SP – Architettura di dettaglio

Autore: Simone Ballarin

Data: 25/06/18

Destinatari: Athesys

Diario delle modifiche

Data	Descrizione	Autore
27/06/18	Creazione documento. Inserimento Istruzioni generali per usare il progetto e architettura di dettaglio Starter.	Simone Ballarin
28/06/18	Inserimento capitolo Dettaglio Information Retriver, PageGenerator.	Simone Ballarin
29/06/18	Inserimento PiiDataHandle, modifica Starter, aggiunto metodo sendVerificationWork a AspRestImp. Inserito SendAccessInfo.	Simone Ballarin
		Simone Ballarin

Scopo del documento

Il seguente documento *SP – Architettura di dettaglio* ha lo scopo di presentare in maniera dettagliato ogni classe presentata in *SP – Architettura di massima*. Il documento deve fornire una descrizione dei metodi e campi dati presenti.

Sintesi del documento

Il documento espone i vari esecutori in ogni loro classe e metodo. Viene presentato per ogni esecutore il loro diagramma UML e per ogni classe una tabella con Descrizione, Parametri, Pseudo codice ed eventuali note. Gli ADT che sono presenti in diversi esecutori vengono descritti alla loro prima occorrenza e poi solo citati nei diagrammi successivi.

Riferimenti informativi

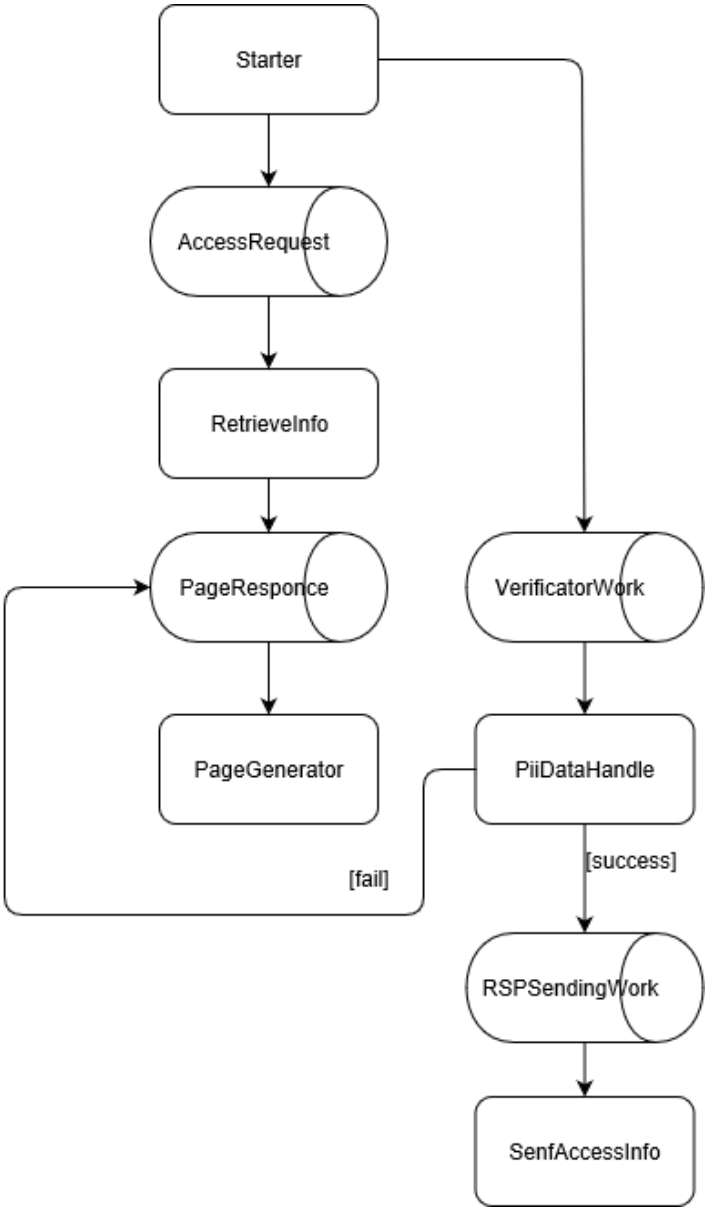
- SP – Analisi dei requisiti;
- http://www.html.it/pag/19603/implementare-un-web-service-restful/
- https://docs.microsoft.com/en-us/aspnet/web-api/overview/older-versions/build-restful-apis-with-aspnet-web-api
- Sample n. 2 sul sito www.MassTransit.com
- https://stackoverflow.com/questions/3850019/running-two-projects-at-once-in-visual-studio

Breve nota sull'ambiente

- Installare erlang (21.0. 21.0.1 non funziona con Windows);
- Installare RabbitMQ;
- Abilitare il plugin RabbitMQAdministrator;
- La dashboard di configurazione si trova in http://localhost:15672/#/connections

Architettura di dettaglio

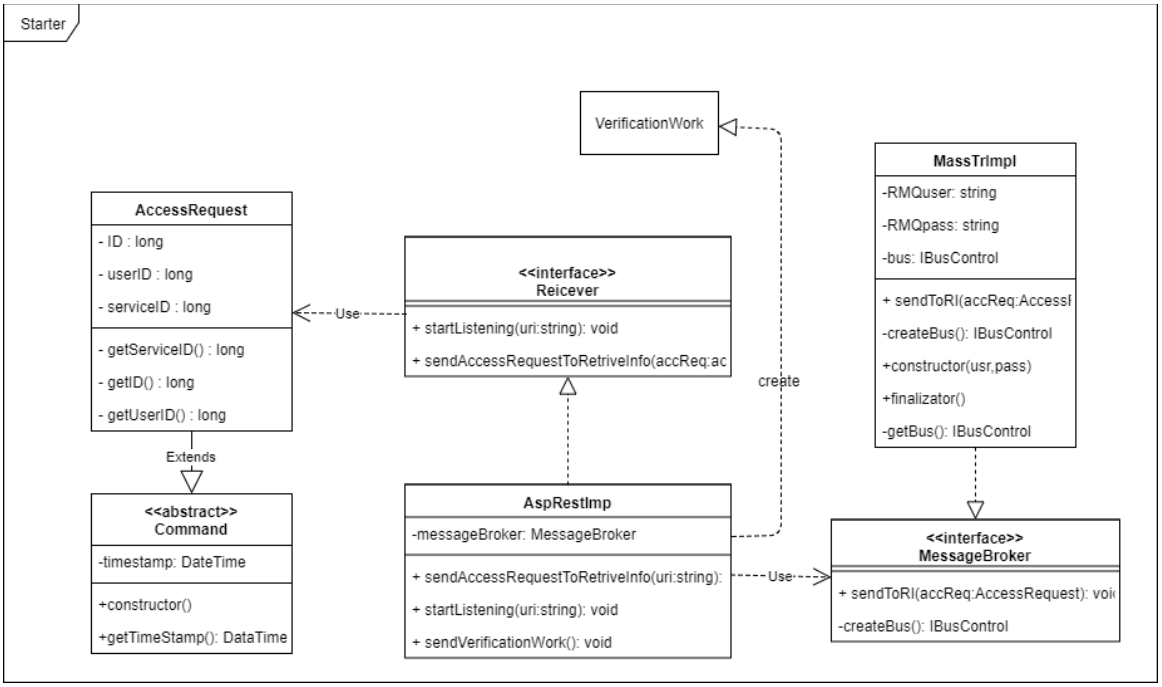
Si consiglia di tenere sempre a mente il flusso generale che il componente SP deve compiere. Il seguente diagramma dovrebbe fornire una prima comprensione di come i vari esecutori comunicano fra loro.



Lo Starter quando riceve una richiesta d'accesso da parte del RSP procede a generare il lavoro di AccessRequest, una volta ricavati tutti i dati necessari per l'accesso da Monokee, viene affidato al PageResponce l'incarico di visualizzare la pagina che richiede l'inserimento del QR. I dati verranno poi inseriti dall'utente e attraverso lo Starter verrà creato un lavoro di verifica dei dati inseriti e se questi sono sufficienti ad accedere al servizio, tramite un'ulteriore accesso a Monokee. In caso di esito positivo viene creato un lavoro di invio dati verso il RSP altrimenti verrà visualizzata una pagina di errore.

Dettaglio Starter

Lo starter rappresenta l'esecutore iniziale. Questo rimane in ascolto di eventuali richieste di accesso inoltrate dagli RSP e da inizio all'esecuzione di questa. Riceve inoltre pure i dati provenienti dai codici QR.



