SP-Studio di fattibilità

Autore: Simone Ballarin

Data: 12/06/18

Destinatari: Athesys

Diario delle modifiche

Data	Descrizione	Autore
15/06/18	Creazione documento. Inserimento dei capitoli Scopo del documento, Sintesi del documento, Descrizione, Studio del dominio.	Simone Ballarin
18/06/18	Inserimento dei capitoli Motivazioni, Conclusioni, Scelta del client Ethereum, scelta del client Hyperledger, scelta libreria grafica. Verifica.	Simone Ballarin

Scopo del documento

Il seguente documento *SP-Studio di fattibilità* ha lo scopo di fornire una macro descrizione ed un primo approccio relativo ai benefici ed ai costi di una eventuale progettazione e messa in produzione di un applicativo gestionale per il Service Provider (SP) che dovrà funzionare nel contesto di un'estensione del prodotto MonoKee basato su blockchain.

Sintesi del documento

Il documento inizia descrivendo le caratteristiche del prodotto MonoKee e di come il SP si cali in questo contesto. Si prosegue analizzando il dominio applicativo. Da questo emerge un utilizzo da personale specializzato in orario lavorativo. Si è effettuata, poi, un'analisi sui due principali tipi di sviluppo: distribuito o centralizzato. Alla fine di una breve analisi emerge una preferenza per l'ultimo. All'interno del documento sono presenti anche una trattazione di una serie di tecnologie (sia a livello di librerie per la comunicazione con la rete, che di librerie front end) che lo sviluppo di un applicativo di questo tipo potrebbe avere bisogno.

Riferimenti

1. Blockchain: The Dawn of Decentralized Identity (G00303143), Homan Farahmand per Gartner
2. ITIL Service Design Book.
3. blogs.oracle.com/java/integrating-the-ethereum-blockchain-into-java-apps
4. http://ethdocs.org/en/latest/ethereum-clients/choosing-a-client.html
5. github.com/ethereum/yellowpaper

