
Università di Pisa

Scuola di Ingegneria

Corso di Laurea in Ingegneria Informatica per la Gestione d'Azienda

*Tesi di Laurea*

# Soluzioni GIS per l'accesso a dati ambientali georeferenziati: applicazione al monitoraggio della Laguna di Venezia

*Relatori:*

Ing. Alessio Bechini

Prof. Gigliola Vaglini

Dott. Alessandro Sarretta

*Candidato:*

Giuseppe Sucameli

ANNO ACCADEMICO 2014/2015

*A Manila e ai miei genitori*

# Ringraziamenti

Innanzi tutto desidero ringraziare il mio primo relatore, l'Ing. Alessio Bechini, per avermi seguito nel lavoro di tesi con preziosi ed utili consigli, dimostrando grande disponibilità e pazienza.

Desidero inoltre ringraziare il Dott. Alessandro Sarretta per l'aiuto fornitomi nel dirimere numerosi dubbi e per la grande cortesia dimostratami.

Ringrazio inoltre CNR-ISMAR che mi ha onorato di questa collaborazione e in particolare Stefano Menegon, sempre disponibile quando necessario.

Un grazie di cuore va ai miei compagni di studio, ma anche e soprattutto amici: Alessia, Antonella, Dario e Emilio. Di giornate insieme ne abbiamo trascorse tante, qualcuna è anche rimasta nella storia, ma la cosa più bella che è rimasta è di certo la grande amicizia che ci lega.

Grazie a Mario e Francesca, amici insostituibili, sempre presenti e pronti a sostenermi ed incoraggiarmi, divertenti, allegri, insomma... Mario e Fra!

Grazie ai miei suoceri e a Dany per essere ogni giorno dei supporter eccezionali, sempre a tifare per me e per i miei successi.

Grazie a Matteo e Leda per essermi stati sempre vicini e per essere un punto certo della mia vita. Non vedo l'ora di abbracciarvi.

Un immenso ringraziamento va ai miei genitori, per l'incrollabile sostegno che hanno saputo mettere in campo in questi lunghissimi 6 anni, senza mai perdersi d'animo, nemmeno quando i miei obiettivi sembravano essere diventati altri, dimostrandomi ogni giorno il loro Amore incondizionato.

Ma il più grande ringraziamento va certamente all'unico e incommensurabile Amore della mia vita: Manila. Non sarei riuscito a concludere questo percorso senza il tuo supporto, senza la tua forza, senza il tuo entusiasmo, senza la tua dolcezza. Se sono riuscito a raggiungere il traguardo è merito tuo oltre che mio... e anche un po' del fatto che il 13 maggio 2016 è oramai alle porte :). Ti Amo.

"I must not fear.
Fear is the mind-killer.
Fear is the little-death that brings total obliteration.
I will face my fear.
I will permit it to pass over me and through me.
And when it has gone past I will turn the inner eye to see its path.
Where the fear has gone there will be nothing. Only I will remain."

— Frank Herbert, *Dune*

# Indice

# Elenco delle figure

# Introduzione

I recenti progressi tecnologici nella microelettronica e nel campo della comunicazione wireless hanno permesso il diffondersi di piccoli apparecchi a bassa potenza in grado di misurare grandezze fisiche, elaborare dati e comunicare tra loro e con il mondo esterno.

Questi apparecchi, chiamati nodi sensori, hanno solitamente un costo di produzione, dimensioni e peso trascurabili e vengono disposti all'interno o in prossimità di fenomeni da studiare o tenere sotto controllo a formare una rete di sensori.

Essendo in grado di misurare grandezze fisiche, uno dei più comuni impieghi delle reti di sensori è nel monitoraggio ambientale. Il controllo di zone soggette a terremoti, la verifica dei livelli di inquinamento dell'aria, il monitoraggio del traffico di grandi città, lo studio dell'innalzamento del livello del mare, sono alcuni dei possibili scenari nei quali si può impiegare una rete di sensori.

L'analisi di tali dati presuppone l'uso di strumenti che riescano ad interpretare correttamente, oltre i valori misurati, anche la componente spaziale associata al dato, ovvero le informazioni sulla posizione del sensore al momento della rilevazione. La comprensione di fenomeni fisici sia semplici che complessi non è infatti realizzabile senza l'identificazione dell'ambiente in cui si manifestano, quindi senza il supporto delle informazioni spaziali.

Proprio per questo l'Open Geospatial Consortium (OGC) ha sviluppato un pacchetto di standard, chiamato Sensor Web Enablement (SWE), con l'obiettivo di rendere accessibili e utilizzabili tramite web i dati di qualunque tipo di sensore.

Di questi standard fa parte Sensor Observation Service (SOS), l'interfaccia da utilizzare per la realizzazione di un servizio web interoperabile che permetta di ottenere informazioni sui sensori e sulle loro osservazioni, ovvero sui dati da essi rilevati.

Sebbene esistano vari client SOS, la maggior parte di essi sono stati realizzati per l'utilizzo dei dati con strumenti di analisi statistica, quali ad esempio R, o per la fruizione del servizio SOS tramite web client, per la visualizzazione dei dati e la creazione di mappe.

Le implementazioni di client SOS per software GIS di tipo desktop sono invece molto limitate, a dispetto delle ampie possibilità di analisi che strumenti di questo genere possano offrire. Specialmente nel panorama del GFOSS (Geografic Free and Open Source Software), a fronte di una grande varietà di soluzioni GIS desktop esistenti, alcune ampiamente utilizzate, le implementazioni dello standard SOS sono soltanto un paio.

Data l'importanza strategica che l'accesso ai dati ambientali ha nella vita di tutti i giorni, soprattutto per chi, come le Pubbliche Amministrazioni, si occupa della gestione dell'ambiente e del territorio e prende decisioni che impattano sulla vita dei cittadini, è di fondamentale importanza avere degli strumenti che permettano l'accesso a dati ambientali, e in generale provenienti da sensori, mediante soluzioni GIS.

Per tale motivo questo lavoro di tesi, oltre a focalizzarsi sullo studio della problematica fin qui espressa, ha trovato naturale prosecuzione nella realizzazione di uno strumento software atto a semplificare concretamente l'accesso

ai dati ambientali tramite GIS desktop.

La scelta è ricaduta su QGIS, un software GIS libero che negli ultimi anni ha preso piede nell'ambito italiano ed internazionale grazie alla sua semplicità d'uso e ad una sempre più fiorente community di sviluppatori e utenti.

Il lavoro di tesi è stato svolto in collaborazione con CNR-ISMAR, istituto che insieme al Comune di Venezia ed altri Enti ed Istituzioni venete ha realizzato e porta avanti il progetto l'Atlante delle Laguna con l'obiettivo di rendere le informazioni ambientali accessibili ai cittadini e alla Pubblica Amministazione.

Nel primo capito saranno descritte le caratteristiche generali delle reti di sensori, partendo dalla struttura dei singoli nodi sensori e dei loro componenti per arrivare a parlare delle tipologie di reti di sensori esistenti e delle problematiche ad esse connesse. Successivamente sarà posta attenzione all'iniziativa Sensor Web Enablement di OGC al fine di fornire una panoramica sugli obiettivi che si propone di risolvere.

Il secondo capitolo sarà dedicato a Sensor Observation Service, lo standard di interfacciamento e accesso ai dati provenienti da sistemi di sensori eterogenei tramite web. Saranno descritti i modelli di dati e le operazioni che tale standard definisce per rendere accessibili ed interrogabili via web sensori, sistemi sensoristici e archivi contenenti dati di osservazioni, in modo coerente e indipendente dalla loro tipologia, che siano sensori fissi, mobili, in-situ o sensori remoti. Saranno quindi affrontate le problematiche che un sistema incontra nell'interfacciarsi con un servizio SOS, principalmente la grande quantità di dati di osservazioni che devono scambiarsi e il supporto opzionale di alcuni formati, e come tali problemi possano essere risolti.

Nel terzo capitolo saranno analizzate le scelte fatte per al realizzazione dello strumento software integrato in QGIS che agirà come client SOS. In

particolare si parlerà di QGIS, la sua struttura interna e delle modalità di interfacciamento esistenti. Sarà poi descritto il formato e il modello dati utilizzato per la memorizzazione in locale dei metadati del servizio SOS e dei dati delle osservazioni, le operazioni che il client mette a disposizione dell'utente e l'interfaccia grafica realizzata.

Il quarto capitolo sarà principalmente dedicato al caso di studio Laguna di Venezia. Dopo una panoramica sulla Laguna, parleremo del percorso che i dati ambientali fanno dai sensori della Laguna ai server di CNR-ISMAR dove saranno resi accessibili via SOS. Vedremo quindi l'applicazione del nostro software all'ambito lagunare.

Le conclusioni che emergono dallo studio condotto sono esposte nel successivo e ultimo capitolo.

# Capitolo 1

# Dai sensori al Sensor Web

Negli ultimi anni a fronte del riscaldamento globale e dei suoi effetti devastanti sul pianeta, i dati ambientali sono diventati sempre più indispensabili per il monitoraggio della salute della Terra. Con la nascita e lo sviluppo delle reti di sensori, la disponibilità di dati ambientali è cresciuta considerevolmente ed è stato necessario costruire Sistemi Informativi atti a gestirli: dalla trasmissione di tali dati, alla loro memorizzazione ed accesso.

Gestire molteplici reti di sensori non è affatto semplice, soprattutto perché bisogna far fronte a problematiche di interoperabilità, scalabilità e flessibilità difficilmente risolvibili con l'utilizzo di SI etereogenei.

Per prendere decisioni bisogna spesso analizzare i dati delle osservazioni di più fenomeni, spesso provenienti da reti di sensori differenti. Tali dati potrebbero altresì essere di varia natura e gestiti da SI diversi. Per permettere l'interoperabilità di tali sistemi, e quindi l'utilizzo di sorgenti dati diverse vi è la necessità di definire e utilizzare modalità di accesso e formati dati omogenei.

In una situazione così variegata, diventa inoltre complicato rendere i dati dei sensori fruibili tramite web, per l'accesso da parte degli utenti finali o dei

sistemi che li vogliano utilizzare. La standardizzazione è dunque il requisito fondamentale per la comunicazione sia di informazioni sui sensori, sia dei dati essi acquisiti, ma anche per confrontare e combinare informazioni provenienti da sensori diversi.

## 1.1 Sensor Network

Una **rete di sensori** (anche detta sensor network) è costituita da un insieme di dispositivi che vengono disposti all'interno di un fenomeno da osservare, o in sua prossimità. Questi dispositivi chiamati **nodi sensori** (o sensor node, in gergo mote) sono in grado di rilevare i valori di parametri fisici dell'ambiente circostante, di eseguire delle semplici elaborazioni e comunicare con altri dispositivi della rete.

Una volta messo in opera, ogni nodo sensore deve lavorare in maniera autonoma, senza l'intervento umano. Per tale motivo le reti di sensori vengono utilizzate anche in contesti difficilmente accessibili, che vanno da applicazioni militari e ambientali all'ambito medico-sanitario a quello industriale, oltre che per le sempre più comuni applicazioni domestiche.

Per dare una definizione di nodo sensore è bene fare riferimento allo standard IEEE 1451 *"Standard for Smart Transducer Interface for Sensor and Actuators"*. Obiettivo di tale standard è quello di stabilire una serie di interfacce comuni per connettere sensori, e più in generale trasduttori, con dispositivi a microprocessore. Lo standard definire inoltre il prototipo di uno *smart transducer* indipendente dalla rete all'interno della quale sarà inserito.

Lo standard IEEE 1451.2 definisce uno **smart transducer** come:

> "un trasduttore che integra le funzioni necessarie alla corretta rappresentazione della grandezza misurata o controllata. Queste

> funzionalità tipicamente sono in grado di semplificare l'integrazione del trasduttore in applicazioni che utilizzino strutture di rete"[9].

Il termine *transducer* nello standard viene utilizzato come termine generico per sensori e attuatori; in questo elaborato faremo riferimento solo ai sensori, gli *smart sensors*, o nodi sensori appunto.

I nodi sensori sono costituiti da 5 componenti essenziali:

- sensing unit: composta da uno o più sensori ed eventualmente uno convertitore analogico-digitale. Un sensore è "un trasduttore che si trova in diretta interazione con il sistema misurato"[23], il cui proposito è individuare degli eventi o variazioni di parametri ambientali e restituire un output corrispondente. Il convertitore AD penserà a convertire gli output analogici in digitale per la fruizione da parte di un microprocessore.

- processing unit: composta da un microprocessore, pilota gli altri componenti del mote per portare a termine i task assegnati. In particolare riceve i task da portare a termine, riceve e memorizza i valori misurati dai sensori e provvede a comunicare e collaborare con gli altri nodi della rete.

- memory unit: si tratta di una memoria esterna utilizzata per immagazzinare i dati grezzi misurati e gli eventuali dati elaborati.

- transceiver unit: è l'unità che connette il nodo alla rete.

- power unit: si occupa di fornire l'energia necessaria al funzionamento del nodo sensore.

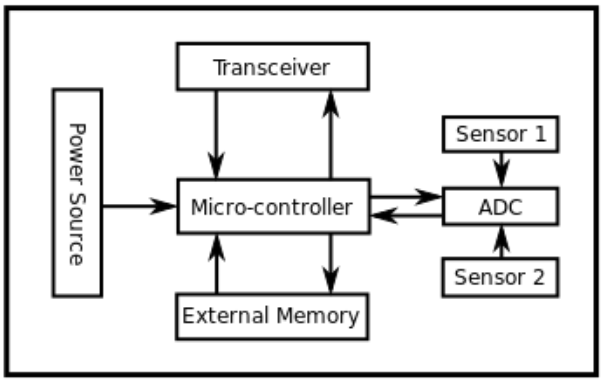


Figura 1.1: Struttura di un nodo sensore

Lo scopo principale di una rete di sensori è quello produrre un'informazione globale significativa a partire da dati locali provenienti dai singoli sensori per un periodo esteso di tempo.Per far questo devono essere realizzate per essere fault-tollerant, non dipendendenti da una particolare configurazione dei nodi.

## 1.2 Wireless Sensor Network

Una tipica rete di sensori è costituita da un numero elevato di nodi collegati tra loro.

Le prime realizzazioni prevedevano l'utilizzo di interfacce di comunicazione cablate. L'utilizzo di cavi consente l'impiego di dispositivi che non hanno limitazioni di potenza: laddove si è in grado di portare una connessione cablata sarà certamente possibile prevedere delle linee di alimentazione.

Tuttavia le reti cablate soffrono di gravi limitazioni, dovute principalmente al mezzo fisico di connessione che rende la struttura della rete inadatta ai cambiamenti. L'aggiunta di un nodo alla rete, ma anche il suo semplice spostamento, ha un impatto notevole sulla struttura, richiedendo spesso la

necessità di una revisione integrale del progetto originale. C'è poi da considerare la difficoltà di installazione di una rete cablata, specialmente in ambienti inospitali o difficilmente raggiungibili dall'uomo. Terzo aspetto da non sottovalutare, la realizzazione di sistemi ad alta densità di nodi sensore cablati è praticamente irrealizzabile o, nei casi in cui fosse fisicamente possibile, risulterebbe comunque economicamente improponibile.

L'uso di comunicazioni wireless risolve egregiamente questi problemi, sebbene ne introduca di nuovi. Bisogna infatti far fronte alla limitata energia di cui dispone il dispositivo, ma anche tenere conto dei limiti normativi sulle comunicazioni wireless, la diradazione del segnale e la sicurezza della comunicazione. Per molti di questi problemi esistono delle soluzioni efficaci che tuttavia implicano un notevole aumento di complessità del sistema che si riperquote sulle caratteristiche e il costo dei singoli componenti.

A parte i problemi non prettamente collegati alle reti di sensori wireless, quali le prestazioni, la sicurezza e in generale quelle che affliggono ogni tipologia di rete wireless, per i quali sono stati sviluppate negli anni svariate soluzioni, le Wireless Sensor Network (WSN) presentano delle particolarità che richiedono l'utilizzo di tecniche di rete specifiche.