Reicever (interfaccia)

Questa classe ha lo scopo di ricevere e inoltrare i messaggi alla coda successiva (RetrieveInfo). Rappresenta il driver del componente Starter.

Metodi:

Public void startListening(uri: string)

Descrizione	Il metodo ha il compito di rimanere in ascolto di eventuali comunicazioni da parte dei RSP
Parametri	• uriPrefixes []: string
Pseudo codice	Non presente
Note	

Private void sendAccessRequestToRetriveInfo (accReq: AccessRequest)

Descrizione	Il metodo ha il compito di rimanere inviare le richieste di accesso incapsulate negli oggetti AccessRequest alla coda del RetrieveInfo
Parametri	• accReq: AccessRequest
Pseudo codice	Non presente
Note	

Private void sendVerificationWorl (vw:VerificationWork)

Descrizione	Il metodo ha il compito di inviare le richieste di verifica.
Parametri	• vw: VerificationWork
Pseudo codice	Non presente

Note	

AspRestImp (implementa Reicever)

Campi dati:

messageBroker: è un riferimento all'oggetto che implementa il messageBroker, tramite questo oggetto vengono inoltrate le richieste alla coda del RetrieveInfo.

Metodi:

void startListening(uri: string)

Descrizione	Il metodo ha il compito di rimanere in ascolto di eventuali comunicazioni http
Parametri	• uriPrefixes[]: string

Pseudo codice	check (uriPrefixes not null);
	check(uriPrefixes not empty)
	listener = HttpListener;
	foreach (uri in uriPrefixes) {
	  listener.add(uri);
	}
	listener.start();
	while true {
	  context = listener.getContext(); //bloccante
	  request = context.request;
	  response = context.responce;
	  if (response is an access Req){
	  accReq = new AccessRequest(getData(responce));
	  sendAccessRequestToRetriveInfo(accReq);
	}
	If (responce is an ver work)
	  This.sendVerificationWork(response);
	}
	Else ignore();

Note	https://docs.microsoft.com/it-it/dotnet/api/system.net.httplistener?view=netframework-4.7.1#definition

Private void sendAccessRequestToRetriveInfo (accReq: AccessRequest)

Descrizione	Il metodo ha il compito di rimanere inviare le richieste di accesso incapsulate negli oggetti AccessRequest alla coda del RetrieveInfo
Parametri	• accReq: AccessRequest
Pseudo codice	messageBroke.sendAccessRequestToRetriveInfo(accReq);
Note	

Private void sendVerificationWorl (vw:VerificationWork)

Descrizione	Il metodo ha il compito di inviare le richieste di verifica.
Parametri	• vw: VerificationWork
Pseudo codice	messageBroker.sendVerificaitionWork(vw);
Note	

MessageBroker (interfaccia)

Metodi:

Public void sendToRI (accReq: AccessRequest)

Descrizione	Il metodo ha il compito di rimanere inviare le richieste di accesso incapsulate negli oggetti AccessRequest alla coda del RetrieveInfo
Parametri	• accReq: AccessRequest
Pseudo codice	Non presente
Note	

Private IBusControl createBus()

Descrizione	Il metodo ha il compito creare il canale di comunicazione verso la coda del componente RetrieveInfo
Parametri	• user: string • pass: string
Pseudo codice	Non presente

MassTrImpl (implementa MessageBroker)

Questa classe implementa il MessageBroker utilizzano la libreria MassTransit che a sua volta usa RabbitMQ. Per queste ragioni è necessario avere installato RabbitMQ e preparato gli utenti.

Campi dati:

RMQuser: stringa che contiene il nome dell'account RabbitMQ da usare per inviare i messaggi.

RMQpass: stringa che contiene la password dell'account RabbitMQ da usare per inviare i messaggi.

bus: riferimento all'oggetto IBusControl cha rappresenta il canale di comunicazione, se questo non è stato creato, o la sua creazione ha fallito presenta un valore a null.

Metodi:

Public constructor (user:string, pass: string)

Descrizione	Il metodo ha il compito costruire l'oggetto MassTrImpl e di stabilire la connessione
Parametri	• user: string • pass: string

Pseudo codice	This.bus = createBus();
	this.user = user;
	this.pass = pass;
	bus.start()
	return this.bus;

Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Public finalizator (user:string, pass: string)

Descrizione	Il metodo ha il compito di rilasciare tutte le risorse di sistema ottenuto durante la vita dell'oggetto.

Parametri	
Pseudo codice	Bus.stop; rilascio altre risorse;
Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Public IBusControl getBus()

Descrizione	Il metodo ha il compito ritornare l'oggetto IBusControl creato dal costruttore
Parametri	
Pseudo codice	Check (bus not null); return this.bus;
Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Public void sendToRI (accReq: AccessRequest)

Descrizione	Il metodo ha il compito di rimanere inviare le richieste di accesso incapsulate negli oggetti AccessRequest alla coda del RetrieveInfo
Parametri	• accReq: AccessRequest

Pseudo codice	getBus();
	client.createRequesClient();
	client.request(accReq)

Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Private IBusControl createBus(user:string, pass:string)

Descrizione	Il metodo ha il compito creare il canale di comunicazione verso la coda del componente RetrieveInfo
Parametri	• user: string • pass: string

Pseudo codice	`create b = new Bus.Factory.CreateUsingRabbitMQ;`
	`setUsername(user);`
	`setPassword(pass);`
	setDatiServerRabbitMQ();
	return b;

Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Command (abstract)

Questa classa astratta rappresenta un generico messaggio che può essere trasportato dal MessageBroker.

Campi dati:

timestamp: è un riferimento all'oggetto DateTime che contiene il timestamp della creazione del Command.

Metodi:

Public construct ()

Descrizione	Il metodo ha il compito costruire l'oggetto Command.

Parametri	
Pseudo codice	This.timeStamp = DataTime.UtcNow
Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Public DateTIme getTimeStamp()

Descrizione	Il metodo ha il compito di restituire il timestamp che rappresenta il momento di creazione dell'oggetto Command
Parametri	
Pseudo codice	Return this.timestamp;
Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

AccessRequest (estende Command)

Rappresenta una richiesta di accesso, nella visione Event Driven rappresenta un evento iniziale.

Campi dati:

ID: è un long che rappresenta una chiave della richiesta, questo codice deve dovrebbe condiviso anche dall'ITF, Monokee e RSP.

userID: è un long che rappresenta una chiave dell'utente che ha richiesto l'accesso, questo codice dovrebbe essere condiviso anche dall'ITF, Monokee e RSP.

serviceID: è un long che rappresenta una chiave del servizio a cui l'utente a richiesto l'accesso, questo codice dovrebbe essere condiviso anche dall'ITF, Monokee e RSP.

Metodi:

Public long getID()

Descrizione	Il metodo ha il compito restituire l'ID della richiesta d'accesso al servizio
Parametri	
Pseudo codice	Return this.ID;
Note	

Public long getUserID()

