Agenzia nazionale per le nuove tecnologie, l'energia e lo sviluppo economico sostenibile

Ministero della Transizione Ecologica

Ricerca di Sistema elettrico

Big Data Integration per la valorizzazione del virtuosismo energetico con tecniche di Blockchain

S. Bergamaschi, L. Ferretti, D. Beneventano, M. Colajanni, G. Simonini, M. Andreolini, L. Gagliardelli, L. Zecchini, F. Magnanini

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Ingegneria "Enzo Ferrari"

Big Data Integration per la valorizzazione del virtuosismo energetico con tecniche di Blockchain

S. Bergamaschi, L. Ferretti, D. Beneventano, M. Colajanni, G. Simonini, M. Andreolini, L. Gagliardelli, L. Zecchini, F. Magnanini

Aprile 2021

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dello Sviluppo Economico - ENEA
Piano Triennale di Realizzazione 2019-2021 - II annualità
Obiettivo: Tecnologie
Progetto: Tecnologie per la penetrazione efficiente del vettore elettrico negli usi finali
Work package: Local Energy District
Linea di attività: "Progettazione di un'applicazione e di una architettura dati per la valorizzazione del virtuosismo energetico" e "Integrazione dell'applicazione e dell'architettura dati per la valorizzazione del virtuosismo energetico"
Responsabile del Progetto: Claudia Meloni, ENEA
Responsabile del Work package: Claudia Meloni, ENEA

Il presente documento descrive le attività di ricerca svolte all'interno dell'Accordo di collaborazione tra ENEA e Dipartimento di Ingegneria "Enzo Ferrari", Università di Modena e Reggio Emilia
Responsabile scientifico ENEA: Nicola Gessa
Responsabile scientifico: Sonia Bergamaschi

Indice

Sommario

Nella prima parte di questo rapporto tecnico si descrive la progettazione di un'architettura per la gestione dei dati che saranno rilasciati dai diversi utenti in forma eterogenea e che quindi devono essere acquisiti e gestiti in maniera appropriata tramite la profilazione per il recupero e la memorizzazione dei dati da diverse sorgenti e utenze (private e pubbliche) e la predisposizione all'uso dei dati per l'analisi, elaborazione e visualizzazione. Vengono presentate le varie architetture proposte e discusse durante il progetto, basate su due requisiti fondamentali

- pre-processare in maniera batch, periodica, i dati, e memorizzarli in modo che siano disponibili per le analisi senza la necessità di ricalcolarli ogni volta;
- integrare dati provenienti da più differenti sorgenti.

Per quello che riguarda il primo requisito verrà considerata una soluzione basata su *data lake*, in particolare sul componente Delta Lake per memorizzare i dati ed eliminare sia i problemi dell'eccessivo tempo di esecuzione nell'import di alcuni dati dal DBMS sia i problemi di import di file di grosse dimensioni.

Per quello che riguarda invece il secondo punto verranno applicate tutte le tecniche e le metodologie di Big Data Integration sviluppate dal DBGroup e centrate sul sistema di Data Integration MOMIS: in questo modo si potrà ottenere una unica sorgente unificata, priva di ridondanze e di conflitti tra dati.

Nella seconda parte di questo rapporto tecnico si descrive la progettazione di alto livello di un sistema di Distributed Application (DApp) per la tokenizzazione dei consumi energetici su tecnologie blockchain per l'incentivazione di comportamenti virtuosi all'interno della comunità energetica. La progettazione ha affrontato diverse attività:

- l'analisi dello scenario e dei casi d'uso al fine di modellare il sistema da realizzare, inclusa l'identificazione degli attori coinvolti, i requisiti funzionali e non funzionali;
- l'analisi dello stato dell'arte nell'ambito delle conoscenze e delle tecnologie utili alla definizione di una soluzione;
- la descrizione del progetto di alto livello della soluzione individuata;
- la descrizione di un meccanismo di gestione di chiavi crittografiche pensato la gestione sicura ed efficiente di dati memorizzati su blockchain.

2 USE CASES AND SCENARIOS

In questa sezione vengono raccolti i principali use case e scenari che sono alla base e che costituiscono requisiti fondamentali del progetto. Questi use case derivano da documentazione messa a disposizione da ENEA; scopo di questa sezione e di riassumerli brevemente, rimandando appunto ai documenti già disponibili per ulteriori dettagli.

2.1 *SelfUser Use case*

Caso d'uso sulla raccolta e comunicazione dei dati di consumo e produzione di un blocco di appartamenti tramite il servizio di gestore dei meter che eseguono le misurazioni dei consumi.
Descrizione sommaria e semplificata del caso d'uso di monitoraggio dei dati di un condominio dotato di capacità di produzione energia e di smart meter per ogni utenza.
Questa descrizione viene definita nell'ambito di un progetto che ha un duplice obiettivo:

1) massimizzare l'autoconsumo secondo la definizione dell'Autoconsumo Collettivo (di cui al decreto milleproroghe del 2020) favorendo la consapevolezza dell'utente ed individuando le modifiche di comportamento auspicabili, in maniera da meglio sfruttare la capacità di produzione autonoma con fotovoltaico per soddisfare i consumi del condominio (ovvero massimizzare autoconsumo).
2) promuovere, quindi comunicare, l'approccio evidenziandone i vantaggi ed i benefici.

Tuttavia, per le caratteristiche generali se ne possono trarre indicazioni per una più ampia classe di applicazioni per le Comunità Energetiche.

Lo scenario

L'attività si svolge all'interno di un progetto dimostrativo incentrato su un condominio che viene usato come pilota per promuovere l'approccio dell'autoconsumo e per individuare i parametri ottimali per una CE (comunità energetica) in edilizia civile. Si tratta di un condominio con n appartamenti, utenze condominiali e capacità di produzione fotovoltaica. I dati vengono raccolti tramite service di terzi che si interfacciano con la piattaforma ENEA. C'è una attività di messa a punto di un **modello** elettrico della rete locale condotta da un soggetto esterno che però è a fini di studio e tuning dei parametri e non è detto che interagisca con l'interfaccia.
In prospettiva la **SCP (Smart City Platform) di ENEA** dovrebbe rendere disponibili all'esterno dati di sintesi (KPI) in formato standard Urban Dataset tramite API REST ad uso di applicazioni di terzi. La parte relativa agli usi interni agli appartamenti ed alla caldaia viene per ora tralasciata.
Di seguito l'architettura generale.

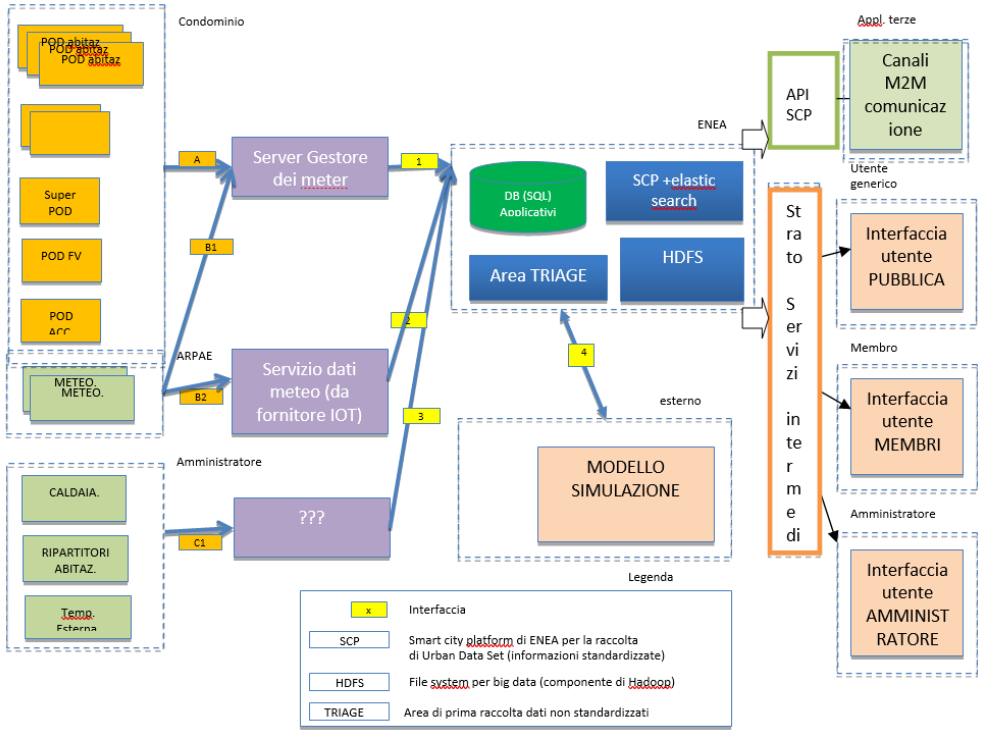

Figura 1 - SelfUser Use case: Architettura generale

I Casi d'uso che possono essere realizzati su questo scenario sono relativi a diversi flussi di dati:

1. Raccolta dati per il sistema di simulazione -> **letture dei dati di consumo fatte al secondo**
2. Raccolta dati per il monitoraggio -> **vengono presi dati quartorari.**
3. Visualizzazione dati ad uso utente generico (pubblico)
4. Visualizzazione dati ad uso membro della comunità
5. Visualizzazione dati ad uso amministratore

Per ora la parte dati meteo non viene considerata.

SelfUser - Caso 1 (Dati Consumo istantanei al secondo)

Denominazione	**Dati Consumo istantanei al secondo**
Descrizione	dati di consumo/produzione **AL SECONDO** raccolti da appartamenti privati e condominio tramite servizio offerto da server di ESCO fornitore degli smart meter ed **inviati ogni settimana** (da dispositivi NILM)
Tipologia dei DATI e granularità	dati istantanei da singolo POD al secondo
Caratteristica flusso di dati	flusso continuo periodicità: invio settimanale; intervallo temporale: almeno un anno (da ottobre 2020 in avanti)
Canale recupero dei dati	inviati via SFTP su server **DATAHALL**
Formato	csv
Numerosità e dimensione	Numero record: 56 POD x 86400 letture (1 al secondo) x 7 giorni = 33.868.800 righe/settimana

SelfUser - Caso 2 (Dati Consumo istantanei quartorari)

Denominazione	**Dati Consumo istantanei quartorari**
Descrizione	dati di consumo/produzione **AL QUARTO d'ORA** raccolti da appartamenti privati e condominio tramite servizio offerto da server di ESCO fornitore degli smart meter (da dispositivi NILM e CHAIN2) **inviati ogni quarto d'ora**
Tipologia dei DATI e granularità	dati istantanei da singolo POD ogni quarto d'ora
Caratteristica flusso di dati	flusso continuo periodicità: invio quartorario intervallo temporale; almeno un anno (da ottobre 2020 in avanti)
Canale recupero dei dati	inviati via MQTT su server **DATAHALL** tramite rotta KARAF+CAMEL
Formato	csv
Numerosità e dimensione	56 POD x 1 lettura ogni 15 minuti (35040 invii all'anno)

SelfUser - Caso 3 (dati centraline meteo)

Denominazione	**dati centraline meteo**
Descrizione	dati da centraline meteo presso condominio; dati raccolti **OGNI MINUTO** su server del fornitore che rende disponibile API per scaricarli (con accesso fino a ogni 5 minuti)
Tipologia dei DATI e granularità	dati da server di centraline - temperatura - irraggiamento solare
Caratteristica flusso di dati	Dati al minuto, sincronizzazione ogni 15 minuti
Canale recupero dei dati	da recuperare da API del fornitore e successivo storage su server DATAHALL
Formato	JSON
Numerosità e dimensione	4 centraline x 1 lettura ogni 15 minuti (35040 invii all'anno)

Documento di riferimento **WP450-002-v4-schema_caso_uso_self_user**

2.2 PELL Use case

Il caso studio PELL Illuminazione Pubblica è caratterizzato da due tipologie di dati distinte:

- statica, in cui le informazioni relative agli impianti di illuminazione sono immagazzinate in un DB relazionale MySQL
- dinamica, in cui le informazioni relative alle misure dei contatori installati nei quadri elettrici sono immagazzinate in HDFS di Hadoop sotto forma di Urban Dataset (Counter Reading).

Alcuni dei servizi offerti (es. KPI), richiedono l'elaborazione di informazioni provenienti da entrambe le tipologie di dato. In questo scenario, è stato adottato Spark [1] come data integrator.
La Figura 2 mostra lo scenario AS IS, in cui un utente richiede il calcolo di un servizio da web, che avvia il processo di retrieving dei dati in Spark. L'importazione dei dati da MySQL e Hadoop dà luogo a due DataFrame, su cui vengono fatte le opportune operazioni preprocessign e di calcolo. Il risultato ottenuto è ancora in formato DataFrame, che può essere poi servito in formato parquet (ma potrebbe essere utilizzato un qualsiasi altro formato, quale JSON o CSV), leggibile dalle librerie JS e quindi servibile su web.