Tra queste, requisiti specifici di una WSN sono:

- il numero dei nodi, che può essere molto maggiore rispetto ad una rete ad hoc, anche di qualche ordine di grandezza;
- l'elevata densità dei nodi, si può infatti arrivare ad avere centinaia di sensori in pochi metri;
- la topologia di rete che può variare in modo molto frequente a causa di guasi o alla mobilità dei nodi;

- la comunicazione tra i nodi, principalmente di tipo broadcast, a differenza delle reti ad hoc nelle quali di solito le comunicazioni sono punto-punto;
- le limitare risorse a disposizione dei singoli nodi, sia in termini di memoria e di capacità di calcolo che di alimentazione;
- la necessità di un identificativo univoco globalmente valido per i nodi;
- la sincronizzazione temporale dei nodi, che deve essere molto precisa, deterministica, per relazionare correttamente grandezze rilevate da sensori diversi;
- la tolleranza ai guasti della rete.

I requisiti menzionati in precedenza impongono dei vincoli molto stringenti sui protocolli e sugli algoritmi da adottare nelle WSN. In particolare, per mantenere i consumi contenuti si continuano a studiare protocolli e algoritmi power-aware, che ottimizzano il consumo energetico del dispositivo.

Dato il ridotto budget energetico e l'elevata densità dei nodi, l'uso delle tecnologie wireless a corto raggio e a bassa potenza costituisce un requisito imprescindibile, insieme con l'uso di algoritmi multi-hop che abilitano al loro utilizzo. Tuttavia realizzare un rete wireless di comunicazione multi-hop, che sia robusta ai guasti ed in grado di auto-configurarsi anche in ambienti ostili è tutt'oggi una sfida tecnologica notevole.

È inoltre necessario prevedere dei robusti meccanismi di sincronizzazione dei dispositivi, dal momento che la risoluzione temporale nelle applicazioni più stringenti può essere nell'ordine del microsecondo.

Lo sviluppo di reti wireless dedicate al mondo dei sensori ha visto nell'ultimo periodo l'affermazione delle *Low-Rate Wireless Personal Area Network*

(LR-WPAN), reti caratterizzate da dimensioni contenute (meno di 10 metri) e al contempo da bassi transfer-rate. Lo standard IEEE 802.15.4 rappresenta oramai un punto fermo nello sviluppo dei livelli PHY (Physical) e MAC (Media Access Control) per questa tipologia di reti. I livelli superiori dello stack ISO-OSI possono essere invece implementati seguendo la specifica ZigBee e le più recenti proposte emerse in ambito IETF per le cosiddette *low power lossy network*.

## 1.3 Sensor Web

Ai giorni nostri vi è sempre più spesso la necessità di utilizzare dati provenienti da sistemi di sensori differenti, metterli insieme ed analizzarli come se provenissero da un'unica sorgente dati. L'eterogeneità di tali dati rende tuttavia questa attività particolarmente complicata a causa della mancanza di operazioni e rappresentazioni uniformi per i dati di sensori.

La standardizzazione delle metodologie e dei formati per l'accesso, il recupero e la memorizzazione di informazioni sui sensori e delle loro osservazioni diventa quindi un aspetto di fondamentale importanza per garantire l'interoperabilità dei SI costruiti attorno ai sistemi di sensori.

Il concetto di **sensor web** venne utilizzato per la prima volta nel 1997 nel Jet Propulsion Laboratory della Nasa per descrivere un sistema di piattaforme sensoriali spazialmente distribuite e comunicanti tra loro dette *pods*[6],[7]. Ogni pod invia i dati raccolti a tutti gli altri pod del sensor web, in tal modo ognuno di essi è in grado di sapere ciò sta avvenendo intorno a se. Ti tratta quindi un sistema di sensori autonomo e distribuito, capace di identificare particolari condizioni sui dati in ingresso e reagire di consequenza in maniera coordinata.

Questa descrizione, nata nel contesto delle esplorazioni spaziali, si è via via ampliata ed evoluta. Il termine **Sensor Web** viene oggi utilizzato per indicare un universo di sistemi di sensori accessibili via web che agiscono in maniera collaborativa e autonoma per produrre un valore maggiore di quello risultante delle singole osservazioni[12].

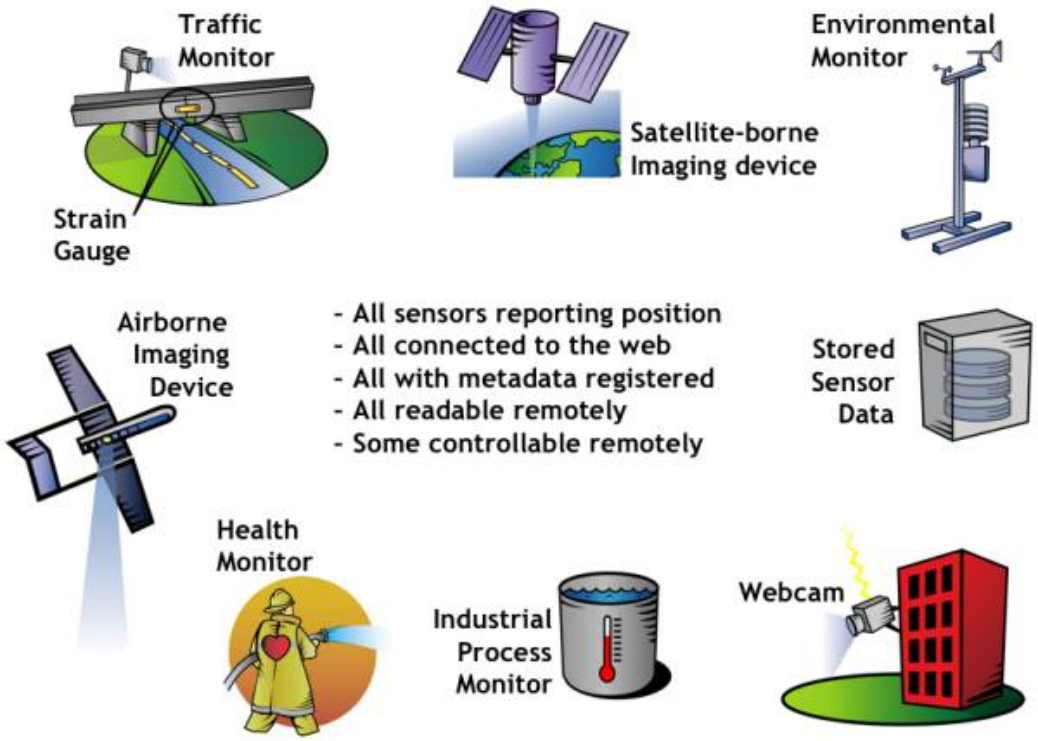


Figura 1.2: Esempi di sensori nel Sensor Web

Si tratta in sostanza di una infrastruttura che fornisce accesso a sistemi di sensori e dati di osservazioni eterogenei, dislocati fisicamente in posti differenti, di tipologia e con mobilità diversa, come se si trattasse di un sistema unico ed integrato.

L'architettura Sensor Web permette l'integrazione di sensori e i sistemi di sensori gestiti da organizzazioni diverse, con diversi vincoli di accesso e sicurezza, con differenti qualità dei dati e requisiti di performance, senza la

necessità di modifiche ai sistemi legacy esistenti. Nel Sensor Web i sensori e le reti di sensori sono risorse condivise, riutilizzabili da applicazioni differenti, ricercabili, accessibili e controllabili attraverso il World Wide Web[2].

## 1.4 Sensor Web Enablement

Molte differenti architetture sono state proposte per la realizzazione del Sensor Web, ma quella che ha avuto maggior successo è l'iniziativa Sensor Web Enablement (SWE)[3] di Open Geospatial Consortium (OGC).

Obiettivo primario di OGC è promuovere l'interoperabilità dei Sistemi Informativi Geografici (GIS) e SWE rientra in quest'ottica. Il gruppo di lavoro SWE ha eleborato e pubblicato un insieme di standard per la definizione di meccanismi di collaborazione così come modelli concettuali e formati per l'interscambio di dati astraendo dall'etereogeneità della comunicazione tra reti di sensori.

I dati scambiati possono essere di varia natura, dalle osservazioni ad informazioni descrittive sui sistemi di sensori coinvolti piuttosto che le procedure utilizzate per le misurazioni. La collaborazione è invece resa possibile tramite la definizione di interfacce standard, disponibili secondo il paradigma SOA, che abilitano produttori e consumatori di dati a interagire con il sistema.

SWE agisce come un layer middleware tra gli asset (siano essi sensori fisici, archivi di osservazioni, simulazioni o algoritmi di elaborazione di osservazioni) e gli strumenti che vogliono utilizzarli, come mostrato in figura 1.3.

Il framework SWE è stato realizzato per offrire le seguenti funzionalità:

**Registrare un servizio** - fornisce la possibilità ad un service provider di registrare un servizio SWE su un service registry.

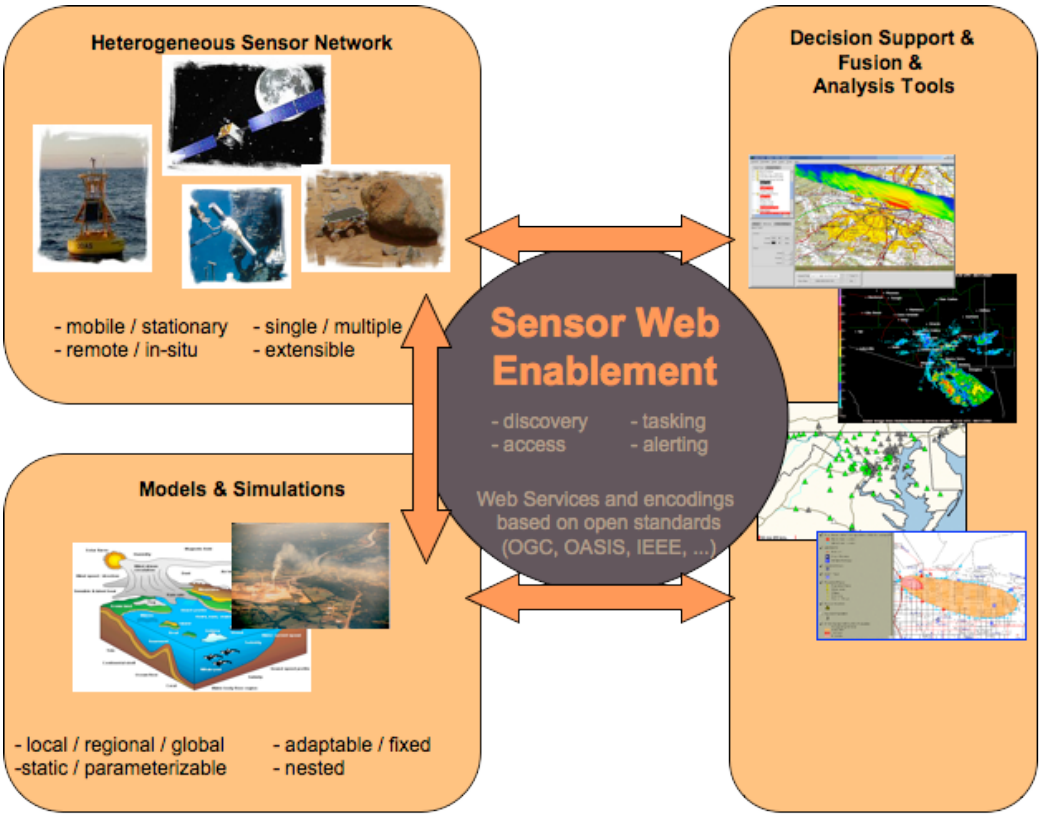


Figura 1.3: Framework SWE[3]

**Ricercare un servizio** - fornisce la possibilità di ricercare un servizio SWE registrato tramite una richiesta ad un service registry.

**Ottenere le capabilities del servizio** - fornisce all'utente i dettagli sulle funzionalità che il servizio mette a disposizione.

**Ottenere le osservazioni real-time** - permette all'utente di inviare un comando al servizio per ricevere un flusso di osservazioni real-time definendo eventuali filtri spaziali e non sulle osservazioni che verranno restituite.

**Recuperare le serie temporali delle osservazioni** - fornisce all'utente la possibilità di inviare un comando al servizio per recuperare dati

archiviati, permettendo di definire eventuali filtri spaziali, temporali e tematici sulle osservazioni che verranno restituite.

**Far eseguire un'attività ad un sensore** - fornisce all'utente la possibilità di inviare un comando ad uno specifico sensore per l'esecuzione di una attività.

**Sottoscrivere un allarme di un sensore** - permette all'utente di sottoscrivere un servizio di allarme per essere avvisato quando avviene un evento specificato dall'utente, come ad esempio se un indicatore in un'area di interesse supera una soglia stabilita dall'utente.

Per realizzare le precedenti funzionalità sono state sviluppate 3 modelli di codifica per descrivere sensori e relative osservazioni:

**Sensor Model Language (SensorML)** - modelli standard e XML Schema per descrivere sistemi di sensori e processi. Forniscono le informazioni necessarie per il discovery dei sensori, la localizzazione di osservazioni, l'elaborazione a basso livello delle osservazioni, la lista delle attività da che un sensore può eseguire.

**Observation and Measurements Schema (O&M)** - modelli standard e XML Schema per la codifica delle osservazioni e delle misurazioni provenienti da sensori, sia archiviate che real-time.

**Transducer Model Language (TransducerML)** - approccio concettuale e XML Schema per il supporto al streaming di dati real-time da e verso i sistemi di sensori.

Per la realizzazione dei web service sono invece state definite 4 interfacce standard:

**Sensor Observation Service (SOS)** - l'interfaccia del servizio per richiedere, filtrare e recuperare le osservazioni e le informazioni sui sistemi di sensori.

**Sensor Planning Service (SPS)** - l'interfaccia del servizio per richiedere acquisizioni e osservazioni, per ri-calibrare un sensore o far eseguire un'attività alla rete di sensori.

**Sensor Event Service (SES)** - l'interfaccia del servizio per pubblicare e sottoscrivere allarmi di sensori.

**Web Notification Service (WNS)** - l'interfaccia del servizio per la ricezione e la notifica asincrona di messaggi di allarme.

In figura 1.4 è raffigurato un esempio di deployment dei servizi SWE in uno scenario idrologico.

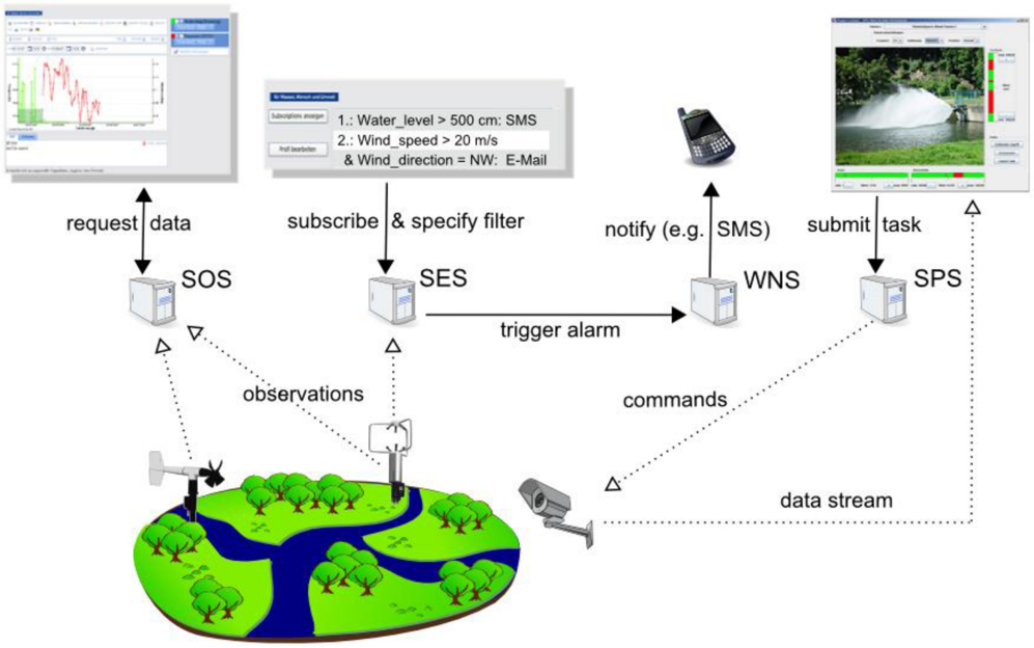


Figura 1.4: Scenario di deployment servizi SWE[1]

Tutti i precedenti standard e modelli facenti parte del framework SWE richiederebbero uno studio approfondito. Tuttavia solo l'applicazione pratica di questi strumenti permette di apprezzare appieno lo sforzo fatto dai membri di OGC per rendere il Sensor Web realizzabile.