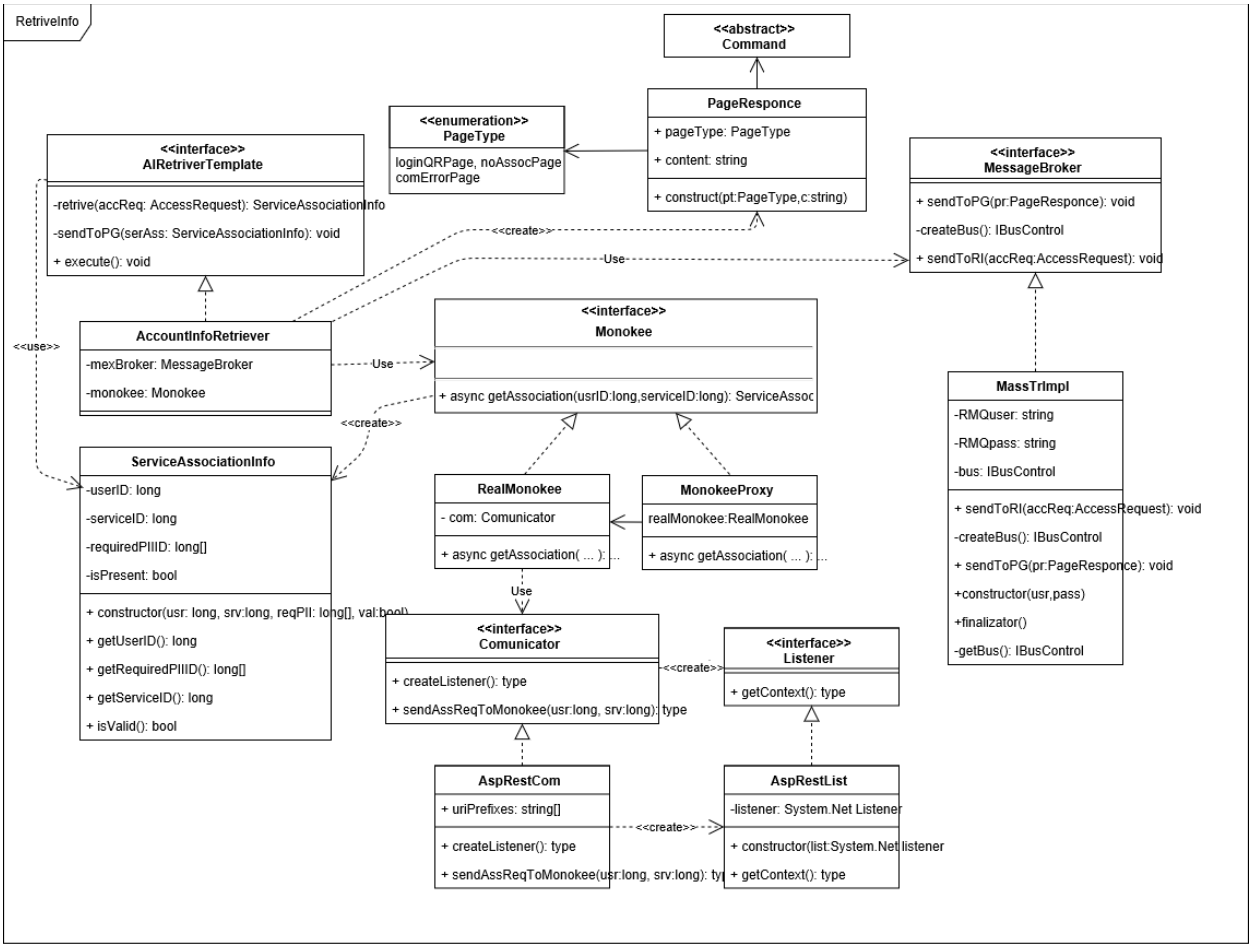
Descrizione	Il metodo ha il compito restituire l'ID dell'utente che ha effettuato la richiesta di accesso.
Parametri	
Pseudo codice	Return this.userID;
Note	

Public long getServiceID()

Descrizione	Il metodo ha il compito restituire l'ID del servizio a cui la richiesta di accesso si riferisce
Parametri	
Pseudo codice	Return this.serviceID;
Note	

Dettaglio RetriveInfo

È l'esecutore con il compito di ottenere le informazioni necessarie da Monokee. In caso di associazione presente manda il lavoro di creazione pagina di login, in caso di associazione non presente manda il lavoro di creazione pagina di errore.



AIRetriverTemplate (interfaccia)

Questa interfaccia ha lo scopo di ricevere gestire e inoltrare i messaggi alla coda successiva (RetrieveInfo). Rappresenta un template (Templete pattern) del driver dell'esecutore RetriveInfo.

Metodi:

private ServiceAssociationInfo retrive (accReq: AccessRequest)

Descrizione	Il metodo ha il compito interrogare Monokee e quindi restituire le informazioni relative all'Access Request richiesta.
Parametri	• accReq: AccessRequest rappresenta la richesta di accesso di cui si vogliono avere le informazioni da Monokee.
Pseudo codice	Non presente
Note	

Private void sendToPG (serAss: ServiceAssocciationInfo)

Descrizione	Il metodo ha il compito mandare all'esecutore PageGenerator il lavoro di visualizzazione della pagina di risposta. Questa pagina varia in base al ServiceAssociationInfo e quindi può essere una pagina di errore o di login QR
Parametri	• serAss: rappresenta le informazioni ottenute da Monokee.
Pseudo codice	Non presente
Note	

Public void execute ()

Descrizione	Nell'ottica del template method patter questo metodo rappresenta esegue l'esecuzione iterativa dell'algoritmo. Chiama il metodo retrive e sendToPG passandogli facendo da passante per i risultati intermedi.
Parametri	
Pseudo codice	Non presente
Note	

AccountInfoRetriever (implementa AIRetriverTemplate)

Campi dati:

mexBroker: è un riferimento alla classe che si occupa di gestire le code, quindi di inviare i messaggi e di riceverli

monokee: è un riferimento alla classe che rappresenta la risorsa esterna Monokee.

Metodi:

private ServiceAssociationInfo retrive (accReq: AccessRequest)

Descrizione	Il metodo ha il compito interrogare Monokee e quindi restituire le informazioni relative all'Access Request richiesta.
Parametri	• accReq: AccessRequest rappresenta la richesta di accesso di cui si vogliono avere le informazioni da Monokee.
Pseudo codice	Ass = Monokee.getAssociation(accReq); return ass;
Note	

Private void sendToPG (serAss: ServiceAssocciationInfo)

Descrizione	Il metodo ha il compito mandare all'esecutore PageGenerator il lavoro di visualizzazione della pagina di risposta. Questa pagina varia in base al ServiceAssociationInfo e quindi può essere una pagina di errore o di login QR
Parametri	• serAss: rappresenta le informazioni ottenute da Monokee.
Pseudo codice	Non presente

Note	Val = serAss.associationIsValid();
	if(val is null) page new pageResponce(error)
	If (val) page = new PageResponce(loginQRPage);
	Else page = new PageResponce(failed);
	getBus();
	client.createRequesClient();
	client.request(page);

Public void execute ()

Descrizione	Nell'ottica del template method patter questo metodo rappresenta esegue l'esecuzione iterativa dell'algoritmo. Chiama il metodo retrive e sendToPG passandogli facendo da passante per i risultati intermedi.
Parametri	

Pseudo codice	ConfigLog(xmlConfig);
	`Log4NetLogWriterFactory.Use();`
	`Log4NetLogger.Use();`
	`Retrive{`
	`  Overload start () {x -> this.reiceve(x)}`
	`}`
	`return (int)HostFactory.Run(x => x.Service<Retriver>());`
	`//avvia host`

Note	

ServiceAssociationInfo

Questa classe rappresenta le informazioni ottenute da Monokee, riporta i dati della AccessRequest che l'ha generata e ci aggiunge l'informazione relativa alla lista delle PII richieste e la presenta o meno della associazione in Monokee. È un oggetto immutabile.

Campi dati:

userID: è una chiave che identifica l'utente che ha effettuato la richiesta di accesso.

serviceID: è una chiave che identifica il servizio per cui l'utente ha effettuato la richiesta di accesso.

requiredPIIID: è una lista di chiavi che identifica i PII necessari per effettuare l'accesso al servizio .

isPresent: è un booleano che rappresenta la possibilità dell'utente ad accedere al servizio richiesto.

Metodi:

Public constructor (usr:long, srv:long, reqPII: long[], val:bool)

Descrizione	Questo metodo ha il compito di creare l'oggetto, è l'unico modo per inserire i dati, in quanto l'oggetto è immutabile.
Parametri	• userID: long • serviceID: long • requiredPIIID: long[] • isPresent:bool
Pseudo codice	This.userID = userID; This.serviceID = serviceID; This.requiredPIIID = requiredPIIID; This.isPresent = isPresent;
Note	

Public long getUserID()

Descrizione	Questo metodo restituisca la chiave dell'utente a cui fanno riferimento le informazioni di associazione
Parametri	
Pseudo codice	Return userID;
Note	

Public long getRequiredPIIID()

Descrizione	Questo metodo restituisca le chiave delle PII richieste per effettuare l'accesso.
Parametri	
Pseudo codice	Return requiredPIID;
Note	Ritorna null in caso l'associazione non sia presente.

Public long getServiceID()

Descrizione	Questo metodo restituisca la chiave del servizio a cui le informazioni fanno riferimento.
Parametri	
Pseudo codice	Return serviceID;
Note	

Public Boolean isValid()

Descrizione	Questo metodo restituisce true se l'utente è associato al servizio richiesto, false se non lo è.
Parametri	
Pseudo codice	Return isPresent;
Note	

PageType (enumeration)

Questo enumeration enumera le varie possibili pagine creabili dall'esecutore PageGenerator.