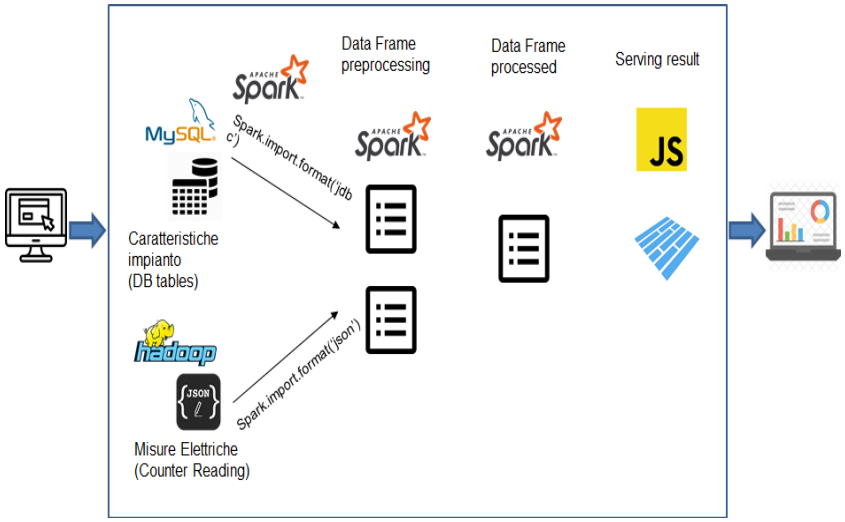

Figura 2 - Scenario AS-IS

In questo scenario i dati dalle due tipologie di sorgenti vengono ad ogni richiesta ricalcolati, per cui è un'inefficienza che al crescere della dimensione diventa sempre più marcata.

Pell - Caso 1

Tabelle:

- `counterreading`. Tabella contenente tutte le misure derivanti dall'UD CounterReading
- `kpivalues`. Tabella contenente i valori dei kpi .
- `kpidescriptions`. Tabella contenente le descrizioni dei kpi.
- `t_frequencies`. Lookup table per le frequenze di calcolo dei kpi.

Documento di riferimento **WP224-027-v1-Architettura_Dati_-_PELL_use_case**

2.3 *Local Energy Community*

I dati che verranno raccolti per il funzionamento della LEC riguarderanno principalmente la descrizione di utente energetiche e loro consumi/produzione/storage di energia. Saranno di due tipologie:

- Dati statici: raccolti una sola volta (ad esempio al momento dell'iscrizione) e conservati in tabelle di un DB relazionale. Tali dati possono riguardare delle persone, degli edifici o appartamenti, etc;
- Dati dinamici: dati raccolti da sensori in continuo o con periodicità differenti a seconda dei casi o recuperati da sistemi esterni alla LEC ad esempio aggregatori esterni, fornitori di energia o del servizio elettrico etc.

L'insieme dei **dati statici** contiene:

- Anagrafica degli utenti (nome, mail, etc);
- Tipologia di utenza e caratterizzazione (domestica, aziendale, condominiale);
- Georeferenziazione dell'edificio o degli edifici sottoposti a misurazione/monitoraggio;
- Caratteristiche dell'involucro e degli edifici (infissi, dispositivi tecnici etc);
- Tipologia di tecnologia per la climatizzazione invernale o estiva;
- Definizione dei carichi elettrici specificando potenza installata, differibilità, attenuabilità classe energetica etc. Il carico elettrico è inteso come qualsiasi oggetto che consuma corrente;
- Caratterizzazione delle sorgenti da fonte rinnovabile;
- Potenza dell'accumulo.

L'insieme dei **dati dinamici** contiene:

- Dati di consumo (con differente intervallo di tempo, tipicamente il quarto d'ora);
- Dati di produzione (con differente intervallo di tempo, tipicamente il quarto d'ora);
- Andamento dell'accumulo (carica/scarica con differente intervallo di tempo, tipicamente il quarto d'ora).

Documento di riferimento
NN224-037-v1-Caratterizzazione_dei_dati_per_la_Local_Energy_Communities_-_20201020

3 BIG DATA PLATFORM ARCHITECTURE

In questa sezione vengono presentate le varie architetture proposte e discusse durante il progetto. Il punto di partenza è stato lo scenario AS IS mostrato in Figura 2 - Scenario AS-IS si possono individuare due principali requisiti

- pre-processare in maniera batch, periodica, i dati, e memorizzarli in modo che siano disponibili per le analisi senza la necessità di ricalcolarli ogni volta;
- integrare dati provenienti da più differenti sorgenti.

Per quello che riguarda il primo requisito è stato individuato nel componente Delta Lake [DELTA] lo strumento opportuno: memorizzando i dati nel Delta Lake si dovrebbero eliminare sia i problemi dell'eccessivo tempo di esecuzione nell'import di alcuni dati dal DBMS sia i problemi di import di file di grosse dimensioni.

Per quello che riguarda invece il secondo punto verranno applicate tutte le tecniche e le metodologie di Big Data Integration sviluppate dal DBGroup e centrate sul sistema di Data Integration MOMIS: in questo modo si potrà ottenere una unica sorgente unificata, priva di ridondanze e di conflitti tra dati.

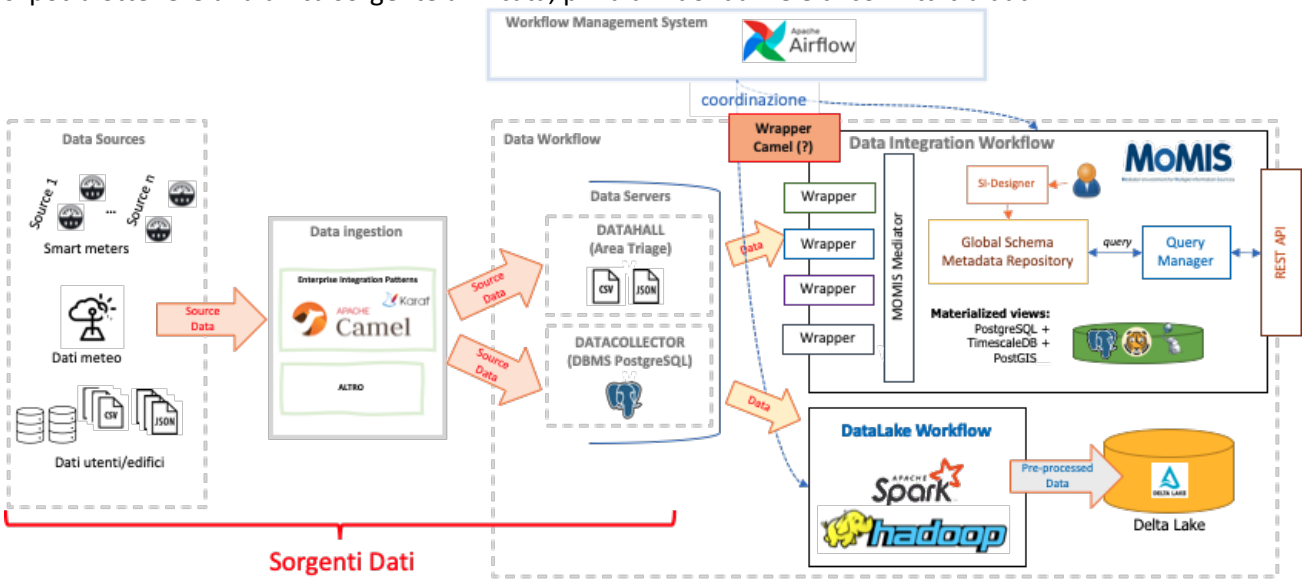

Figura 3 Overview dell'architettura – PRIMA VERSIONE

Con riferimento alla Figura 3.

1. **Data Source**: Le sorgenti dati sono molteplici e potranno sicuramente aumentare e variare nel tempo. Le tipologie principali sono quelle descritte nella sezione precedente.
2. **Data Ingestion:**
   a. Invio dati tramite JSON/CSV (es: email)
   b. Apache Camel (+ Karaf): Java library per integrare diversi framework: muovere dati da un sistema a un altro. Caratteristiche:
      i. DSL per scrivere connettori (Camel Language)
      ii. Propone una serie di pattern notevoli, es: splittare un messaggio in linee multiple, effettuare un *routing* in base al contenuto del messaggio
3. **Data Servers:**

   L'elemento centrale è il server DATAHALL, su cui si trova la cosiddetta "area triage", riferimento per lo storage di dati non strutturati, che contiene i file (CSV, JSON, ecc.) così come sono acquisiti dall'esterno dagli strumenti per la data ingestion, senza applicarvi ulteriori operazioni di processing. Da DATAHALL si può accedere via remote port forwarding al server DATACOLLECTOR, su cui si trova un DBMS PostgreSQL, riferimento per lo storage di dati strutturati (contiene ad esempio i dati su

consumi e presenze relativi alle palazzine gemelle A e B del centro ENEA di Bologna). È contemplata inoltre la possibilità di sfruttare altri DMBS di test presenti su altri server nella rete locale.

4. **Data Workflow:**

   Questo modulo funzionale ha il compito di integrare i dati e renderli fruibili in maniera sempre aggiornata, con uno schema omogeneo.

   a. Per garantire il massimo di flessibilità sono individuati due workflow (e relativi moduli funzionali): *Data Integration workflow* per integrare i dati, che fornisce il principale punto di accesso ai dati unificati per ogni applicazione;
   b. *Data Lake workflow* che permette di conservare tutti i dati storici in maniera "grezza" (i.e., non integrata) e manipolabile all'occorrenza da utenti esperti tramite la piattaforma DeltaLake/Spark.
   c. Entrambi i sistemi dovrebbero essere coordinati tramite **Data Workflow management (c**), con in quale vengono programmati i meccanismi necessari di coordinamento per il flusso dei dati (es: nuovi dati in input che scatenano trigger e ETL; *alert di failures*; *error-handling*).

Di seguito la descrizione dei moduli appena elencati.

3.1 *Data Integration Workflow - MOMIS*

La Big Data Integration (BDI) è l'ambito di ricerca principale del gruppo di ricerca database (DBGroup - www.dbgroup.unimore.it), dove da oltre 20 anni sono sviluppate tecnologie per supportare l'estrazione e l'integrazione di dati, utili alla generazione di nuova conoscenza. È stato realizzato il sistema di Data Integration MOMIS (Mediator EnvirOnment for Multiple Information Sources) in grado di integrare dati, provenienti da fonti eterogenee e distribuite in maniera semi-automatica. Una versione open source di MOMIS viene distribuita da DataRiver (www.datariver.it)

Il sistema MOMIS è basato sull'architettura wrapper/mediatore (vedere Figura 4 - Il sistema di Data Integration MOMIS): il wrapper rende disponibile una sorgente dati locale da integrare, il mediatore genera uno schema mediato, detto anche Global Virtual View (GVV) o Global Schema (GS), degli schemi delle sorgenti locali. Una volta ottenuto lo schema globale, l'utente interroga lo schema ed ottiene, in modo totalmente trasparente, una vista completa ed unificata dei dati contenuti nelle sorgenti locali. In particolare, MOMIS esegue Data Fusion, ovvero il processo di fusione di più record che rappresentano lo stesso oggetto del mondo reale in una rappresentazione unica, coerente e pulita; a tale scopo, nel sistema MOMIS sono disponibili diverse strategie di gestione dei conflitti.

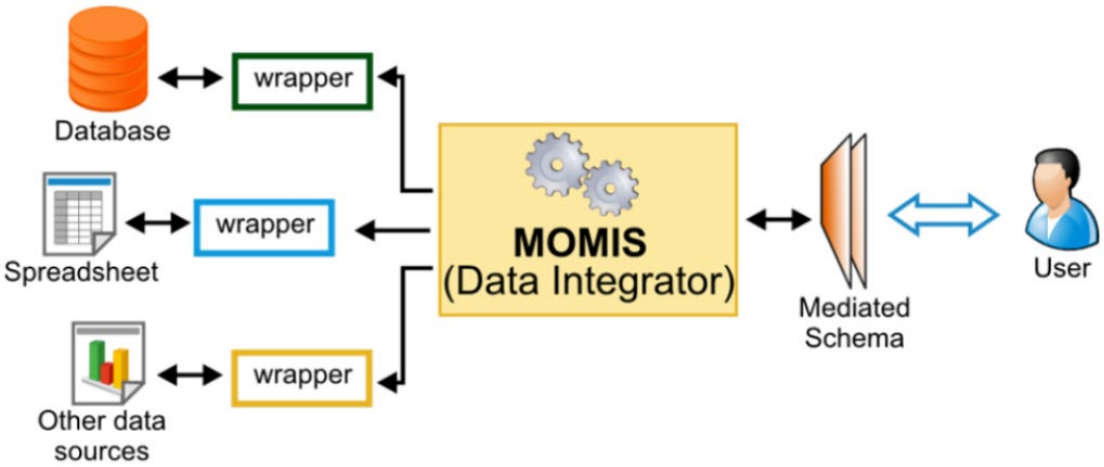

Figura 4 - Il sistema di Data Integration MOMIS

Una delle caratteristiche fondamentali del sistema MOMIS è quello di generare in modo semi-automatico il Global Schema, come una vista riconciliata, integrata e virtuale delle sorgenti sottostanti; riportiamo in modo molto sintetico le fasi principali di generazione del Global Schema:

1. *Local Source Selection*: selezione delle sorgenti dati e dei relativi elementi (tabelle, file) da includere nel Global Schema
2. *Local Source Annotation*: assegnamento di un significato agli elementi (tabelle, attributi, …) rispetto ad una ontologia e/o una risorsa lessicale, quale WordNet (https://wordnet.princeton.edu)
3. *Semantic Relationships Extraction*: generazione di un insieme di relazioni semantiche intra/inter-schema sulla base delle annotazioni fatte nella fase precedente
4. *Global Class Generation*: generazione di cluster di elementi sulla base delle relazioni semantiche (global class GC) e mapping rispetto agli elementi locali
5. *Mapping Refinement*: raffinamento dei mapping per definire particolari trasformazioni dei dati locali rispetto alla Global Class. In sostanza, lo schema globale generato automaticamente può essere rifinito manualmente dal System Integration Designer.

