riscontrati e soluzioni adottate, interazione con l'ambiente, sensori utilizzati e il loro collegamento ...

Per qualsiasi dubbio in merito fare riferimento agli autori.

2 Visione

La visione che guida questo progetto consiste nel raggiungere rapidamente l'ideale di cittá intelligente, quindi con un ambiente aumentato, capace di prendere decisioni e agire tempestivamente per far fronte a casi specifici e capace di comunicare direttamente con chi si trova immerso in esso, per facilitare la vita di tutti i giorni.

In particolare vogliamo prepararci ad imparare, modellare e costruire sistemi che si integreranno in questo contesto, visto l'andamento stesso del mercato che sta sempre piú rendendo disponibili risorse di elaborazione e sensoristica a minor prezzo.

3 Obbiettivi

Lo scopo del progetto é quello di implementare concretamente un'applicazione di domotica. Affrontando quindi tutte le problematiche ad essa annesse e fornire una possibile soluzione a queste. Ci auguriamo che questa possa essere di spunto per applicazioni simili e che possa quindi favorirne lo sviluppo.

Sfruttando questo progetto, vogliamo esplorare e apprendere la teoria e i concetti affrontati nel corso di smart city. Quindi tutti gli aspetti riguardanti la gestione di sensori e input provenienti dall'ambiente esterno. Uscendo dalla tipica zona di confort dei sistemi software.

4 Requisiti

Si vuole monitorare lo stato ambientale di una stanza. In particolare si vogliono monitorare lo stato di: luce, temperatura e movimento, mantenendo la possibilitá di aggiungere altre tipologie di sensori.

Il sistema dovrá dare all'utente la possibilitá di inserire, attraverso un'interfaccia web, per ogni valore misurato, un'apposito range che indichi i valori ammessi all'interno della stanza in modo che, in caso uno dei valori misurati non risulti conforme alle specifiche, venga indicata una notifica di allarme sull'interfaccia stessa. Questo con l'idea di simulare la possibilitá di eseguire delle azioni collegate all'allarme (ad esempio, accensione delle luci o del riscaldamento)

L'utente potrá inoltre visualizzare all'interno del sito i valori misurati in tempo reale e il valore dei vari sensori nel tempo, potendone quindi consultare la storia.

4.1 Requisiti Non Funzionali

Aggiungiamo nei requsiti non funzionali tutte le proprietá che il sistema deve avere e che non sono state ufficialmente formalizzate.

- Logging: Possibilitá di controllare il fluire delle informazioni e delle operazioni svolte attraverso una qualsivoglia forma di logging
- Separazione dei compiti e indipendenza tra le parti: garantire i principi SOLID dove é possibile evitare che un qualsiasi componente sia strettamente legato ad un'altro.

5 Acquisto Hardware

Sfortunatamente il primo problema incontrato in un progetto come il seguente e' stata la necessitá di acquistare la parte hardware del sistema che si andrá costruire. Di consequenza si e' messo in atto un processo di ricerca dei sensori, cavi e quant'altro per riuscire a soddisfare i requisiti

5.1 Dispositivi di Computazione

Prima di tutto necessitiamo di un dispositivo in grado di computare i dati emessi dai vari sensori e che sia interamente programmabile. Nel corso abbiamo visto due possibilita' che hanno avuto molto successo recentemente:

– Arduino
– Raspberry Pi

Abbiamo scelto la seconda opzione data la maggior familiarita' con il dispositivo e dal momento in cui risulta piu' facile il riutilizzo dello stesso una volta terminato questo progetto.

Costo del dispositivo: 44,50 €

5.2 Sensori

Un'altra cosa fondamentale riguarda i sensori necessari per catturare i parametri richiesti. Abbiamo Quindi scelto i sequenti sensori

Parametri Ambientali	**Sensori Costo**
Temperatura	
Luce	
Movimento	

Table 1. Sensor Table

5.3 Hardware Aggiuntivo

Hardware	**Costo**
Breadboard	
Wires	
Resistors	

Table 2. Addictional Hardware

6 Analisi dei Requisiti

6.1 Casi D'Uso

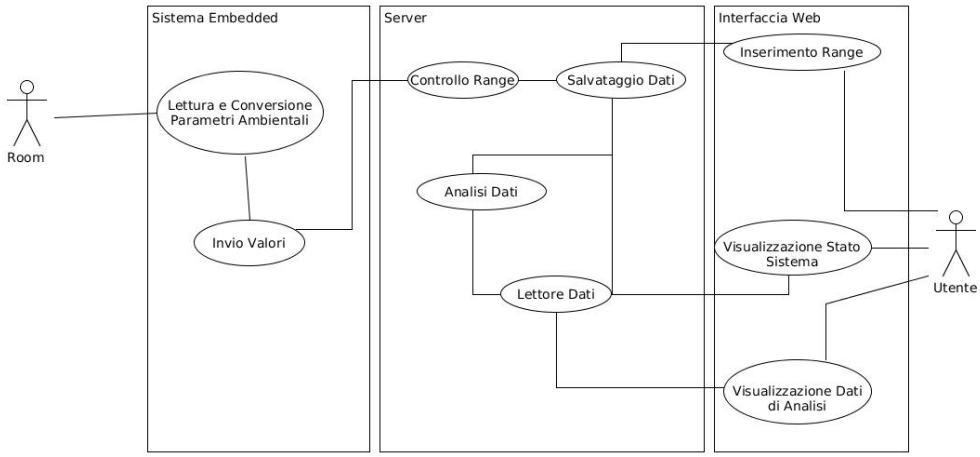

Fig. 1. Casi d'Uso

Nell'immagine sopra si possono vedere le macro operazioni principali effettuate dal sistema e le interazioni con l'esterno. In particolare gli attori che interagiscono con il sistema saranno:

- La stanza: con questo attore si intendono i vari parametri che si possono rilevare attraverso i sensori e che quindi saranno di input per il sistema.
- Utente: con questo attore rappresenta l'utente che puó interagire con il sistema.

Il sistema é stato volutamente suddiviso in tre parti distinte con l'idea di seguire un modello MVC dove la parte di modello non viene aggiornata solamente attraverso l'input inserito dall'utente, ma anche e soprattutto dall'input dei sensori. L'organizzazione hardware ha fortemente influito su questa suddivisione.

É inoltre possibile visualizzare le macro operazioni effettuate e modellate dal sistema. Si vedano gli scenari di seguito per avere una piú dettagliata

visualizzazione dell'interazione tra le varie parti che lo schema sovrastante vuole rappresentare.

6.2 Scenario

In questa sezione verranno illustrati le principali modalitá di utilizzo del sistema. Gli scenari elencati di seguito riguardano l'utente e di conseguenza si prevede l'accesso da parte di questo all'interfaccia di input.

Si prevede che il sistema sia opportunamente configurato e settato a livello hardware, senza errori durante la fase di start up.

6.2.1 Inserimento Range

1. Attraverso un apposito menú l'utente é in grado di accedere alla funzionalitá di settaggio dei range associati ai parametri ambientali.
2. Il server conosce giá i sensori collegati al raspberry. Appena questi inviano qualche dato l'utente é in grado di visualizzare i controlli relativi ad ogni tipologia di sensore attualmente connesso. Conseguentemente l'utente é in grado di modificare tali intervalli.
3. Al termine della modifica degli intervalli l'utente dovrá confermare le modifiche attraverso un'apposito pulsante.
4. Il sistema mostra un messaggio di conferma o di errore.

6.2.2 Visualizzazione Stato Realtime

All'accesso del sistema l'utente visualizza lo stato realtime dei valori dei sensori ed eventuali notifiche:

- Se i valori vanno oltre gli intervalli correnti.
- Sullo stato dei sensori, se sono o meno attivi al momento.

In questa modalitá l'utente non puó effettuare alcuna operazione.

6.2.3 Visualizzazione Dati

Attraverso un apposito menú l'utente é in grado di accedere alla visualizzazione dei dati storici dei vari sensori.

6.3 Modello del Dominio

In questa sezione vogliamo cercare di modellare le entitá inserite all'interno dei requisiti senza fare riferimento alla parti tecnologiche e hardware. In questo modo siamo in grado di decidere le interfaccie con cui vogliamo lavorare, costruendo la parte software, aumentando il disaccoppiamento con la parte fisica, aumentando anche la possibilitá di utilizzare volendo lo stesso software su diverse configurazioni. In questo progetto, ad ogni modo, si utilizzerá solamente la configurazione descritta in questo report.

Suddivisione del Sistema: visto che é emerso giá dai casi d'uso la separazione del sistema in varie parti, ci é sembrato giusto iniziare a modellare dividendolo fin da subito in modo da:

- semplificare il processo
- suddividere il lavoro di implementazione successivamente tra i membri del gruppo.

In particolare le parti individuate sono tre:

- Sistema Embedded: che nel nostro caso si occuperá di catturare, convertire e inviare i dati alle altre parti
- Server: Sará la parte che riceve i dati e si occupa di effettuare le varie elaborazioni come ad esempio, il salvataggio dei dati, il calcolo delle statistiche e il controllo dei range. Infine questa parte dovrá rendere disponibili i risultati alla parte successiva.
- Web site: parte che, prendendo i risultati dalle parti precedenti, li mostra all'utente rispettando i casi d'uso predecenti.