I possibili tipi sono:

- loginQRPage, è una pagina mostra cattura il codice QR con le informazioni necessarie;
- noAssocPage, è una pagina che comunica che l'utente che ha effettuato una richiesta per un servizio a cui non è abilitato (non possiede l'associazione in Monokee);
- comErrorPage, è una pagina che comunica un generico errore di comunicazione;

Command (abstract)

Questa classa astratta rappresenta un generico messaggio che può essere trasportato dal MessageBroker.

Campi dati:

timestamp: è un riferimento all'oggetto DateTime che contiene il timestamp della creazione del Command.

Metodi:

Public construct ()

Descrizione	Il metodo ha il compito costruire l'oggetto Command.
Parametri	
Pseudo codice	This.timeStamp = DataTime.UtcNow
Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Public DateTIme getTimeStamp()

Descrizione	Il metodo ha il compito di restituire il timestamp che rappresenta il momento di creazione dell'oggetto Command
Parametri	
Pseudo codice	Return this.timestamp;
Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

PageResponce (estende Command)

Questa classe estende Command, e nel contesto dell'architettura event driven rappresenta un evento di processamento che rappresenta la creazione di generazione e visualizzazione pagina. È un oggetto immutabile.

Campi dati:

pageType: indica il tipo della pagina che si vuole far creare e visualizzare dall'esecutore. Questa informazione è rappresentata dall'enumeratore PageType.

content: è una stringa contenente delle generiche informazioni che si vuole far visualizzare nella pagina che verrà creata.

Metodi:

public construct (pt:PageType, content:string)

Descrizione	Il metodo ha il compito costruire l'oggetto PageResponce. Rappresenta l'unica modo per inserire le informazioni all'interno dell'oggetto.
Parametri	• pageType: PageType • content: string
Pseudo codice	This.pageType = pageType; This.content = content;
Note	

MessageBroker (interfaccia)

Metodi:

Public void sendToRI (accReq: AccessRequest)

Descrizione	Il metodo ha il compito di inviare le richieste di accesso incapsulate negli oggetti AccessRequest alla coda del RetrieveInfo
Parametri	• accReq: AccessRequest
Pseudo codice	Non presente
Note	

Public void sendToPG (sInfo: ServiceAssociationInfo): void

Descrizione	Il metodo ha il compito di inviare le richieste di accesso incapsulate negli oggetti PageResponce alla coda del PageResponce
Parametri	• pageResponce: PageResponce
Pseudo codice	Non presente
Note	

Private IBusControl createBus()

Descrizione	Il metodo ha il compito creare il canale di comunicazione verso la coda del componente RetrieveInfo
Parametri	• user: string • pass: string
Pseudo codice	Non presente
Note	

MassTrImpl (implementa MessageBroker)

Questa classe implementa il MessageBroker utilizzano la libreria MassTransit che a sua volta usa RabbitMQ. Per queste ragioni è necessario avere installato RabbitMQ e preparato gli utenti.

Campi dati:

RMQuser: stringa che contiene il nome dell'account RabbitMQ da usare per inviare i messaggi.

RMQpass: stringa che contiene la password dell'account RabbitMQ da usare per inviare i messaggi.

bus: riferimento all'oggetto IBusControl cha rappresenta il canale di comunicazione, se questo non è stato creato, o la sua creazione ha fallito presenta un valore a null.

Metodi:

Public constructor (user:string, pass: string)

Descrizione	Il metodo ha il compito costruire l'oggetto MassTrImpl e di stabilire la connessione
Parametri	• user: string • pass: string

Pseudo codice	This.bus = createBus();
	this.user = user;
	this.pass = pass;
	bus.start()
	return this.bus;

Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Public finalizator (user:string, pass: string)

Descrizione	Il metodo ha il compito di rilasciare tutte le risorse di sistema ottenuto durante la vita dell'oggetto.
Parametri	
Pseudo codice	Bus.stop; rilascio altre risorse;
Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Public IBusControl getBus()

Descrizione	Il metodo ha il compito ritornare l'oggetto IBusControl creato dal costruttore
Parametri	

Pseudo codice	Check (bus not null); return this.bus;
Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Public void sendToRI (accReq: AccessRequest)

Descrizione	Il metodo ha il compito di rimanere inviare le richieste di accesso incapsulate negli oggetti AccessRequest alla coda del RetrieveInfo
Parametri	• accReq: AccessRequest

Pseudo codice	getBus();
	client.createRequesClient();
	client.request(accReq)

Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Public void sendToPG (sInfo: ServiceAssociationInfo): void

Descrizione	Il metodo ha il compito di rimanere inviare le richieste di accesso incapsulate negli oggetti PageResponce alla coda del PageResponce
Parametri	• pageResponce: PageResponce

Pseudo codice	Non getBus();
	client.createRequesClient();
	client.request(pageResponce);

Note	

Private IBusControl createBus(user:string, pass:string)

Descrizione	Il metodo ha il compito creare il canale di comunicazione verso la coda del componente RetrieveInfo
Parametri	• user: string • pass: string

Pseudo codice	`create b = new Bus.Factory.CreateUsingRabbitMQ;`
	`setUsername(user);`
	`setPassword(pass);`
	setDatiServerRabbitMQ();
	return b;

Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Monokee (interfaccia)

Questa interfaccia rappresenta l'entità Monokee, con le classi RealMonokee e MonokeeProxy partecipa ad un'applicazione di un proxy pattern.

Metodi:

Public async ServiceAssociationInfo getAssociation(userID: long, serviceID:long)

Descrizione	Il metodo ha il compito di restituire le informazioni riguardo all'associazione tra utente e servizio.
Parametri	• userID: long che rappresenta la chiave dell'utente • serviceID: long che rappresenta la chiave del servizio richiesto
Pseudo codice	Non presente
Note	

RealMonokee (implementa Monokee)

Rappresenta una reale istanza del servizio Monokee. Con Monokee e MonokeeProxy rappresenta un'applicazione del pattern Proxy.

Campi dati:

comunicator: è un riferimento di un oggetto che implementa l'interfaccia Comunicator.

Metodi:

Public async ServiceAssociationInfo getAssociation(userID: long, serviceID:long)

Descrizione	Il metodo ha il compito di restituire le informazioni riguardo all'associazione tra utente e servizio. Questa operazione è asincrona.
Parametri	• userID: long che rappresenta la chiave dell'utente • serviceID: long che rappresenta la chiave del servizio richiesto
Pseudo codice	Return Async Communicator.sendAssToMonokee(userID,serviceID);
Note	

MonokeeProxy (implementa Monokee)

Rappresenta un proxy remoto del servizio Monokee. Applica una politica di acquisizione pigra. Con Monokee e RealMonokee rappresenta un'applicazione del pattern Proxy.

Campi dati:

realMonokee: è un riferimento di un oggetto RealMonokee.

Metodi:

Public async ServiceAssociationInfo getAssociation(userID: long, serviceID:long)

Descrizione	Il metodo ha il compito di restituire le informazioni riguardo all'associazione tra utente e servizio. Questa operazione è asincrona.
Parametri	• userID: long che rappresenta la chiave dell'utente • serviceID: long che rappresenta la chiave del servizio richiesto
Pseudo codice	applyPolicy1; applyPolicy2; …. applyPolicyN; realMonokee.getAssociation(userID, serviceID)
Note	

Comunicator (interfaccia)

Questa classe fornisce un'interfaccia per gestire tutte le informazioni attraverso fonti esterne. Deve essere usata per comunicare con Monokee e il Real Service Provider.

Metodi:

Public Listener createListener()

Descrizione	Il metodo ha il compito di restituire un oggetto listener che ascolto le richieste da parte di Monokee.
Parametri	
Pseudo codice	Non presente
Note	

Public void sendAssReqToMonokee(usr:long, serv:long)

Descrizione	Il metodo ha il compito di inviare una richiesta di associazione a Monokee.
Parametri	• usr: long che rappresenta la chiave dell'utente • serv: long che rappresenta la chiave del servizio richiesto
Pseudo codice	Non presente
Note	

AspRestCom (implementa Comunicator)

Campi dati:

uriPrefixes: lista stringhe che rappresentano gli uri a cui il listener ascolta.