Per la stessa integrazione è possibile produrre più schemi globali in base alle esigenze.

Gli schemi globali prodotti possono essere interrogati tramite il Query Manager dall'utente finale e/o applicazioni di terze parti.
Una caratteristica fondamentale del sistema di Data Integration MOMIS è il fatto che è un sistema di Data Integration *virtuale*, ovvero i dati risiedono nelle sorgenti e vengono recuperati solo durante un'interrogazione, rendendo il processo trasparente all'utilizzatore. In questo modo i dati non vengono replicati e la risposta ad ogni interrogazione riporta i dati aggiornati.
D'altra parte, il sistema di Data Integration MOMIS consente di *materializzare* delle classi globali (*materialized views*) sulla base di opportune politiche di ottimizzazione: dati utilizzati spesso o di particolare importanza vengono materializzati a livello del mediatore e quindi le interrogazioni che li coinvolgono non necessitano di accedere alle relative sorgenti per recuperare i dati.
Per una descrizione generale del sistema MOMIS fare riferimento alla seguente pubblicazione: [2].
Per una documentazione completa del sistema MOMIS fare riferimento al sito DBGROUP https://dbgroup.unimore.it/site/home/research.html

3.2 *Data Lake Workflow - Delta Lake*

I dati grezzi (quindi senza alcun pre-processing) vengono memorizzati all'interno di un data lake che si appoggia su HDFS—la dimensione stimata dei dati e le risorse a disposizione lo permettono. In particolare, si è scelto di utilizzare *Delta Lake* [3] come data lake. *Delta Lake* è un progetto Open Source che consente di gestire grandi quantità di dati utilizzando sistemi di memorizzazione esistenti come HDFS.
Le motivazioni per cui abbiamo scelto *Delta Lake* sono le seguenti:

- Supporta le transazioni ACID garantendo il massimo livello di isolamento (*serializability*). Grazie a questo è possibile eseguire contemporaneamente letture e scritture senza dover preoccuparsi dell'integrità dei dati.
- Utilizzo di un formato aperto. I dati sono memorizzati nel formato open source Apache Parquet. Questo formato fornisce il vantaggio di attuare una compressione dei dati, e quindi lo spazio richiesto per la memorizzazione dei dati è di molto inferiore rispetto a quello originario (tipicamente CSV).
- I metadati vengono gestiti come se fossero dati, in questo modo è possibile gestire tabelle della dimensione del petabyte.
- Integrazione con Apache Spark. Apache Spark può leggere e scrivere nativamente su *Delta Lake*, consentendo così una completa integrazione tra i due sistemi. In questo modo si ha accesso a tutto

lo stack di Spark, consentendo di eseguire elaborazioni efficienti sui dati sia in modalità batch che streaming.

- Data versioning. Utilizzando un sistema di log, *Delta Lake* consente di tenere traccia delle operazioni che vengono eseguite sui dati, consentendo di recuperare versioni precedenti (*rollback*) dei dati che sono state sovrascritte.
- Evoluzione dello schema. *Delta Lake* consente di modificare lo schema dei dati, che normalmente può evolversi nel tempo, senza dover modificare i dati che si erano raccolti fino a quel momento.
- Memorizzando i dati in modo strutturato consente facilmente di trovare tutti i record relativi ad uno specifico utente, garantendo di poter rispettare la GDPR (es. nel caso in cui si debbano eliminare tutti i dati di uno specifico utente).
- Maturità dello strumento. *Delta Lake* viene utilizzato da numerosi vendor di servizi cloud come Databricks e Microsoft Azure, che dichiarano di essere in grado di gestire Petabyte di dati con questo strumento.

Lo scopo del Data Lake workflow è quello di supportare l'analisi di dati da parte di utenti avanzati che hanno necessità di accedere ai dati grezzi per ricavare nuove informazioni che non sarebbe possibile ricavare dai dati integrati che provengono dal Data Integration workflow.
Ad esempio, i dati grezzi dei sensori raccolti a livello del secondo sono memorizzati all'interno di Delta Lake, mentre nel Data Integration Workflow sono aggregati al livello del quarto d'ora, quindi se ad un utente avanzato servono i dati al secondo dovrà utilizzare il Data Lake workflow. Inoltre, essendo Delta Lake integrato con tutto l'ecosistema Apache Spark, permette l'utilizzo naturale di tutte le sue librerie (es: MLlib per il machine learning) e della computazione streaming e batch.
Per una descrizione generale del sistema *Delta Lake* fare riferimento alla seguente pubblicazione:
[4]. Gli aspetti fondamentali del Delta Lake sono stati riassunti ed elaborati con riferimento alle problematiche del progetto nel seguente documento [5].

3.3 *Data Workflow management - Airflow*

Lo scopo del Data Workflow management è quello di gestire l'orchestrazione dei moduli architetturali. Infatti, quando nuovi dati vengono resi disponibili dalle sorgenti, devono essere presenti meccanismi che li identifichino automaticamente e scatenino l'avvio degli opportuni workflow (ad esempio l'avvio di un Delta Lake workflow).
Inoltre, è necessario che lo strumento di gestione dei workflow sia in grado di: (i) gestire la presenza di errori imprevisti, in modo da prevedere meccanismi di risoluzione e comunicazione di questi; (ii) permettere la composizione di workflow a partire da moduli software componibili all'esigenza (es: per gestione di errori).
Lo strumento deve essere programmabile, semplice da utilizzare e campabile con ogni modulo software che può comporre un data workflow (i.e., indipendente da linguaggi di programmazione e piattaforme software utilizzate). Gli sviluppatori dei data workflow devono poter programmare con semplici script i meccanismi che gestiscono: il trigger dei workflow, la lettura dei dati, come devono essere loggati/gestiti gli errori.

Dati i requisiti sopra elencati è stato individuato Apache Airflow (http://airflow.apache.org) come tool candidato per il Data Work Management, infatti Apache Airflow [6] permette di:

1) Gestire workflow di task, programmando semplici script in linguaggio Python:
   a) I task possono essere eseguiti da tool di ogni tipo (i moduli sono delle "blackbox") e implementati con qualsiasi linguaggio di programmazione (i.e., Python è solo il "collante")
   b) gli script definiscono le dipendenze tra l'output di un task e l'inizio di un altro task
      i) Task eseguiti con un ordine ben preciso, rappresentabili con un DAG: *directed acyclic graph*
2) Gestire lo scheduling, il monitoraggio e gli eventuali errori dei task che costituiscono un data workflow
   a) Permette di specificare tramite script le azioni da prendere se qualche task fallisce (trigger rules)

b) Ha una user interface con funzionalità molto ricche per monitorare il workflow anche da interfaccia grafica.

3.4 *Architettura – varianti PRIMA VERSIONE*

Questa sezione descrive le varie architetture delineate e discusse durante il progetto, evidenziandone di ciascuna i pro e i contro, anche con riferimento ai singoli use case. Le principali differenze tra queste varie architetture risiedono in particolare nel ruolo del Delta Lake e nel flusso dei dati da/verso tale componente, soprattutto rispetto al sistema di Data Integration.

3.4.1 Variante architettura: Delta Lake come generica sorgente

Questa variante nasce dalla constatazione che nella prima versione dell'Architettura di Figura 3 Overview dell'architettura – PRIMA VERSIONE la parte relativa Delta Lake & Spark costituisce un percorso alternativo rispetto a quello di Data Integration: come discusso in precedenza il Delta Lake workflow ha lo scopo di immagazzinare tutti i dati grezzi per eventuali ulteriori elaborazioni.
D'altra parte, ci sono diverse situazioni e motivazioni in cui è molto utile che il sistema di Data Integration possa accedere anche ai dati memorizzati nel Delta Lake, ovvero è importante considerare il Delta Lake stesso come una generica sorgente del sistema di Data Integration. Le principali motivazioni di questa scelta sono le seguenti:

- Storico dei dati: visto che Delta Lake implementa un meccanismo di *data versioning* è possibile recuperare versioni precedenti dei dati che possono essere utili in fase di analisi;
- Analisi specifica sui dati: nel caso in cui sia necessario recuperare la versione originale di un dato nel suo formato originale è possibile recuperarla da Delta Lake in cui è memorizzata.

In Figura 5 viene introdotta quindi una variante dell'architettura originale per considerare appunto il Delta Lake come generica sorgente del Data Integration.

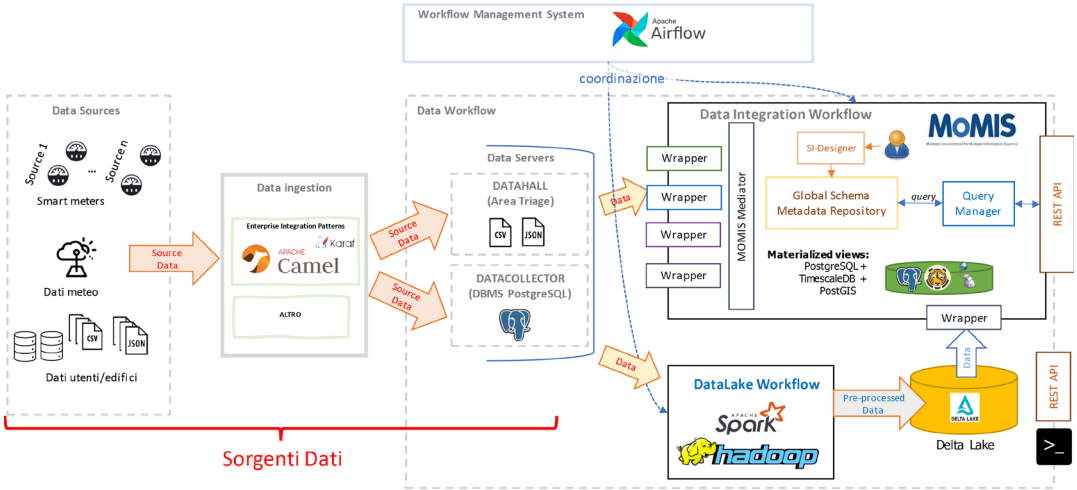

Figura 5 - Variante architettura: Delta Lake come generica sorgente del Data Integration

L'efficacia e l'efficienza di tale soluzione dovranno essere valutate in fase di implementazione, considerando ovviamente le peculiarità del sistema di Data Integration utilizzato. In particolare, nel caso di utilizzazione di un sistema di Data Integration *virtuale* quale sistema MOMIS, si deve considerare innanzitutto il fatto che una generica richiesta, una generica query sul sistema di Data Integration comporta che venga inviata una *query locale* al sorgente dati Delta Lake per riceverne un risultato che verrà elaborato/integrato dal Query Manager.
Il problema principale da mettere in conto in tale situazione è il fatto che Spark, pur mantenendo online l'infrastruttura (master e workers), mantiene i dati a livello di applicazione, quindi ogni volta che viene mandata in esecuzione una richiesta di dati da una tabella deve ricaricare i dati in memoria prima di eseguire

l'interrogazione: questo potrebbe causare una latenza troppo alta e quindi un rallentamento generale del sistema di Data Integration.
Per questo motivo sono state valutate ulteriori varianti all'architettura, varianti che prendono in considerazione anche la possibilità prevista nel sistema di Data Integration MOMIS di *materializzare* alcuni risultati tramite la componente *Materialized views* mostrata in Figura 5 - Variante architettura: Delta Lake come generica sorgente del Data Integration.
Queste varianti sono discusse nella prossima sezione.

3.4.2 Variante architettura: Delta Lake per le materialized view

Una prima variante da prendere in considerazione e da valutare in fase di implementazione è la seguente: pur conservando la possibilità da parte del sistema di Data Integration di accedere al Delta Lake come ad una generica sorgente dati si dovrebbe ridurre al minimo la lettura dati dal Delta Lake per evitare i problemi discussi sopra. In particolare, si dovrebbe utilizzare il Delta Lake quasi esclusivamente come una sorgente di Dati Storici, come discusso nella sezione precedente.
Da questa considerazione scaturisce un'altra variante molto importante da prendere in considerazione, quella di utilizzare il Delta Lake principalmente *in scrittura* per memorizzare i risultati del sistema di Data Integration. Di conseguenza la materializzazione dei dati (*materialized view*) del sistema di Data Integration non dovrebbe avviene esclusivamente su un DBMS (quale ad esempio il framework Postgres con PostgresSQL, TimescaleDB e PostGIS mostrato in Figura 5) ma potrebbe avvenire anche sul Delta Lake. Con tale configurazione, il sistema di Data Integration MOMIS potrebbe gestire non solo il Data Integration workflow ma anche il, o almeno parte del, Data Lake workflow (vedi Figura 5).
Naturalmente il confronto e la discussione sui vari scenari possibili possono essere fatti solo con riferimento alle specificità del sistema di Data Integration, e quindi vengono demandate alla fase di implementazione.

