SP – Architettura di massima

Autore: Simone Ballarin

Data: 25/06/18

Destinatari: Athesys

# Diario delle modifiche

|  |  |  |
| --- | --- | --- |
| Data | Descrizione | Autore |
| 25/06/2018 | Creazione documento e stesura dei capitoli: Scopo del documento, Riferimenti, Descrizione. | Simone Ballarin |

# Scopo del documento

Il seguente documento *SP – Architettura di massima* ha lo scopo di presentare e dimostrare una prima architettura di massima per il componente (IW) che dovrà funzionare nel contesto di un’estensione del prodotto MonoKee basato su blockchain.

# Sintesi del documento

Bla bla bla mi chiamo

# Riferimenti

* SP – Analisi requisiti;
* SP – Studio di fattibilità;
* [www.math.unipd.it/~rcardin/sweb/2018/Software%20Architecture%20Patterns\_4x4.pdf](http://www.math.unipd.it/~rcardin/sweb/2018/Software%20Architecture%20Patterns_4x4.pdf)

# Descrizione

Il progetto ha come scopo la creazione di un Identity Wallet (IW). L’applicativo si colloca nel contesto di un’estensione del servizio Monokee basato su blockchain. L’estensione offre un sistema di Identity Access Management (IAM) composto da quattro principali fattori:

* Identity Wallet (IW)
* Service Provider (SP)
* Identity Trust Fabric (ITF)
* Trusted Third Party (TTP)

In sintesi l’estensione dovrà operare al fine di fornire la possibilità ad un utente di registrare e gestire la propria identità autonomamente tramite l’IW, mandare i propri dati all’ITF, la quale custodirà la sua identità e farà da garante per le asserzioni proveniente dai TTP. Inoltre il SP dovrà essere in grado con le informazioni provenienti da IW e ITF di garantire o meno l’accesso ai propri servizi.

Il software SP, più dettagliatamente, dovrà assolvere ai seguenti compito:

nell’ambito della ricezione dei dati da un Identity Wallet (IW) deve:

* ricevere da parte dell’IW la chiave pubblica (o l’hash di questa);
* ricevere un riferimento alla locazione dell’hash della chiave pubblica all’interno dell’ITF;
* ricevere altre informazioni necessarie da parte dell’IW con relativo riferimento all’interno dell’ITF;
* gestire il trasferimento dei dati tramite codice QR.

Nell’ambito della verifica dei dati provenienti dall’IW deve:

* usare la chiave pubblica dell’IW e il riferimento per verificare l’identità e le varie altre informazione passate dall’Identity Wallet;
* generare e comparare gli hash dei valori ottenuti con quelli presenti nell’ITF;
* verificare che l’identità e le altre informazioni ottenute siamo sufficienti a garantire l’accesso al servizio.

Nell’ambito dell’accesso il SP deve:

* a seguito della verifica comunica il risultato all’organizzazione che fornisce il servizio, in modo tale da garantire l’accesso all’utente dell’IW.

Si intende, quindi, sviluppare il componente Service come un’applicazione server che opera in collaborazione con l’ITF al fine di la veridicità dei dati provenienti dall’IW e quindi garantire o meno l’accesso al servizio. Questa connessione deve avvenire tramite il protocollo JSON-RFC descritto nello Yellow Paper. Questi dati vengono presentati sotto forma di codice QR. L’applicazione deve inoltre avere un’interfaccia web accessibile tramite Internet dalla quale il personale del fornitore può configurare il servizio. Si fa notare come con SP non si intenda il servizio o il fornitore dei servizi, ma semplicemente il componente MonoKee che ha lo scopo di interfacciarsi con questi.

# Architettura di massima

Il componente Service Provider è sviluppato come applicazione server, questo implica un possibile accesso al servizio multiplo da parte di vari Real Service Provider (RSP) che ci inoltrano le loro richieste di accesso. L’applicativo fa uso di diverse fonti per espletare le proprie funzioni. Più dettagliatamente queste sono: Monokee, RSP e ITF. Da questo primo studio architetturale non sembrerebbe necessario l’uso di una base di dati locale. Considerato quanto appena detto si è ritenuta particolarmente adatta un’architettura Event Driven basata sull’utilizzo di code. Per la comunicazione con il RSP e con Monokee si è deciso di utilizzare un approccio basato sulle API RESTful. Invece per la comunicazione verso l’ITF si è deciso di utilizzare un client Ethereum.

# Architettura Event Driven

Questo tipologia di architettura rappresenta uno dei principali esempi di pattern architettura asincrono. Produce applicati altamente scalabili e facilmente adattabili ad ogni carico di utilizza. Se applicata bene fornisce la possibilità di avere eventi con un singolo scopo (Single Responsibility Principle) con un basso livello di accoppiamento. Questo è reso possibile dalla gestione asincrona di questi eventi.