Metodi:

Public Listener createListener()

Descrizione	Il metodo ha il compito di restituire un oggetto listener che ascolto le richieste da parte di Monokee.
Parametri	

Pseudo codice	check (uriPrefixes not null);
	check(uriPrefixes not empty)
	listener = HttpListener;
	foreach (uri in uriPrefixes) {
	  listener.add(uri);
	}
	listener.start();
	return new AspRest(listener);

Note	

Public void sendAssReqToMonokee(usr:long, serv:long)

Descrizione	Il metodo ha il compito di inviare una richiesta di associazione a Monokee.
Parametri	• usr: long che rappresenta la chiave dell'utente • serv: long che rappresenta la chiave del servizio richiesto

Pseudo codice	string content = userID+":" serviceID;
	ASCIIEncoding encoding=new ASCIIEncoding();
	byte[] buffer =encoding.GetBytes(content);
	request.ContentType="application/x-www-associationInfo ";.
	request.ContentLength=content.Length;
	Stream newStream=request.GetRequestStream();
	newStream.Write(buffer,0,buffer.Length);
	newStream.Close();

Note	http://www.dotnethell.it/tips/SendPOSTHttp.aspx

Listener (interfaccia)

È un oggetto con il compito di rimanere in ascolto su determinati uri. È un oggetto non mutabile.

Metodi:

Public void getContext()

Descrizione	Il metodo ha il compito di ritornare appena arriva un messaggio inviato da uno degli uri specificati nella AspRestComp.

Parametri	
Pseudo codice	Non presente
Note	

AspRestList (implementa Listerner)

È un oggetto con il compito di rimanere in ascolto su determinati uri. È un oggetto non mutabile. Questa classe è un wrapper del listener di System.Net.

Campi dati:

listener: è il listener fornito dal System.Net. Creato da AspRestComp

Metodi:

Public constructor()

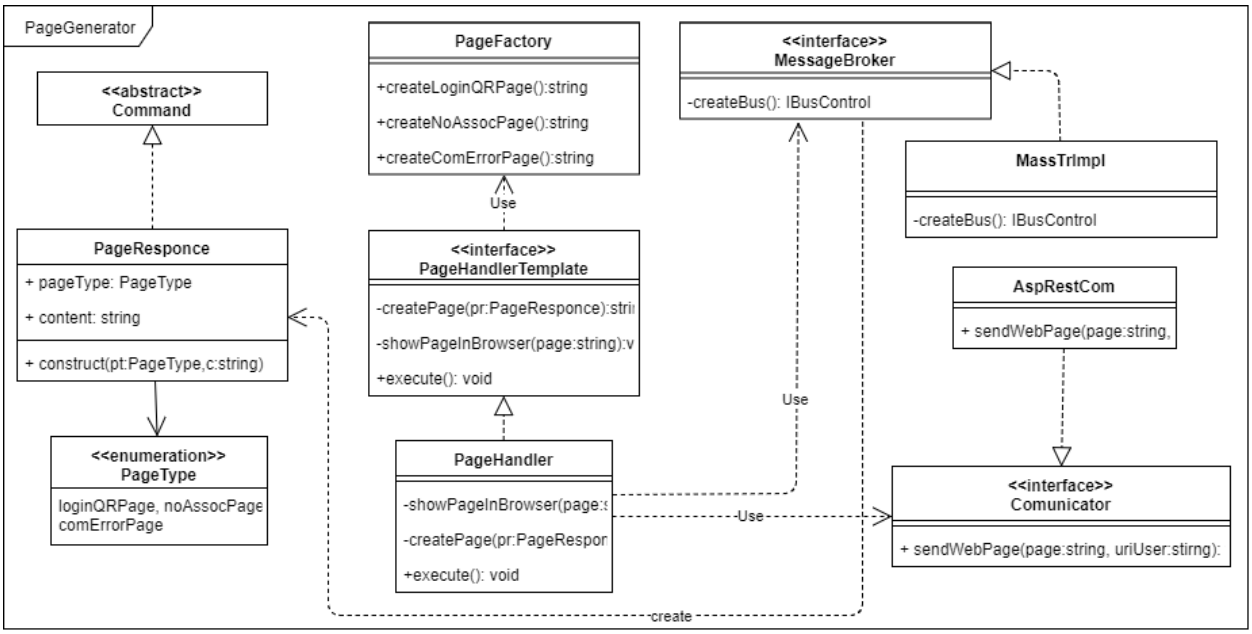
Descrizione	Il metodo ha il compito di costruire l'oggetto Listener da un'istanza del listener fornito da System.Net
Parametri	• List: è un'implementazione di System.Net listener.
Pseudo codice	This.listener = List;
Note	

Public void getContext()

Descrizione	Il metodo ha il compito di ritornare appena arriva un messaggio inviato da uno degli uri specificati nella AspRestComp.
Parametri	
Pseudo codice	Listener.getContext();
Note	

Dettaglio PageGenerator

È l'esecutore con il compito di ascoltare le richieste di creazione pagina provenienti dagli altri esecutori e quindi di generarle come richiesto e di inviarle al browser del richiedente.



PageHandlerTemplate (interfaccia)

Questa interfaccia consiste in un template per la creazione di un driver per la gestione e l'esecuzione dei Command di PageGenerator.

Metodi:

private string createPage(pr:PageResponce)

Descrizione	Questa metodo rappresenta lo step dell'algoritmo in cui si crea la pagina da visualizzare in base alla richiesta pervenuta rappresentata dalla PageResponce.
Parametri	• Pr: è la PageResponce pescata dalla coda dell'esecutore, in questo oggetto è contenuto la pagina che si vuole creare.
Pseudo codice	Non presente
Note	

private void showPageInBrowser(page:string)

Descrizione	Questa metodo rappresenta il secondo step dell'algoritmo, cioè quello in cui la pagina precedentemente creata viene visualizzata nel browser di chi ha fatto la richiesta.
Parametri	• Page: è una stringa che rappresenta la pagina che si vuole stampare nel browser.
Pseudo codice	Non presente
Note	

private void execute ()

Descrizione	Questa metodo rappresenta è l'esecutore dei vari step, gli esegue in ordine in modo tale da gestire un lavoro di PageResponce.

Parametri	
Pseudo codice	Non presente
Note	

PageHandler (implementa PageHandlerTemplate)

Questa classe implementa la struttura fornita da PageHandlerTemplate. Gestice il Command PageResponce.

Campi dati:

pageFactory: è un'istanza dell'oggetto PageFactory, serve per permettere al PageHandler di generare la pagine da inviare ai Browser.

communicator: è un'istanza dell'oggetto Comunicator che fornisce i metodi necessari per effettuare le comunicazioni verso Monokee e l'utente.

Metodi:

private string createPage(pr:PageResponce)

Descrizione	Questa metodo rappresenta lo step dell'algoritmo in cui si crea la pagina da visualizzare in base alla richiesta pervenuta rappresentata dalla PageResponce.
Parametri	• Pr: è la PageResponce pescata dalla coda dell'esecutore, in questo oggetto è contenuto la pagina che si vuole creare.

Pseudo codice	Switch(Pr.pageType){
	Case: loginQRPage;
	pageFactory.createLoginQRPage(pg.content);
	break;
	case: noAssocPage;
	pageFactory.createNoAssocPage(pg.content);
	break;
	case: comErrorPage;
	pageFactory.createComErrorPage(pg.content)
	}
	Return createdPage;

Note	

private void showPageInBrowser(page:string)

Descrizione	Questa metodo rappresenta il secondo step dell'algoritmo, cioè quello in cui la pagina precedentemente creata viene visualizzata nel browser di chi ha fatto la richiesta.
Parametri	• Page: è una stringa che rappresenta la pagina che si vuole stampare nel browser. • userUri: è una stringa che rappresenta l'uri dell'utente a cui inviare la pagina web.
Pseudo codice	Communicator.sendWebPage(page, userUri)
Note	

private void execute ()

Descrizione	Questa metodo rappresenta è l'esecutore dei vari step, gli esegue in ordine in modo tale da gestire un lavoro di PageResponce. Una volta

	eseguito un PageResponce rimane in attesa del prossimo e lo esegue. Prosegue iterativamente in questa maniera.
Parametri	

Pseudo codice	ConfigLog(xmlConfig); //file ausiliari di config
	`Log4NetLogWriterFactory.Use();`
	`Log4NetLogger.Use();`
	`Retrive{`
	`  Overload start () {x ->`
	`  {p=this.createPage(x); this.sendWebPage(p);}`
	`}`
	`return (int)HostFactory.Run(x => x.Service<Retriver>());`
	`//avvia host`

Note	

Comunicator (interfaccia)

Questa classe fornisce degli strumenti per interagire con l'esterno. Vengono presentate solo i metodi di interesse per questo esecutore.