Descrizione

Studio del dominio

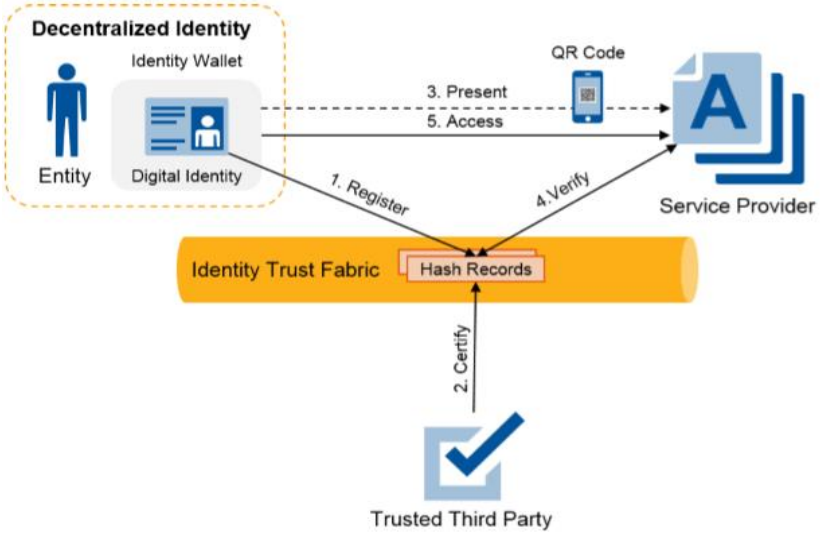
Dominio applicativo

L'applicativo SP dovrà essere usato come strumento abilitatore da parte dei vari fornitori di servizi a partecipare al progetto MonoKee. Da un primo studio si pensa che il target di questi servizi sarà lavorativo, successivamente potrà essere considerato l'introduzione di servizi Consumer. Si tratta sostanzialmente di un'applicazione di tipo Server con scopi essenzialmente di comunicazione. Data la vasta varietà di servizi e di necessità che potrebbe avere il fornitore non risulta definibile un comportamento standard che il SP dovrà tenere, ma si dovrà adattare caso per caso. Possiamo comunque ipotizzare che il suo funzionamento sia necessario solo durante l'orario di ufficio, quindi dalle 7.00 alle 18.00, fuori questi orari sarà possibile fare manutenzione. Nell'ottica dell'introduzione di servizi consumer si dovrebbe comunque tener conto di una disponibilità maggiore. L'applicativo dovrebbe offrire un'interfaccia di manutenzione, accessibile tramite interfaccia grafica da parte del personale del fornitore del servizio. Il software deve essere utilizzato dal personale IT delle varie organizzazione che utilizzano il servizio, per questa ragione si può dare per scontato che l'utente generico possegga delle competenze informatiche avanzate.

Dominio tecnologico

Proposta architettura di massima

Il service provider deve operare come intermediario tra l'IW, l'ITF e il reale fornitore del servizio. Le comunicazioni dovrebbero seguire lo schema di seguito proposto. A seguito dello studio di fattibilità relativo all'IW è emerso come la connessione tra IW e SP debba avvenire tramite protocollo REST. Mentre la comunicazione con l'ITF deve avvenire tramite blockchain. Relativamente alla comunicazione verso il fornitore vero e proprio non si possono fare considerazione in quanto queste possono variare significativamente.



Si evidenziano essenzialmente due principali opzioni per la costruzione di questo applicativo. Il primo è l'utilizzo, anche per esso come per l'ITF, di un approccio totalmente distribuito basato su blockchain. Il secondo approccio consiste i un'applicazione tradizionale.

Sviluppo distribuito

Questo approccio prevede che la logica applicativa sia totalmente affidata a codice eseguito su blockchain. Questo comporterebbe una disponibilità continuativa sempre garantita e altre caratteristiche di

affidabilità e sicurezza. Come punto negativo si evidenzia che una soluzione del genere implicherebbe l'uso di molte tecnologie non completamente mature e di linguaggi in molti casi incompleti. Inoltre questa scelta implicherebbe l'utilizzo della stessa blockchain presente nell'ITF. I vantaggi attribuibili a questo approccio non sono considerati fondamentali al fine di una buona implementazione del SP, di contro gli svantaggi risultano particolarmente pesanti. L'applicazione di un approccio di questo genere anche se possibile risulta sconsigliato. Uno sviluppo di questo tipo, almeno in reti di tipo Permissionless, comporta un cambio di stile di programmazione dovuto all'alto costo delle operazioni. Come descritto nello studio tecnologico Ethereum ciò comporta le seguenti diversità:

- alcuni pattern non risultano applicabili;
- complessità lineari sono difficilmente giustificabili;
- uso di pattern ad hoc.

Sviluppo tradizionale

La seconda opzione risulta essere una più tradizionale applicazione server che comunica tramite librerie alla blockchain. Si consiglia l'uso di linguaggi fortemente tipati quali:

- C++;
- C#;
- Java.

Data l'alta diffusione di Javascript e NodeJS nella comunità Ethereum si consigliano pure questi.

In base al linguaggio scelto e alla tecnologia blockchain scelta, si dovranno utilizzare differenti librerie per effettuare la comunicazione tra la rete e l'applicativo. Presupponendo una scelta di Ethereum si propongono le seguenti librerie:

Linguaggio	Libreria	Note
C++	cpp-ethereum	
C#	Nethereum e un client	Questa soluzione si collega particolarmente bene alla scelta di Xamarin per l'IW.
JS e NodeJS	Web3 e un client	
Java	Web3j e un client	

Di seguito si procederà alla trattazione di alcuni degli strumenti che si ritengono più utili.