Ci sono due possibili approcci a questa architettura:

* Mediator topology;
* Broker topology.

## Mediator topology

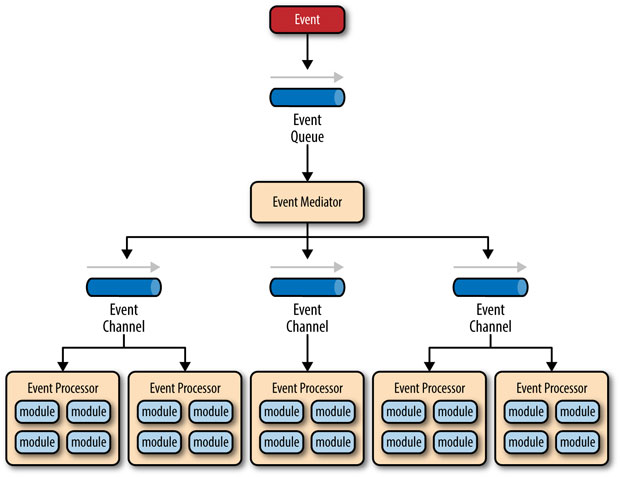
Un evento generalmente possiede una serie di passi ordinati per essere eseguito. In questa approccio ci sono quattro componenti che interagiscono fra loro:

* una o più code di eventi;
* un mediatore di eventi;
* uno o più esecutori di eventi;
* dei canali di eventi.

Gli eventi possono essere di due tipi:

* eventi iniziali;
* eventi di processamento.

Di seguito si riporta una generica architettura Event Driven Mediator Topology.



### Mediatore di eventi

Il mediatore (l’Event Mediator) ha il compito di orchestrare i passi necessari per rispondere ad un evento iniziale; per ogni passo invia uno specifico evento di processamento ad un canale (Event Channel). Il mediatore non applica nessun tipo di logica, conosce solo i passi necessari per gestire l’evento iniziale e quindi li genera.

### Canale di eventi

Si tratta generalmente di un canale di comunicazione asincrono. Questo può essere di due tipi:

* coda di messaggi;
* topic di messaggi.

### Esecutore di eventi

Contiene la vera logica di business per processo ogni evento. Sono auto contenuti, indipendenti ed scarsamente accoppiati.

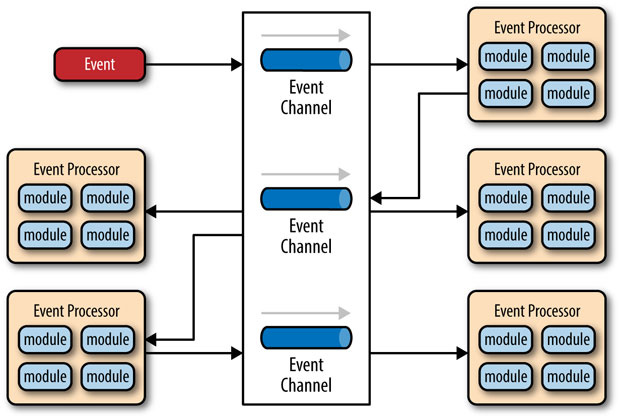
## Broker topology

In questo approccio non è presente un mediatore centrale. Il flusso dei messaggi viene distribuito dai vari esecutori, creando una coda di eventi che generano a loro volta altri eventi. Risulta molto utile nel caso in cui il flusso sia molto semplice.

In questo approccio ci sono due principali componenti:

* un broker che contiene tutti i canali;
* vari esecutori di eventi.

Di seguito si riporta una generica architettura Event Driven Broker Topology.



## Considerazioni

Di seguito si evidenziano alcune vantaggi e svantaggi in maniera analitica:

|  |  |
| --- | --- |
| **Caratteristica** | **Considerazioni** |
| Agilità generale | I cambiamenti sono generalmente isolati e possono essere fatti velocemente con piccoli impatti. |
| Facilità di deploy | È dovuta all’alto disaccoppiamento degli esecutori. Questa nota vale particolarmente per la tipologia Broker in quanto non presenta il mediatore. |
| Testabilità | Richiede strumenti specializzati per generare eventi, questo potrebbe rendere i test di sistema difficoltosi. I test di unità invece sono facilmente implementabili. |
| Scalabilità | La natura indipendente dei componenti rende facile scalare questi in base alle necessità permettendo così un tuning delle risorse molto fine. |
| Facilità di sviluppo | È il principale svantaggio di queste architettura. |

Uno dei principali svantaggi di questo tipo di architettura è la complessità di implementazione, dovuta al fatto che operazioni sono completamente asincrone e distribuite. Si è comunque ritenuta questa architettura nella sua variante Broker Topology adatta allo scopo soprattutto per questioni di performance, scalabilità e facilità di deploy.