Chiaramente in questa fase tutto viene semplificato e ridotto a poche entitá. Questo peró non significa che, ogni entitá presente negli schemi sottostanti, non si riveli essere poi a sua volta un sottosistema piú complesso.

Lo scopo di questa fase é appunto quella riflettere, in un primo modello, i requisiti. Successivamente, iniziare a sviluppare il progetto cercando di mantenere coerenza nelle interfaccie principali che indicano le interazioni piú importanti.

6.3.1 Sistema Embedded

Per quanto riguarda il sistema embedded é necessario prima effettuare delle indagini su come interagire e comunicare con i sensori in modo da modellare adeguatamente il tutto e quindi assicurarsi che in seguito sará semplice riuscire a integrare il tutto con la tecnologia che avremo intenzione di utilizzare. Questa é una piccola eccezione che é necessario fare a questo livello. Tuttavia si fa riferimento a un paradigma piú che ad una tecnologia specifica.

Il modello di interrogazione dei sensori é a polling di consequenza il nostro modello dovrá riflettere questa modalitá di interazione. Sfortunatamente questo implica che a livello di modello giá **un'entitá attiva** che si occupa di reperire i valori visto, che in una modalitá a polling devo esplicitamente chiedere ai sensori i valori.

Struttura

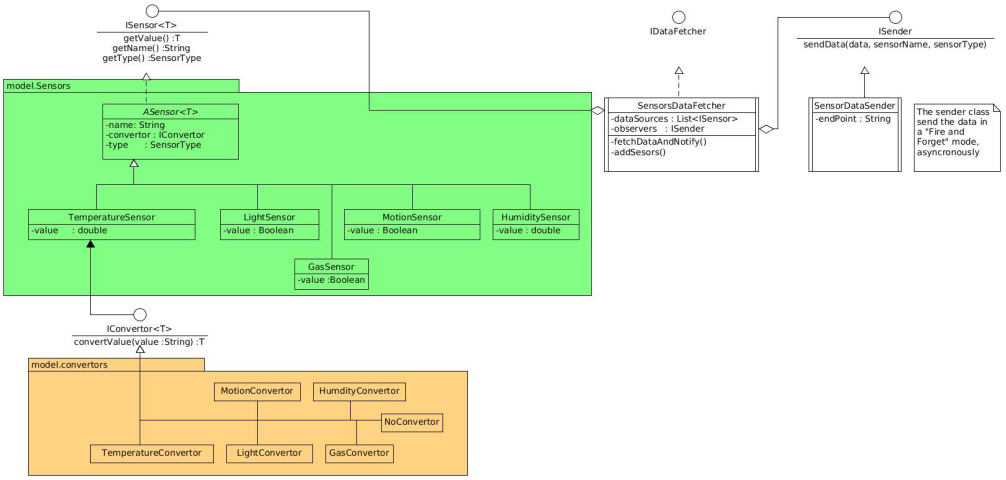

Fig. 2. Sistema Embedded, Struttura

Dalla struttura é possibile individuare subito le entitá principali presenti nei requisiti. In particolare la presenza di una specifica gerarchia per i sensori che condividono la stessa interfaccia che consente agilmente di ottenere il

valore corrente del sensore. Sono stati inseriti solamente i sensori citati nei requisiti, ma si puó facilmente intuire come qualsiasi tipo di sensore sia facilmente modellabile secondo questa struttura.

Si noti inoltre come viene anche inserita l'entitá IConvertor che si occoperá di convertire il valore di uno specifico sensore in un'unitá piú consona per la sua gestione. Chiaramente questo viene affrontato fin da questo livello perché si immagina l'operazione di conversione come un'operazione quasi istantanea e necessaria.

Infine sono presenti le entitá che si occupano, attivamente, di interrogare i sensori ogni intervallo di tempo predefinito e quindi di inviarli altrove. In questo caso é stato tutto ridotto ad un singolo endpoint, anche se poi possono essere facilmente piú di uno. Come si puó visionare nel commento, l'invio dei dati avviene con una metodologia di tipo *fire and forget*, quindi non avvengono reinvii dei dati e vengono ignorati eventuali errori. Chiaramente tutto questo é dovuto all'idea che i cicli di invio siano abbastanza brevi da potersi permettere eventuali perdite.

Interazione

Prima di iniziare si vuole evidenziare come viene riportato solamente un diagramma di interazione perché é presente solamente un'entitá attiva. Tuttavia nel futuro potrebbero esserci piú task e potranno essere necessari piú diagrammi dell'interazione.

Nello schema di interazione vengono evidenziate le varie fasi del Sistema, in particolare la fase iniziale di setup, dove, conoscendo quali sensori sono presenti, questi vengono aggiunti nella memoria dell'entitá principale in modo che in seguito siano facilmente interrogabili.

Terminata la fase di *Init* inizia il loop infinito che aspetta inizialmente un piccolo lasso di tempo per poi interrogare iterativamente tutti i sensori che sono stati aggiunti precedentemente, raccogliendo i dati e interrogando asincronamente l'entitá di invio, per poi ricominciare il ciclo stesso.

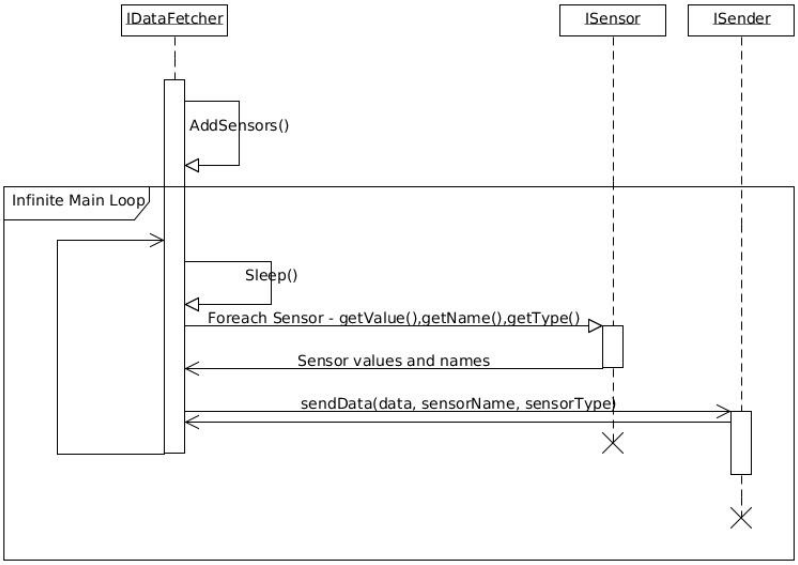

Fig. 3. Sistema Embedded, Interazione

Comportamento

Nel diagramma del comportamento viene semplicemente riportato quanto é stato precedentemente discusso attraverso una state machine.

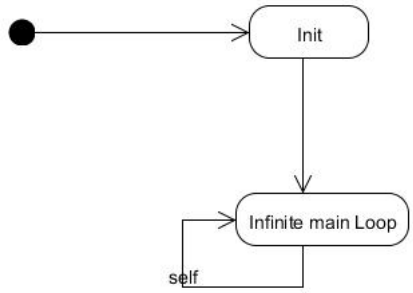

Fig. 4. Sistema Embedded, Comportamento

6.3.2 Server

La parte server é quella piú importante di tutto il sistema in quanto é quella che concentra tutta la logica applicativa del sistema stesso.

Struttura

Le entitá principali di questo schema sono:

- IPersistentStore, che si occuperá di salvare opportunamente i dati provenienti dai sensori e dall'utente
- IDataReceiver, che sará sempre in ascolto per ogni messaggio proveniente dal sistema embedded e quindi notificherá opportunamente il sistema ad ogni nuovo arrivo
- IPresentator che si occuperá di ottenere i dati necessari per le viste da mostrare all'utente nel momento in cui una nuova richiesta verrá inoltrata.

Chiaramente ognuna di queste entitá é in parte citata nei casi d'uso.

Si vedano i diagrammi dell'interazione per i dettagli di come avviene la comunicazione di tutte queste entitá al fronte di garantire il funzionamento e il soddisfacimento dei requisiti.