Il lavoro di tesi si focalizza preferenzialmente su SOS, l'interfaccia standard di accesso ai dati sui sensori e alle misurazioni delle loro osservazioni, e sul modello dati delle osservazioni O&M, al fine di individuare e analizzare le criticità cui bisogna far fronte nel realizzare un client SOS.

# Capitolo 2

# Sensor Observation Service

Sensor Observation Service (SOS) è uno standard OGC che può essere utilizzato in tutti quei casi in cui è necessario gestire dati provenienti da sensori in maniera interoperabile.

Lo scopo principale di SOS è fornire un'interfaccia standard per la gestione e il recupero di metadati e osservazioni da sistemi di sensori eterogenei. Ai giorni nostri la maggior parte dei dati geospaziali utilizzati sono provenienti da sistemi di sensori. Questi includono sensori in-situ (ad esempio i rilevatori del livello dei fiumi), piattaforme di sensori mobili (ad esempio i satelliti) e reti di sensori fissi (ad esempio i sensori sismici).

Prima di tutto è bene chiarire le differenze che intercorrono tra i vari tipi di sensori dal momento che questi concetti verranno successivamente utilizzati.

I sensori possono essere distinti in base a 2 principali caratteristiche: la mobilità e la modalità di raccolta dei dati. Relativamente alla mobilità possiamo distinguere i *sensori fissi*, la cui posizione non cambia nel tempo, dai *sensori mobili*, la cui posizione è invece variabile. Differenziando i sensori in base alla modalità di raccolta dati avremo invece i *sensori in-situ*, che

devono essere posizionati direttamente nel punto nel quale il fenomeno da misurare si manifesta, e i *sensori remoti*, che si trovano a notevole distanza dall'oggetto di interesse, come nel caso di sensori operanti da satellite.

Storicamente gli utenti di dati di sensori erano divisi in due gruppi ben distinti proprio in base a quest'ultima caratteristica: vi erano chi aveva a che fare con sensori in-situ e quelli che principalmente utilizzavano dati di sensori remoti. Riuscire a mettere insieme questi due mondi, aventi terminologia, prospettive ed aspettative diverse ha costituito per anni una grande sfida nella realizzazione del Sensor Web.

Nell'ultimo decennio però la collaborazione tra OGC ed ISO ha portato a definire una base terminologica, modelli e protocolli che hanno abilitato allo sviluppo del Sensor Web in ambito geospaziale[8], tra cui lo standard Sensor Data Model for Imagery and Gridded Data[18], o ISO 19130, e chiaramente le specifiche OGC Sensor Web Enablement.

Prima di descrivere le operazioni fornite da un servizio SOS è necessario attenzionare il modello delle osservazioni come specificato dallo standard OGC O&M. Insieme al modello forniremo anche delle definizioni utili a comprendere i concetti alla base di tale modello.

Nei paragrafi successivi faremo riferimento alle specifiche SOS 2.0.

## 2.1 Osservations and Measurements

Lo standard OGC O&M (Observations and Measurements), o ISO 19156, definisce dei modelli di dati generici per la descrizione delle osservazioni.

O&M definisce una **observation** (o *osservazione*) come:

"l'azione, associata ad un istante o periodo di tempo, attraverso la quale un numero, un termine o altro simbolo è assegnato

ad un fenomeno"[5].

La figura 2.1 mostra il modello di una generica **observation**.

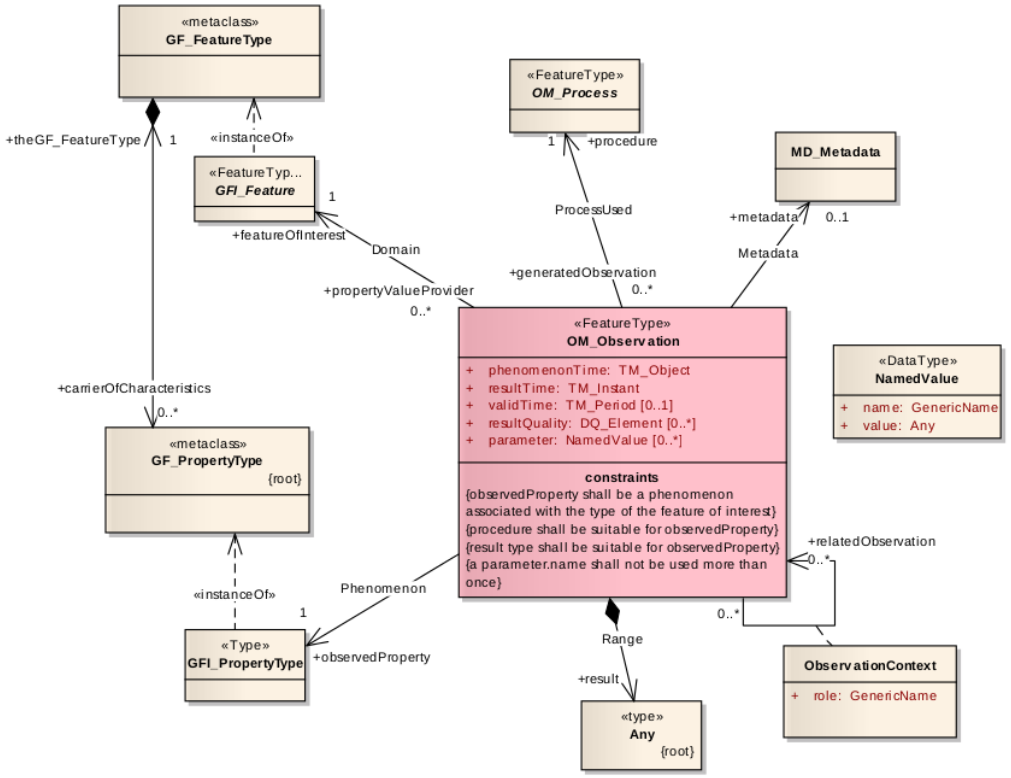


Figura 2.1: Modello base di un oggetto Observation

Lo schema UML mostra le varie relazioni e proprietà che contraddistinguono una generica Observation.

La **Feature of Interest**, abbreviato *FoI*, è la rappresentazione del fenomeno del mondo reale preso come target dell'osservazione. La FoI permetteno di associare più osservazioni allo stesso fenomeno, inoltre fungono da collante con lo standard ISO 19109 - General Feature Model (GFM) che definisce i concetti base di *Feature* e *Property*. Esempi di FoI sono la torre di Pisa, il golfo del Messico, ma anche il tornado Katrina.

La **Observed Property** è la descrizione associata alla proprietà osservata, ad esempio temperatura, livello del mare, direzione del vento.

Il **Result** è una stima del valore di una proprietà di una FoI. Il modello non definisce il tipo di Result poiché questo è strettamente collegato alla Observed Property. Un Result può essere un valore numerico o un termine di un dizionario qualora la proprietà sia scalare (come il peso, l'altezza, la densità) oppure un oggetto strutturato qualora si tratti di una proprietà complessa (quale il tempo, la posizione, il colore). Nel primo caso l'osservazione viene anche detta *measurement* (o misurazione), nel secondo si parla invece di *classification* (o classificazione). Esistono poi Result che sono funzioni dello spazio o del tempo, qualora la proprietà sia variabile, come la posizione di un veicolo, la temperatura misurata da una stazione metereologica, o la salinità lungo un estuario. In tal caso il Result sarà una *Function* o *Coverage*. Una Coverage può essere codificata in due modi distinti:

- una funzione dominio-valori, rappresentata da una griglia di valori, comunemente utilizzata per le immagini;
- una lista di coppie geometria-valore, usata invece per le serie temporali.

La **Procedure** è il processo che ha effettuato l'osservazione, ovvero la procedura che applicata sulla FoI ha generato il Result corrispondente. Questa può essere generica e riusabile oppure specifica per l'osservazione per cui viene utilizzata. Esempi di procedure sono un sensore fisico, un algoritmo o una simulazione. Una procedure può essere descritta utilizzando le specifiche fornite dallo standard OGC SensorML la cui trattazione esula dagli scopi di questa tesi.

Relativamente ai tempi associati all'osservazione distinguiamo:

- **phenomenonTime** che è l'istante o il range temporale durante il quale è stata effettuata l'osservazione, cioé quando è avvenuta l'interazione tra la Procedure e la FoI;

- **resultTime** che rappresenta il momento in cui il risultato è diventato disponibile;

- **validTime** che invece rappresenta il range temporale di valida del risultato, cioé il periodo di tempo durante il quale tale dato dovrebbe essere usato.

Dal punto di vista della localizzazione dell'osservazione, la separazione tra FoI e Procedure permette al modello di mappare sia le osservazioni il cui fenomeno sia osservabile in-situ che quelle remote relative a fenomeni non accessibili. Tuttavia esistono anche dei casi in cui il fenomeno di interesse abbia proprietà non direttamente osservabili.

Per meglio inquadrare nel modello questi casi sono stati introdotti i concetti di *Sampling Feature* e di *Specimen*. Lo schema in figura 2.2 rappresenta la loro relazione con la classe Observation.

La **Sampling feature** (o fenomeno campione) rappresenta la parte del fenomeno su cui l'osservazione è stata effettuata. Spesso infatti le proprietà osservabili di un fenomeno non sono quelle di nostro ultimo interesse. Un esempio potrebbe essere l'analisi del tipo di vegetazione fatta attraverso l'osservazione della riflessione dei colori tramite sistemi satellitari. In tal caso la riflessione dei colori è una proprietà dell'immagine fotografica del lotto di terreno (*Sampling Feature*), mentre la proprietà che si vuole analizzare è legata piuttosto al lotto (*Sampled Feature*).

Con il termine **Specimen** (o campione) si identifica invece l'eventuale campione fisico di materia su cui sono effettuate le operazioni di osservazione.

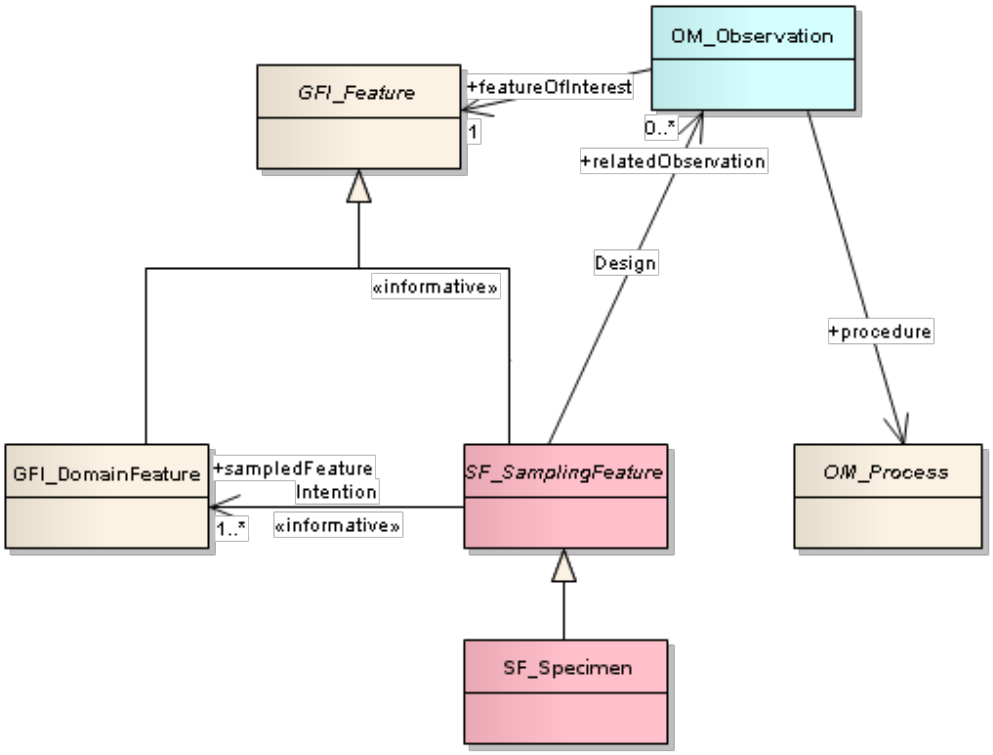


Figura 2.2: Relazioni tra Observation, Sampling Feature e Specimen

Ci sono infatti casi in cui il dominio della FoI non è completamente osservabile (ad esempio è troppo ampio) e si deve ricorrere a dei campioni per stimarlo. Un esempio sono le analisi fatte in laboratorio.

Riepilogando, O&M permette di modellare i seguenti tipi di osservazioni:

- osservazioni in-situ, nelle quali la procedura viene applicata direttamente sulla FoI (ad esempio sensori in-situ);
- osservazioni remote, quelle cioé relative a fenomeni non accessibili perché distanti dal luogo in cui la procedura è stata applicata (come nel caso di sensori remoti);

- osservazioni di fenomeni le cui proprietà non sono direttamente osservabili;
- osservazioni ex-situ, quelle fatte su campioni prelevati dal luogo in cui il fenomeno si manifesta;
- osservazioni in cui l'informazione sul luogo dell'osservazione non è disponibile perché sconosciuta oppure concettualmente non valida (come nel caso di alcune simulazioni).

Facciamo adesso un esempio di come lo standard O&M permetta di schematizzare una osservazione.

Prendiamo l'osservazione "Sample WMC997t collected at Empire Dam on 1996-03-30 was found to have 5.6 g/T Au as measured by ICPMS at ABC Labs on 1996-05-31"[4].

Lo schema O&M dell'osservazione è mostrato in figura 2.3.

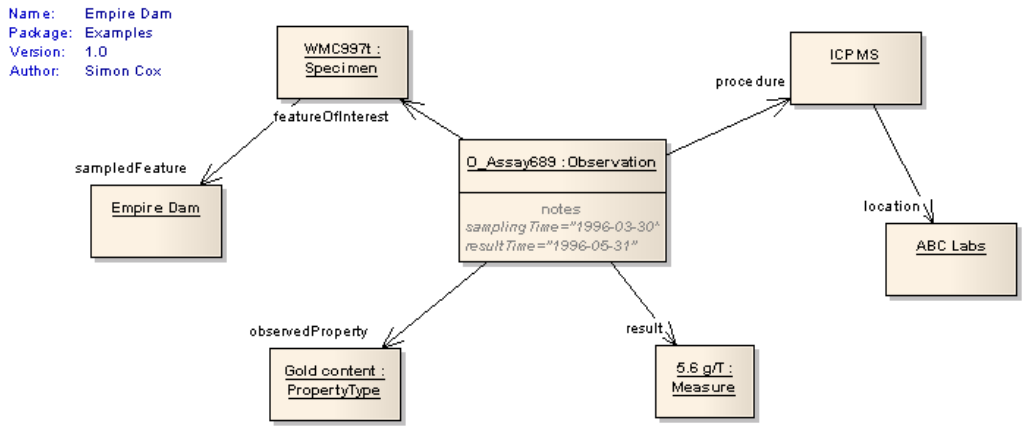


Figura 2.3: Schema O&M dell'osservazione Empire Dam

## 2.2 Specifiche SOS

Un servizio SOS è costituito da vari moduli ognuno dei quali realizza un insieme delle funzionalità del servizio.

In figura 2.4 sono rappresentati i vari moduli SOS come definiti nella versione 2.0 dello standard e le specifiche OGC di base che questi utilizzano.