3.5 *Interazione tra il sistema di Data Integration ed il Delta Lake*

In questa sezione vogliamo discutere i principali aspetti emersi nelle sezioni precedenti e relativi all'interazione tra il sistema di Data Integration MOMIS ed il Delta Lake. Innanzitutto, precisiamo che questo è un nuovo aspetto, una nuova configurazione del sistema MOMIS rispetto a quelle note, trattate/implementate nel passato.
Nella precedente sezione abbiamo evidenziato il problema di una latenza troppo alta - e quindi di un rallentamento generale del sistema di Data Integration – quando il Delta Lake viene interrogato dal sistema MOMIS. Innanzitutto, constatiamo che è sicuramente necessario un intervento a livello della infrastruttura che gestisce l'interazione tra il sistema MOMIS e Spark (da un punto di vista concettuale, possiamo dire a livello del wrapper del Delta Lake).
D'altra parte, è altrettanto importante considerare anche aspetti legati all'elaborazione delle interrogazioni da parte del Query Manager; ricordiamo infatti che il Query Manager riscrive una generica query generando un insieme di subquery (*query locali*) che verranno eseguite sulle sorgenti locali collegate tramite wrapper e che i risultati parziali ottenuti dalle sorgenti verranno poi *fusi* dal Query Manager per ottenere una risposta unificata.
Considerato che nel Delta Lake ci sono migliaia di *tabelle* con enormi dimensioni, si dovrebbero prestare particolare attenzione alle ottimizzazioni effettuabile dal Query Manager, ottimizzazioni basate soprattutto sulla *semplificazione/riduzione* delle query locali:

1. *pushdown* delle condizioni sulle query locali, per massimizzare la selettività delle query locali, in modo da ridurre i dati da trasferire
2. semplificazione delle operazioni di full-outer join in inner join;
3. *riduzione* del numero di query locali, eliminando quelle ridondanti

Tutte queste ottimizzazioni, che sono di natura generale e quindi applicabili ad ogni sorgente (vedere la seguente tesi di dottorato http://dbgroup.ing.unimo.it/tesi/phdcarlos.pdf), sono sicuramente più auspicabili nel caso di una sorgente come il Delta Lake.

Un altro importante aspetto che si deve prendere in considerazione nell'interazione tra il sistema di Data Integration MOMIS ed il Delta Lake è quello relativo all'utilizzo del Delta Lake per le *materialized view* (vedi sezione precedente). Premesso che le *materialized view* per essere utili devono contenere dati aggiornati, è chiaro che oltre alla semplice soluzione di ri-creare e salvare l'intera view, si devono valutare anche opportune politiche di modifica delle view esistenti. Nel caso di materialized views nel Delta Lake, non si può naturalmente prescindere dalle sue peculiarità e specificità; a tale riguardo segnaliamo che nell'articolo di presentazione del Delta Lake c'è una interessante discussione a riguardo (Sezione 4.2 Efficient UPSERT, DELETE and MERGE), in particolare con riferimento all'SQL MERGE statement.

4 Blockchain Architecture for Transparent Energy Tokenization

In questa sezione si descrive la progettazione di alto livello di un sistema di *Distributed Application (DApp)* per la *tokenizzazione* dei consumi energetici su tecnologie blockchain per l'incentivazione di comportamenti virtuosi all'interno della comunità energetica. La progettazione ha affrontato diverse attività:

- l'analisi dello scenario e dei casi d'uso al fine di modellare il sistema da realizzare, inclusa l'identificazione degli attori coinvolti, i requisiti funzionali e non funzionali (Sezione ?);
- l'analisi dello stato dell'arte nell'ambito delle conoscenze e delle tecnologie utili alla definizione di una soluzione (Sezione 4.2);
- la descrizione del progetto di alto livello della soluzione individuata (Sezione 4.3);
- la descrizione di un meccanismo di gestione di chiavi crittografiche pensato la gestione sicura ed efficiente di dati memorizzati su blockchain (Sezione 4.4);

4.1 *Analisi dello scenario e modellazione del sistema e dei requisiti*

Il progetto richiede lo studio di una soluzione basata su tecnologie blockchain per la gestione e la valorizzazione dei comportamenti in tema di consumi energetici. La soluzione deve fornire alcune caratteristiche fondamentali che la distinguono da una tipica applicazione Web centralizzata, quali: l'impiego di *architetture distribuite* che non richiedono di assumere come fidato un singolo attore o componente; l'impiego di tecniche per garantire la *trasparenza* delle operazioni presso gli utenti finali; la garanzia di impiego di tecniche per mantenere la *privacy* dei dati degli utenti. A queste caratteristiche si unisce la necessità di ottenere una soluzione *scalabile* e *performante* impiegabile nell'ambito di workload realistici in termini di quantità di dati gestiti e di operazioni effettuate sul sistema. Dal punto di vista più strettamente tecnico, si richiede l'impiego di conoscenze pubbliche e tecnologie *open* (nessun vincolo nell'ambito di licenze proprietarie o di impiego di tecnologie che potrebbero implicare *lock-in*). Implicitamente, si introduce anche il requisito di impiegare conoscenze e tecnologie *consolidate*, in grado di garantire ottime tolleranze in termini di *affidabilità* e di *sicurezza* del sistema finale.

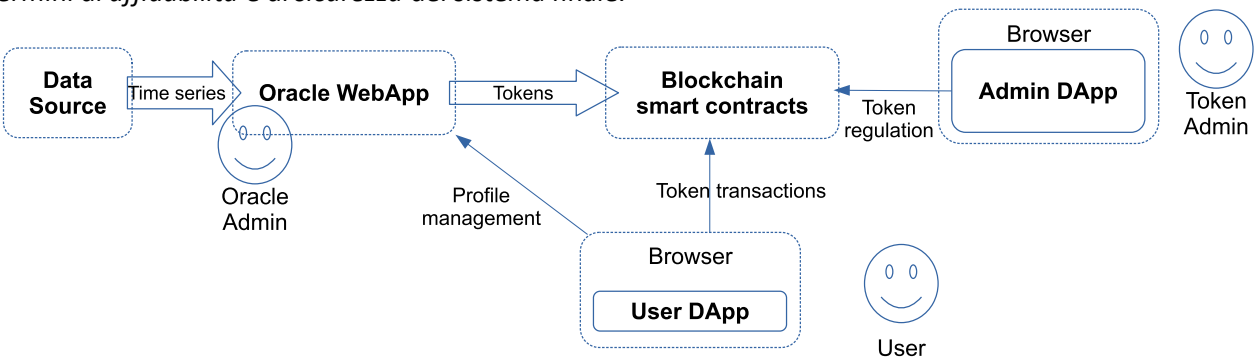

Figura 6 - Schema di alto livello dello scenario e della soluzione di riferimento

In Figura 6 si propone lo schema di alto livello dei componenti e dei ruoli dello scenario. Dal punto di vista funzionale, il sistema deve realizzare da alcune macro-componenti di alto livello: **Oracolo**, **Blockchain smart contracts** e **Distributed Applications (DApp)**. L'Oracolo si pone a valle dei componenti di estrazione e integrazione dei dati analizzati nei precedenti capitoli, e da essi riceve serie temporali che che rappresentano i consumi di ciascun utente aggregati per un certo intervallo di tempo. L'oracolo deve poter applicare dei filtri

sui dati ricevuti per selezionare i dati rilevanti o applicare degli algoritmi di analisi delle serie temporali per individuare informazioni di interesse, ad esempio comportamenti virtuosi degli utenti. Il sistema non deve definire in modo preventivo i filtri o gli algoritmi da applicare, né queste logiche devono essere studiate in questa fase progettuale, ma deve permettere l'impiego di certe classi di logiche e la configurabilità delle stesse da parte di **amministratori del sistema**. A valle dei filtri e delle analisi, l'oracolo definisce le regole di assegnazione dei token agli utenti e ne pilota l'inserimento nei sistemi blockchain distribuiti. La DApp include la logica applicativa in esecuzione sulle tecnologie blockchain (gli *smart contract*) per consentire agli **utenti** di consultare e interagire con il sistema dei token, scambiandoli con altri utenti o utilizzandoli per acquisire servizi messi a disposizione nell'ecosistema blockchain (questi servizi sono considerati componenti esterni non studiati dal sistema in analisi). Per consentire queste funzionalità, la gestione dei token deve essere realizzata con sistemi standard noti e interoperabili. Gli smart contract devono essere configurabili, per consentire la modifica delle regole di scambio dei token o il loro uso all'interno dell'ecosistema blockchain da parte di un **admin**. Inoltre, la DApp deve anche mettere a disposizione degli utenti delle interfacce per consentire agli utenti di accedere alle funzionalità degli smart contract. Il sistema risultante è distribuito perché è previsto che gli utenti stessi eseguano il codice delle interfacce interagendo direttamente con le tecnologie blockchain distribuite, senza utilizzare componenti intermedi centralizzati.
Il sistema deve preoccuparsi di gestire la **confidenzialità** dei dati degli utenti, senza memorizzare informazioni sui consumi sul sistema stesso. Allo stesso tempo, il sistema deve essere **trasparente**, nel senso che deve consentire agli utenti di monitorarne il corretto comportamento. La soddisfazione di entrambi i requisiti è una delle difficoltà principali dello scenario e si prende in analisi nella seguente sezione.
Il sistema deve essere in grado di permettere la gestione degli utenti del sistema, inclusa la registrazione, l'abilitazione e la revoca dei permessi per l'impiego del sistema di tokenizzazione. La seconda funzionalità, presa in considerazione come estensione in seconda fase, riguarda la possibilità di integrare funzionalità per la gestione dei wallet degli utenti, nel caso in cui questi non riescano a gestirli sui propri client.

4.2 *Analisi delle soluzioni esistenti per tokenizzazione con garanzie di confidenzialità*

Consideriamo i sistemi noti di *tokenizzazione* su tecnologie *blockchain*, ovvero sistemi decentralizzati che consentono a degli *utenti* di rappresentare degli *asset* in proprio possesso tramite *token* e di scambiarli secondo determinate regole tramite delle *transazioni*. Garantire la privacy di un sistema di *tokenizzazione* può riguardare la protezione di due aspetti principali (qui presentati ad alto livello, e che potrebbero essere ulteriormente analizzati in dettaglio a seconda dei contesti):

- la quantità dei token in possesso di ciascun utente e le informazioni all'interno di ogni transazione (ad esempio, la quantità di token scambiati).
- il grafo delle interazioni fra gli utenti.

Discutere del livello di privacy di un sistema di *tokenizzazione* è molto simile al discutere del livello di privacy di un sistema blockchain, in quanto le operazioni che vogliamo effettuare sui token è del tutto simile a quelle che possiamo operare su una moneta virtuale conservata da un sistema blockchain. Nel caso in cui si desiderino logiche applicative più complesse, la situazione si complica ulteriormente perché le tecniche utilizzate per nascondere la privacy dovrebbero avere ulteriori obiettivi oltre alla validazione di transazioni.
In questo ambito, è noto come alcuni dei sistemi più noti, come *bitcoin* ed *ethereum*, non garantiscano alcuna garanzia di privacy all'interno del sistema. L'unica forma di garanzia è data dal fatto che le identità degli utenti non sono memorizzate all'interno del sistema blockchain stesso, in quanto ogni utente utilizza una coppia di chiave crittografiche che non conservano alcun metadato riguardante la sua identità legale o virtuale. Questo tipo di garanzia è definita *pseudonimizzazione,* perché la chiave pubblica di un utente agisce come uno pseudonimo, ma tutte le operazioni effettuate da quella chiave pubblica sono completamente pubbliche. In particolare, ogni transazione viene memorizzata in chiaro all'interno dei dati della blockchain e la validità delle transazioni viene effettuata semplicemente controllando:

- la consistenza della logica applicativa della transazione stessa (ad esempio, effettivo possesso della quantità di *coin* spesi in *bitcoin*, e consistenza della logica applicativa dello *smart contract* in *ethereum*);

- la validità della *firma digitale* dell'utente che ha richiesto l'esecuzione della transazione.