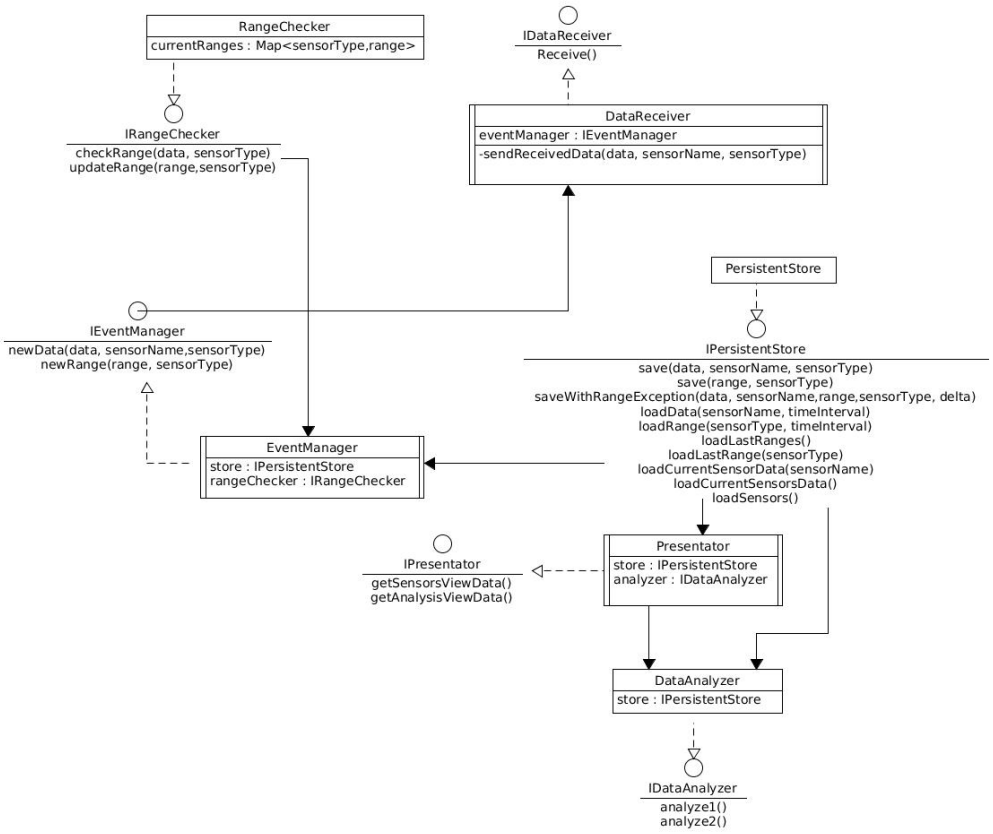

Fig. 5. Server, Struttura

Interazione

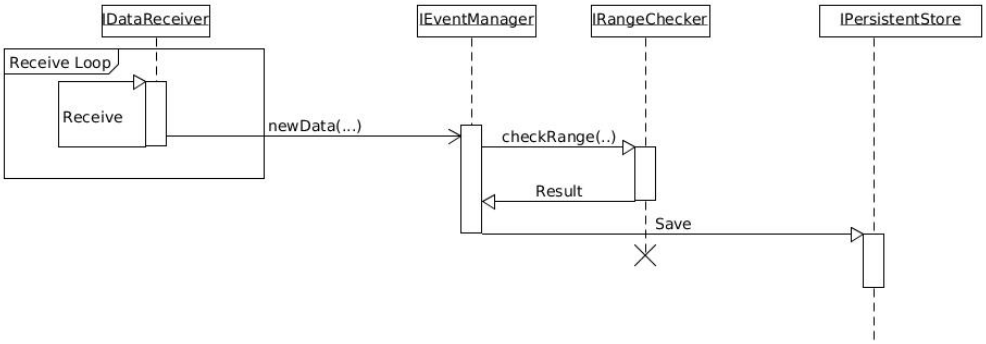

Fig. 6. Server, Interazione, Dati dai Sensori

Abbiamo deciso di omettere la parte di inizializzazione del sistema in questa fase dal momento che risulterá meglio definita nell'analisi del problema, dovendo aggiungere entitá che non sono direttamente correlate con i requisiti principali ma con requisiti non funzionali.

Nello schema sovrastante si puó osservare l'interazione nel caso in cui i sensori emettano un nuovo valore. Si vuole porre particolare enfasi su come l'entitá che riceve i dati rimanga sempre in ascolto di nuovi arrivi delegando, tramite una chiamata asincrona alle altre entitá, il compito di salvare i dati appena ricevuti.

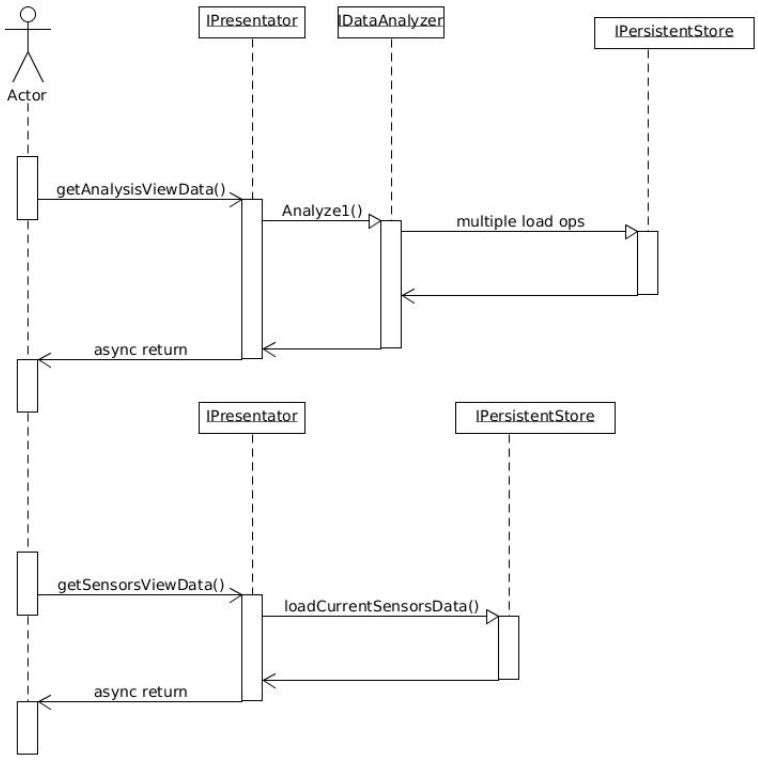

Fig. 7. Server, Interazione, Richiesta Visualizzazione Dati

In quest'altro schema dell'interazione si puó osservare che entitá vengono coinvolte a fronte di una chiamata di visualizzazione dati da parte dell'utente. Anche in questo caso la chiamata iniziale é stata effettuata in maniera asincrona in modo da lasciare il sistema libero di rispondere ad eventuali altre richieste per poterle gestire in mniera concorrente.

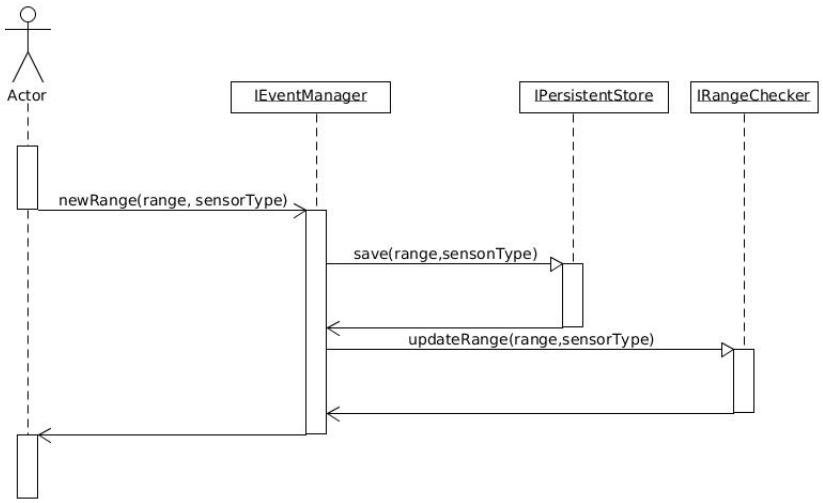

Fig. 8. Server, Interazione, Nuovo Range

In questo caso invece viene illustata l'interazione per quanto riguarda la richiesta da parte dell'utente dell'inserimento di un nuovo range. Si nota come la chiamata, questa volta, arrivi all'entitá che gestisce gli eventi essendo questo effettivamente un caso incui avviene un cambiamento nei dati e quindi tutto va gestito appropriatamente per evitare i conflitti.

Comportamento

In questa fase si mostrano le entitá che sono effettivamente attive e che quindi operano cambiamenti sul sistema. Si é deciso di omettere tutte le entitá che vengono solamente chiamate ed effettuano una unica operazione al fine di evitare di formalizzare troppe cose a questo livello.

Si vuole far notare inoltre come tutte queste entitá abbiano effettivamente uno stato di attesa di un qualche messaggio o evento. Questo aspetto puó essere molto importante in seguito nelle scelte per la progettazione.