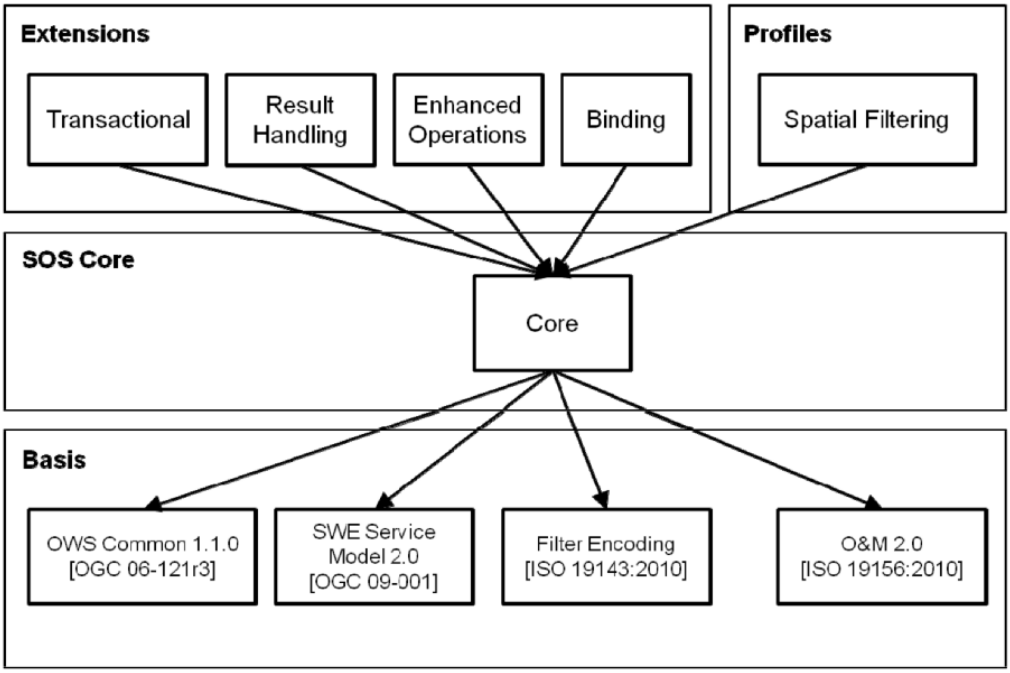


Figura 2.4: Dipendenza tra SOS Core, Extensions e specifiche OGC di base

Il nucleo di un servizio SOS è il modulo *SOS Core*, che rappresenta l'interfaccia minima che ogni servizio SOS deve implementare.

SOS Core è costituito dalle seguenti 3 operazioni:

**GetCapabilities** - utilizzata per recuperare informazioni descrittive sul servizio, i cosidetti metadati, e informazioni dettagliate sulle operazio-

ni che il servizio mette a disposizione, quali le modalità di richiesta supportate per le varie operazioni, i parametri etc.

**DescribeSensor** - permette di interrogare il servizio relativamente ai metadati dei sensori e sistemi di sensori disponibili.

**GetObservation** - permette l'accesso ai dati delle osservazioni. Nella richiesta è possibile specificare eventuali filtri da utilizzare, spaziali, temporali e/o tematici.

Esistono poi delle estensioni del Core che aggiungono al servizio ulteriori funzionalità. Tra queste la specifica SOS v2.0 definisce *Enhanced Operations Extension*, *Transactionali Extension* e *Result Handling Extension*.

Le operazioni definite in Enhanced Operations Extension realizzano una prima estenzione delle funzionalità di base per l'accesso ai dati sulle FoI e alle osservazioni:

**GetFeatureOfInterest** - da accesso alle informazioni sulle Feature per le quali il servizio rende disponibili le osservazioni.

**GetObservationById** - permette di accedere specifiche osservazioni conoscendo soltanto il loro ID.

Transactional Extension permette di interagire con il servizio SOS per l'inserimento di sensori e osservazioni:

**InsertSensor** - permette la registrazione di un nuovo sensore.

**DeleteSensor** - permette l'eliminazione di un sensore precedentemente registrato e di tutte le osservazioni ad esso associate.

**InsertObservation** - operazione che permette l'inserimento di nuove osservazioni sul server SOS.

L'estensione Result Handling Extension permette invece di recuperare i soli risultati delle osservazioni senza i relativi metadati:

**InsertResultTemplate** - permette l'inserimento di un template per le osservazioni, contenente i metadati e la struttura dei risultati, necessario per inserire risultati di osservazioni tramite l'operazione InsertResult.

**InsertResult** - permette di inserire nuovi risultati di osservazioni in base ad un template precedentemente inserito.

**GetResultTemplate** - permette di accedere ad un template contenente la struttura dei risultati restituiti dalle successive invocazioni dell'operazione GetResult.

**GetResult** - operazione che permette di recuperare i risultati di osservazioni senza i relativi metadati né informazioni sulla struttura di tali risultati.

Per eseguire una operazione, un client può inviare una richiesta GET o POST all'URL del servizio SOS specificando il nome dell'operazione. Ogni operazione inoltre prevede un set di parametri, alcuni dei quali obbligatori, che il client può fornire nella richiesta.

Le informazioni scambiate tra client e server devono essere codificate. A tale scopo lo standard SOS definisce un set minimo di **binding** che un servizio SOS 2.0 deve supportare per essere compliant:

- i binding XML, spesso chiamato POX o Plain Old XML, per i quali lo standard fornisce gli Schema XML sia delle richieste che delle risposte;
- i binding SOAP che racchiudono richieste e risposte XML all'interno di un envelope SOAP;

- i binding KVP (Key-Value Pair), unico formato di codifica disponibile per le richieste HTTP GET, in cui i parametri della richiesta sono codificati con coppie chiave-valore ed inseriti nella query-string. Le risposte saranno codificate in XML se non diversamente specificato nella richiesta.

Ogni server può tuttavia supportare codifiche aggiuntive tramite l'implementazione di una estensione che realizzi uno o più binding differenti.

Vediamo più nel dettaglio le operazioni fondamentali per il recupero dei dati delle osservazioni da un server SOS.

### 2.2.1 GetCapabilities

Per comunicare con un servizio web la prima cosa da fare è sicuramente reperire alcune informazioni sul servizio. Bisogna infatti sapere cosa fa il servizio, quali sono le operazioni disponibili e i loro parametri, i formati e le codifiche supportate. Oltre queste informazioni di carattere generale, ogni servizio web può fornire delle informazioni caratteristiche dello specifico servizio. Nel caso di un servizio SOS è ad esempio necessario conoscere la lista dei sensori (procedure) disponibili, l'insieme delle proprietà osservate, le feature di riferimento. A tale scopo i servizi OGC mettono a disposizione l'operazione **GetCapabilities**. Una richiesta GetCapabilities permette ad un client di recuperare tutta una serie di informazioni descrittive sul servizio, i cosiddetti metadati del servizio.

I listati 1 e 2 mostrano due esempi di richiesta GetCapabilities per un servizio SOS versione 2.0 raggiungibile all'URL http://www.example.org/sos.

La prima è una richiesta minimale codificata in KVP, la seconda è codificata in XML e specifica anche quali sezioni di metadati recuperare.

```
http://www.example.org/sos?service=SOS&request=GetCapabilities
```

Listing 1: Esempio di richiesta GetCapabilities minimale codificata in KVP

```
<?xml version="1.0" encoding="UTF-8"?>
<sos:GetCapabilities service="SOS"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sos="http://www.opengis.net/sos/2.0"
    xmlns:ows="http://www.opengis.net/ows/1.1"
    xmlns:swe="http://www.opengis.net/swe/2.0"
    xsi:schemaLocation="http://www.opengis.net/sos/2.0
    http://schemas.opengis.net/sos/2.0/sosGetCapabilities.xsd">
    <ows:AcceptVersions>
        <ows:Version>2.0.0</ows:Version>
    </ows:AcceptVersions>
    <ows:Sections>
        <ows:Section>ServiceIdentification</ows:Section>
        <ows:Section>ServiceProvider</ows:Section>
        <ows:Section>OperationsMetadata</ows:Section>
        <ows:Section>Contents</ows:Section>
    </ows:Sections>
</sos:GetCapabilities>
```

Listing 2: Esempio di richiesta GetCapabilities codificata in XML

La risposta è divisa in sezioni, ognuna delle quali contiene delle informazioni relativi a contesti differenti:

**ServiceIdentification** - contiene le informazioni per identificare il servizio SOS, in particolare titolo e abstract del servizio, il tipo di servizio (SOS nel nostro caso) e le versioni supportate;

**ServiceProvider** - contiene nome e informazioni di contatto del provider che fornisce il servizio;

**OperationsMetadata** - elenca le operazioni disponibili per il servizio, specificando per ognuno l'URL a cui bisogna fare la richiesta, il tipo di richiesta (ad esempio HTTP GET o POST), gli eventuali parametri e le codifiche supportate;

**FilterCapabilities** - elenca quali sono gli operatori e gli operandi disponibili per la definizione un filtro temporale, spaziale e/o tematico sui risultati;

**Contents** - elenca i dati che il servizio mette a disposizione, organizzati in *Offerings*.

Un servizio SOS organizza collezioni di osservazioni correlate nelle cosiddette **Observation Offerings**, in breve Offerings. Ogni Offering deve specificare un insieme di parametri utili ad identificare i dati in essa contenuti:

- la procedura che ha fornito tutte le osservazioni;
- il periodo temporale all'interno del quale ricadono le osservazioni dell'Offering;
- i fenomeni cui le osservazioni fanno riferimento;
- l'area geografica che contiene i sensori;
- l'area geografica oggetto delle osservazioni, che risulta essere distinta dalla precedente nel caso di sensori remoti.

### 2.2.2 GetFeatureOfInterest

Spesso è utile recuperare la lista delle FoI gestiti da un servizio SOS per visualizzarli ad esempio su una mappa. L'operazione GetFeatureOfIntereset permette di farsi restituire un tutte le FoI dal server SOS, oppure un loro sotto-insieme mediante l'utilizzo di filtri spaziali o definendo filtrando le FoI in base ai valori delle proprietà delle osservazioni ad essi collegati, quali la Observed Property o la Procedure utilizzata.

Segue un esempio di richiesta GetFeatureOfInterest per recuperare solo le Feature che stanno all'interno dell'area passata come parametro del filtro spaziale (listato 3).

```
<?xml version="1.0" encoding="UTF-8"?>
<sos:GetFeatureOfInterest service="SOS" version="2.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sos="http://www.opengis.net/sos/2.0"
    xmlns:fes="http://www.opengis.net/fes/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:swe="http://www.opengis.net/swe/2.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:swes="http://www.opengis.net/swes/2.0"
     ↪   xsi:schemaLocation="http://www.opengis.net/sos/2.0
     ↪   http://schemas.opengis.net/sos/2.0/sos.xsd">
    <sos:spatialFilter>
        <fes:BBOX>
            <fes:ValueReference>sams:shape</fes:ValueReference>
            <gml:Envelope srsName="http://www.opengis.net/def/crs/EPSG/0/4326">
                <gml:lowerCorner>0 0</gml:lowerCorner>
                <gml:upperCorner>60 60</gml:upperCorner>
            </gml:Envelope>
        </fes:BBOX>
    </sos:spatialFilter>
</sos:GetFeatureOfInterest>
```

Listing 3: Esempio di richiesta GetFeatureOfInterest

### 2.2.3 GetObservation

L'operazione GetObservation permette ad un client di recuperare la lista delle osservazioni. Di solito si è interessati alle osservazioni di uno specifico fenomeno o relative ad una particolare proprietà osservabile che si sono verificate in un range di tempo definito. A tale scopo l'operazione GetObservation permette di impostare un filtro temporale, spaziale e/o tematico per farsi restituire solo le osservazioni di interesse.

Il listato 4 mostra un esempio di richiesta GetObservation effettuata per recuperare le osservazioni del livello del fiume Arno rilevate a Pisa - Ponte di Mezzo il giorno 27 Novembre 2015.

Oltre ai vari filtri è stato anche specificato il formato con cui saranno codificati i dati delle osservazioni nella risposta, O&M 2.0 nel precedente esempio, tuttavia un client può indicare qualunque formato supportato dal server elencato nella sezione OperationsMetadata della risposta alla GetCapabilities per

```
<?xml version="1.0" encoding="UTF-8"?>
<sos:GetObservation service="SOS" version="2.0.0"
    xmlns:sos="http://www.opengis.net/sos/2.0"
    xmlns:fes="http://www.opengis.net/fes/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:swe="http://www.opengis.net/swe/2.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:swes="http://www.opengis.net/swes/2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/sos/2.0
     ↪  http://schemas.opengis.net/sos/2.0/sos.xsd">
    <sos:observedProperty>http://www.example.org/observableProperty/waterlevel</sos:observe
     ↪  dProperty>
    <sos:temporalFilter>
        <fes:During>
            <fes:ValueReference>phenomenonTime</fes:ValueReference>
            <gml:TimePeriod gml:id="tp_1">
                <gml:beginPosition>2015-11-27T00:00:00.000Z</gml:beginPosition>
                <gml:endPosition>2015-11-27T23:59:59.999Z</gml:endPosition>
            </gml:TimePeriod>
        </fes:During>
    </sos:temporalFilter>
    <sos:featureOfInterest>http://www.example.org/featureOfInterest/arno/pisa_pontedimezzo<
     ↪  /sos:featureOfInterest>
    <sos:responseFormat>http://www.opengis.net/om/2.0</sos:responseFormat>
</sos:GetObservation>
```

Listing 4: Esempio di richiesta GetObservation

l'operazione GetObservation.

La risposta elenca tutte le osservazioni che rispondono alle caratteristiche specificate nella richiesta, codificate in XML secondo lo standard O&M 2.0.

## 2.3 The Angle Bracket Tax: limiti e soluzioni

La comunicazione con un servizio SOS avviene solitamente mediante scambio di messaggi XML. A parte casi particolari quali le richieste codificate in KVP, tutto il resto dei dati scambiati tra client e server è codificato in XML (o SOAP/XML): le richieste, le risposte, ma anche i dati relativi ai sensori e quelli delle osservazioni.

XML è un formato molto flessibile, auto-descrittivo ed estendibile, permette di creare strutture complesse e, grazie agli Schema XML, permette

la tipizzazione dei dati e fornisce metodi di validazione del contenuto: ha cioè molte caratteristiche utili alla realizzazione di sistemi interoperabili. Per questo motivo rappresenta l'encoding di riferimento per servizi web realizzati secondo il paradigma SOA (Service Oriented Architecture).

Tuttavia, se la quantità di dati che un server deve inviare al client è ingente, un formato verboso come XML produce parecchio overhead sia lato server, che dovrà codificare i dati e spedirli, che lato client, che dovrà attendere la risposta e fare il parsing del messaggio ricevuto.

L'overhead è sostanzialmente proporzionale alla dimensione dei messaggi scambiati. Ciò implica che, a parità di banda, per trasferire un messaggio XML sia necessario un tempo più lungo rispetto ad un formato sviluppato appositamente per mantenere contenuta l'occupazione di banda. Si stima infatti che un messaggio XML in plain text sia circa l'80% più lungo del messaggio JSON contenente le medesime informazioni[10].

Questo problema è riferito spesso come *The Angle Bracket Tax*. Se per anni XML non ha avuto concorrenti, ai giorni nostri si sta assistendo ad una graduale presa di piede dei formati lightweight, tra i quali JSON.

La ragione di questo cambiamento è dovuta alla crescita esponenziale di produttori e consumatori di dati via web. Infatti se da una parte aumentano i dispositivi client sempre connessi che fanno un uso intensivo di webservices, dall'altra sempre più informazioni vengono rese disponibili tramite servizi web, buona parte delle quali proviene proprio da reti di sensori.

Se consideriamo ad esempio che un singolo sensore è in grado di effettuare anche centinaia di migliaia di misurazioni al giorno (ad esempio una ogni secondo o sua frazione), possiamo avere in mente quale sia l'impatto del sensor web sui sistemi che dovranno gestire tali dati, sebbene nella maggior parte dei casi i dati di sensori non vengano utilizzati direttamente, ma vengano

piuttosto aggregati già prima di essere memorizzati.

Per risolvere questo problema lo standard SOS 2.0 mette a disposizione delle operazioni che permettono di ridurre l'overhead quando si vogliano recuperare grandi moli di dati. Tali operazioni sono quelle fornite dall'estensione SOS Result Handling Extension, cioé *GetResultTemplate* e *GetResult*.

Una volta che server e client sono allineati sul template da usare, il client può richiedere in modo separato i metadati dei campi definiti nel template, che risultano identici per tutto il gruppo di osservazioni richieste, e i relativi valori, diversi invece per ogni osservazione.

In questo modo per prima cosa viene eliminata la necessità di recuperare per ogni osservazione anche i relativi metadati. In secondo luogo, se nella risposta contenente i metadati questi sono ancora codificati in XML, per l'elenco dei valori viene invece utilizzato un formato DSV (Delimiter-Separated Values), costituito solo da valori e separatori, così da ridurre notevolmente la dimensione del messaggio da trasferire al client.

### 2.3.1 GetResultTemplate

L'operazione GetResultTemplate permette ad un client di richiedere al server SOS i metadati di un gruppo di osservazioni secondo la struttura definita tramite una precedente richiesta InsertResultTemplate. Lo scopo principale è quello di separare il recupero dei metadati delle osservazioni dai dati ad esse associati. I parametri da specificare nella richiesta sono Offering e Observed Property.

Il listato 5 mostra un esempio di richiesta GetResultTemplate.