Garantire la confidenzialità di dati che vengono impiegati nell'ambito della logica applicativa degli smart contract è estremamente complesso, perché per garantire la consistenza della logica applicativa i nodi del sistema blockchain stesso devono essere in grado di leggere le informazioni memorizzate nella blockchain. Ad esempio, nel caso in cui le quantità di token attribuiti agli utenti venissero cifrati con sistemi "standard" (ad esempio, schemi di cifratura IND-CPA), i nodi del sistema non potrebbero accedere alle informazioni ed effettuare nessuna delle verifiche necessarie per garantire la consistenza delle transazioni. Ad esempio, sarebbe impossibile sapere se un utente che vuole effettuare un pagamento possiede effettivamente la quantità di *token* necessaria.
Per riuscire a garantire garanzie di privacy, è necessario adottare degli schemi particolari che consentono di nascondere delle informazioni, ma che comunque consentono al sistema di effettuare certe verifiche. Ad oggi, le tecniche note allo scopo sono molteplici e non esiste una soluzione consolidata considerata in assoluto la migliore e da utilizzare. Infatti, lo studio di queste tecniche è ancora oggetto della ricerca accademica e industriale sia in ambito applicativo (ad esempio, trovare nuove applicazioni di schemi e tecniche note) sia in ambito teorico (ad esempio, creazione di nuovi schemi crittografici), con l'obiettivo di risolvere alcuni difetti delle tecniche note. Ad esempio, abilitare nuove garanzie di sicurezza, mitigare o eliminare alcune assunzioni di sicurezza o di sistema, o migliorare le prestazioni finali del sistema.
L'attuale stato dell'arte per quanto riguarda le garanzie di privacy in ambiti blockchain permissionless è rappresentato da ZCash [7], che sfrutta tecniche per garantire la totale confidenzialità degli utenti e delle transazioni effettuate tramite cosiddette *shielded transactions*. Anche se teoricamente ZCash costituisce il sistema ideale, in realtà il sistema è al momento ancora afflitto da alcuni difetti molto rilevanti:

- a causa dei requisiti in termini di risorse computazionali e di memoria, il sistema consente comunque di eseguire transazioni "non protette" dalle tecniche più avanzate, per cui una rilevante percentuale di transazioni vengono eseguite in modalità simile alle blockchain come bitcoin. Inoltre, le informazioni rivelate dalle transazioni non protette possono anche permettere di *inferire* informazioni riguardo le transazioni protette;
- le tecniche di "assoluta" riservatezza non sono considerate completamente vantaggiose in alcuni scenari a causa dell'impossibilità di poter monitorare, controllare o supervisionare le operazioni, neanche da parte di attori con privilegi superiori preposti allo scopo;
- le tecniche per garantire la confidenzialità delle transazioni si basano su tecniche crittografiche relativamente nuove (*pairing cryptography, zk-SNARKs)*, che non possono essere considerate ancora altrettanto "solide" quanto tecniche consolidate come firme digitali. Ad esempio, senza discutere di argomenti più teorici riguardo il tipo di assunzioni di sicurezza su cui si basano i protocolli *zk-SNARKs*, il livello di sicurezza dei parametri della crittografia di pairing su cui si basa sono stati modificati nel 2016 in seguito all'individuazione di nuovi attacchi [8].

Si ricorda inoltre che una simile approccio è al momento utilizzato nell'ambito di un sistema che non prevede lo sviluppo di smart contract arbitrari. In questo ambito esistono dei progetti proposti, ma ad oggi si devono considerare come soluzioni ancora sperimentali e in fase di analisi. Una ulteriore tecnica che può essere impiegata per garantire confidenzialità e trasparenza in sistemi di *tokenizzazione* fa uso di schemi di *commitment omomorfico* [9]. Queste tecniche permettono di assegnare token agli utenti senza rivelarne il contenuto e utilizzano tecniche di crittografia più consolidate, ma impongono comunque degli overhead in termini di performance sulle tecnologie blockchain e dei limiti sulle funzionalità implementabili su smart contract.

Esistono una serie di ulteriori tecniche per proteggere la confidenzialità delle informazioni nell'ambito delle tecnologie blockchain permissionless [10], ma non esiste una soluzione ottimale. È invece necessario convergere su soluzioni che rappresentano dei trade-off fra confidenzialità, performance e funzionalità:

- alcune soluzioni si basano comunque sulle costruzioni *zk-SNARKs*, in alcuni casi facendone un uso minore per ottenere performance migliori a scapito di alcune garanzie di privacy, ma ereditandone comunque le stesse assunzioni in termini di sicurezza;
- un gran numero di esse sono basate su tecniche di *offuscamento*, che si basano su strategie le cui reali garanzie di sicurezza non sono provabili o misurabili.

Si evidenzia come, anche nel caso di blockchain *permissioned*, per nascondere informazioni riguardo le transazioni possano esistere soluzioni architetturali (ad esempio, controllo degli accessi per impedire l'accesso a certi servizi), ma per nascondere certe informazioni ai nodi partecipanti sia comunque necessario ricorrere a soluzioni basate su protocolli crittografici avanzati [11]. Inoltre, gli utenti che possono accedere "direttamente" alla blockchain *permissioned* per ragioni di trasparenza, tutti i controlli applicati solitamente sul "perimetro" non possono essere impiegati e i trade-off in termini di confidenzialità sono analoghi a quelli delle soluzioni *permissionless*.

4.2.1 Logica di funzionamento dell'approccio scelto

I compromessi in gioco nell'ambito delle garanzie di confidenzialità, trasparenza, funzionalità e performance in sistemi di tokenizzazione ha portato a optare per una soluzione di compromesso che impiega le seguenti strategie:

- il sistema non ha l'obiettivo di garantire la confidenzialità del grafo delle transazioni o delle quantità dei token utilizzati dagli smart contract;
- il sistema ha l'obiettivo di non rendere note le precise informazioni che riguardano i parametri con cui vengono calcolati i token, quali i consumi energetici degli utenti e altri parametri di calcolo impiegati dai modelli;
- le garanzie di confidenzialità delle informazioni appena nominate possono portare ad ottenere un sistema poco trasparente, in cui gli utenti non hanno la certezza che i token a loro assegnati sono stati assegnati correttamente secondo i modelli dichiarati dall'infrastruttura. Per risolvere questo problema, il sistema utilizza un sistema crittografico per certificare che i parametri impiegati per il calcolo dei token sono quelli legittimi. Gli utenti interagire con il sistema per monitorare periodicamente che i propri dati siano effettivamente quelli corretti.

Le operazioni di alto livello che il sistema esegue per implementare queste garanzie sono le seguenti:

- il componente Oracolo analizza le serie temporali ricevute dai componenti di estrazione dati e definisce l'ammontare dei token da assegnare a ciascun utente. I token vengono memorizzati in chiari sugli smart contract delle tecnologie blockchain distribuite;
- per ogni inserimento di token, il componente Oracolo *notarizza* i parametri impiegati per assegnare i token sulle stesse tecnologie blockchain distribuite tramite schemi di *commitment confidenziale*.

Gli smart contract possono agire senza impedimenti sui token, che non sono protetti da tecniche di crittografia. Allo stesso tempo, gli utenti possono utilizzare i valori di commitment notarizzati per ispezionare la correttezza dei parametri impiegati per l'assegnazione dei token. Si nota che il trade-off risiede nel fatto che dal numero di token assegnati sia possibile ipotizzare i parametri impiegati per il loro calcolo, ma non si hanno informazioni certe a meno di non riuscire ad *aprire* il commitment notarizzato. Le informazioni necessarie per aprire il commitment notarizzato vengono mantenute dall'Oracolo, che le concede su richiesta solo agli utenti assegnatari dei token durante apposite procedure di *audit*. Si descrive più in dettaglio la logica di funzionamento di alto livello del sistema individuato tramite la Figura 7.

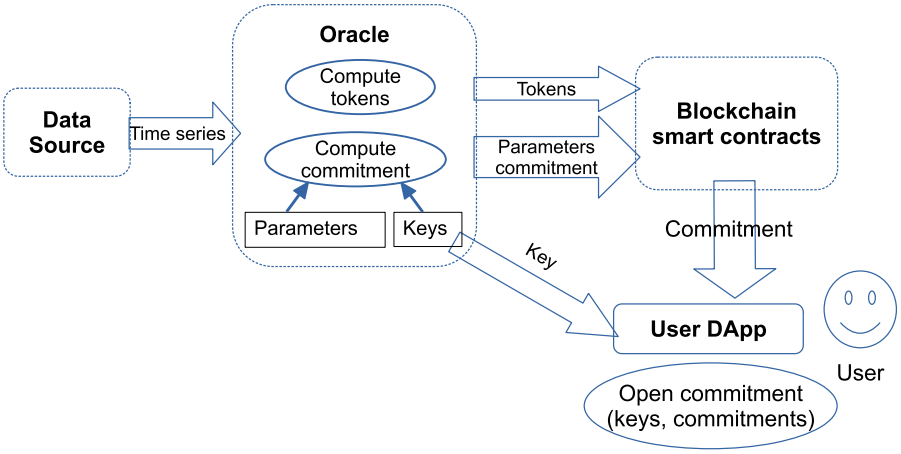

Figura 7 - Flusso dei dati con commitment confidenziale dei parametri di tokenizzazione

Come descritto nello scenario, l'Oracolo riceve i dati sotto forma di time series dai componenti di estrazione (qui semplificati con l'oggetto di alto livello *data source*). L'Oracolo applica internamente le logiche di filtraggio e calcolo dei token, e inserisce nel sistema di smart contract blockchain i token calcolati (*tokens* nella figura*)*. Inoltre, per ciascun token calcola un commitment (*parameters commitment*) utilizzando come input i parametri utilizzati per effettuare il calcolo e una chiave chiave di protezione gestita internamente (*key*). Un utente può effettuare un controllo dei parametri impiegati per calcolare i token a lui associati richiedendo all'Oracolo di restituire le chiavi di protezione. Questo schema richiede la gestione di due aspetti di dettaglio: le modalità di calcolo del commitment, e la gestione di un numero di chiavi di protezione potenzialmente di grandi dimensioni. I dettagli di queste informazioni vengono descritte nella Sezione 4.4.

4.3 *Progettazione di alto livello del sistema di tokenizzazione*

Si utilizza la Figura 8 per rappresentare l'architettura di alto livello del sistema blockchain per la gestione dei token.

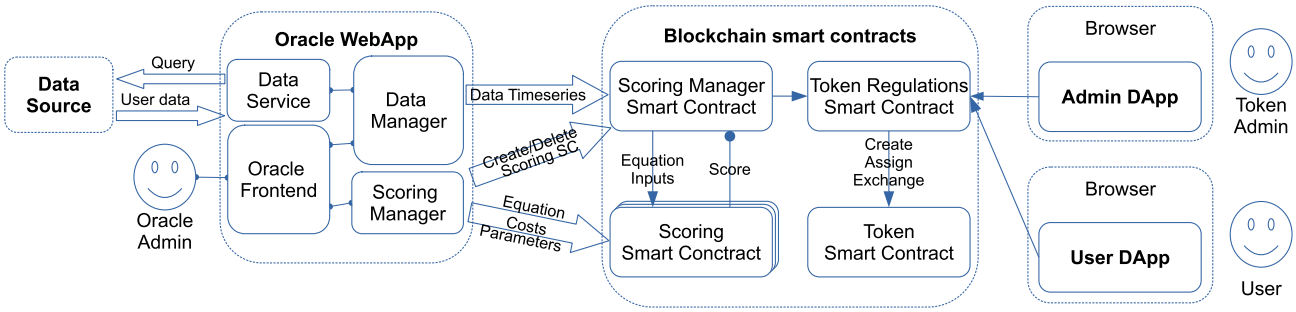

Figura 8 - Architettura di Alto Livello del sistema blockchain con oracolo e distributed applications

L'architettura è composta da quattro componenti principali: l'applicazione web che implementa Oracolo **(Oracle WebApp)**, il sistema di smart contract distribuito su blockchain **(Blockchain Smart contracts)**, l'applicazione client-side per l'interfacciamento con il sistema blockchain **(User DApp)**, l'applicazione client-side per la gestione del sistema di regolamentazione dei token su blockchain **(Admin DApp)**.
Il sistema è modellato per interagire con un componente software esterno e tre profili utente. Il componente software esterno è il **Data Source**, che fornisce i dati energetici dati energetici come descritto nei capitoli precedenti del report. I tre profili utente sono:

- l'amministratore delle funzionalità dell'Oracolo **(Oracle Admin)**;
- l'amministratore delle funzionalità relative alle regole di scambio e gestione dei token sul sistema blockchain **(Token Admin)**;
- gli utenti che utilizzano il sistema per interagire fra di loro per scambiare token **(Token User).**

Si osserva che in questo disegno architetturale non c'è alcun dettaglio riguardo le modalità di autenticazione e autorizzazione degli attori con cui il sistema interagisce, ovvero come il sistema è in grado di riconoscere

effettivamente il **Data Source** e i diversi profili utente. Le componenti architetturali di autenticazione e autorizzazione non sono interesse di questo documento e possono essere realizzati utilizzando soluzioni standard senza alcun tipo di vincolo.