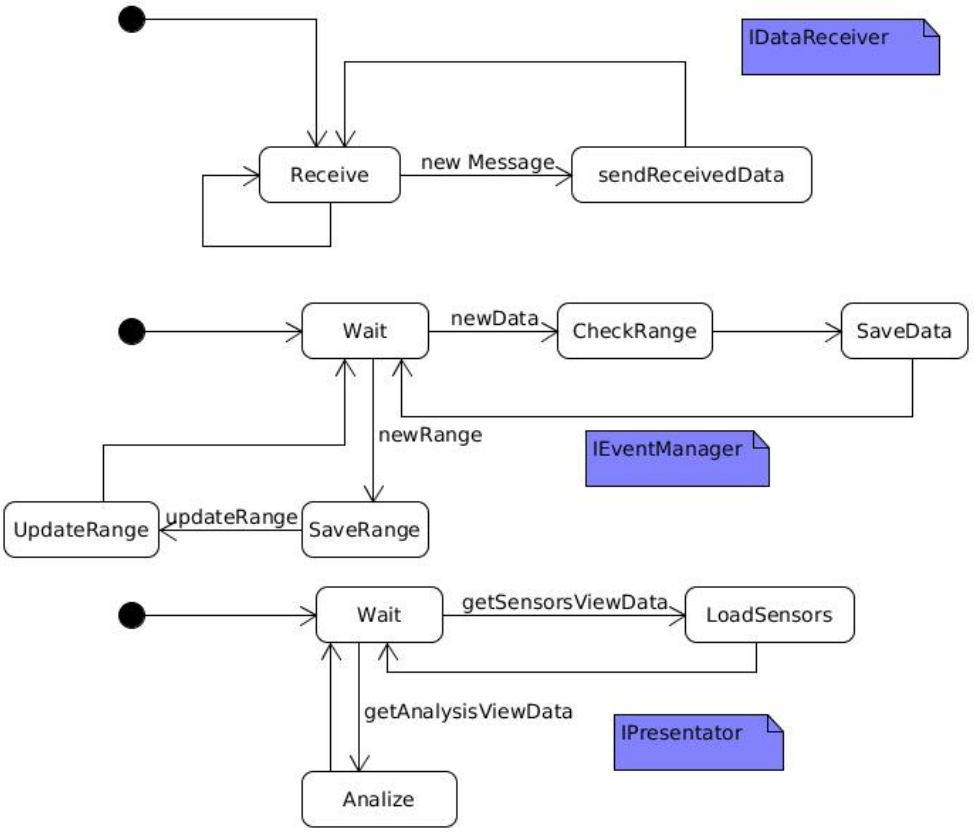

Fig. 9. Server, Comportamento

6.3.3 Web Site

Dopo un'attenta analisi abbiamo ritenuto superflua la necessitá di modellare e analizzare la parte di visualizzazione in quanto non é effettivamente presente nessuna business logic ma solamente la parte che effettua le richieste di dati alle parti precedenti visuzalizzando tutto su schermo.

La documentazione relativa a questa parte si limiterá solamente a come utilizzare il software attraverso l'interfaccia.

Possiamo pensare di definire i link a cui l'interfaccia dovrá rispondere appropriatamente attraverso una pagina web. Questo ci consente di impostare

comunque dei test che esulano dal contenuto della pagina, garantendoci peró che questa sia effettivamente online in maniera automatica.

gli URL scelti sono:

- *http://...../DomoticRoom/Status*
- *http://...../DomoticRoom/NewRange*
- *http://...../DomoticRoom/Analysis*

7 Analisi del Problema

Assunzioni Prima di iniziare l'analisi del problema abbiamo ritenuto necessario effettuare delle assuzioni riguardanti al paradigma che ci consentirebbero di effettuare un prodotto di qualitá migliore e con meno sforzi.

Il paradigma di programmazione di riferimento é il *reactive programming* perché la concezione di flusso di dati é proprio quello che ci interessa modellare in quanto anche nel nostro sistema sará presente un flusso di dati dal client al server. Per la comunicazione attraverso la rete questo ci consente di sfruttare chiamate asincrore aumentando il disaccoppiamento tra client e server.

Per questo abbiamo deciso di utilizzare per i dati un database NoSQL per via dell'estrema dinamicitá, in quanto ci consente di aggiungere dei campi anche in seguito e un miglioramento di performance nell'accesso a dati che normalmente richiederebbero dei join.

Per ulteriori informazioni piú dettagliate sul paradigma si rimanda ad un'altra sede.

7.1 Architettura Logica

Per la costruzione dell'architettura logica ci si baserá ovviamente sulla precedente analisi dei requisiti in modo da mantenere il contatto con questi e non rischiare di uscire fuori specifica. Anche in questo caso si andrá separare l'analisi in due parti:

- Sistema Embedded
- Server e Website

La parte del website é stata incorporata direttamente nella parte server proprio per la sua assenza di business logic.

Modellazione Reactive Prima di iniziare a costruire la modellazione sulla base del paradigma appena citato é necessario capire come fare a modellare lo stream stesso all'interno della nostra architettura logica, conseguentemente si é deciso di utilizzare i *marable diagrams*. Questi sono una rappresentazione di come il flusso di dati nel tempo avviene e quali tipologie di trasformazioni consentono di effettuare.

Con questi schemi ci viene concesso ancora una volta di concentrarci prima sul "cosa" e non sul "come" che invece viene delegata alla parte progettuale, dove si cercherá effettivamnete di implementare il risultato finale dell'analisi

7.1.1 Sistema Embedded

Flussi Dati

Alla luce del paradigma di programmazione che si é deciso di adottare sono stati effettuati dei cambiamenti anche alla struttura che é stata esposta precedentemente proprio perché ci sono state fornite da questa assunzione nuovi livelli di astrazione, primo tra tutti il concetto di **Stream/Flusso**. Quindi in questa fase abbiamo ritenuto molto utile iniziare a visualizzare effettivamente questi flussi tramite un'apposito diagramma.

In particolare si puo' notare come in questo diagramma il compito di convertire i valori di un sensore sia stato disaccoppiato dal sensore stesso che quindi si deve solamente occupare di invitare i dati all'interno del flusso. É stato inoltre aggiunto anche una nuova entitá *packager* che si occuperá di formattare i valori secondo il protocollo che deve essere definito tra client e server per l'apposita comunicazione. In particolare ogni dato proveniente da un sensore avrá un suo formato per via: della differenziazione dei dati, della tempistica di rilevazione che puó anche risultare diversa e dell'idea percui la comunicazione in ogni caso debba essere asincrona.

Sulla base di questa ultima frase si vuole sottolineare che ogni sensore quindi avrá un suo flusso asincrono, ma che in quest'immagine si é deciso di condensare in un'unica immagine per comoditá.

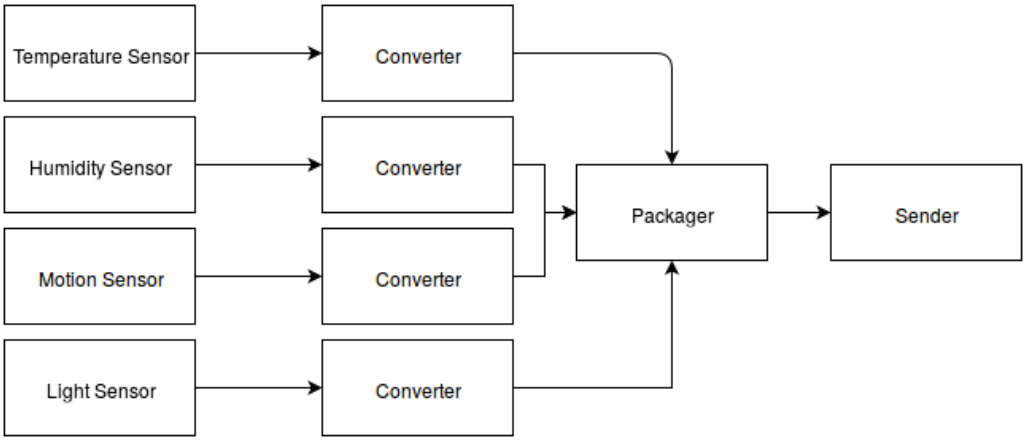

Fig. 10. Diagramma di flusso, per sensore, dei dati

Di seguito si indica un primo marable diagram che mostra le varie trasformazioni che verranno applicate. Questo serve soprattutto per iniziare a capire poi come interpretare eventuali altri schemi come questo.