La risposta contiene solo i metadati dei campi definiti nel template, senza i relativi valori per recuperare i quali è necessario effettuare una richiesta GetResult. Inoltre sono indicati i delimitatori utilizzati per codificare in

```
<?xml version="1.0" encoding="UTF-8"?>
<sos:GetResultTemplate service="SOS" version="2.0.0"
    xmlns:sos="http://www.opengis.net/sos/2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/sos/2.0
     ↪  http://schemas.opengis.net/sos/2.0/sos.xsd">
    <sos:offering>http://www.example.org/offering/arno/pisa_pontedimezzo_2015</sos:offerin
     ↪  g>
    <sos:observedProperty>http://www.example.org/observableProperty/waterlevel</sos:observe
     ↪  dProperty>
</sos:GetResultTemplate>
```

Listing 5: Esempio di richiesta GetResultTemplate

formato DSV i valori dei campi del template restituiti come risposta ad una eventuale richiesta GetResult.

### 2.3.2 GetResult

Contestualmente al recupero dei metadati è necessario recuperare i valori associati ai campi del template. L'operazione GetResult restituisce tutti e solo i valori dei campi delle osservazioni definiti nel template di risultati.

I parametri da specificare sono gli stessi utilizzati nella richiesta GetResultTemplate, cioé Offering e Observed Property.

Il listato 6 mostra un esempio di richiesta GetResult.

```
<?xml version="1.0" encoding="UTF-8"?>
<sos:GetResult service="SOS" version="2.0.0"
    xmlns:sos="http://www.opengis.net/sos/2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/sos/2.0
     ↪  http://schemas.opengis.net/sos/2.0/sos.xsd">
    <sos:offering>http://www.example.org/offering/arno/pisa_pontedimezzo_2015</sos:offerin
     ↪  g>
    <sos:observedProperty>http://www.example.org/observableProperty/waterlevel</sos:observe
     ↪  dProperty>
</sos:GetResult>
```

Listing 6: Esempio di richiesta GetResult

La risposta contiene per ogni osservazione i valori dei campi definiti nel template. Tali valori sono codificati in un formato DSV ed inclusi in una intestazione XML. I delimitatori utilizzati come separatori di record e valori sono quelli restituiti dalla precedente richiesta GetResultTemplate.

# Capitolo 3

# Realizzazione client SOS per QGIS

Dopo aver analizzato nei precendenti capitoli lo standard OGC Sensor Observation Service (SOS) e i modelli utilizzati per le osservazioni di dati ambientali provenienti da reti di sensori, possiamo presentare il client software che abbiamo realizzato per rendere tali dati accessibili e analizzabili tramite il software QGIS.

In questo capitolo verranno descritte le problematiche cui abbiamo dovuto far fronte per la realizzazione del client SOS per QGIS chiamato **QGIS2SOS**. In particolare analizzeremo la modalità di integrazione tra lo strumento sviluppato e il software GIS suddetto e la memorizzazione in locale dei dati dei sensori.

Partiremo prima di tutto fornendo al lettore i motivi alla base della scelta di QGIS. Seguirà quindi una breve introduzione a QGIS, la sua struttura e i modelli utilizzati per rappresentare l'informazione geospaziale. Queste informazioni sono necessarie nei paragrafi seguenti per meglio capire le questioni che sono state affrontate.

## 3.1 Perché QGIS

Certamente non possiamo esimerci dallo spiegare quale siano i motivi che hanno portato a scegliere QGIS tra le tante soluzioni GIS open-source che non supportano SOS.

Principalmente perché è uno dei software GIS desktop più usati al mondo, sia da parte dei professionisti del settore che dalle Pubbliche Amministrazioni, e la mancanza di supporto SOS rappresenterebbe nel prossimo futuro un grosso limite per gli utenti.

In secondo luogo perché si tratta di **GFOSS**, software geografico libero.

In Italia dal 2014 l'Agenda Digitale ha stabilito che le PA devono dare la precedenza al FOSS su soluzioni proprietarie equivalenti. Il software libero presenta infatti delle peculiarità: chiunque può studiare, modificare, eseguire e ridistribuire il software e il suo codice sorgente. In ottica di riuso, un componente sviluppato da una PA può essere utilizzato dalle altre, ma anche essere modificato per adattarlo a specifiche esigenze e le modifiche possono altresì essere ridistribuite.

Altro motivo, tutto personale, è che dal 2009 participo attivamente al progetto. Faccio infatti parte del team di sviluppo di QGIS, i cosiddetti *Core developer*, e credo fermamente che QGIS sia una valida alternativa alle soluzioni GIS proprietarie esistenti, non solo a livello di funzionalità, quanto più per il suo modello di sviluppo community-driven.

### 3.1.1 I numeri

QGIS è un software GIS (Geografic Information System) open source multipiattaforma ed è uno dei progetti ufficiali della Open Source Geospatial Foundation (OSGeo).

Nato nel 2002 come semplice visualizzatore di dati GIS, ai giorni nostri QGIS è tradotto in più di 25 lingue ed ha raggiunto uno stato di maturità tale da essere utilizzato da sempre più persone per il loro lavoro quotidiano in ambito GIS, sia in Europa che in campo internazionale. È sviluppato in linguaggio C++ e utilizza le librerie Qt (vedi Appendice B), ciò gli permette di essere compilato e distribuito per svariate piattaforme.

Stanti ai dati di OpenHub, il progetto conta più di 250 contributori[13] di cui circa 50 sono sviluppatori core, aventi cioé accesso in scrittura al repository ufficiale di QGIS (vedi `https://github.com/qgis`). Riguardo la platea di utenti che lo utilizza, non vi sono dati ufficiali circa il loro numero. Tuttavia se nel 2011 erano stati stimati essere 100k, oggi si parla di circa 250k utenti[19].

QGIS supporta nativamente un considerevole numero di formati raster e vettoriali[16]:

- tabelle e viste spaziali realizzate con PostGIS, Spatialite, Oracle Spatial, MS SQL Spatial
- formati vettoriali supportati dalla libreria OGR (vedi `http://www.gdal.org/ogr_formats.html`)
- formati raster supportati dalla libreria GDAL (vedi `http://www.gdal.org/formats_list.html`)
- vettori e raster GRASS GIS (vedi `https://grass.osgeo.org/`),
- dati spaziali accessibili tramite Web Service OGC, quali WMS, WMTS, WMS-C, WFS, WFS-T, WCS.

Permette inoltre di effettuare sofisticate analisi spaziali sui dati tramite lo strumento Processing che, oltre a fornire un'interfaccia unica e semplificata

per l'uso di routine di analisi sia native che provenienti da altri programmi come GDAL, SAGA, OTB, GRASS, fTools, permette all'utente di combinare tra loro più algoritmi in maniera grafica grazie al Processing Modeler.

È anche possibile estendere le funzionalità esistenti installando dei plugin aggiuntivi. Il repository dei plugin (vedi `https://plugins.qgis.org/`) ne contiene più di 500 che coprono le esigenze più particolari, mentre quelli che si rivelano di uso generale vengono via via rilasciati in bundle con QGIS.

### 3.1.2 QGIS internals

La finestra principale di QGIS si presenta all'utente con una mappa centrale, il cosiddetto *canvas*, sulla quale possono essere disposti vari *layer* (strati).

Ad ogni layer è associato un sistema di riferimento, o *CRS* (Coordinate Reference System), e uno stile personalizzabile dall'utente che definisce come gli oggetti contenuti debbano essere disegnati sul canvas. Un layer può essere di 2 tipi:

- **layer vettoriale**, contenente geometrie (punti, linee, poligoni) a cui sono associate un insieme di informazioni;
- **layer raster**, che rappresenta una griglia georiferita, nel qual caso le informazioni sono rappresentate dal valore associato alle singole celle della matrice.

Ogni layer ha un *data provider* associato che si occupa di recuperare i dati dal *data source* (sorgente dati) ed esporli al layer soprastante. Sarà poi il *renderer* ad occuparsi di disegnare sulla mappa e nella corretta posizione gli oggetti contenuti nel layer, utilizzando lo stile che l'utente ha predisposto

e tenendo conto sia del sistema di riferimento associato al layer che di quello del progetto.

In figura 3.1 è mostrato lo schema ad alto livello dell'architettura di QGIS.

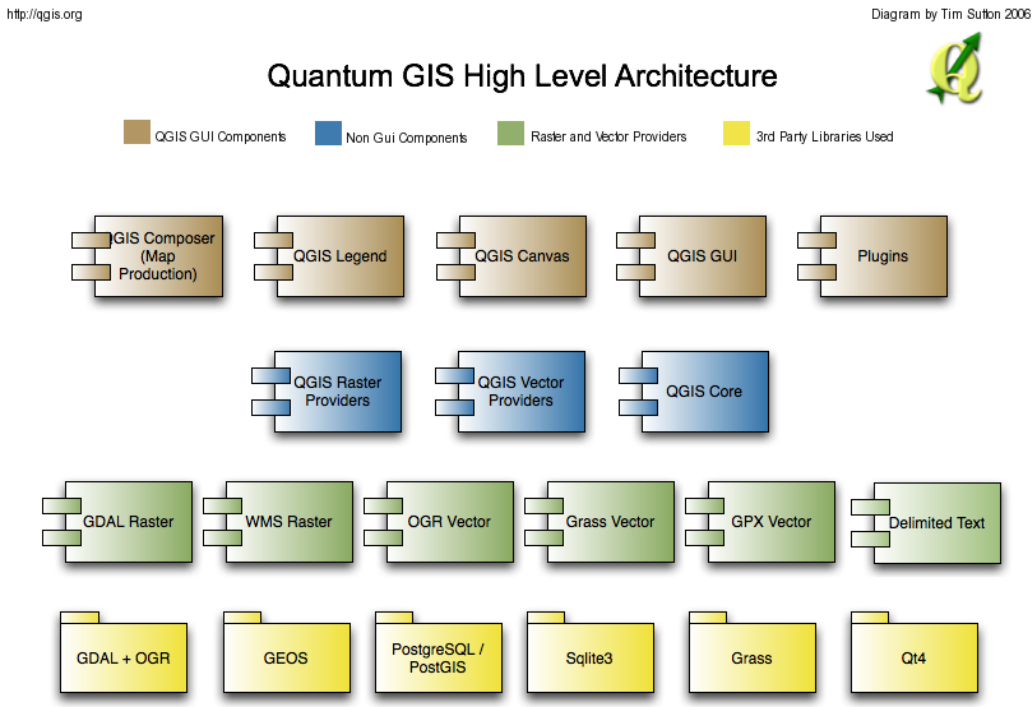


Figura 3.1: Architettura QGIS [fonte: `www.qgis.org`]

QGIS è composto da parecchi moduli, ognuno dei quali fornisce delle funzionalità specifiche. Possiamo distinguere i moduli in 4 strati:

- superiormente si trova lo strato GUI, contenente i moduli che fungono da interfaccia per l'utente o che, come il package Plugin, permettono l'estensione delle funzionalità utente.

- sotto troviamo il layer Core, contenente i moduli che realizzano le funzionalità GIS vere e proprie e le classi di l'astrazione dei provider dati del layer sottostante;

- lo strato Provider contiene i moduli di interfaccia con le sorgenti dati supportate;
- l'ultimo strato contiene le librerie di terze parti utilizzate da QGIS.

## 3.2 Obiettivi

Lo scopo dello strumento che abbiamo sviluppato è quello di permettere all'utente di visualizzare e analizzare i dati delle osservazione provenienti da un servizio SOS 2.0 standard.

QGIS non supporta nativamente lo standard SOS. Lo strumento sviluppato deve quindi per prima cosa occuparsi della parte di comunicazione col servizio per recuperare i dati delle osservazioni. Deve quindi rendere tali dati visualizzabili in QGIS, cioé deve mostrare sul canvas, ma anche permettere all'utente di analizzarli utilizzando tutti gli strumenti che QGIS mette a disposizione. In sostanza deve essere il più possibile trasparente per l'utente.

A parte le operazioni specifiche o non supportate, una volta che i dati sono stati caricati come layer in QGIS, l'utente deve poter effettuare tutte le comuni operazioni che normalmente effettua su altri layer.

Partiamo ad analizzare più nel dettaglio quelli che sono gli aspetti principali di cui l'applicazione dovrà tener conto.

## 3.3 Visualizzazione delle osservazioni

L'interazione dell'utente con le osservazioni ambientali provenienti da un servizio SOS deve sicuramente partire dalla loro visualizzazione in QGIS. È quindi necessario stabilire cosa deve essere visualizzato e come.

Sappiamo dal precedente capitolo che l'informazione spaziale associata ad una osservazione non è unica. Possiamo infatti distinguere la posizione associata al sensore (o più in generale alla procedura) che ha effettuato l'osservazione da quella relativa al fenomeno osservato, la FoI.

Relativamente alla visualizzazione in mappa, è fondamentale mostrare la posizione del fenomeno target delle osservazioni, dal momento che l'obiettivo è quello di monitorarlo e/o studiarlo. La posizione del sensore che ha effettuato le rilevazioni invece, qualora diversa da quella della FoI, può risultare utile in particolare scenari per convalidare i dati dell'osservazione, ad esempio per scartare una misura in base alla distanza del sensore dal fenomeno osservato.

Oltre alle FoI, le osservazioni sono associate ad una Observed Property e una Procedure. Bisogna quindi fare in modo anche tali informazioni siano mostrate. Fondamentale è inoltre la visualizzazione dei risultati delle osservazioni, cioé i valori delle misurazioni delle Observed Properties connesse ad ogni osservazione.

Tuttavia nel caso delle osservazioni non basta mostrare nella mappa la FoI e i valori delle proprietà osservate per una determinata FoI, ma è necessario tenere conto di un'ulteriore dimensione: il tempo. La visualizzazione delle osservazioni quindi può essere effettuata secondo varie modalità, in base alle variabili che si prendono come riferimento.

Lo strumento dovrà permettere all'utente di visualizzare i dati secondo varie tipologie di viste mostrando l'andamento dei risultati delle osservazioni nel tempo e lasciando all'utente la possibilità di decidere le proprietà e le FoI di cui si vogliono visualizzare i dati.

## 3.4 Integrazione in QGIS

Lo sviluppo del client su QGIS può seguire strade tra loro alternative. È bene valutare quale possa essere quella più adatta allo scopo dello strumento che deve essere realizzato.

Una prima possibilità è la realizzazione di un **data provider C++** che si occupi di implementare il protocollo di comunicazione con il webservice SOS, che costruisca quindi le richieste da inviare al servizio ed interpreti le risposte da esso ricevute.

Una valida alternativa all'implementazione di un provider SOS è quella di sviluppare un **plugin Python** che effettui le richieste al webservice, recuperi i dati delle osservazioni e li memorizzi in locale, quindi sul client, in un formato già supportato da QGIS, ovvero per il quale esista già un provider.

### 3.4.1 Provider SOS...

La realizzazione di un nuovo provider è una scelta che solitamente viene effettuata nei casi in cui la funzionalità sviluppata sia general-purpose, poiché basterà integrarlo nel core di QGIS così che venga compilato e distribuito insieme ad esso. In caso contrario è difficile che l'aggiunta di un nuovo provider venga approvata dal QGIS Board (l'organismo di controllo del progetto), poiché ciò comporta un ampliamento della base di codice che deve essere manutenuta.

Nel nostro caso questo non rappresenta un problema: infatti è oramai chiaro che i dati di sensori nei prossimi anni saranno destinati a rappresentare la maggior parte dei dati geospaziali e che SOS sia la via da seguire nella realizzazione di sistemi interoperabili che rendano disponibili tali dati.

Per trovare delle obiezioni alla realizzazione di un provider SOS bisogna indagare il funzionamento e la struttura interna di QGIS.

Per prima cosa, i provider interrogano il datasource (cioé il webservice nel nostro caso) ad ogni pan/zoom della mappa. Sebbene si possa fare in modo di effettuare richieste solo per l'*exten* (estensione di mappa) mostrata in quel momento ed utilizzare altresì una cache per limitare l'overhead di richieste successive per i medesimi dati, l'operazione di recupero dati potrebbe impiegare comunque parecchio tempo, considerando soprattutto la quantità di osservazioni con cui potremmo avere a che fare.

In secondo luogo la struttura del provider risulta piuttosto rigida per adattarsi alle molteplici viste dei medesimi dati SOS. Ricordiamo infatti che, oltre che per FoI, le osservazioni possono essere raggruppate per Offering, per Observed Property, ma anche per Procedure. I provider di QGIS invece permettono soltanto di segmentare le Features in insiemi distinti tramite i cosiddetti *sublayers*.