Metodi:

public void sendWebPage(page:string, usrUri: string)

Descrizione	Questa metodo mostra la pagina web voluta nel browser dell'utente.
Parametri	• page: stringa che contiene il codice html della pagina da visualizzare nel browser dell'utente. • usrUri: stringa che contiene l'indirizzo che intende visualizzare la pagina.
Pseudo codice	Non presente
Note	

AspRestComunicator

Campi dati:

Metodi:

public void sendWebPage(page:string, usrUri:string)

Descrizione	Questa metodo mostra la pagina web voluta nel browser dell'utente.
Parametri	• page: stringa che contiene il codice html della pagina da visualizzare nel browser dell'utente. • usrUri: stringa che contiene l'indirizzo che intende visualizzare la pagina.

Pseudo codice	string content = page;
	ASCIIEncoding encoding=new ASCIIEncoding();
	byte[] buffer =encoding.GetBytes(content);
	request.ContentType="application/x-www-webPage ";.
	request.ContentLength=content.Length;
	Stream newStream=request.GetRequestStream();
	newStream.Write(buffer,0,buffer.Length);
	newStream.Close();

Note	https://docs.microsoft.com/en-us/aspnet/web-api/overview/older-versions/build-restful-apis-with-aspnet-web-api

MessageBroker (interfaccia)

Questa interfaccia ha il compito di gestire la coda. Nel contesto di questo iteratore viene usato soltanto il metodo createBus.

Metodi:

public IBusControl createBus()

Descrizione	Il metodo ha il compito creare il canale di comunicazione verso la coda del componente RetrieveInfo
Parametri	• user: string • pass: string
Pseudo codice	Non presente

MassTrImpl (implementa MessageBroker)

Questa classe implementa il MessageBroker utilizzano la libreria MassTransit che a sua volta usa RabbitMQ. Per queste ragioni è necessario avere installato RabbitMQ e preparato gli utenti.

Campi dati:

RMQuser: stringa che contiene il nome dell'account RabbitMQ da usare per inviare i messaggi.

RMQpass: stringa che contiene la password dell'account RabbitMQ da usare per inviare i messaggi.

bus: riferimento all'oggetto IBusControl cha rappresenta il canale di comunicazione, se questo non è stato creato, o la sua creazione ha fallito presenta un valore a null.

Metodi:

Public constructor (user:string, pass: string)

Descrizione	Il metodo ha il compito costruire l'oggetto MassTrImpl e di stabilire la connessione
Parametri	• user: string • pass: string

Pseudo codice	This.bus = createBus();
	this.user = user;
	this.pass = pass;
	bus.start()
	return this.bus;

Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Public finalizator (user:string, pass: string)

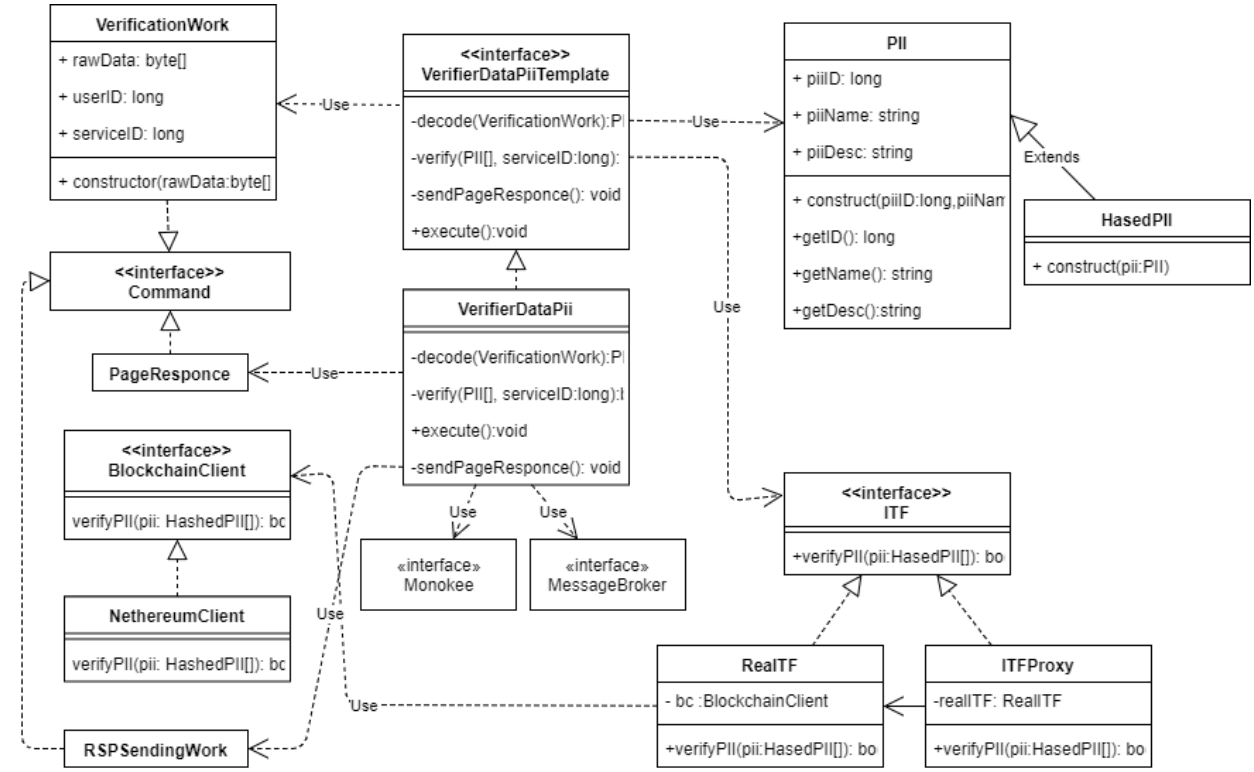
Descrizione	Il metodo ha il compito di rilasciare tutte le risorse di sistema ottenuto durante la vita dell'oggetto.
Parametri	
Pseudo codice	Bus.stop; rilascio altre risorse;
Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Public IBusControl getBus()

Descrizione	Il metodo ha il compito ritornare l'oggetto IBusControl creato dal costruttore
Parametri	
Pseudo codice	Check (bus not null); return this.bus;
Note	masstransit-project.com/MassTransit/learn/samples/request-response.html

Dettaglio PIIDataHandle

Questo esecutore ha il compito di ricevere nella propria coda dallo Starter una serie di lavoro di verifica e quindi di verificare questi sia verso Monokee sia verso l'ITF. In questo esecutore sono presenti le interfacce Command, Monokee e MessageBroker che non verranno presentate. Per una trattazione di Queste fare riferimento agli esecutori precedenti.



VerificationWork(implementa Command)

Questa classe rappresenta il lavoro di questo esecutore che consiste in una serie di PII da verificare presso l'ITF (queste essendo ottenuto da codice QR sono in formato grezzo) e verificare che queste siano sufficienti per effettuare l'accesso al servizio. L'oggetto è immutabile.

Campi dati:

rawData: è un array da dati grezzi byte che viene scansionato dal codice. Contiene le informazioni sui vari PII forniti dall'utente.

userID: è un long che rappresenta la chiave dell'utente che ha fornito i dati.

serviceID: è un long che rappresenta la chiave del servizio per cui è stato richiesto l'accesso.

Metodi:

constructor (rawData: byte [], userID: long, serviceID:long)

Descrizione	Il metodo ha il compito di costruire l'oggetto richiesta. Essendo l'oggetto immutabile questo è l'unico modo per inserire informazioni in esso.
Parametri	• rawData: byte[] con le info grezze del QR • userID: long con la chiave dell'utente

	• serviceID: long con la chiave del servizio per cui è richiesto l'accesso.
Pseudo codice	Non presente
Note	

VerificationDataPiiTemplate (interfaccia)

Questa interfaccia rappresenta il template da seguire per eseguire l'algoritmo di gestione di un VerificationPiiWork. Dalla ricezione fino al suo esaurimento.