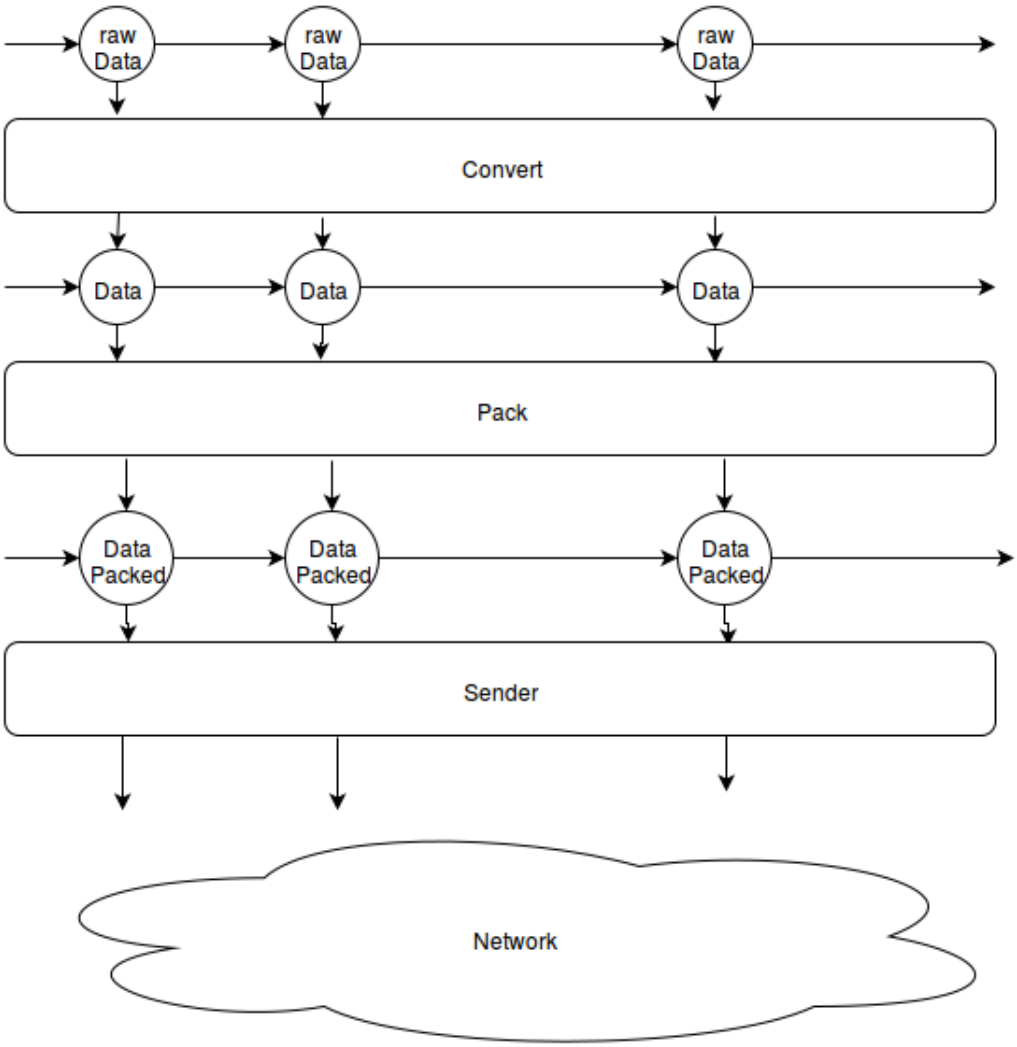

Fig. 11. Marable diagram dei singoli dati proveniente da una sorgente

Struttura

Chiaramente la struttura, che si basa sul precedente step di analisi dei requisiti, risulta cambiata anche alla luce delle assunzioni effettuate e quindi si possono notare l'introduzione di alcune entitá che servono per la gestione del sistema embedded e per impostare i flussi di dati provenienti dai sensori.

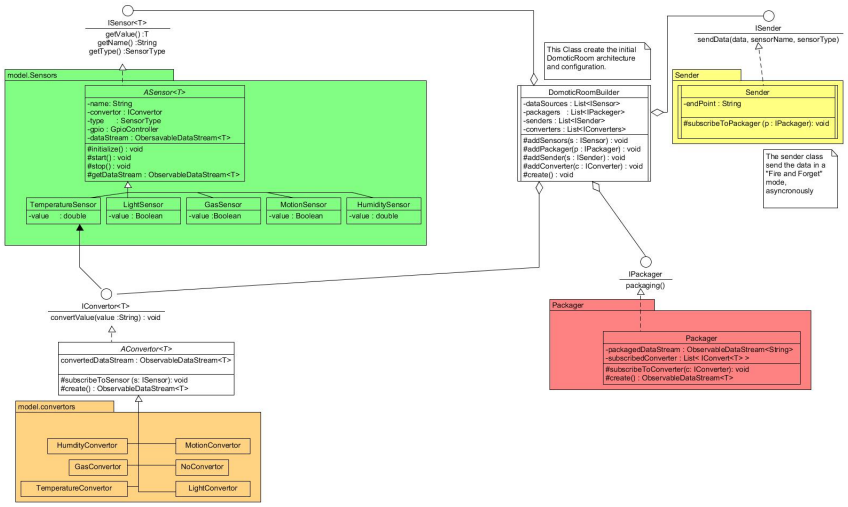

Fig. 12. Logic Architecture, Structure

Come si mostra nello schema soprastante, la struttura logica non ne esce particolarmente modificata rispetto all'analisi iniziale.
Il *SensorDataFetcher* è stato eliminato, poichè l'implementazione di polling per ricevere dati dai sensori viene sostituita dal pattern Observable, utilizzato per sfruttare tutte le potenzialità del paradigma Reactive.
La nuova entitá *DomoticRoomBuilder* ha il ruolo centrale di inizializzare il sistema, creando l'architettura e la configurazione iniziale dei suoi componenti.
L'entitá *Sensor* ha il ruolo centrale di ricevere i dati dai sensori e inviarli su uno stream dati osservabile. L'entitá *Convertor* si occupa di convertire i dati che riceve dalla sorgente **Sensor** a cui è sottoscritto; inoltre genera un nuovo Stream dove invia i dati convertiti che riceve dai *Sensor* a cui è sottoscritto.

L'entitá *Packager* unisce tutti i flussi in entrata, dai *Converter* in un unico flusso, il suo compito è preparare i dati ad essere inviati al Server.
L'entitá *Sender* si occupa di inviare i dati che riceve dal Packager sulla rete, come pensato nella struttura iniziale.

Interazione

La parte di interazione viene drasticamente modificata proprio perché l'assunzione del paradigma reactive risulta principalmente improntato su questo, in particolare é possibile condensare in meno codice, e piú dichiarativo. Vengono in ogni caso mostrati le varie operazioni effettuate sullo stream per mantenere il contatto con quanto mostrato attraverso i marable diagrams. Particolare attenzione va posta poi sulla parte che converte da *procedure calls* a *reactive streams*.
Il sistema non ha una interazione continua; come invece suggerisce il paradigma, esso "reagisce" al verificarsi di un evento di particolare interesse.
In questo caso, il sistema reagisce autonomamente alla ricezione di un valore inviato dal sensore. L'entitá *Sensor* si occupa di incanalare queste letture periodiche in un flusso potenzialmente infinito di valori.
L'entitá *Convertor* é a sua volta interessata all'arrivo di un nuovo valore sul flusso creato dal *Sensor*; il Convertor applicherá ad ogni nuovo dato una funzione atta a convertire in valore in un dato di maggiore utilità per il sistema.
L'entitá *Packager* sarà invece di ascolto sui flussi generati da ogni *Convertor*, appena un nuovo valore convertito è inviato sul flusso; a questo punto il suo compito é "impacchettare" il dato, secondo un protocollo di trasmissione per poterlo inviare al Server.
Quest'ultima parte é gestita dal *Sender* che appena un dato impacchettato é pronto lo invierá al Server.

Comportamento

7.1.2 Server

Interazione e Flussi Dati

Anche per quanto riguarda la parte server é necessario definire gli steam che saranno presenti all'interno del sistema. Inquanto questa parte é notevolmente piú corposa rispetto a quella dell'embedded system per varie ragioni,

prima tra tutte la capacitá di calcolo, saranno necessari piú stream, in particolare si é pensato di creare uno stream per ogni funzionalitá sulla base delle operazioni che quindi sono da compiere.

Il primo flusso che andremo ad analizzare riguarda la ricezione dei dati da parte del sistema embedded. In particolare si vuole cercare di riutilizzare le entitá che si sono introdotte nell'analisi dei requisiti proprio perché queste sono maggiormente collegate al dominio. Si puø' inoltre notare altre entitá chiamate *IRawDataFormatter e IDBDataFormatter*, queste entitá sono utili per preparare la corretta formattazione dei dati provenienti dal flusso e dalla successiva elaborazione per essere salvati poi sul database. Si é deciso in particolare di inserirlo per andare a separare i compiti il piú possibile e per facilitare la modifica nel caso si voglia adottare uno standard dei dati interno rispetto a quello del client in modo da effettuare una netta indipendenza tra server e client. Soprattutto perché in un'eventuale futuro si puó facilmente immaginare un'eterogeneitá dei dati dovuti ai vari client disponibili e dalla loro provenienza (altre aziende, protocolli proprietari . . .)

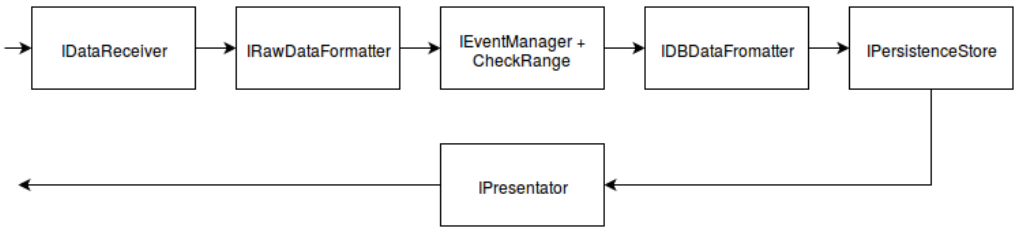

Fig. 13. Server, Flusso Dati Ricezione

Il secondo flusso che si vuole introdurre invece riguarda l'aggiornamento del range da parte dell'utente, in particolare anche in questo caso l'IEventManager viene chiamato in causa il quale si occuperá di notificare il cambiamento sia a livello di database che al componente incaricato di controllare il range. Questa parte non subisce particolari cambiamenti rispetto a quella dell'analisi dei requisiti perché rimane molto coerente anche a fronte del cambio di paradigma. Un cambiamento significativo riguarda il fatto che il cambiamento del range non avviene piú direttamente dall'event manager ma dal database, questo per enfatizzare che il cambiamento del range non é effettivo se non viene prima registrato in maniera permanente.