Nethereum: è un tool che permette una facile integrazione con il client in applicazioni .NET. Fornisce una suite di librerie open source che aiutano a prototipare applicazione .NET velocemente. E' disponibile anche nel sotto insieme Xamarin. La documentazione è presente è sembra ben strutturata e di ottima qualità. Inoltre potrebbe rappresentare una buona soluzione in caso di scelta di Microsoft Azure Blockchain. Il sito del progetto è il seguente www.nethereum.com.

Web3: è una collezione di librerie che permettono di interagire con un nodo remoto o locale usando una connessione HTTP o IPC. Web3 è presente in npm, meteor, pure js. Per il suo funzionamento è necessario

avere un client attivo nel proprio computer. Web3 supporta Mist e Metamask. Il sito del progetto è il seguente: web3js.readthedocs.io.

Web3J: si tratta di una libreria analoga a Web3 per Java.

Mist: è un browser sviluppato direttamente dal team Ethereum in grado di operare transazioni direttamente nella blockchain senza la necessità di possedere un intero nodo. È estremamente immaturo e non utilizzabile in produzione. Si riporta di seguito il sito del progetto: github.com/ethereum/mist.

Metamask: è uno plugin disponibile per i browser Chrome, Firefox e Opera che permette di interfacciarsi alla rete Ethereum senza la necessità di eseguire in intero nodo della rete. Il plugin include un wallet con cui l'utente può inserire il proprio account tramite la chiave privata. Una volta inserito l'account il plugin farà da tramite tra l'applicazione e la rete.

In caso, invece, si opti per la scelta di Hyperledger la scelta risulterebbe molto più semplice in quanto la blockchain in questione fornisce un'API per la comunicazione REST.

Scelta del client Ethereum

Il client è un componente che implementa il protocollo di comunicazione di Ethereum. Ethereum diversamente da Hyperledger presenta una realtà molto varia e frattagliata. Solo dal punto di vista del client ci sono multiple implementazioni per differenti sistemi operativi ed in differenti linguaggi. Questa diversità viene vista dalla community come un indicatore di salute per l'intero ecosistema. Il protocollo in ogni caso è sempre lo stesso ed è definito nel così detto Yellow Paper in nota 5, in sostanza si basa sull'utilizzo di file Json.

Fino al settembre 2016 erano presente le seguenti alternative:

Client	Language	Developers
Go-ethereum	Go	Ethereum Foundation
Parity	Rust	Ethcore
Cpp-ethereum	C++	Ethereum Foundation
Pyethapp	Python	Ethereum Foundation
Ethereumjs-lib	Javascript	Ethereum Foundation
Ethereum(J)	Java	<ether.camp>
Ruby-ethereum	Ruby	Jan Xie
EthereumH	Haskell	BlockApps

I client appena citati sono accessibili tramite apposite librerie, un buon esempio potrebbe essere web3 per Javascript.

Scelta del client Hyperledger

In caso si dovesse optare per una scelta basata su Hyperledger quale base dell'ITF bisognerà ricadere sull'unica soluzione proposta dal team di sviluppo, cioè quella di utilizzare direttamente una comunicazione REST. Il team offre uno strumento detto Composer con il quale si potrà definire un'interfaccia per operare la comunicazione REST. Al fine di poter gestire efficientemente queste chiamate lato applicazione SP si consigliano le seguenti librerie:

Linguaggio	Librerie	Sito
.NET	WCF REST Starter Kit	www.asp.net/downloads/starter-kits/wcf-rest

	OpenRasta	www.openrasta.org
	Service Stack	www.servicestack.net
Java	Jersey	www.jersey.java.net
	RESREasy	www.jboss.org/resteasy
	Restlet	www.restlet.org
C++	linavajo	www.libnavajo.org
	C++ RESTful framework	www.github.com/corvusoft/restbed
	C++ REST SDK	www.github.com/Microsoft/cpprestsdk

Data l'amplissima diffusione delle API REST e di queste librerie non si procede ad una trattazione analitica.

Conclusioni scelta sviluppo

Considerando quanto precedentemente detto lo sviluppo tradizionale sembrerebbe avrebbe meno incognite e un ampio repertorio di librerie utilizzabili. Si propone, quindi un'architettura del secondo tipo.