Metodi:

Private PII[] decode(rawData: byte[])

Descrizione	Il metodo ha il compito eseguire la decifratura dei dati grezzi provenienti dalla lettura del codice QR e quindi di restituire un oggetto PII.
Parametri	• rawData: byte[] con le info grezze del QR
Pseudo codice	Non presente
Note	

Private bool verify(pii:PII, serviceID: long)

Descrizione	Il metodo ha il compito eseguire la verifica dei vari PII e di verificare che questi siano sufficienti. Se queste verifiche risultano entrambe positive allora il metodo ritorna true, altrimenti false.
Parametri	• PII[]: lista delle PII da verificare nella ITF • serviceID: long della chiave del servizio a cui effettuare l'accesso
Pseudo codice	Non presente
Note	

Private void sendPageResponce(pageType:PageType, content: string)

Descrizione	Il metodo ha il inviare la richiesta al PageGenerator di visualizzare la pagina di risposta alla verifica.
Parametri	• pageType: indica il tipo di pagina che si vuole far visualizzare • content: stringa con un messaggio da visualizzare sulla pagina dell'utente.
Pseudo codice	Non presente
Note	

Private void execute()

Descrizione	Il metodo ha il inviare rimanere in ascolto dei vari verification work
Parametri	
Pseudo codice	Non presente
Note	

VerificationDataPii (implementa VerificationDataPiiTemplate)

Questa classe rappresenta un'applicazione del template da seguire per eseguire l'algoritmo di gestione di un VerificationPiiWork. Dalla ricezione fino al suo esaurimento.

Campi dati:

monokee: è un riferimento ad un oggetto che implementa l'interfaccia Monokee.

ITF: è un riferimento ad un oggetto che implementa l'interfaccia ITF.

messageBroker: è un riferimento ad un oggetto che implenta l'interfaccia MessageBroker.

Metodi:

Private PII[] decode(rawData: byte[])

Descrizione	Il metodo ha il compito eseguire la decifratura dei dati grezzi provenienti dalla lettura del codice QR e quindi di restituire un oggetto PII.
Parametri	• rawData: byte[] con le info grezze del QR

Pseudo codice	rawData.toString();
	while(string end){
	getName;
	getDesc;
	getID;
	createPII(name,desc,ID);
	arr.putPII;
	}
	Return arr;

Note	

Private bool verify(pii:PII[], serviceID: long)

Descrizione	Il metodo ha il compito eseguire la verifica dei vari PII e di verificare che questi siano sufficienti. Se queste verifiche risultano entrambe positive allora il metodo ritorna true, altrimenti false.
Parametri	• pii: lista delle PII da verificare nella ITF • serviceID: long della chiave del servizio a cui effettuare l'accesso

Pseudo codice	serAss = Monokee.getAssociation(userID,serviceID);
	piiReq = serAss. getRequiredPII();
	bool verificati = true;
	foreach (piiID in piiReq && verificati) {
	bool trovato= false;
	foreach(p in pii && !trovato){
	If (p.getID() == piiID;) trovato=true;
	Else trovato = false;
	}
	Return verificati;
	}

Note	

Private void sendPageResponce(pageType:PageType, content: string)

Descrizione	Il metodo ha il inviare la richiesta al PageGenerator di visualizzare la pagina di risposta alla verifica.
Parametri	• pageType: indica il tipo di pagina che si vuole far visualizzare • content: stringa con un messaggio da visualizzare sulla pagina dell'utente.
Pseudo codice	messageBroker.sendToPG(new PageResponge(pageType,content);
Note	

Private void execute()

Descrizione	Il metodo ha il inviare rimanere in ascolto dei vari verification work E in base al risultato invia un Command al SendAccessInfo o la creazione di una pagina di errore.
Parametri	

Pseudo codice	ConfigLog(xmlConfig);
	`Log4NetLogWriterFactory.Use();`
	`Log4NetLogger.Use();`
	`Execute{`
	`  Overload start () {x -> this.decode(x); this.verify(…);`
	`if verify == false this.sendPageResponce(…)`
	`Else messageBroker.sendSendingWork(new RSPSendingWork)}`
	`}`
	`return (int)HostFactory.Run(x => x.Service<Retriver>());`
	`//avvia host`

Note	

PII

Rappresenta una PII in chiaro, è la stessa classe già presentata in RetriveInfo.

HashedPII (estende PII)

Questa classe rappresenta una PII con le informazioni piiName e piiDesc sotto forma di Hash, questa classe dev'essere usata per comunicare con l'ITF. L'oggetto è immutabile.

Campi dati:

rispetto a PII non presenta nessuna differenza, se non per il fatto che piiName e piiDesc sono sotto forma di hash.

Metodi:

public constructor(pii: PII)

Descrizione	Il metodo ha il compito di costruire l'oggetto hashedPII a partire da un oggetto PII.
Parametri	• pii: è un riferimento all'oggetto PII di cui si vuole creare la versione hashedPII.
Pseudo codice	This.ID = pii.getID(); This.piiName = pii.getName().hashed(); This.piiDesc = pii.getDesc().hashed();
Note	

ITF (interfaccia)

Questa interfaccia rappresenta il componente ITF. Con RealITF e ITFProxy partecipano ad un'applicazione del Proxy Pattern.

Metodi:

public verifyPII(pii: HashedPII[]):bool

Descrizione	Il metodo ha il compito di verificare presso l'ITF le informazioni PII
Parametri	• pii: è una lista di oggetti hashedPII da verificare nell'ITF.
Pseudo codice	Non presente.
Note	

RealITF (implementa ITF)

Questa interfaccia rappresenta il reale componente ITF. Si occupa di instaurare la comunicazione tramite l'uso di un BlockchainClient. Con ITF e ITFProxy partecipano ad un'applicazione del Proxy Pattern.

Campi dati:

blockClient: è un riferimento ad un oggetto che implementa l'interfaccia BlockchainClient. Si occupare di chiamare i metodi dei vari SmartContract.

Metodi:

public verifyPII(pii: HashedPII[]):bool

Descrizione	Il metodo ha il compito di verificare presso l'ITF le informazioni PII
Parametri	• pii: è una lista di oggetti hashedPII da verificare nell'ITF.
Pseudo codice	Return blockClient.verifyPII(pii);
Note	

ITFProxy (implementa ITF)

Questa interfaccia rappresenta un remote proxy del componente ITF. Si occupa applicare una serie di politiche. Con ITF e ITFProxy partecipano ad un'applicazione del Proxy Pattern.

Campi dati:

realITF: è un riferimento ad un oggetto RealITF.

Metodi:

public verifyPII(pii: HashedPII[]):bool

Descrizione	Il metodo ha il compito di verificare presso l'ITF le informazioni PII
Parametri	• pii: è una lista di oggetti hashedPII da verificare nell'ITF.
Pseudo codice	Return realITF.verifyPII(pii);
Note	

BlockchainClient (interfaccia)

Questa interfaccia ha il compito di rappresentare un canale di comunicazione verso gli SmartContract. Deve essere atea rispetto alla tipologia di blockchain usata.

Metodi:

public bool verifyPII(pii: HashedPII[])

Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che verifica i dati.
Parametri	• pii: è una lista di oggetti hashedPII da verificare nell'ITF.
Pseudo codice	Non presente
Note	L'implementazione sarà sensibilmente diversa in base alla specifica blockchain usata.

NethereumClient (implementa BlockchainClient)

Questa classe rappresentare un canale di comunicazione verso gli SmartContract di una rete Ethereum. Fa uso della libreria .NET Nethereum per instaurare la comunicazione.