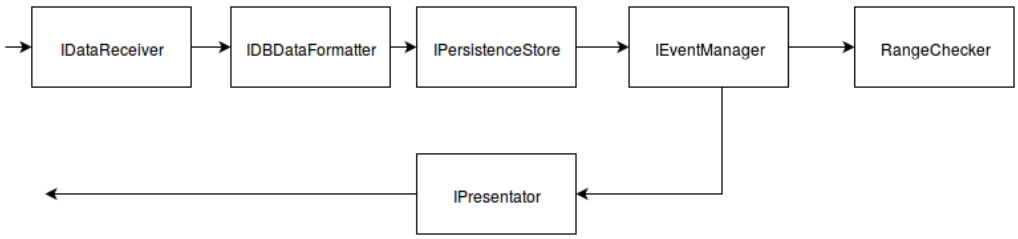

Fig. 14. Server, Flusso Dati Nuovo Range

L'ultimo flusso di dati che verrá trattato riguardano le richieste di dati stessi per la visualizzazione da parte dell'utente. Probabilmente per quanto riguarda le seguenti operazioni si poteva anche pensare di gestire il tutto attraverso delle semplici chiamate a procedura, ma, per mantenere una certa coerenza, per sfruttare la dichiaratividad del paradigma reactive e per evitare di incorrere nel problema noto come *Callback Hell* si é scelto di utilizzare comunque un flusso. Un'ulteriore vantaggio di gestire attraverso l'infrastruttura reactive consiste nella possibilitá di aggiornare realtime l'interfaccia ogni qualvolta avviene l'inserimento di un nuovo valore. Concludendo si vuole far notare la presenza di un'entitá che entra in campo quando viene richiesta un'analisi sui dati e non semplicemente una sua visualizzazione. La forma a nuvola indica proprio la possibilitá che questo oggetto entri a far parte o meno nel flusso.

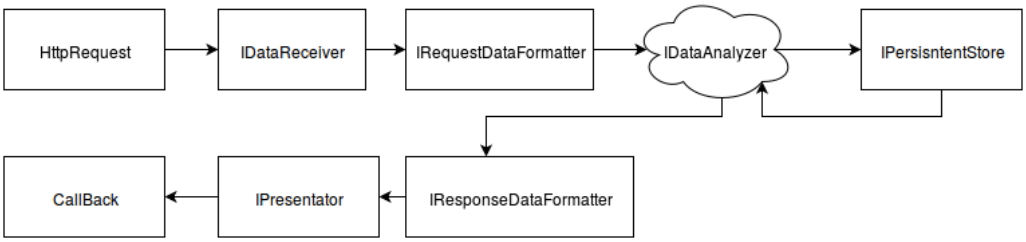

Fig. 15. Server, Flusso Dati Richieste Dati

Prima di passare alla struttura é secondo noi utile andare a definire l'interazione del configurator in quanto é l'unica entitá attiva che non si basa effettivamente sui flussi e che contiene l'entry point di tutto il server.

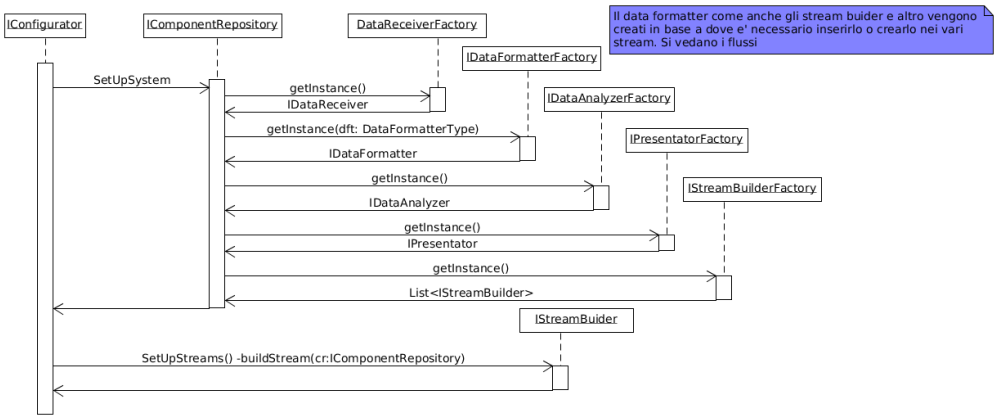

Fig. 16. Server, IConfigurator Interaction

Struttura

Nella struttura dell'analisi del problema abbiamo varie differenze, anche a livello di interfaccia, dovute al cambio di paradigma o ad altre motivazioni, di seguito verranno elencate per rendendere tutto piú chiaro:

- IDataReceiver: non esiste piú il metodo *Receive* perché non si ipotizza piú un'approccio a polling e quindi non é necessaria questa funzionalitá, mentre invece viene esposto un metodo che da la possibilitá di ottenere lo stream di dati associato all'arrivo di un nuovo valore dall'esterno.
- Tutto ció che in precedenza, nel diagramma dell'interazione, era modellato attraverso una chiamata a procedura e veniva effettuata all'arrivo di un nuovo dato, ora viene convertito in funzioni che prendono in ingresso un dato stream e lo modificano restituendo un nuovo stream all'inizio del sistema in modo che, in questa fase, si é in grado di costruire e comporre i vari flussi che poi verranno elaborati attraverso l'infrastruttura.
- Per separare ancora di piú i vari flussi di stream indicati precedentemente e per separare i compiti di creazione dei vari stream si sono ideate delle classi *factories* che assembleranno i vari componenti a tale scopo.
- Sono stati impostati come oggetti sigleton i componenti *EventManager e PersistenceStore*
- É stato ampliato la parte del persistenceStore in modo da differenziare i vari compiti di salvataggio e caricamento attraverso diversi oggetti.

– Il DataReceiver é stato differenziato tra dati provenienti dai sensori e dati provenienti dall'utente, come le richieste di un nuovo range o la richiesta di dati. Questo ha di fatto portato all'eliminazione dell'oggetto presentator individuato precedentemente.

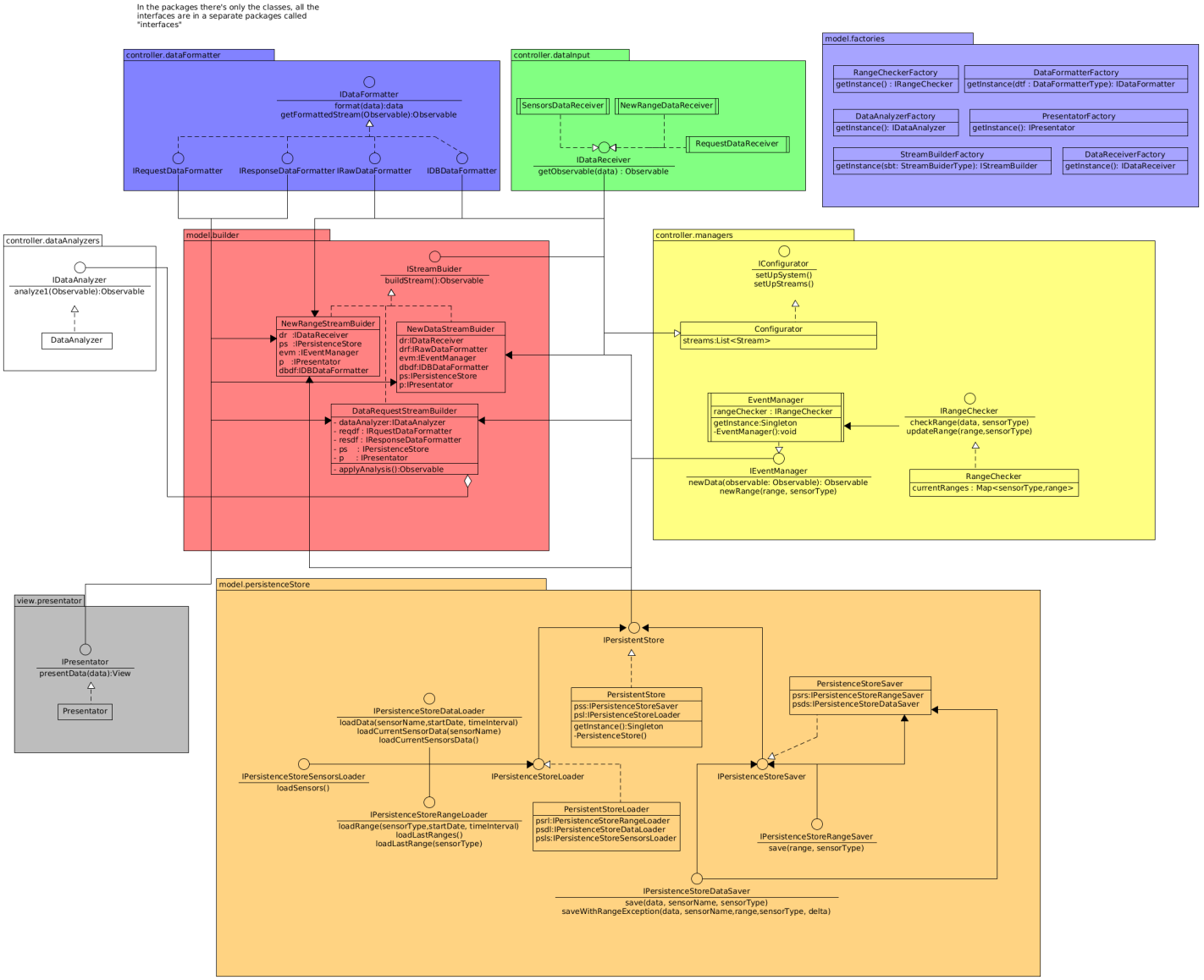

Fig. 17. Server, Struttura Architettura Logica

Comportamento

Per quanto riguarda il comportamento delle varie parti del sistema verranno illustrate quelle principali e descritti gli stati principali incui questi componenti verranno a trovarsi. In particolare verranno omessi tutti quei componenti che effettivamente si occupano solamente della trasformazione degli stream in quanto non posseggono effettivamente uno stato, ma consistono in degli algoritmi che, dato un flusso in ingresso, gli applicano delle apposite trasformazioni e restituiscono un flusso in uscita.