4.3.1 Blockchain Smart Contracts

La parte centrale del sistema è il sistema blockchain per la gestione dei token degli utenti **(Blockchain Smart Contracts)**. Sulla base delle scelte progettuali effettuate nell'ambito di questo sistema possono dipendere diverse scelte progettuali nell'ambito degli altri sistemi dell'architettura. In particolare, oltre chiaramente alla diversa realizzazione del sistema blockchain stesso, la realizzazione di meccanismi avanzati per gestire i dati con garanzie di confidenzialità in un ambito di pubblico accesso al sistema può richiedere funzionalità aggiuntive per la trasformazione dei dati per prima della scrittura e dopo la lettura. Come definito nella precedente sezione, si è optato per un sistema in cui la gestione dei token avviene in chiaro, e in cui le informazioni che vengono protette sono i parametri utilizzati per calcolarli. In questa sezione quindi si descrive il sistema di smart contract per la gestione dei token senza considerare strumenti di protezione dei dati. In modo analogo ma differente, la scelta di sistemi permissioned o permissionless ha implicazioni nelle funzionalità da realizzare negli ambiti di gestione dell'identità degli utenti e dei sistemi di autorizzazione. L'architettura scelta prevede che gli utenti autorizzati accedano direttamente alla blockchain. In particolare, la piattaforma tipo di riferimento per cui viene pensato il sistema è la piattaforma di smart contract di **Ethereum.** Piattaforme compatibili con lo stesso sistema di smart contract possono essere utilizzate senza alcuna problematica, come ad esempio il progetto Hyperledger Besu. Il sistema viene modellato tramite quattro tipo di smart contract specializzati:

- **Token Smart Contract**: è lo smart contract per la modellazione del token di basso livello. Da questo punto di vista, la scelta può orientarsi su interfacce di token già consolidate, come ad esempio il modello standard di token ERC20. Le interfacce di questo smart contract consentono di agire sui token senza alcun controllo aggiuntivo (ad esempio, effettuare transazioni secondo *fee* gestite dai modelli, oppure creazione di token). La presenza di queste interfacce è necessaria se è necessario consentire possibili azioni di regolamentazione attiva da parte dei Token admin (ad esempio, riassegnazione di token colti in comportamenti non accettati dalla comunità). Se queste funzionalità non sono previste, le interfacce "dirette" ai token ERC20 possono essere non essere inserite, e lo smart contract può essere "collassato" con il seguente smart contract (*token regulation smart contract*) che abilita le azioni utilizzabili dagli utenti della comunità secondo le modalità definite dai modelli in uso;
- **Token Regulation Smart Contract:** è lo smart contract che regola tutte le transazioni dei token, che offre delle interfacce di più alto livello rispetto all'interfaccia ERC20 descritto sopra, e che permette la gestione in modo intermediato tutte le operazioni fondamentali (ad esempio, operazioni di *creazione*, *assegnazione* e *scambio* dei token fra gli utenti). In questo smart contract possono essere presenti i metodi invocabili direttamente dagli **utenti** del sistema. Inoltre, possono essere codificate le regole per stabilire delle modalità di gestione dei token (ad esempio, sistemi di "tassazione"). Si assume che queste regole siano *configurabili* da parte del **Token Admin** tramite dei metodi appositi (gestiti ad alto livello tramite un'interfaccia grafica resa disponibile da un'apposita applicazione, la **Token DApp**). Le normali interfacce utilizzabili dagli utenti (**Token User**) saranno invece utilizzabili tramite le interfacce grafiche messe a disposizione dall'applicazione **User DApp**;
- **Scoring Smart Contract (SSC):** gli smart contract per il calcolo di punteggi per l'assegnazione di token, che concettualmente implementano dei modelli di valutazione di comportamenti sulla base di input (i dati degli utenti). Per essere in grado di gestire in maniera flessibile diverse situazioni con differenti modalità di calcolo, ci possono essere più smart contract che gestiscono diversi modelli di

valutazione. Il numero di questi smart contract può aumentare o diminuire quindi in base alle necessità e all'evoluzione della comunità energetica. Nonostante la flessibilità, questi smart contract saranno tutti sviluppati seguendo uno schema generale ben preciso, e quindi condividono e implementano le medesime interfacce per poter essere gestiti in modo trasparente da altri smart contract (es. fornendo funzioni comuni come impostaParametro(*nomeParam*, *value*), o calcolaToken(*dati*), in maniera analoga a come sono concepiti gli smart contract che implementano le interfacce ERC20. Ad esempio, possono accettare una lista di input e restituire un valore numerico come punteggio di valutazione. Inoltre (come accennato nell'esempio precedente), questo tipo di smart contract prevede dei metodi per modificare dei parametri nei modelli di calcolo implementati; inoltre facendo riferimento allo schema di Figura 8, attraverso la funzione di create/delete SSC è possibile generare nuovi SSC che implementano modelli con logiche nuove, partendo da template di SSC generali. Il sistema di templating può essere gestito nella Webapp. L'Oracle Admin (o il Remuneration Model Admin) può in altre parole fornire altre formule e parametri che vengono "importate" in un template generale per creare nuovi SSC, come implementazioni dei modelli da aggiungere.

- **Scoring Manager Smart Contract:** lo smart contract che gestisce e smista l'esecuzione degli SSC in base all'immissione dei dati degli utenti nel sistema blockchain dall'esterno. Lo SMSC processa/verifica i dati per stabilire quale SSC invocare per assegnare i token a ciascun utente. La parte di processing viene effettuata in due fasi: i dati utente vengono inviati in modo appropriato agli Scoring Smart Contract per il calcolo dei punteggi di "valutazione" degli utenti; i punteggi vengono eventualmente "riaggregati" secondo un ulteriore modello per calcolare effettivamente la quantità di token da assegnare. L'assegnazione avviene invocando appositi metodi del Token Regulation Smart Contract (che è il contratto incaricato di gestire le possibili assegnazioni di token all'utente. Una possibile alternativa può prevedere lo spostamento delle funzionalità di "scoring" al di fuori dalla blockchain, e prevedere invece che l'oracolo invochi direttamente gli scoring smart contract opportuni.

Si osserva che nel caso di utilizzo di una piattaforma permissionless preesistente, differenti realizzazioni di ciascuno smart contract, così come una ripartizione differente degli smart contract stessi, può incidere fortemente sui costi di mantenimento del sistema (costo di creazione degli smart contract e costo per ciascuna operazione eseguita – costo di elaborazione). In presenza di un sistema permissioned, l'organizzazione degli smart contract può incidere invece soprattutto sui costi di esecuzione. In particolare, l'organizzazione delle strutture dati possono individuare differenti trade-off in termini di costi di rete (di interesse sono soprattutto i costi fra le applicazioni client-side in esecuzione nella User DApp e nell'Admin DApp.

4.3.2 Oracle WebApp

La **Oracle WebApp** è un servizio centralizzato che ha tre funzionalità principali:

- interagisce con le sorgenti dati (**Data Source**), per ottenere i dati di riferimento degli utenti. Per gestire l'immissione di dati, la Oracle WebApp può agire in due modi:
    - *modalità pull*: connettersi in maniera configurabile attraverso il Data Service ad una sorgente dati, invocarla e poi ricevere i dati di interesse tramite protocolli request/response. Questa funzionalità sembra essere la più semplice e quella che ha meno impatto nei componenti interni dedicati all'estrazione ed elaborazione dati. Esempi possono essere query ai database, eventualmente tramite viste in sola lettura (approccio poco disaccoppiato),

oppure la creazione di Web API aggiuntive apposite (approccio più disaccoppiato ed indipendente dalle tecnologie e dalle strutture database dei componenti di gestione dati);
    - *modalità push*: come alternativa potrebbe mettere a disposizione delle Web API, oppure altri meccanismi più efficienti a seconda della natura del trasferimento dei dati ed essere invocata da esterno;
- permette all'**Oracle Admin** di agire su diversi parametri di configurazione dell'oracolo (ad es. insieme di dati da estrarre, tempistiche delle operazioni da eseguire) tramite un'interfaccia grafica **(Oracle Frontend)**. Si osserva che dal punto di vista tecnologico questa interfaccia potrebbe utilizzare tecnologie Web parzialmente basate su javascript e quindi essere eseguita in parte anche sul browser dell'amministratore;
- interagire con il sistema blockchain per configurare gli smart contract che gestiscono l'assegnazione dei token agli utenti sulla base dei dati ricevuti. In particolare, ci sono due funzionalità principali:
    - il servizio **Data Manager (ex Data Service)**, che, in base a configurazioni dell'Oracle Admin, attiva il Data Service, riceve i dati utente da esso e, sempre secondo le *configurazioni* indicate dall'Oracle Admin, li elabora (in base alla complessità dei dati che sono stati ricevuti) e attiva gli smart contract dedicati alla valutazione; se si fa a meno dello Scoring Manager Smart Contract, la selezioni dei modelli (leggi SSC) da invocare sui dati deve essere fatta dal Data Manager.
    - una funzionalità **Scoring Manager** per la creazione, gestione (modifica di parametri), deployment e cancellazione degli Scoring Smart Contract. Questa funzionalità gestisce il template (potrebbero essere più di uno ma sarebbe meglio per semplicità cercare di averne uno generale) per gli SSC.

4.4 *Strategia di derivazione di chiavi per verifica selettiva efficiente dei dati*

Si discute l'utilizzo di una strategia basata su funzioni di derivazione di chiavi, note con il termine inglese di *Key Derivation Function* (KDF), per consentire l'audit a granularità fine dei commitment per la verifica dei parametri impiegati per il calcolo dei token negli smart contract. In particolare, le KDF sono utilizzate per calcolare le chiavi impiegate nel calcolo degli hash impiegati come commitment degli input impiegati per il calcolo dei token. La prima parte della discussione considera l'impiego in *black-box* di una funzione KDF che accetta come input una chiave e un contesto di utilizzo, e genera come output la chiave derivata. In questa fase non si considera alcuna specifica implementazione della funzione KDF, ma si focalizza sull'impiego della funzione agli scopi della soluzione proposta. Nella fase successiva si propone come implementare effettivamente la funzione di derivazione considerando funzioni crittografiche standard consolidate e diffuse in librerie crittografiche considerate per lo sviluppo del progetto.

4.4.1 Strategia di derivazione di alto livello

Definiamo la funzione KDF secondo la seguente rappresentazione:

$$K2 = KDF(K1, INFO)$$

dove:

- K1 è la chiave crittografica originale che si usa come input alla funzione di derivazione;
- INFO identifica i parametri che rappresentano l'ambito applicativo per cui si utilizzerà la chiave derivata;
- K2 è la chiave crittografica derivata che viene generate dalla funzione KDF.

Si osserva che questa rappresentazione è semplificata rispetto a quella utilizzata solitamente dagli standard delle funzioni KDF, ma è sufficiente per il contesto di utilizzo della funzione nel sistema progettato.
La strategia di utilizzo della funzione KDF nel contesto del sistema progettato prevede due aspetti:

- l'uso di un sistema di naming gerarchico per codifica INFO in modo da rispecchiare correttamente i contesti di utilizzo delle chiavi derivate generate (approccio tipico nell'ambito di modellazione di informazioni nativamente gerarchiche)
- l'uso ricorsivo della funzione di derivazione per consentire di distribuire meno chiavi in fase di audit e render quindi più efficiente la procedura (approccio tipico nell'ambito della costruzione di soluzioni *access tree* per il controllo degli accessi)

Si propone un esempio per chiarire l'approccio proposto. Si consideri che il sistema di smart contract utilizzi dati specifici per ogni utente generati con granularità di un'ora. Si propone uno schema di derivazione che consenta allo schema di commitment di utilizzare una chiave univoca per ogni dato inserito partendo dalla *chiave "master" di comunità* K_C.

Chiavi di primo livello associate agli utenti. Si calcola prima di tutto una chiave K_U specifica per l'utente per cui stiamo effettuando il commit dei dati

$$K_U = KDF(K_C, "1.user:" \mid user)$$

Dove "1.user:" è una stringa costante di contesto per le variabili degli utenti e *user* è la variabile che rappresenta l'identificativo dell'utente (si nota che la variabile *user* potrebbe in realtà rappresentare l'identificativo di uno specifico contratto dell'utente a seconda del contesto applicativo).

Chiavi di secondo livello associate a informazioni temporali a grana grossa. Si calcola una chiave $K_{U,M}$ specifica per l'utente e per un mese di consumo:

$$K_{U,M} = KDF(K_U, "1.user:" \mid user \mid "2.month" \mid year.month)$$

Dove "2.month" è una stringa costante di contesto per le variabili dei mesi e *year.month* è la variabile che rappresenta in modo univoco un mese (in questo caso, ad esempio, la stringa che contiene l'anno e il mese).

Chiavi di terzo livello associate a informazioni temporali a grana fine. Si calcola una terza chiave $K_{U,M,H}$ specifica per l'orario:

$$K_{U,M,H} = KDF(K_{U,M}, "1.user:" \mid user \mid "2.month" \mid year.month \mid "3.hour" \mid day.hour)$$

Dove "3.hour" è una stringa costante di contesto per le variabili delle ore e *day.hour* è la variabile che rappresenta in modo univoco un'orario di una giornata (ad esempio, la stringa costruita sulla base del numero del giorno e l'ora in formato 24h).

Con questa strategia, si ottengono le seguenti funzionalità:

- la conoscenza di K_U consente di verificare tutti e soli i dati relativi ad un utente;
- la conoscenza di $K_{U,M}$ consente di verificare tutti e soli i dati relativi ad un utente in un certo mese;
- la conoscenza di $K_{U,M,H}$ consente di verificare una sola entry all'interno dello smart contract (perché abbiamo assunto che l'orario sia il livello di granularità più fine sulla base del quale vengono calcolati i token);
- la conoscenza di una chiave di livello N consente di calcolare tutte le chiavi da essa derivate di livello N + 1, ma non il contrario, permettendo di istanziare strategie di accesso ai dati a granularità variabile senza incappare in overhead eccessivi di chiavi da distribuire.

Questo approccio è vantaggioso nel seguente contesto di utilizzo:

- tramite il rilascio di K_U ad un utente, è possibile consentire all'utente di verificare tutti i dati a lui associati. Si nota che nell'uso normale del sistema il rilascio di questa chiave a un utente è molto rischioso perché, a meno di strategie aggiuntive di rotazione della chiave master di comunità, consente potenzialmente di rivelare informazioni su tutti i dati passati e futuri di un utente. D'altra parte, in situazioni particolari avere un'unica chiave per l'accesso a tutti i dati dell'utente può essere utile;
- tramite il rilascio di $K_{U,M}$ ad un utente, è possibile consentire all'utente di verificare tutti i dati a lui associati durante un mese di utilizzo. Questa sembra essere la situazione può simile all'uso comune che un utente richiede al sistema;

- in una *potenziale futura modalità di utilizzo* in cui si consentano audit da parte di enti terze, rilasciare chiavi $K_{U,M,H}$ a granularità minima possono consentire verifiche probabilistiche riguardo la correttezza dei dati impiegati senza rivelare informazioni significative su particolari utenti o fasce orarie (che si ipotizza possa essere utile sia per la protezione della privacy degli utenti sia per la privacy della comunità stessa).