Inoltre, si vuole fare notare come l'utilizzo delle soluzioni .NET potrebbero rivelarsi molto vantaggioso in quanto facilmente integrabili con il IW e Azure Blockchain.

Scelta libreria grafica

Dall'analisi applicativa emerge la necessità di creare un'interfaccia da cui gli utenti saranno in grado di configurare il servizio. Di seguito si procederà ad una breve trattazione delle più diffuse tecnologie utilizzate per creare interfacce grafiche.

Ci sono due modi per accedere ad una interfaccia: la prima è l'utilizzo di un sito internet, la seconda consiste nell'utilizzo di un programma locale. Considerato il dominio applicativo si tende a preferire la prima opzione in quanto più immediata e soprattutto accessibile da ogni computer senza necessità di installazione. I framework più consigliato sono:

React: è una libreria JavaScript sviluppata e mantenuta da FaceBook. Permette di costruire interfacce utente dinamiche e complesse rimanendo semplice e intuitivo da utilizzare. Permette una facile creazione di Single page applications (SPA). È una tecnologia ampiameneo utilizzata e diffusa. Il collegamento tra l'interfaccia è la logica avviene tramite la libreria Redux. Il sito del progetto è il seguente: github.com/facebook/react.

Play: è un framework per Java e Scala basato su Akka. Permette la costruzione di interfacce web e mobili. La libreria risulta essere estremamente scalabile e si basa su un funzionamento di tipo asincrono. È ampliamente utilizzata nel mondo Java e ben documentata. Il sito del progetto è il seguente: www.playframework.com.

Vaadin: consiste in un insieme omnicomprensivo di componenti web, framework Java, temi, strumenti per creare interfacce web. Vaadin si può affidare sia alla piattaforma Java, sia a piattaforme web. Rappresenta una piattaforma stabile e ampiamente utilizzata. Il sito del progetto è il seguente: www. vaadin.com.

NancyFX: è un framework estremamente leggere per costruire servizi su .NET e Mono. Usa della lambda per identificare i percorsi relativi e gli argomenti. Utilizza il pattern MVC. Questo risulta particolarmente utile in caso non si usi un Server Windows. Il sito del progetto è www.nancyfx.org.

WT: è una libreria per la costruzione di interfacce Web disponibile per C++. Wt è in grado di gestire tutte le richieste e tutti i rendiring necessari, lasciando la possibilità di focalizzarsi sulle funzionalità. Si basa

sull'uso di widget per comporre le interfacce. Utilizza dei componenti stateful. Il sito del progetto è www.webtoolkit.eu.

I framework precedentemente citati sono tutti ampiamente diffusi e usati. Lo scopo dell'elenco è solo quello di presentare i principali strumenti a disposizione e non quella di fare una comparazione. La scelta dipenderà fortemente dal linguaggio utilizzato dal backend, in ogni caso si risulta ben coperti.

Motivazioni

Aspetti positivi

A seguito dell'analisi sopra proposta sono stati individuati i seguenti aspetti positivi:

- lo sviluppo di un'applicazione server tradizionale comporta uno sviluppo molto semplice e immediato, grazie anche alla disponibilità di un ampio repertorio di librerie;
- la comunicazione con la blockchain risulta in ogni caso facilmente implementabile grazie all'utilizzo di apposite librerie.
- esiste un'ampia scelta di librerie front end per ogni possibile linguaggio di sviluppo.

Fattori di rischio

Invece per quanto riguarda i fattori di rischio, sono emersi i seguenti punti:

- la creazione dell'interfaccia web potrebbe richiedere un alto numero di ore.

Conclusioni

Dal presente studio emerge come la creazione di un SP sviluppato come applicativo server e non come applicazione distribuita possa essere un'ottima opzione implementativa. Lo studio non presenta particolari rischi. Si ritiene, quindi, che un approccio di questo tipo sia fattibile nei tempi dello stage.