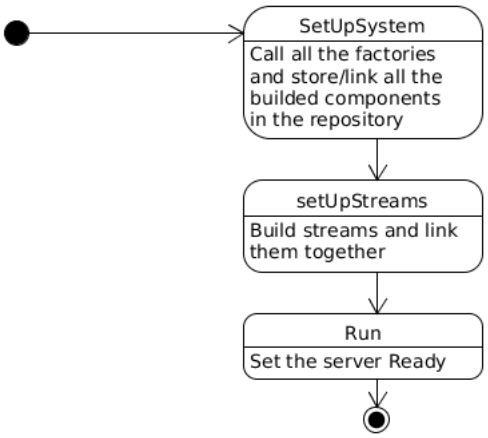

Fig. 18. Server, IConfigurator Behavior

Come si puó vedere dalla figura sovrastante il componente IConfigurator sará quello che si occuperá di attivare tutto il sistema server occupandosi di rendere tutto operativo e pronto per le richieste da soddisfare. Questo avviene appunto in fasi, incui prima viene settato il tutto attraverso una fase di creazione di tutti i componenti necessari, poi avviene la creazione dei flussi che si occuperanno del flow dei dati ed infine si attiveranno questi flussi in modo che siano attivi.

Il comportamento del persistence store rimane molto semplice: una volta creato, attende la richiesta da parte del sistema per la memorizzazione dei dati. Quando una richiesta avviene allora si attiva una delle parti che si occupano di leggere o scrivere all'interno del persistence store e effettuano

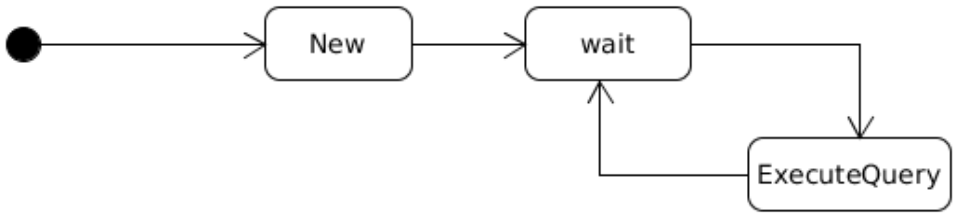

Fig. 19. Server, IPersistenceStore Behavior

l'operazione, per poi tornare in attesa di ulteriori operazioni. Si aggiunge inoltre che questo componente come l'eventManager sono componenti singleton, e quindi é presente solamente un'istanza di questo in tutto il sistema.

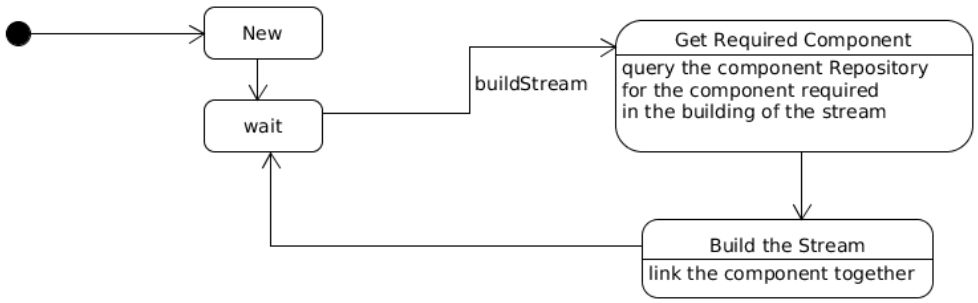

Fig. 20. Server, IStreamBuilder Behavior

Lo stream builder anch'esso risulta molto semplice perché viene chiamato da altri componenti, nel suo caso dal configurator stesso, e per questo presenta due stati come il caso precedente, *New e Wait*. All'arrivo di una richiesta il componente richiede i vari blocchi necessari per la costruzione dello stream direttamente dal componentRepository e, una volta ottenuti li collega assieme per costruire il flusso che poi restituirá al configurator stesso.

7.2 Gap di Astrazione

In questa sezione aggiungeremo tutti i tipi di astrazioni richiesti per affrontare il progetto e che non sono direttamente fruibili attraverso la tecnologia di riferimento.

1. Web Server: Data la necessitá di comunicare attraverso la rete é necessario che si utilizzi un paradigma a message-passing o attraverso chiamate asincrone, soprattutto per la comunicazione che avviene tra il raspberry e il server.
2. Continuous Integration, Testing and collaborative source control: Lavorando in gruppo sullo stesso repository é necessario impostare il lavoro affinché sia possibile effettuare modifiche in maniera indipendente gli uni dagli altri e allo stesso modo sia possibile controllare automaticamente che i test predisposti e le modifiche effettuate siano coerenti con le specifiche e che il building del progetto sia in ogni caso garantito.
3. Paradigmi Eterogenei: si é deciso di utilizzare dei paradigmi diversi dal OOP classico e questo puó portare a problematiche di utilizzo per via dell'inesperienza. Tuttavia queste non vengono fornite direttamente dal linguaggio e quindi si necessitano soluzioni al problema.

7.3 Analisi dei Rischi

In questa sezione verranno elencati i rischi che si potranno incontrare in un progetto di questo tipo fonrmalizzandoli fin da subito, prima di eseguire l'effettiva realizzazione dello stesso, in modo da poterli affontare e discutere preventivamente per essere pronti nel caso questi si verifichino.

- Aumento dei Costi: é necessario porre particolare attenzione alla struttura hardware del sistema e di come i singoli componenti vanno ad interconnettersi assieme per evitare che si debbano affrontare dei costi aggiuntivi, a progetto giá avviato, causati da una qualche mancanza o per via di un'estensione del progetto in corso d'opera.
- Quantitativo della memoria: l'adozione di un database NoSQL porta con se un certo livello di ridondanza e quindi questo puó portare ad un aumento di utilizzo della memoria e di spazio. Il tutto va valutato accuratamente durante il progetto, magari attraverso una serie di prove in base a come é stutturato il dato inizialmente. Se il rischio quindi é reale e quanto é critico.

– Integrazione tra le tecnologie: un possibile rischio riguarda la difficoltá nell'integrare tutte le tecnologie che devono essere sfruttate nel progetto per riuscire a colmare l'abstraction gap e quindi riuscire a completare il progetto attraverso l'analisi appena completata.

8 Work Plan

Il piano di lavoro che abbiamo intenzione di attuare si divide in varie fasi:

1. Controllo del Funzionamento Hardware: la prima cosa che é necessario fare é quella di controllare che tutta la sensoristica sia effettivamente compatibile e che la parte sistemistica sia assemblabile e funzionante.
2. Ricerca e Analisi Problematiche: Viste le problematiche sollevate precedentemente nell'abstraction gap é necessario soffermarsi prima di partire con il progetto per andare ad individuare i tools che possono coprire queste mancanze e quindi avere un'impatto significativo sulla realizzazione del progetto stesso. Queste possono cambiare drasticamente anche la realizzazione del progetto stesso che peró deve rimanere fedele all'analisi del problema.
3. Modello del Progetto: costruzione del modello del progetto alla luce delle considerazioni precedenti
4. Impostazione dell'Environment: setup di tutti i tools precedenti e controllo del loro corretto funzionamento
5. Suddivisione dei Compiti: quando tutto é formalizzato adeguatamente é possibile suddividere i vari compiti e quindi parallelizzare il lavoro
6. Cicli di Feedback: é molto utile ai fini della realizzazione del progetto, organizzare dei meeting periodici al fine di effettuare un check sullo stato dei lavori e quindi controllare eventuali incongruenze, discutere i problemi, rivedere i modelli precedenti e altro
7. Integrazione: Integrare tutti i progetti assieme per costruire tutto il sistema assicurandosi che sia corretta l'interazione tra le parti
8. Testing: La parte di testing deve essere sviluppata assieme al codice stesso se possibile sulla base del modello in modo da arrivare alle fasi finali da poter automaticamente capire cosa funziona e cosa no.
9. Deploy: Per la parte di deploy non si porrá particolare attenzione in questa prima versione dal momemento in cui non é effettivamente richiesta una particolare complessitá nel deploy su piú macchine. Comunque

si tratta di un'aspetto che puó essere affrontato anche a progetto finito nel quale si potrebbe procedere ad apportare le modifiche richieste per realizzare un deploy piú articolato.

8.1 Strumenti e Framework

In questa sottosezione vengono illustrati i tools utilizzati durante questo progetto utili a cercare di colmare l'abstraction gap e per il supporto alla gestione del progetto stesso.