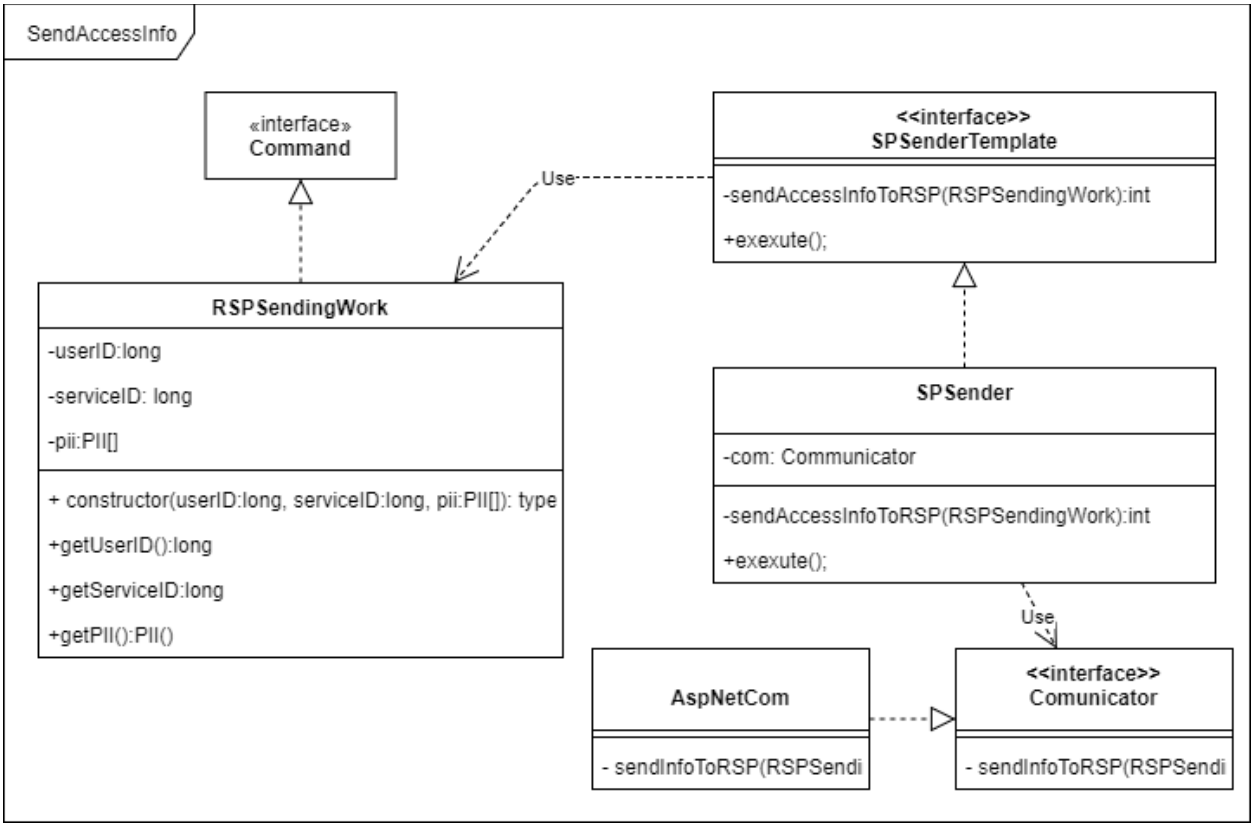
Descrizione	Il metodo ha il compito di chiamare il metodo presente negli smartContract che verifica i dati.
Parametri	• pii: è una lista di oggetti hashedPII da verificare nell'ITF.
Pseudo codice	Nethereum.Web3.Web3("https://ITFclient.com "); web3. Eth. Transactions.verifyMethod.Call; web3. Eth. Transactions.GetTransactionReceipt; return result;
Note	http://nethereum.com/

RSPSendingWork (implementa Command)

Questa classe rappresenta un lavoro da eseguire in caso di login possibile. Consiste nell'inviare le informazioni al RSP. Questa classe verrà presentata nel capitolo che tratta l'esecutore SendAccessInfo.

Dettaglio SendAccessInfo

Questo esecutore ha il compito di inviare le informazioni necessarie per effettuare il Login al RSP. Queste informazione deve essere fornite in forma non di Hash. I Command provengono da PIIDataHandle.



RSPSendingWork (implementa Command)

Questa classe rappresenta il lavoro di invio degli attributi di accesso al RSP. Contiene tutte le informazioni necessarie ad espletare il compito. La classe è immutabile.

Campi dati:

userID: è un long che rappresenta la chiave dell'utente che ha effettuato l'accesso.

serviceID: è un long che rappresenta la chiave del servizio a cui è stato effettuato l'accesso.

pii: è una lista di riferimenti ad oggetti PII, che rappresentano gli attributi di accesso in chiaro.

Metodi:

public constructor(userID:long, serviceID:long, pii:PII[])

Descrizione	Il metodo ha il compito di costruire l'oggetto RSPSendingWork.
Parametri	• userID: è un long che rappresenta la chiave dell'utente che ha effettuato la richiesta. • serviceID: è un long che rappresenta la chiave del servizio. • Pii: è una lista di PII che contiene tutti gli attributi di accesso necessari per effettuare l'accesso al servizio.

Pseudo codice	This.userID=userID; This.serviceID=serviceID; This.pii = pii;
Note	

public long getUserID()

Descrizione	Il metodo restituisce un long che rappresenta la chiave dell'utente che ha effettuato la richiesta.
Parametri	
Pseudo codice	Return userID;
Note	

public long getServiceID()

Descrizione	Il metodo restituisce un long che rappresenta la chiave del servizio a cui l'utente ha richiesto l'accesso.
Parametri	
Pseudo codice	Return serviceID;
Note	

public PII[] getPII()

Descrizione	Il metodo restituisce un array di oggetti PII che rappresentano le PII necessaria ad accedere il servizio. Le Pii devono essere in chiaro.
Parametri	
Pseudo codice	Return pii;
Note	

RSPSenderTemplate (interfaccia)

Questa interfaccia rappresenta il template da seguire per eseguire il lavoro dell'esecutore. In altre parole rappresenta l'algoritmo da eseguire per espletare il lavoro di invio attributi al RSP.

Metodi:

private void sendAccessInfoToRSP(info:RSPSendingWork)

Descrizione	Il metodo ha il compito di costruire l'oggetto inviare le informazioni di accesso al RSP.
Parametri	• Info: è un riferimento ad un oggetto di tipo RSPSendingWork che contiene tutte le informazioni necessarie per espletare l'invio al RSP.
Pseudo codice	Non presente
Note	

Public void execute()

Descrizione	Il metodo ha il compito di rimanere in attesa di messaggi ed in caso ne arrivi uno di eseguire l'invio.
Parametri	
Pseudo codice	Non presente
Note	

RSPSender (implementa RSPSenderTemplate)

Questa classe implementa il template da seguire per eseguire il lavoro dell'esecutore. In altre parole rappresenta l'algoritmo da eseguire per espletare il lavoro di invio attributi al RSP.

Campi dati:

com: è un riferimento ad un oggetto che implementa l'interfaccia Comunicator. Effettua tutte le operazioni di comunicazione con i RSP.

Metodi:

private void sendAccessInfoToRSP(info:RSPSendingWork)

Descrizione	Il metodo ha il compito di costruire l'oggetto inviare le informazioni di accesso al RSP.
Parametri	• Info : è un riferimento ad un oggetto di tipo RSPSendingWork che contiene tutte le informazioni necessarie per espletare l'invio al RSP.
Pseudo codice	Com.sendInfoToRSP(info);
Note	

Public void execute()

Descrizione	Il metodo ha il compito di rimanere in attesa di messaggi ed in caso ne arrivi uno di eseguire l'invio.
Parametri	

Pseudo codice	ConfigLog(xmlConfig);
	`Log4NetLogWriterFactory.Use();`
	`Log4NetLogger.Use();`
	`Execute{`
	`  Overload start () {x -> this.sendInfoToRSP(x);`
	`}`
	`return (int)HostFactory.Run(x => x.Service<Execute>());`
	`//avvia host`

Note	

Comunicator (interfaccia)

Questa classe fornisce degli strumenti per interagire con l'esterno. Vengono presentate solo i metodi di interesse per questo esecutore.

Metodi:

public void sendInfoToRSP(info: RSPSendingWork)

Descrizione	Questa metodo invio gli attributi di accesso al RSP.
Parametri	• Info : è un riferimento ad un oggetto di tipo RSPSendingWork che contiene tutte le informazioni necessarie per espletare l'invio al RSP.
Pseudo codice	Non presente
Note	

AspRestComunicator

Campi dati:

Metodi:

public void sendWebPage(page:string, usrUri:string)

Descrizione	Questa metodo invio gli attributi di accesso al RSP.
Parametri	• Info : è un riferimento ad un oggetto di tipo RSPSendingWork che contiene tutte le informazioni necessarie per espletare l'invio al RSP.

Pseudo codice	RetriveUri(info.getServiceID());//magari farselo passare.
	Content = info.getUserID()+info.getServiceID();
	Var pii = info.getPII();
	Foreach(p in pii){
	Conten.append(p.toString());
	}
	ASCIIEncoding encoding=new ASCIIEncoding();
	byte[] buffer =encoding.GetBytes(content);
	request.ContentType="application/x-www-accessAttributs ";.
	request.ContentLength=content.Length;
	Stream newStream=request.GetRequestStream();
	newStream.Write(buffer,0,buffer.Length);
	newStream.Close();

Note	https://docs.microsoft.com/en-us/aspnet/web-api/overview/older-versions/build-restful-apis-with-aspnet-web-api