Considerazioni sulle strategie di derivazione. L'esempio proposto può essere anche utilizzato così com'è nell'ambito del prototipo di sistema, ed è possibile ridurre o aumentare il numero di livelli di derivazione a seconda del tipo di verificabilità che si reputa necessario. Si osserva che una strategia alternativa potrebbe prevedere l'impiego di chiavi di primo livello associate a informazioni temporali a grana grossa. D'altra parte, se la resistenza dei dati passati inseriti a sistema di importanza rilevante, sarebbe più opportuno introdurre un sistema di rotazione delle chiavi master di comunità nell'ambito della piattaforma KMS da integrare in una futura versione del sistema. Comunque, la definizione della specifica strategia dovrebbe essere definita specificamente rispetto ai requisiti di verificabilità, e si stima che eventuali modifiche della strategia in fasi avanzate del progetto non richiedano modifiche al sistema generale.

4.4.2 Note di implementazione della funzione KDF

Uno schema standard consolidato per realizzare una funzione KDF è lo standard HMAC-based Key Derivation Function (HKDF, Standard IETF, RFC5869). Questa funzione può essere utilizzata così com'è per sostituire la funzione proposta (si nota infatti che la modellazione proposta rappresenta un sotto-insieme delle funzionalità disponibili nello standard HKDF). Nel caso la funzione HKDF non sia disponibile nell'ambito delle librerie impiegate nel sistema, si reputa che la funzione KDF possa essere implementata tramite una funzione HMAC, in cui la chiave di input della funzione di derivazione viene utilizzata come chiave della funzione HMAC, e il campo INFO viene impiegato come input della funzione HMAC. L'impiego della funzione HMAC al posto della funzione HKDF è considerato sicuro se viene rispettato il requisito di utilizzate chiavi crittografiche forti nell'ambito delle chiavi di comunità (ovvero, sequenze di bit random o pseudo-random di dimensioni pari al livello di sicurezza richiesto dallo schema in cui la chiave viene impiegata). L'utilizzo di informazioni di altra natura, seppur segrete, potrebbe intaccare la sicurezza delle chiavi derivate. Si nota infatti che l'impiego dello standard HMAC costituisce un subset della funzione HKDF denominato di *"expand"* utilizzato per lo *scoping* delle chiavi derivate rispetto ai contesti applicativi di utilizzo, ma omette completamente la fase preliminare di *"extract"* utilizzata per "irrobustire" chiavi che, anche nel caso di alto contenuto di entropia, non rispettano alcuni dei requisiti per essere "nativamente" chiavi crittografiche forti. Lo schema inoltre sarebbe completamente insicuro nel caso di utilizzo di segreti a basso contenuto di entropia come password scelte dagli utenti.

4.5 Estensione dell'architettura per gestione server-side dei wallet degli utenti

L'architettura proposta è orientata alla gestione dei wallet per l'esecuzione di transazioni degli utenti "lato client", ovvero ogni utente mantiene le chiavi crittografiche necessarie per firmare digitalmente le proprie transazioni. Questo paradigma è quello che rispetta più fedelmente i principi di decentralizzazione e controllo del sistema e dei dati da parte degli utenti, perché nessun altro oltre agli utenti può eseguire transazioni sugli smart contract, neanche i gestori della piattaforma stessa. Lo svantaggio di questo approccio è la necessità di richiedere agli utenti una maggiore consapevolezza e competenza nella gestione delle proprie chiavi crittografiche.

In particolare, la violazione del wallet di un utente da parte di attori malevoli può consentire l'impersonificazione dell'utente e la potenziale esecuzione di transazioni sulla piattaforma blockchain da parte dell'utente. Sebbene questo sia un evento che può avvenire anche nel caso di wallet gestiti lato server, la compromissione di wallet lato utente può essere considerata più rischiosa perché potenzialmente più facile da realizzare nel caso di utenti non competenti per la gestione appropriata di queste operazioni. Nel caso di un sistema permissioned, sistemi di mitigazione analoghi a quelli impiegati nel caso di smarrimento di credenziali di autenticazione, come la revoca e il rinnovo di un nuovo wallet (questo processo non è invece

possibile in un sistema permissionless completamente decentralizzato). Un popolare strumento per migliorare la sicurezza dei wallet gestiti lato utente sono i cosiddetti *wallet hardware*, ovvero sistemi hardware embedded che conservano il wallet dell'utente e che provvedono all'esecuzione delle operazioni di firma digitale. La violazione di un wallet hardware è molto più complicata per un avversario da realizzare. D'altra parte, l'impiego di wallet hardware introduce costi di gestione della piattaforma (il costo dei dispositivi hardware degli utenti) e potenzialmente una riduzione dell'usabilità finale del sistema da parte degli utenti, che devono preoccuparsi della gestione "fisica" del dispositivo fornito. Una discussione analoga è valida anche nel caso di smarrimento del wallet da parte degli utenti.
Per evitare le problematiche appena descritte riguardo la gestione dei wallet da parte degli utenti, è possibile impiegare un sistema alternativo in cui i wallet vengono gestiti da parte di servizi delegati dagli utenti allo scopo. In questo caso, gli utenti potranno interagire con il sistema in modo analogo a un qualsiasi servizio Web, perché tutte le operazioni di firma digitale eseguite sono realizzate dal servizio che gestisce il wallet. Questo servizio può essere considerato ortogonale alla piattaforma che istanzia l'infrastruttura blockchain, e può anche essere ipotizzata l'integrazione con servizi di gestione dei wallet esterni. Di seguito si propongono due architetture di alto livello volte ad per istanziare una soluzione per la gestione dei wallet degli utenti. Le soluzioni proposte si pongono l'obiettivo di avere un approccio modulare e di impiegare protocolli standard, di sfruttare componenti istanziabili tramite prodotti open-source, e di prestare attenzione ai requisiti principali di sicurezza dei componenti critici. Le prime due soluzioni di distinguono nell'ambito del livello di disaccoppiamento dei componenti volti alla gestione dei wallet degli utenti, ma entrambe mantengono un single point of trust nell'ambito della gestione dei wallet.

- la prima versione segue un approccio in cui si assume che un solo gestore abbia il controllo sia dei servizi blockchain sia dei wallet, in cui è possibile progettare una soluzione in cui tutti i componenti sono sotto il controllo della stessa autorità e condividono una maggiore fiducia reciproca. Questa soluzione è tecnicamente più semplice da realizzare, ma è meno flessibile;
- la seconda versione segue un approccio più disaccoppiato, in cui potenzialmente le due classi di servizio sono istanziate da autorità differenti. Questo approccio è più complesso da realizzare, ma più flessibile e attuabile anche nell'ambito di architetture gestite da entità differenti.

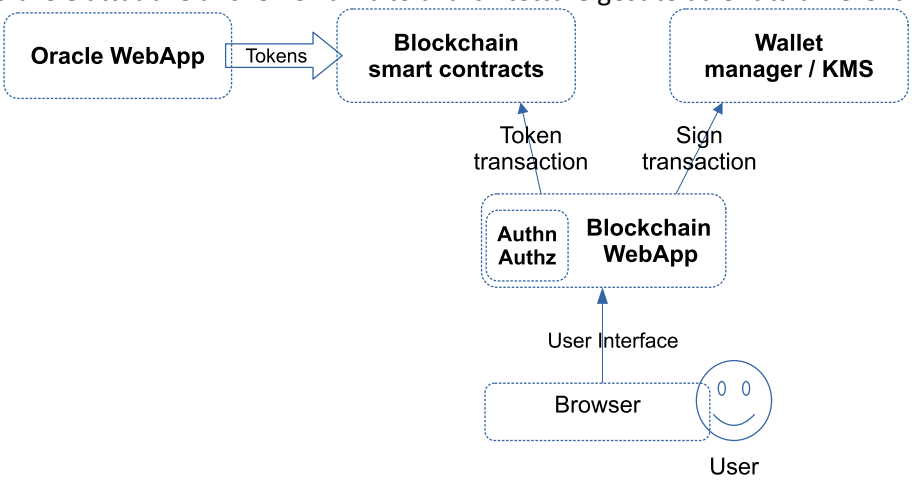

Figura 9 - Gestione Wallet lato infrastruttura con Webapp di gestione dell'accesso intermedia

Si rappresenta la prima versione dell'architettura in Figura 9. Questa architettura trasforma la soluzione originale basata su DApp facendolo diventare più simile a una tradizionale applicazione Web, in cui l'utente utilizza un browser per accedere a servizi offerti dall'infrastruttura del gestore. In questo caso, la *blockchain webapp* ospita completamente la logica applicativa dell'utente, senza richiedere la creazione di transazioni blockchain dallo stesso utente (per questo non è più denominata *dapp)*. La *blockchain webapp* esegue le funzionalità di autenticazione e autorizzazione come solitamente svolto dalle tipiche applicazioni (potenzialmente distribuendo le diverse funzioni ad altri componenti specializzati dell'architettura, dettaglio che però in questo caso è puramente tecnico e non rilevante per l'utente e per i componenti da lui eseguiti). Quanto l'utente effettua delle operazioni, la *blockchain webapp* può richiedere a un ulteriore componente, *il Wallet manager*, di firmare le transazioni da eseguire per conto dell'utente. Si evidenzia che questo

componente conserva le chiavi crittografiche di tutti gli utenti. Il motivo per cui si separano i due componenti (*blockchain webapp* e *wallet manager*) è dovuto agli aspetti di sicurezza critici che il *wallet manager* deve soddisfare. In particolare, questo è un componente altamente specializzato che deve solo conservare le chiavi crittografiche ed eseguire algoritmi di firma digitale. Queste funzionalità possono essere individuate in componenti software di *Key Management Service (KMS)*, di cui esistono istanze anche nell'ambito di soluzioni software opensource (ad esempio, HashiCorp Vault [12]).

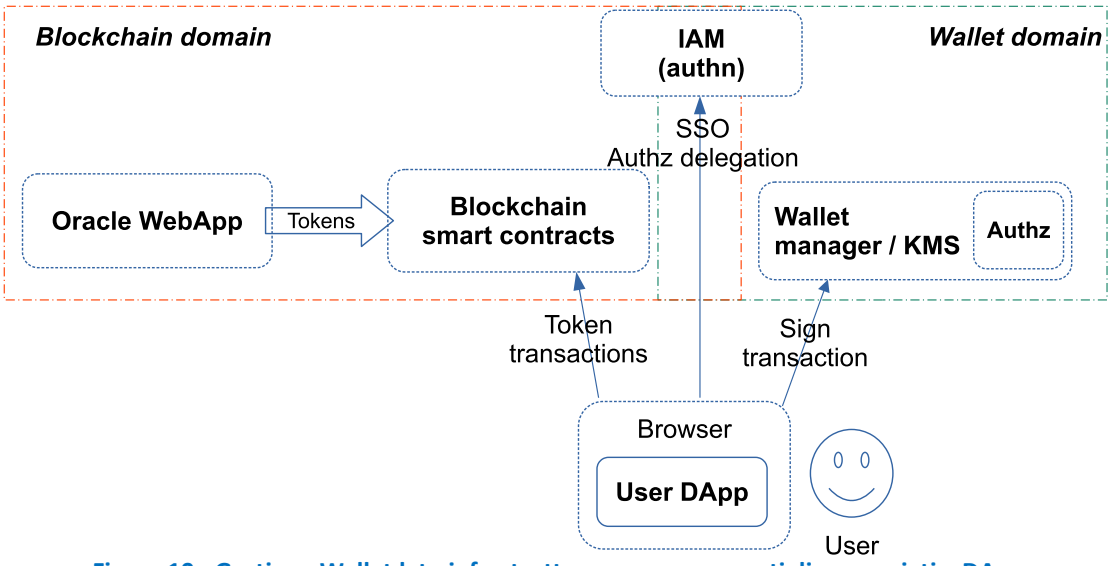

Figura 10 - Gestione Wallet lato infrastruttura con componenti disaccoppiati e DApp

Si rappresenta la seconda versione di architettura in Figura 10. Questa architettura ha l'obiettivo principale di poter istanziare tutti i protocolli necessari al fine di decentralizzare le autorità che gestiscono diversi componenti. In particolare, si evidenzia l'esistenza di due domini autoritativi che hanno in gestione i componenti: il *dominio blockchain (blockchain domain)*, e il *dominio wallet (wallet domain)*. Il *dominio blockchain* esegue tutti i componenti già individuati nella progettazione iniziale del sistema. Il *dominio wallet* invece esegue il *Wallet manager*, e ulteriori funzionalità necessarie per la validazione delle operazioni in questo disegno architetturale. Si evidenzia che questo tipo di architettura può anche prevedere che utenti differenti possano utilizzare *domini wallet* differenti per gestire le proprie chiavi crittografiche. In tutti questi casi, si evidenzia la necessità di scorporare chiaramente le funzionalità di autenticazione (*authn*), svolte da un componente che viene solitamente identificato con il nome di *Identity and Access Management (IAM, ad esempio Keycloak* [13]*)*, dalle funzionalità di autorizzazione (*authz)* eseguite in questo caso all'interno del *Wallet manager*. Si chiarisce il funzionamento di alto livello di questa architettura descrivendo il flusso delle operazioni necessarie all'esecuzione di una transazione blockchain da parte di un utente. Primo, l'utente deve effettuare l'autenticazione Single Sign-On (SSO) presso l'IAM, ottenendo un token di autenticazione. Secondo, l'utente può utilizzare il token di autenticazione presso il *Wallet Manager*. Si nota che in questo caso il *Wallet manager* deve validare direttamente la validità del token ed applicare controlli sulle autorizzazioni associate al token stesso, per consentire all'utente che richiede l'operazione di ottenere transazioni blockchain firmate solo per sé stesso, e non per altri utenti. Infine, l'utente può sottomettere la transazione all'infrastruttura blockchain. Nonostante sia più complessa, questa versione architetturale sembra prestarsi meglio alla soluzione originale perché mantiene la presenza della *User DApp*, e un controllo maggiore da parte degli utenti. Si evidenzia che è possibile realizzare questo tipo di architettura anche se in realtà i due domini sono gestiti dalla stessa autorità. Infatti, questa soluzione utilizza pratiche e protocolli migliori per la gestione di tutte le fasi di gestione degli accessi e di suddivisione dei ruoli rispetto alla prima versione.