### 3.4.2 ... o Plugin?

Lo sviluppo di un plugin permette al contrario di poter definire qualunque dettaglio implementativo, sia sulla parte di recupero dati che su quella di visualizzazione.

I contro di questo approccio sono quello di dover memorizzare in locale i dati delle osservazioni, ma anche il fatto che l'utilizzo del plugin con un nuovo set di dati richiede un tempo di setup iniziale per lo scaricamento delle osservazioni dal server.

Per il salvataggio in locale dei dati delle osservazioni, è fondamentale memorizzare i dati seguendo il modello O&M, così da essere sicuri che le relazioni tra le varie entità rimangano consistenti. Riguardo ai formati, se da un lato

l'uso di formati file-based evita il setup sul sistema di un DBMS, dall'altro quasi tutti quelli supportati da QGIS mal si prestano alla memorizzazione di uno schema complesso.

Uno di quelli che a prima vista sembrava essere valido allo scopo è **NetCDF** (Network Common Data Form), un formato self-describing per la memorizzazione di array multidimensionali che da anni viene utilizzato per lo scambio dati in ambito scientifico, oltre che come input e output di numerose applicazioni meterologiciche e oceanografiche. Sebbene QGIS lo supporti, il supporto è limitato esclusivamente all'utilizzo come sorgente dati raster.

## 3.5 Considerazioni e decisioni

Data l'attuale struttura dei provider di QGIS, realizzati nell'ottica di accesso a dati tabellari e non fortemente articolati, la scelta di realizzare un provider SOS è da escludere. I limiti presenti rendono ardua la gestione di informazioni provenienti da una base di dati complessa come quella SOS. Una modifica in questo senso necessiterebbe di rivedere la maggior parte delle classi Core di QGIS.

La realizzazione di un provider inoltre ci obbligherebbe a realizzare comunque un plugin per supportare le operazioni aggiuntive, quali la visualizzazione delle serie-temporali dei risultati.

La realizzazione di un **plugin python** fornisce invece tutta la flessibilità di cui abbiamo bisogno nel recuperare dati aventi strutture differenti, quali FoI e Observation.

Per quanto riguarda i tempi di scaricamento delle osservazioni, c'è da considerare che una volta che i dati sono stati scaricati dal server e memorizzati in locale, le successive operazioni sulla medesima base di dati non necessitano

di nuove richieste al server. In questo modo la velocità di risposta del plugin non dipende più dalla banda di rete disponibile né dal carico del server, evitando di sovraccaricare il server con continue richieste.

Il formato che abbiamo selezionato per la memorizzazione delle osservazioni sul client è **SQLite/SpatiaLite**, un file-based database che concilia la flessibilità di un R-DBMS nel memorizzare strutture complesse alla semplicità di installazione. **SpatiaLite** è l'estensione spaziale di SQLite che implementa operazioni e modelli di dati OGC compliant per realizzare il supporto dei dati spaziali per i database SQLite.

Per quanto riguarda gli aspetti di visualizzazione, le FoI possono essere mostrate in mappa come punti, linee o poligoni a seconda del valore dell'attributo *shape* dell'oggetto *Sampling feature*. I valori osservati possono invece essere mostrati nella tabella degli attributi del layer visualizzato. Inoltre è necessario realizzare alcuni grafici per la visualizzazione delle serie-temporali dei risultati. A tale scopo si è anche considerato l'uso di un plugin esterno, *Time Manager*, che permette di "animare" il canvas.

Nell'analisi sarebbe stato necessario attenzionare le eventuali problematiche di autenticazione per l'accesso al servizio. QGIS dispone di classi di accesso a webservices che astraggono da questo aspetto, per cui possiamo considerare in maniera marginale la sua discussione. Sarà l'invocazione delle API di QGIS utilizzate a fornire tale funzionalità.

## 3.6 Progettazione database

Il diagramma E-R in figura 3.2 raffigura le relazioni tra le entità Observation, Feature e ObservableProperty.

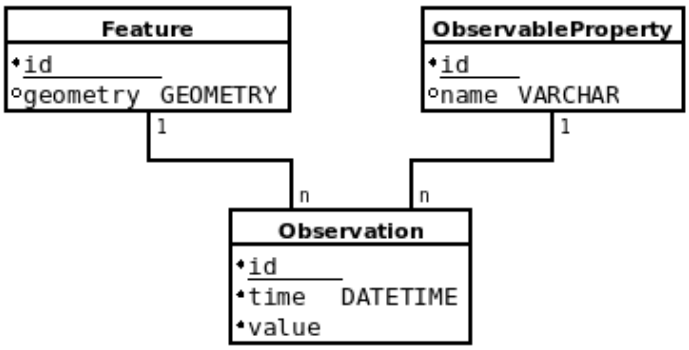


Figura 3.2: Diagramma E-R

Lo schema è stato costruito a partire dalle specifiche e definizioni dello standard SOS:

- una Observation ha una Feature e una ObservableProperty associata,
- più Observations possono appartenere alla stessa Feature,
- possono esistere più Observations con la medesima ObservableProperty associata.

Un tale schema, sebbene rispecchia il modello delle osservazioni, mal si presta ad essere utilizzato in QGIS.

QGIS può utilizzare come layer pressocché ogni tabella o vista presente nel database SQLite/SpatiaLite. Se la tabella/vista possiede una colonna geometrica, il layer viene caricato come layer vettoriale e le sue geometrie vengono visualizzate in canvas, altrimenti il risultato è una semplice tabella alfanumerica.

Se provassimo a caricare in QGIS la tabella delle osservazioni realizzata seguendo il precedente diagramma E-R, ci ritroveremmo con una tabella alfanumerica. Infatti le geometrie sono attributi dell'entita Feature, quindi non presenti nella tabella Observation.

Se creassimo una vista contenente sia i campi di Observation che la corrispondente colonna geometrica di Feature e la caricassimo in QGIS, le geometrie delle FoI sarebbero correttamente visualizzate in mappa.

```
CREATE VIEW "obs_data_view" AS
    SELECT ob.*, foi.geom
    FROM observation as ob
        JOIN featureofinterest as foi
            ON ob.foi = foi.id
```

Questo passaggio però non è sufficiente a permettere un ampio uso dei dati delle osservazioni in QGIS.

La vista infatti presenta una osservazione per riga, rendendo impraticabile realizzare formulazioni complesse basate sul valore di più proprietà nel medesimo istante.

Pensiamo ad esempio di voler visualizza in mappa la direzione e velocità del vento. Se avessimo sulla stessa riga della tabella i valori di entrambe le proprietà richieste, l'istante associato e la geometria della FoI, il problema sarebbe risolto.

Per fare ciò potremmo creare un'ulteriore vista `wind_data_view` che metta in relazione la velocità del vento con la sua direzione:

```
CREATE VIEW "wind_data_view" AS
    SELECT o1.time, o1.foi,
        o1.value as windspeed,
        o2.value as winddirection
    FROM observation as o1
        JOIN observation as o2
            ON o1.time = o2.time AND o1.foi = o2.foi
    WHERE o1.prop = 'Wind speed'
        AND o2.prop = 'Wind direction'
```

Caricando tale vista in QGIS e aprendo la tabella degli attributi del layer si vede facilmente che abbiamo ottenuto il risultato che volevamo.

Proviamo a generalizzare. L'ideale sarebbe avere sulla stessa riga della tabella, a fianco all'istante temporale e la geometria della FoI, i relativi valori di tutte le proprietà misurate in quell'istante per quella FoI.

Purtroppo la creazione di una vista composta da tante JOIN quante sono le proprietà osservabili è una via impraticabile. Per prima cosa, la vista viene acceduta ad ogni pan/zoom della mappa. Con una decina di proprietà osservabili e qualche decina di migliaia di osservazioni per proprietà, la vista inizia a richiedere parecchio tempo per essere visualizzata in QGIS, nell'ordine del secondo. C'è poi da considerare che il contenuto della vista `wind_data_view` è un sottoinsieme di quello che vorremmo fosse il risultato. Infatti la query SQL di creazione della vista richiederebbe l'uso di `FULL OUTER JOIN` in modo da mantenere tutti i valori misurati nel risultato, anche quelli che non hanno corrispettivi per le altre proprietà. Purtroppo però la FULL OUTER JOIN non è supportata attualmente da SQLite.

Una soluzione praticabile è utilizzare dei trigger per popolare una nuova tabella denormalizzata, `FoiTimeObservation`, che abbia la struttura richiesta: timestamp, foi, prop1, ..., propN. La coppia foi e timestamp sono chiave primaria di FoiTimeObservation, mentre prop1, ..., propN sono i campi che conterranno i valori delle osservazioni per le varie prorietà osservabili.

Nella tabella Observation creiamo un trigger per ogni proprietà osservabile, come segue:

```
CREATE TRIGGER "insert_prop_[PROPID]_observation"
INSERT ON observation WHEN NEW.prop IN
    (SELECT id
    FROM observableproperty
    WHERE field = '[PROPID]')
BEGIN
    INSERT OR IGNORE INTO foitimeobservation (foi, time)
        VALUES (NEW.foi, NEW.time);
    UPDATE foiobservation
        SET "[PROPID]" = NEW.value
```

```
        WHERE foi = NEW.foi AND time = NEW.time;
END
```

dove `[PROPID]` è l'identificativo di una proprietà osservabile presente in nella tabella ObservableProperty.

Ad ogni inserimento di una osservazione nella tabella Observation verrà scatenato uno dei trigger, quello che agisce sulla colonna associata alla proprietà dell'osservazione che si sta per inserire.

Per prima cosa il trigger cosa prova ad inserire una nuova riga nella tabelle FoiTimeObservation con i valori di FoI e timestamp della nuova osservazione, se non già presente. Quindi aggiorna il valore del campo associato alla proprietà dell'osservazione.

Tale approccio ha un costo: la duplicazione dei valori delle misurazioni, presenti sia sulla tabella originale Observation che sulla nuova tabella FoiTimeObservation.

Si può però prevedere che una tale tabella venga creata solo quando necessario, quando cioé l'utente ha bisogno di incrociare i dati di osservazioni diverse, per la visualizzazione o per il loro utilizzo con gli strumenti di analisi di QGIS.

## 3.7 Progettazione UI

Nella realizzazione della UI del client SOS è importante tenere presente quali siano le informazioni necessarie all'utente per poter utilizzare lo strumento.

### 3.7.1 UI importazione osservazioni

Nelle figure 3.3 e 3.4 è raffigurata l'interfaccia utente del client di importazione delle osservazioni da un servizio SOS. Una volta inserito l'URL del servizio SOS al quale collegarsi, le informazioni principali vengono mostrate nell'interfaccia utente.

Per la fruizione di un servizio web fondamentali sono i metadati, Titolo e Abstract in primo luogo. Tali informazioni permettono in poche righe di avere una panoramica sui dati che il servizio mette a disposizione. Trattandosi di un servizio SOS, è utile mostrare anche la lista delle Offerings disponibili e delle Procedures utilizzate, le FoI e le Observed Properties di cui il servizio espone i dati. Relativamente alle FoI, oltre che elencarli nell'interfaccia ha senso mostrarli anche sulla mappa di QGIS.

Tale operazione deve essere effettuata automaticamente dal client quando viene interrogato il servizio in modo da permettere all'utente la facile identificazione dell'area contenente le FoI e semplificare di conseguenza la definizione di un eventuale filtro spaziale.

È inoltre possibile definire un filtro per le osservazioni da importare, sia temporale, definendo cioé l'intervallo di date per il quale effettuare l'importazione, che spaziale, tracciando sulla mappa di QGIS l'area di cui si vogliono recuperare le osservazioni. Inoltre l'utente può scegliere se importare solo le osservazioni di specifiche Observable Properties selezionandole dalla lista.

## 3.8 UI visualizzazione dati

In base alle variabili che si prendono come riferimento, è possibile visualizzare i dati delle osservazioni secondo viste differenti. Tali visualizzazione possono essere effettuate direttamente in mappa o tramite grafici appositi.

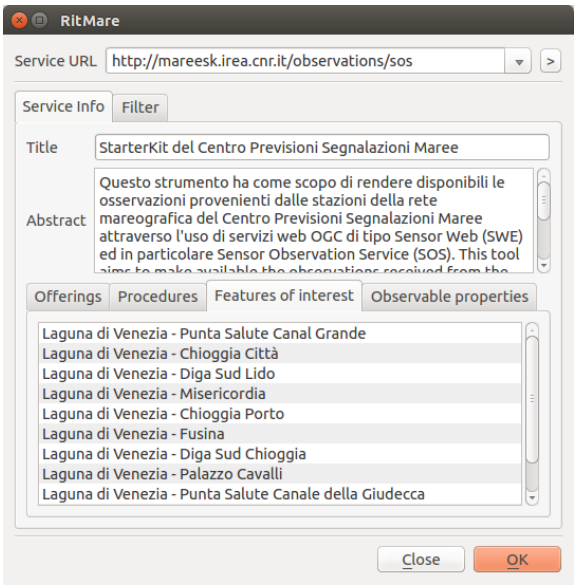


Figura 3.3: Visualizzazione informazioni sul servizio SOS di CNR-ISMAR

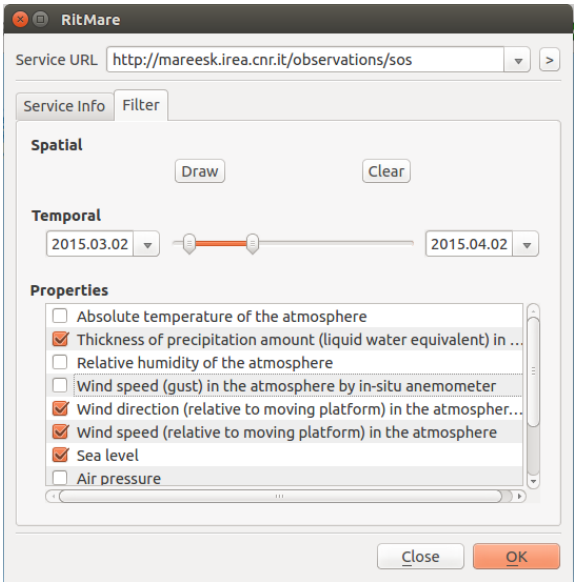


Figura 3.4: Definizione dei fitri sui dati da recuperare

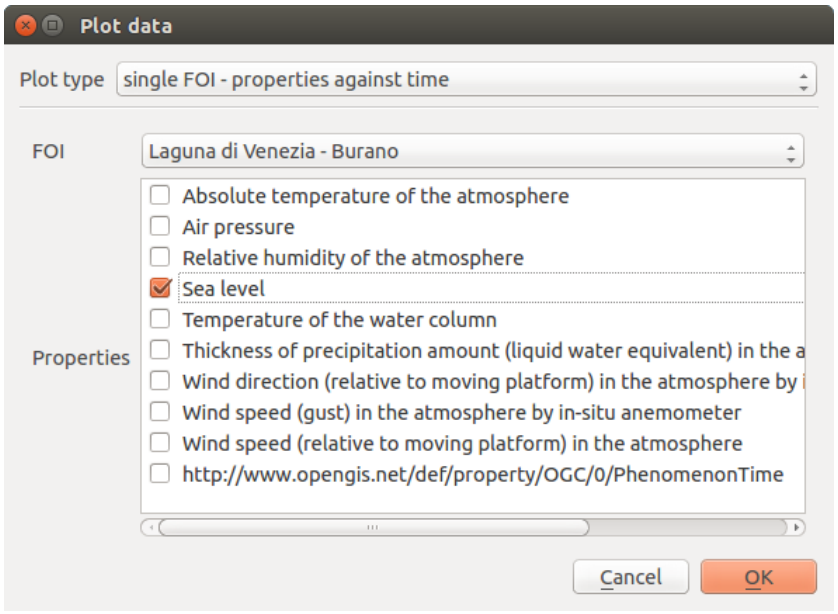


Figura 3.5: Selezione del tipo di grafico e dei suoi parametri

In figura 3.5 è visibile la finestra di selezione della tipologia di grafico e dei suoi parametri. In base al tipo di grafico selezionato nella casella in alto, nella parte bassa della finestra vengono popolate le liste contenenti i parametri utili alla creazione del grafico.