- Umlet: Questo tool é stato utilizzato per creare gli schemi UML che realizzano i modelli di tutto il progetto.[**?**]
- Play Framework and Akka: framework creato da Typesafe che consente di gestire un web server con REST API e di gestire autonomamente le chiamate in maniera asincrona. Akka é tutta l'infrastruttura sottostante che consente di gestire tutto questo. Per ulteriori informazioni é sufficente cercare nel web.[**?**,**?**]
- Activator Template: tool associato al framework precedente che consente di gestire le proprie applicazioni secondo degli standard affermati e delle configurazioni di defalut in modo da velocizzare lo startup di tutto il progetto.[**?**]
- Scala Build Tool: anche se é stato pensato per progetti in scala questo strumento funziona correttamnete anche per java e consente di gestire tutte le dipendenze, eventuali librerie aggiuntive e configurazioni. Molti dei progetti precedenti vengono inseriti all'interno del sistema attraverso questo.[**?**]
- pi4j: Libreria utilizzata per la gestione della comunicazione tra la parte hardware e software, consente di interagire con il dispositivo embedded, in particolare il raspberry, e di ottenere i valori dalla sensoristica ad un livello di astrazione piú alto. [**?**]
- Bootstrap: Si tratta di una libreria grafica per gestire il frontend e renderlo piú piacevole.. . .
- MongoDB: Al fine di coprire alcuni temi del corso si é deciso di adottare un database NoSQL, in particolare perché questo, attraverso una rappresentazione a documenti e del tutto simile al formato JSON, fortemente utilizzato in ambito web, risulta molto adatto al problema. Inoltre ci consente di memorizzare i dati in maniera dinamica in modo che, se in un futuro l'applicazione dovesse ingrandirsi o se devono essere fatte

delle aggiunte/modifiche, queste possano essere fatte in agilmente. Infine,l'ultimo vantaggio consiste nel memorizzare i dati esattamente come possono essere utili al sistema, evitando il prezzo delle join relazionali. [?]
- reactiveMongo: Per l'accesso al database si é deciso di utilizzare una libreria che meglio incarna l'idea di stream effettuando appunto accessi completamente asincroni e quindi rendendo il tutto non bloccante, in piena idea reattiva.[?]
- Flot.js: Libreria javascript per la rappresentazione dei dati sul web. [?]
- New Relic: ...
- Quickcheck: ...
- rxJava: ... [?]

9 Project

9.1 Database

Per la gestione del database si é scelto di utilizzare un database NoSql, nell'accezione MongoDB[?]. In questa sede non si discuteranno i vantaggi, gli svantaggi o le particolaritá di questa tecnologia in quanto é stato fatto giá ampiamente durante il corso. Di seguito si riportano gli schemi dei documenti che sono stati salvati, le collezioni e il significato di alcuni dei valori inseriti.

Collection Ranges

In questa collezione di documenti vengono raggruppati tutti i documenti relativi ai ranges che sono da controllare. In particolare sono stati individuati due tipologie di range in base alla tipologia di sensori.

- Si/No: Per quella tipologia di sensori che non forniscono effettivamente dei valori ma che notificano solamente la presenza o l'assenza di un particolare elemento come il GAS o il movimento.
- Valore: validi quando un sensore effettivamente serve una misurazione di una grandezza, come ad esempio la temperatura. In questo caso é utile sapere se questa grandezza rimane all'interno di determinati range.

```
{ "_id" : ObjectId("569ba0a4c1832917d823d809"), "value" : 0, "type" : 1, "dateCr
eated" : ISODate("2016-01-17T14:09:40.684Z") }
{ "_id" : ObjectId("569ba0abc1832917d823d80a"), "value" : 0, "type" : 2, "dateCr
eated" : ISODate("2016-01-17T14:09:47.312Z") }
```


Fig. 21. Documento dei Ranges Booleani


```
{ "_id" : ObjectId("569b96f009dfabc5d2aa96fe"), "minBound" : 0, "maxBound" : 100
, "type" : 5, "dateCreated" : ISODate("2016-01-17T13:28:16.572Z") }
{ "_id" : ObjectId("569b9f30c1832917d823d808"), "minBound" : 0, "maxBound" : 100
, "type" : 4, "dateCreated" : ISODate("2016-01-17T14:03:28.805Z") }
```


Fig. 22. Documento dei Ranges di Valori

Si vuole far notare come sia comunque sempre presente un attributo di tipo data in modo da poter filtrare i range in base al tempo di inserimento e la presenza di un valore numerico che in questo caso rappresenta il tipo di range.

Per indicare il tipo di range si riporta il listato scala che indica tale tipologia.

```
class RangeType

case class TemperatureRangeType() extends RangeType
case class GasRangeType()         extends RangeType
case class MovementRangeType()    extends RangeType
case class LightRangeType()       extends RangeType
case class HumidityRangeType()    extends RangeType

object RangeTypeUtil{
  def Int2RangeType(rt:Int): RangeType = rt match {
    case 1 => new GasRangeType
    case 2 => new LightRangeType
    case 3 => new MovementRangeType
    case 4 => new TemperatureRangeType
    case 5 => new HumidityRangeType
  }
```

Fig. 23. Valore numerico dei Ranges

Collertion Sensors

Per quanto riguarda la collezione dei sensori si utilizza un'apposita collection. Questa non sará modificata spesso, e serve principalmente per eventuali evoluzioni del progetto, in modo che sia giá presente. Anch'essi sono dotati di di tipo come i range e, per questa implementazione, i tipi coincidono.

A livello di progetto peró si é deciso di mantenerli separati nel caso in un futuro questi divergano. Inseriamo di seguito un'immagine che rappresenta come é strutturato un documento di un sensore.

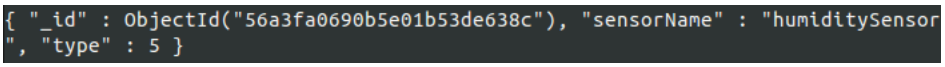

Fig. 24. Rappresentazione a DB dei Sensori

Collection Data

Per quanto riguarda la rappresentazione dei dati raccolti, anche in questo caso sono presenti 2 tipologie di dati:

- Dati Corretti: cioé quella tipologia di dati che rientrano all'interno del range prestabilito per la loro tipologia.
- Dati Incorretti: quelli che invece non rientrano nella tipologia corretta e quindi vanno a violare il range attivo.

Come si puó osservare il primo tipo é piú semplice e ingloba i precedenti. Di seguito come fatto in precedenza si indica un esempio di struttura di un dato corretto e non corretto.

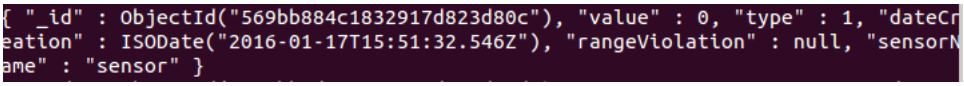

Fig. 25. Dato Corretto salvato in database

Come si puó osservare é stato comunque inserito un valore per la violazione a *null* in modo da poter distinguere meglio i dati violati da quelli invece corretti.

In questo caso appunto il valore della violazione non e' piu' nullo ma contiene un semplice campo *delta* che indicherá la differenza rispetto al range. In particolare questo sará positivo se il valore registrato supera il range attuale, negativo se invece é troppo basso o booleano se il tipo di sensoristica e quindi anche di range é booleano.

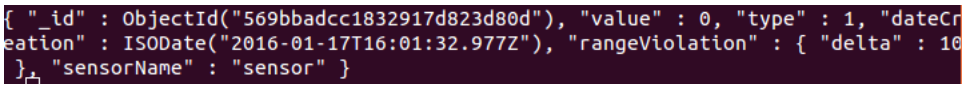

Fig. 26. Dato Non Corretto salvato in database

9.2 Introduzione All'Architettura di Progetto

Nelle prossime sezioni si vuole inserire le modifiche che sono state fatte alla architettura logica appena introdotta, ma cercando di evitare di ripetere aspetti che non sono effettivamnete cambiati. Di conseguenza si é deciso di riportare solamente quegli aspetti che sono stati modificati in base alle scelte tecnologiche effettivamente fatte.
In ogni caso si é deciso di cercare il piú possibile di sfruttare quanto giá é stato concordato dagli analisti cercando di effettuare meno modifiche possibili all'architettura.

9.3 Struttura

9.4 Interazione

In questa parte si riporta la tecnologia che si é utilizzata per effettuare l'interazione che si é analizzata e indicata in fase di analisi, lo stream. In particolare si é notato che, il framework play[**?**] giá gestisce attraverso delle sue strutture la possibilitá di operare attraverso stream di dati. Questa possibilitá é stata introdotta dal framework per gestire meglio dati parziali o casi che richiederebbero molto tempo, come ad esempio un'upload di file cospiquo. Nonostante tutto peró questo rientra nelle nostre necessitá e quindi ci avvarremo di questa astrazione per risolvere il nostro problema. Si riporta in bibliografia quindi la documentazione del framework che spiega il funzionamento di *Enumerators,Iteratees e Enumeratees.* [**?**]

9.5 Comportamento

10 Implementation

In questa fase bisognerebbe riportare il codice che implementa le varie parti, peró, per non appesantire troppo questa relazione che vuole essere una spiegazione alle scelte progettuali e di anali effettuate, si rimanda al lettore la revisione del codice. Di seguito si inserisce il link al repository

contenente tutti i sorgenti del progetto stesso, nonché il sorgente di tale documentazione.

`https://github.com/benkio/DomoticRoom`

11 Testing

12 Deployment

13 Maintenance