In ultima analisi, si osserva che in modo simile ad alcune delle soluzioni più avanzate per la confidenzialità delle transazioni su blockchain (vedi Sezione 4.2), potrebbero anche essere impiegate ulteriori varianti basate su protocolli o tecnologie meno consolidate. Nonostante teoricamente di maggiore sicurezza, si ritengono

queste varianti di difficile realizzazione in contesti industriali consolidati per via della necessità di impiegare protocolli meno consolidati (e potenzialmente di dover sviluppare nuovi componenti software, senza aver la possibilità di affidarsi a software open source preesistente e consolidato) o su assunzioni ancora non ritenute completamente consolidate. Si evidenziano due varianti potenzialmente di particolare interesse:

- nel caso di gestione dell'infrastruttura che esegue il *Wallet manager* da parte dalla stessa entità che esegue l'infrastruttura blockchain, è possibile investigare soluzioni basate su *trusted hardware enclaves* [14], che consentono l'esecuzione di codice in ambienti *untrusted*. Nonostante queste tecnologie siano già ampiamente diffuse nei moderni microprocessori, molte soluzioni note sono vulnerabili a diversi tipi di attacchi di tipo *side channel*. Sembra quindi che una soluzione consolidata sulla base di queste tecnologie non possa ancora essere realizzata;
- nel caso in cui non si voglia delegare una sola autorità con il roulo di gestire il *Wallet manager* (ovvero, si preferisce adottare un approccio decentralizzato analogo a quanto svolto con le tecnologie blockchain), è possibile adottare soluzioni di *crittografia a threshold*, in cui la firma delle transazioni deve essere applicata da diverse entità. Il livello di garanzie che si ottiene è simile a quello dell'adozione di *multi-signature* nel contesto di transazioni su tecnologia *bitcoin* [15]. D'altra parte, non esistono soluzioni software open source consolidate che implementano dei Key Management Services che supportano queste tecnologie.

5 Conclusioni

Questa sezione sintetizza i risultati complessivi del lavoro e riporta eventuali raccomandazioni per possibili ulteriori sviluppi della ricerca.

Nella prima parte di questo rapporto tecnico è stata descritta la progettazione di un'architettura per la gestione dei dati che saranno rilasciati dai diversi utenti in forma eterogenea e che quindi devono essere acquisiti e gestiti in maniera appropriata per la loro predisposizione alla successiva fase di analisi, elaborazione e visualizzazione. Sono state presentate e discusse varie architetture, confrontandole in base a due requisiti fondamentali: pre-processare in maniera batch, periodica, i dati, e memorizzarli in modo che siano disponibili per le analisi senza la necessità di ricalcolarli ogni volta; integrare dati provenienti da più differenti sorgenti. Per quello che riguarda il primo requisito è stato individuato nel componente Delta Lake lo strumento opportuno: memorizzando i dati nel Delta Lake si dovrebbero eliminare sia i problemi dell'eccessivo tempo di esecuzione nell'import di alcuni dati dal DBMS sia i problemi di import di file di grosse dimensioni.

Per quello che riguarda invece il secondo punto sono state considerate le tecniche e le metodologie di Big Data Integration sviluppate dal DBGroup e centrate sul sistema di Data Integration MOMIS allo scopo si ottenere una sorgente unificata, priva di ridondanze e di conflitti tra dati. In questo rapporto tecnico, tali tecniche di integrazione sono solo state introdotte da un punto di vista generale, demandando agli sviluppi futuri, in particolare quelli in collaborazione con DataRiver, la successiva fase di implementazione.

Nella seconda parte di questo rapporto tecnico è stata descritta l'analisi e la progettazione di alto livello di un'architettura per la gestione di un sistema di tokenizzazione dei consumi energetici degli utenti su tecnologia blockchain. In questa parte è stato modellato il sistema da realizzare dati i requisiti dello scenario richiesto, inclusi gli attori che devono interagire con il sistema e i requisiti funzionali e non funzionali. Il sistema di tokenizzazione ha affrontato inoltre la progettazione di due macro-componenti: un sistema *oracolo* che integra i dati eterogenei raccolti nella prima parte del progetto e la loro trasformazione in token; un sistema di smart contract in grado di consentire agli utenti l'utilizzo dei token in un contesto decentralizzato. Il rapporto ha incluso l'analisi di possibilità alternative note nell'ambito dello stato dell'arte di questi sistemi per garantire caratteristiche di confidenzialità, che ha mostrato come approcci votati alla totale confidenzialità dei dati e delle transazioni coinvolte nel sistema non possono essere ancora utilizzabili nei contesti considerati da questo progetti per problematiche di performance o di maturità delle conoscenze o delle tecnologie coinvolte. L'approccio selezionato per garantire la confidenzialità è da considerarsi un compromesso per garantire trasparenza della correttezza delle operazioni svolte dall'*oracolo*. In questo contesto, il report ha presentato anche uno schema di gestione delle chiavi crittografiche da impiegare per la protezione dei dati, che consente di bilanciare sicurezza e prestazioni del sistema di protezione dei dati tokenizzati. Le attività di modellazione, analisi e progettazione di alto livello, unite alle considerazioni in termini di trasparenza del sistema e di protezione dei dati coinvolti, sono stati forniti come input per successive fasi di progettazione di dettaglio, legata a specifiche tecnologie, e di successiva implementazione.

6 Bibliografia

[1] Spark, 2021. [Online]. Available: https://spark.apache.org.

[2] Bergamaschi S., Beneventano D., Guerra F., Orsini M. (2011) «Data Integration». InHandbook of Conceptual Modeling. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-15865-0_14

[3] DeltaLake, 2021. [Online]. Available: https://delta.io.

[4] M. D. T. S. L. Y. B. Z. S. M. M. .. Z. M. Armbrust, «Delta lake: high-performance ACID table storage over cloud object stores,,» *VLDB Endowment, 13(12),* pp. 3411-3424, 2020.

[5] L. Gagliardelli, *Presentazione Delta Lake,* 2021 (http://dbgroup.ing.unimore.it/publication/delta_lake_LucaGagliardelli2021.pptx).

[6] A. Airflow, 2021. [Online]. Available: https://airflow.apache.org/.

[7] Zcash, «Zcash - How It Works,» 23 10 2020. [Online]. Available: https://z.cash/technology/.

[8] S. Yonezawa, «Security of Pairing-Friendly Curves,» 8 Luglio 2019. [Online]. Available: https://tools.ietf.org/id/draft-yonezawa-pairing-friendly-curves-02.html#rfc.section.3.

[9] NIST8301, «NISTIR 8301: Blockchain Networks: Token Design and Management Overview,» 09 02 2021. [Online]. Available: https://csrc.nist.gov/publications/detail/nistir/8301/final.

[10] A. Narayanan e M. Malte, «Obfuscation in bitcoin: Techniques and politics,» in *International Workshop on Obfuscation: Science, Technology, and Theory*, 2017.

[11] E. Androulaki, S. Cocco e C. Ferris, «Private and confidential transactions with Hyperledger Fabric,» 11 Maggio 2018. [Online]. Available: https://developer.ibm.com/tutorials/cl-blockchain-private-confidential-transactions-hyperledger-fabric-zero-knowledge-proof/#privacy-preserving-exchange-of-assets-with-zero-knowledge-asset-transfer-zkat-.

[12] HashiCorp, «Vault Project,» [Online]. Available: https://www.vaultproject.io/. [Consultato il giorno 10 Aprile 2021].

[13] RedHat, «Keycloak,» [Online]. Available: https://www.keycloak.org/. [Consultato il giorno 10 Aprile 2021].

[14] Intel, «Intel SGX,» [Online]. Available: https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html. [Consultato il giorno 10 Aprile 2021].

[15] «Bitcoin Multi-Signature,» [Online]. Available: https://en.bitcoin.it/wiki/Multi-signature. [Consultato il giorno 10 Aprile 2021].

7 Breve curriculum scientifico del gruppo di lavoro

Il gruppo di lavoro UniMoRe è costituido da due unità

- Il gruppo di ricerca DBGroup, diretto dalla Prof.ssa Sonia Bergamaschi e composto da Domenico Beneventano, Giovanni Simonini, Luca Gagliardelli e Luca Zecchini.
  Il gruppo di ricerca DBGroup ha redatto la prima parte di questo rapporto tecnico, in particolare la sezione BIG DATA PLATFORM ARCHITECTURE.
- Il gruppo di ricerca WebLab, diretto dal Prof. Colajanni al momento dell'inizio del progetto (attualmente Professore Ordinario presso l'Università di Bologna). In questo progetto, il gruppo è stato coordinato da Luca Ferretti, e ha visto contributi da parte di Michele Colajanni, Mauro Andreolini e Federico Magnanini. Il gruppo di ricerca WebLab ha redatto la seconda parte di questo rapporto tecnico, in particolare la soluzione di tokenizzazione dei dati descritta nel Capitolo 4.

Il gruppo di ricerca DBGroup (www.dbgroup.ing.unimore.it) di UniMoRe ha iniziato a lavorare su temi di gestione dei dati nei primi anni '90 e ha studiato la preparazione dei dati per più di 20 anni. La maggior parte delle attività di ricerca è stata incentrata sul sistema di Data Integration MOMIS , la cui versione open-source è fornita e gestita da Datariver (www.datariver.it). Una caratteristica fondamentale del sistema MOMIS è quella di fornire approcci basati sulla semantica e altamente automatizzati per l'integrazione dei dati. La quantità di dati è in crescita esponenziale; per consentire l'estrazione del suo enorme valore nascosto il DBGroup sta portando avanti la ricerca su:

- Big Data Management il DBGroup sta adottando e migliorando tecnologie all'avanguardia per gestire i Big Data
- Analisi dei Big Data come ottenere informazioni preziose dai dati e come estrarre informazioni per guidare il processo decisionale, quando i dati sono troppo grandi per gli algoritmi tradizionali
- Big Data Integration: per estrarre il reale valore dei dati in uno scenario big data, integrando fonti dati diverse ed eterogenee, dai dati strutturati a quelli non strutturati, dai dati testuali a quelli multimediali.

Gli argomenti principali che il DBGroup sta studiando sono: metodi di join scalabili distribuiti; metodi di Entity Resolution con un approccio *human-in-the-loop* e tecniche di *machine learning*; integrazione di big data per la gestione delle energie rinnovabili.
Il gruppo di ricerca ha anche una forte esperienza su Semantic Web (motori di ricerca semantici basati su sistemi di integrazione dei dati), Keyword Search su database relazionali e Linked Open Data (metodi di sintesi e query visive). Il gruppo DBGroup è stato coinvolto e coordinato diversi progetti nazionali ed europei, come il progetto PRIN WISDOM, il progetto UE FP5-IST SEWASIE e il progetto FIRB NeP4B.

Il gruppo di ricerca WebLab (https://weblab.ing.unimore.it) dell'Università di Modena e Reggio Emilia svolge attività di ricerca, di formazione e di trasferimento tecnologico in molteplici settori che spaziano dalla sicurezza dei sistemi e dei servizi informatici, alla crittografia, alle architetture ad alte prestazioni, ai sistemi di Edge e Cloud computing. Il gruppo di ricerca comprende professori e ricercatori appartenenti al Dipartimento di Ingegneria "Enzo Ferrari" e al Dipartimento di Fisica, Informatica e Matematica.
Nell'ambito del gruppo di crittografia, le attività includono ricerche affini al progetto in esame, quali le tecnologie blockchain e le soluzioni per la gestione della sicurezza delle informazioni in contesti esternalizzati. Inoltre, le ricerche sono orientate ai sistemi per la protezione dei sistemi *cyber fisici, automotive* e *IoT*. Il gruppo ha inoltre pluriennali competenze nell'applicazione di sistemi di machine learning ai *sistemi di intrusion detection* e di anomaly detection. In tali ambiti, il WebLab ha all'attivo centinaia di pubblicazioni in conferenze e riviste internazionali, partecipa e guida progetti di ricerca nazionali ed europei, e ha molteplici collaborazioni con aziende e organizzazioni del territorio regionale, nazionale ed europeo.