Per la visualizzazione delle serie temporali dei risultati è stato sviluppato un grafico che permette di fissare un valore tra FoI e Observable Property. Fissando ad esempio la FoI, è possibile visualizzare l'andamento nel tempo di più proprietà della medesima FoI, mentre selezionando una specifica proprietà è possibile confrontare l'andamento nel tempo dei valori di tale proprietà per FoI differenti. La figura 3.6 mostra il grafico relativo alle serie temporali.

Per visualizzare le FoI nello spazio dovremmo creare un grafico 3-dimensionale avente le coordinate della FoI sugli assi X e Y e il valore della proprietà selezionata sull'asse Z. Se poi volessimo vedere l'andamento di tali valori nel tempo una soluzione è quella di realizzare un grafico animato.

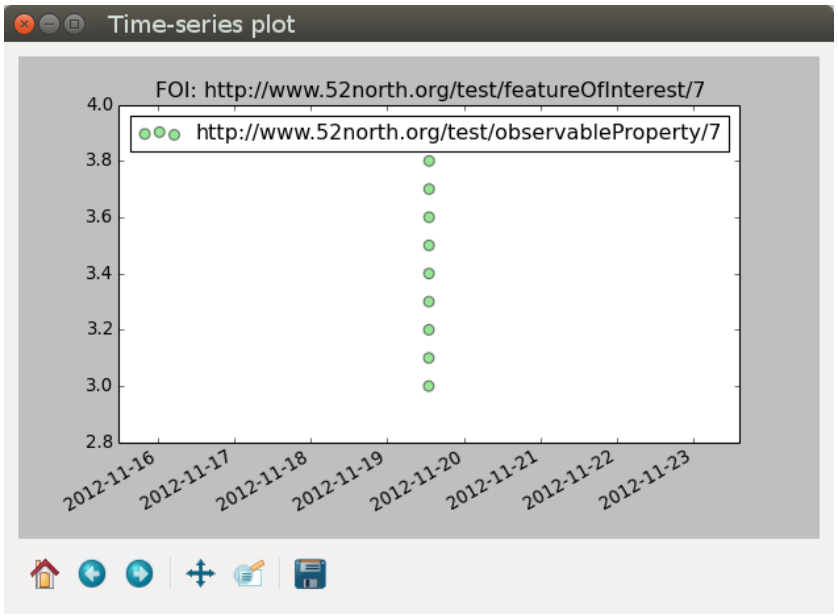


Figura 3.6: Grafico delle serie temporali

Le figure 3.7 e 3.8 mostrano i due grafici 3D "livello del mare" e "velocità e direzione del vento" realizzati e disponibili in QGIS2SOS. Una barra di animazione, presente nella parte bassa della finestra dei due grafici, permette all'utente di animare il grafico nel tempo similmente a quanto fatto da Time-Manager. I punti blu rappresentano le FoI cui i dati delle osservazioni sono associati, mentre i valori intermedi sono calcolati tramite interpolazione.

Alternativamente potremmo pensare di sfruttare direttamente la mappa di QGIS. In tal caso basterà definire uno stile per il layer che tenga conto dei valori delle proprietà che vogliamo visualizzare e utilizzare il plugin *Time Manager*.

**Time Manager** permette di eseguire un'animazione dei dati visualizzati in mappa in funzione del tempo. In figura 3.9 è possibile vedere uno screenshot del plugin all'opera. Una volta specificata l'ampiezza del frame temporale, il plugin divide i dati del layer in frame successivi in base ai valori

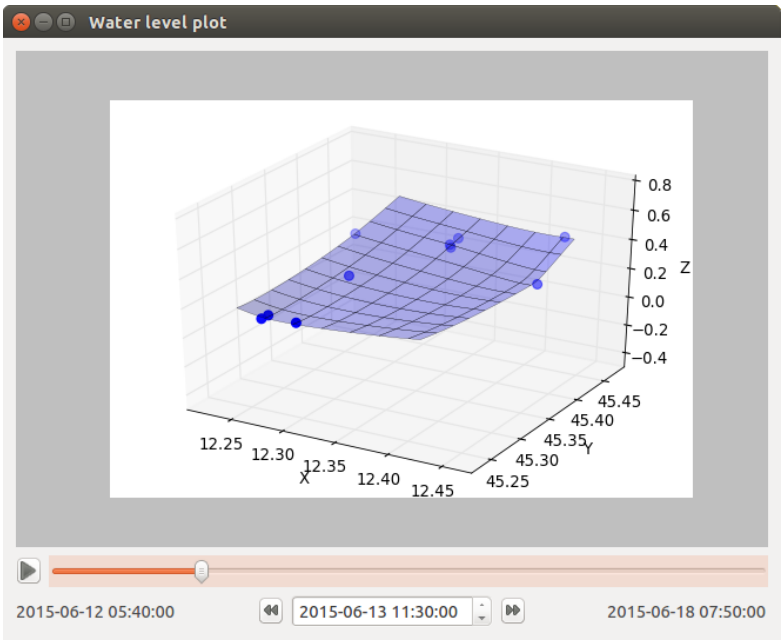


Figura 3.7: Grafico animato del livello dell'acqua

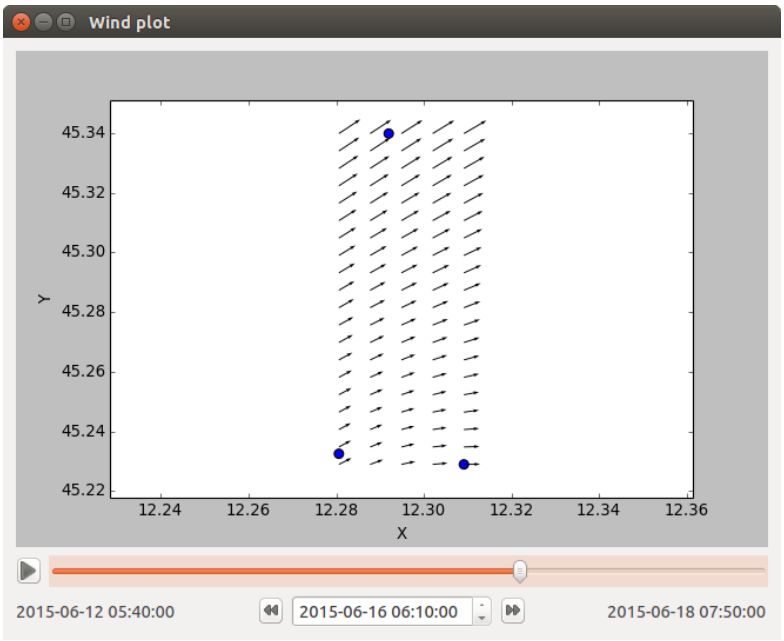


Figura 3.8: Grafico animato di velocità e direzione del vento

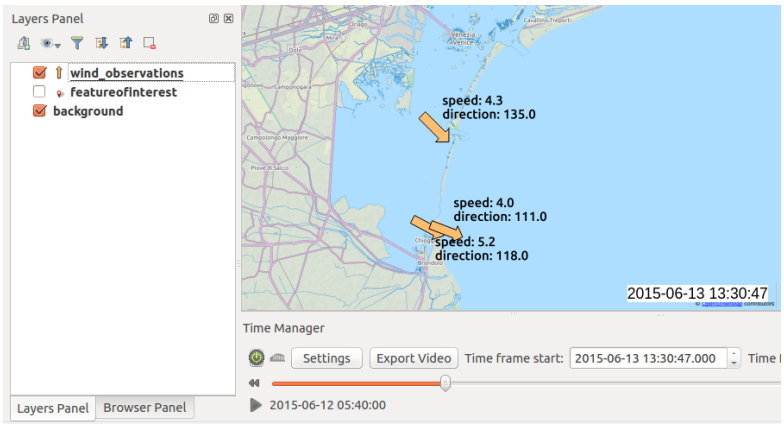


Figura 3.9: Animazione tramite il plugin Time Manager

del campo attributo specificato dall'utente. Solo i dati facenti parte del frame corrente verranno visualizzati. Tramite una barra temporale è quindi possibile spostarsi tra i vari frame, oppure basterà agire sugli appositi pulsanti per avviare l'animazione.

# Capitolo 4

# Case-study: la Laguna di Venezia

## 4.1 La Laguna

La Laguna di Venezia si trova nel Mare Adriatico settentrionale, lungo le coste del Veneto.

La sua superficie è di circa 550 $km^2$. Di questa circa l'8% è occupata da terra, Venezia e le isole minori, sebbene solo il 5% risulta permanentemente emersa[24]. Il 12% della superficie è costantemente coperto d'acqua, mentre il rimanente 80% è fangoso, rappresentato dalle piane di marea, dalle barene, zone di terreno a pelo d'acqua ricoperte da vegetazione particolarmente resistente al sale, e dalle casse di colmata, isole artificiali realizzate negli anni '60 e inizialmente progettate per ospitare la nuova zona industriale di Marghera i cui lavori furono bloccati nel 1969 e definitivamente nel 1973 grazie alla Legge Speciale per Venezia.

La laguna è separata dal mare aperto dai lidi, lunghi cordoni sabbiosi talvolta arginati dall'uomo con opere di varia natura. La comunicazione con l'esterno avviene attraverso le bocche di porto del Lido di Malamocco e di Chioggia. L'acqua entra dal mare ogni sei ore e se ne esce dopo altre sei.

## 4.2 Sistemi di monitoraggio

La laguna che conosciamo ha circa 6000 anni. Prima al suo posto c'era una pianura costituita da sedimenti trasportati dai fiumi. La sua nascita avviene in seguito alla fine dell'ultima glaciazione, dovuta a fenomeni come l'innalzamento del livello del mare e il progressivo consolidamento dei depositi[17].

Il destino della laguna, in mancanza dell'intervento dell'uomo, sarebbe stato il suo graduale interramento, causato dall'apporto dei sedimenti trasportati dagli stessi fiumi responsabili indiretti della sua creazione. Per evitare che ciò accadesse negli anni sono state predisposte varie reti di sensori con l'obiettivo di monitorare lo stato della laguna in modo da poter intervenire qualora necessario.

Due in particolare sono le reti di monitoraggio su cui ci focalizzeremo.

La *rete per il monitoraggio del livello del mare e dei parametri meteorologici* dell'intera laguna e del litorale è costituita da 15 stazioni di misura collegate via radio, attraverso una stazione che funge da ripetitore, ad una stazione centrale che raccoglie tutti i dati. Il livello della marea viene misurato tenendo conto di un livello idrometrico di riferimento che, nel caso di Venezia, è lo zero locale di Punta della Salute, convenzionalmente determinato come il *Livello del Medio Mare* (l.m.m.) del 1897 (prima Rete Altimetrica dello Stato)[11].

La *rete delle stazioni per il monitoraggio del moto ondoso*, realizzata nel 2003 nel centro storico e nella laguna, è composta da 9 stazioni di monitoraggio, ognuna delle quali rileva la distanza del sensore dal pelo libero dell'acqua. I dati registrati permettono di valutare i movimenti della superficie dell'acqua e disegnare il profilo delle onde.

Nel loro percorso i dati subiscono varie trasformazioni. Man mano che passano da un sistema all'altro, i dati vengono memorizzati, elaborati e inviati al successivo nodo della catena.

Le stazioni di monitoraggio dislocate in laguna effettuano rilevazioni con frequenze che variano, in base alla tipologia dei sensori e ai parametri da misurare, fino a 4 misure al secondo. I dati grezzi raccolti dalle stazioni vengono inviati dei speciali nodi della rete responsabili della raccolta dati, i quali provvederanno successivamente a trasmetterli ad un server per la loro memorizzazione e il successivo utilizzo.

Il *Centro Previsioni Segnalazioni Maree* del Comune di Venezia è il centro che si occupa di recuperare i dati grezzi e memorizzarli su un database. I dati grezzi vengono quindi filtrati per rimuovere eventuali valori anomali dovuti ad errori di ricezione o cause esterne. Successivamente sono sottocampionati con un periodo di 5 minuti e inviati a CNR-ISMAR che li archivia su un database PostgreSQL/PostGIS e li rende accessibili per la fruizione attraverso l'infrastruttura GET-IT.

GET-IT (Geoinformation Enabling Toolkit), già RITMARE StarterKit, è l'infrastruttura dati che CNR ha realizzato e continua a sviluppare nell'ambito di del progetto RITMARE-SP7 (Sub Project 7) con lo scopo di permettere l'interoperabilità, la partecipazione e condivisione di informazioni tra le comunità tematiche del progetto RITMARE.

## 4.3 SOS2QGIS per la Laguna

Vedremo adesso come lo strumento SOS2QGIS possa essere impiegato in QGIS per il recupero dei dati dal server SOS di CNR-ISMAR, la loro visualizzazione e analisi.

Una volta attivato il plugin tramite l'apposito pulsate sulla barra degli strumenti di QGIS, viene visualizzata la finestra di importazione delle osservazioni che permette all'utente di inserire l'URL del servizio SOS e recuperarne i metadati. Le figure 3.3 e 3.4 riportate nel capitolo precendente mostrano proprio i metadati del servizio SOS di CNR-ISMAR e la relativa finestra per il filtraggio dei risultati.

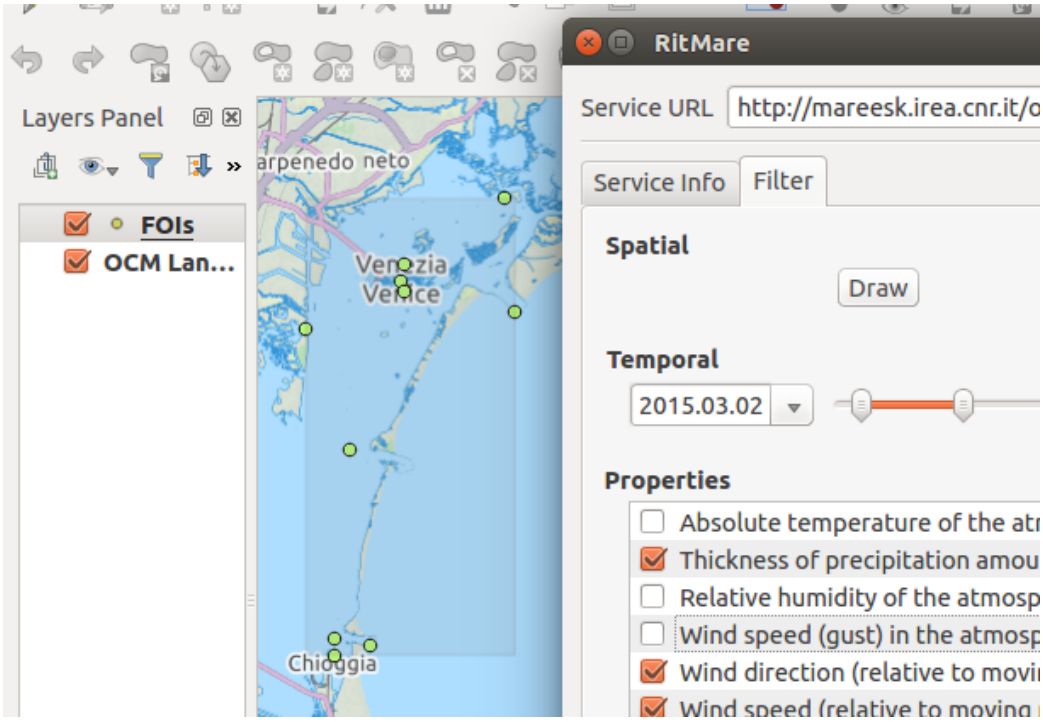


Figura 4.1: Area contenente le FoI della Laguna e filtro spaziale

Contestualmente al recupero dei metadati, il plugin carica in mappa il layer con le FoI del servizio in modo da permettere all'utente di identificare la zona di interesse, come mostrato in figura 4.1. Il rettangolo presente attorno alle FoI rappresenta il filtro spaziale che è stato impostato dall'utente per limitare il recupero delle osservazioni. L'altro layer visibile nella legenda è invece la mappa *OCM Landscape* di **OpenStreetMap** (vedi `https://www.`

`openstreetmap.org/`), utilizzata nell'esempio come sfondo.

Dopo che l'utente ha definito i filtri da utilizzare ed ha selezionato dove memorizzare il DB SQLite/SpatiaLite che conterrà le osservazioni, il plugin inizia a scaricare le osservazioni mostrando una finestra di attesa all'utente. Completato il processo, il layer delle FoI viene mostrato in mappa. La figura 4.2 mostra le FoI relative ai dati delle osservazioni scaricate.

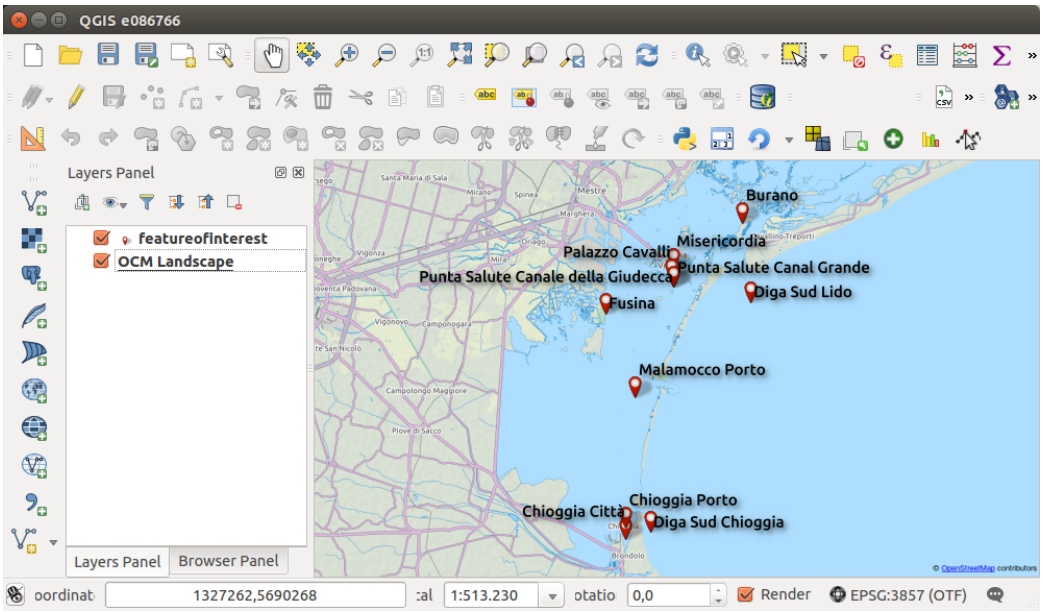


Figura 4.2: Layer delle FoI recuperate dal servizio SOS di CNR-ISMAR

Carichiamo a questo punto il layer con le osservazioni agendo sull'apposito pulsante. La figura 4.3 mostra tre layer delle osservazioni identici, a cui è stato assegnato però uno stile differente in base ai valori dei campi contenuti, renderizzati sopra il layer delle FoI visto precedentemente.

Al layer `wind_observation` è stato assegnato uno stile a forma di freccia, la cui dimensione e inclinazione dipende però dai valori dalle proprietà *Wind Speed* e *Wind Direction* delle osservazioni in esso contenute. Dando un'occhiata al canvas è possibile vedere il risultato del rendering.

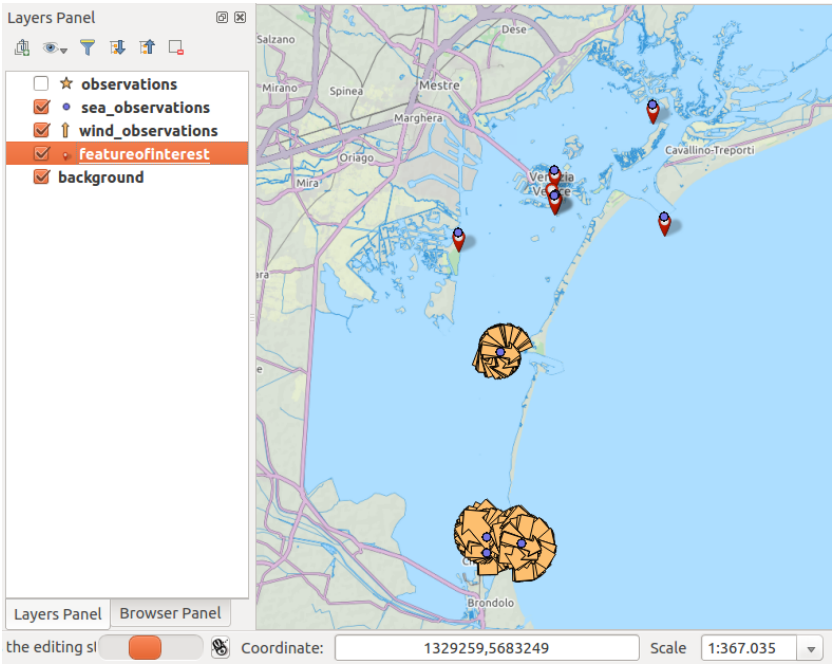


Figura 4.3: Layer delle Observations recuperate dal servizio SOS di CNR-ISMAR

Se però attivamo il plugin Time Manager, aggiungendo anche delle label che mostrano i valori delle proprietà, il risultato ottenuto è quello visibile in figura 4.4.

Proviamo adesso a visualizzare le serie temporali delle osservazioni della proprietà *Sea Level*. Agendo sul pulsante apposito, viene visualizzata la finestra di selezione del grafico riportata in figura 3.5. Selezioniamo il tipo di grafico desiderato, la proprietà Sea Level e le FoI per le quali vogliamo visualizzare le serie temporali. La figura 4.5 mostra il risultato ottenuto.

Dall'andamento dei valori si possono avere informazioni visive che possono dare spunti per indagare più approfonditamente un fenomeno in una particolare area. L'analisi degli eventuali *spike* inoltre può permettere di identificare un problema ad un sensore di una delle stazioni metereologiche.

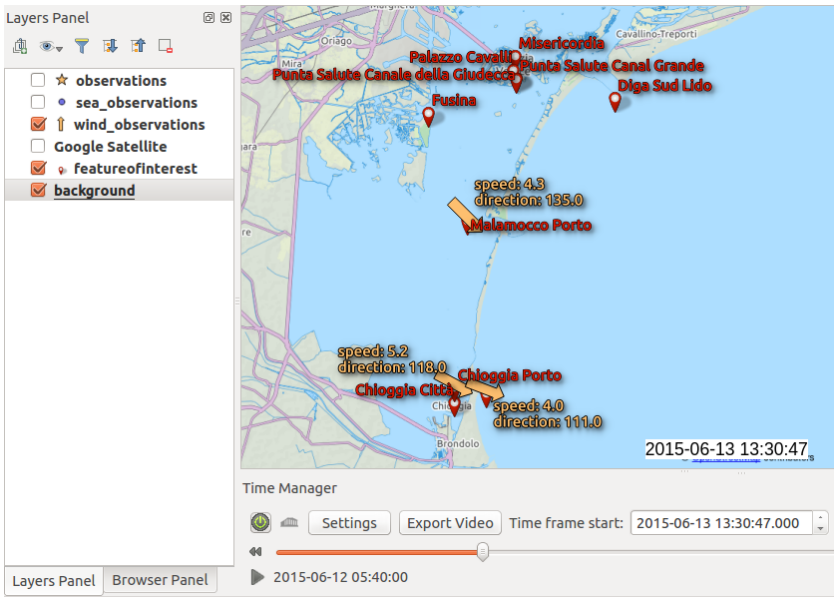


Figura 4.4: Visualizzazione osservazioni del vento in Laguna con il plugin Time Manager

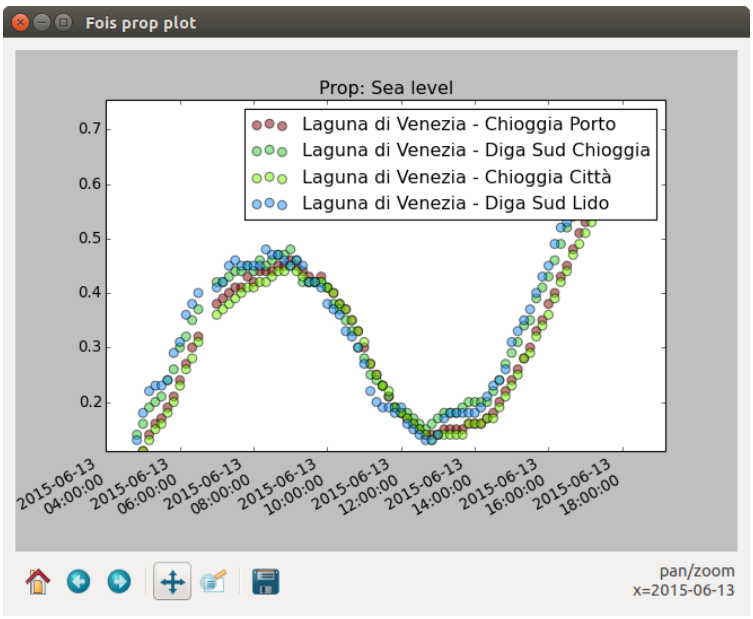


Figura 4.5: Valori della proprietà "Sea level" di un set di FoI selezionate dall'utente

# Capitolo 5

# Conclusioni

Nei capitoli precedenti abbiamo discusso dell'importanza che rivestono i dati provenienti dalle reti di sensori globali per il monitoraggio e l'analisi di fenomeni ambientali.

Abbiamo mostrato successivamente come lo standard SOS nella sua versione 2.0, insieme all'infrastruttura SWE, abbia reso realizzabile il Sensor Web, rendendo tutte le reti di sensori esistenti delle fonti inesauribili di informazioni.

Abbiamo quindi sviluppato uno strumento software che permette a QGIS, uno dei più utilizzati software GIS desktop al mondo, di interagire con un servizio SOS per recuperare i dati delle osservazioni.

Abbiamo infine dimostrato come SOS2QGIS possa essere applicato al monitoraggio di un caso di studio reale, la Laguna di Venezia, per visualizzare l'andamento dei parametri della laguna tramite QGIS e permettere l'analisi di tali dati in maniera semplice e integrata.

Sebbene SOS2QGIS possa già essere utilizzato in ambiti diversi da quello della Laguna di Venezia, potendo comunicare con qualunque servizio SOS 2.0 standard-compliant, la versione attuale dello strumento realizza un numero

ristretto di funzionalità rispetto a quelle che SOS mette a disposizione.

Manca il supporto alle Offerings e alle Procedures, con tutti i vantaggi derivanti dalla loro implementazione. Manca la possibilità di gestire dati di tipo Coverage, che potrebbero essere visualizzati in QGIS tramite layer raster.

Il completo supporto in QGIS dello standard SOS è ancora lontano, ma lo strumento SOS2QGIS rappresenta senza dubbio un enorme passo avanti sulla via del monitoraggio ambientale tramite GIS.

A conferma di ciò, il lavoro realizzato ha già suscitato l'interesse della comunità e non solo. Sebbene non sia ancora stato pubblicato, abbiamo ricevuto un contatto da una azienda europea si occupa dello sviluppo di componenti INSPIRE/SOS/IoT/SensorWeb, la quale si è detta disposta a sponsorizzare i lavori su SOS2QGIS.

Di questo ne siamo fieri, soprattutto perché ci rassicura ed assicura che lo sforzo fatto non sia stato inutile.

## 5.1 Prossimi passi

La prima cosa da fare è certamente rendere SOS2QGIS accessibile al mondo, pubblicandolo sul repository dei plugin per QGIS. In questo modo il lavoro fatto non andrà perduto, inoltre chiunque lo utilizzerà potrà riportare impressioni e eventuali problemi trovati durante il suo utilizzo. La collaborazione è fondamentale per realizzare l'interoperabilà dei sistemi e sotto questo aspetto il software open-source rappresenta certamente la chiave da utilizzare.

Il secondo passo è quello di realizzare le funzionalità mancanti, partendo prima di tutto dal supporto alle Procedures, recuperando dal server la loro

posizione e descrizione e permettendo di utilizzare questi dati dentro QGIS. A seguire bisognerà implementare il supporto alle Offerings e la gestione e visualizzazione dei dati delle Coverages.

# Appendici

# Appendice A

# Sviluppo plugin Python per QGIS

Un plugin python per QGIS è composto da alcuni file fondamentali[15], sempre presenti nella directory del plugin:

**metadata.txt** è il file contenente le informazioni generali sul plugin.

**__init__.py** è il punto di accesso al plugin, permette l'importazione del plugin come package python.

**plugin.py** è il file contenente la definizione della classe del plugin e dei suoi metodi.

Il file `metadata.txt` è un file in formato INI che contiene una sezione `general` al cui interno sono definiti:

- il nome e la descrizione del plugin
- versione del plugin, minima e massima versione di QGIS richiesta per eseguire il plugin
- il nome e email dell'autore
- la categoria, utile per inquadrare il plugin in un determinato menu

- keywords o tags utili per la ricerca
- il repository contenente il codice del plugin
- il bug tracker per la segnalazione di eventuali errori.

Il file `__init__.py` è un file a cui Python assegna un significato particolare, utilizzato per la definizione di packages, cioé directory contenenti moduli python. Se di norma tale file può semplicemente essere vuoto, QGIS necessita che tale file contenga una funzione `classFactory(iface)`, methodo factory utilizzato per la creazione dell'istanza del plugin.

L'argomento del metodo factory, `iface`, è un oggetto di classe `QgisInterface`[14], classe che espone una serie di metodi statici che permettono allo sviluppatore di interagire con la finestra principale di QGIS, con la mappa, la legenda, le toolbar e i menu.

L'accesso in python alle funzionalità di QGIS avviene appunto tramite l'oggetto `iface` che deve essere passato al plugin nel momento in cui viene quest'ultimo viene istanziato.

La classe del plugin è contenuta nel file python `plugin.py`, sebbene si possa utilizzare un qualunque nome per tale file.

Per poter essere caricata in QGIS, la classe del plugin deve definire almeno i seguenti 3 metodi:

**__init__(iface)** cioè il costruttore della classe che prende come argomento l'istanza di `QgisInterface` passata dalla funzione `classFactory` e la memorizza per il successivo utilizzo.

**initGui()** metodo che si occupa della creazione e aggiunta dei pulsanti nei menu e nelle toolbar di QGIS, così da permettere all'utente di richiamare le funzionalità del plugin, ovvero la connessione ai segnali inviati

da QGIS nel caso in cui il plugin debba attivarsi allo scatenarsi di particolari eventi.

**deinitGui()** che deinizializza le risorse allocate dal plugin, rimuovendo gli eventuali pulsanti dalle toolbar e dai menu.

Eventuali altri metodi e file sono a discrezione dello sviluppatore.

# Appendice B

# Framework QT

Qt è un framework multipiattaforma scritto in C++ per lo sviluppo di applicazione desktop, embedded e mobile. Molteplici sono le piattaforme supportare, tra queste le più importanti sono Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS.

Lo sviluppo di Qt inizia nel 1990 per conto di Trolltech. L'azienda, dopo essere stata oggetto di molteplici e successive acquisizioni, oggi è chiamata The Qt Company e insieme ad un allenza di aziende e privati, The Qt Project, contribuisce allo sviluppo di Qt.

Qt è disponibile sotto licenza commerciale, venduta da The Qt Company, ma anche rilasciata sotto licenze GPL e LGPL, adatte quindi per lo sviluppo di software open-source.

Qt utilizza **MOC** (Meta-Object Compiler), un preprocessore che estende il linguaggio C++ con costrutti Qt quali Signal e Slot. Prima della compilazione il MOC fa il parsing dei file di intestazione scritti in C++ Qt-extended e genera i corrispettivi file sorgenti in C++ standard compliant. Per tale ragione applicazioni e librerie che usano il framework QT possono essere compilate da qualunque compilatore C++ standard compliant.

### Signals e Slots

Il meccanismo **Signal-Slot** è una funzionalità di Qt che permette la comunicazione fra oggetti alternativa all'uso delle callback spesso utilizzate da altri strumenti[21].

Le callback hanno due principali difetti. In primo luogo una callback non è type-safe essendo sostanzialmente un puntatore a funzione con argomenti generici, quindi non si può essere certi che verrà invocata con i corretti argomenti. Inoltre le funzioni che vogliono utilizzarle devono sapere il nome della callback da invocare, sono quindi strettamente accoppiate con la callback stessa anche qualora risiendano in oggetti diversi.

In Qt il funzionamento è invece *event-driven*. Un oggetto Qt può notificare che un evento è accaduto emettendo un Signal, mentre gli oggetti che vogliono essere notificati possono usare uno Slot, cioé una funzione che deve essere chiamata in risposta ad un particolare segnale.

Il meccanismo è type-safe. Un Signal è identificato dalla sua signature, ovvero il nome e l'elenco degli argomenti passati attraverso il segnale, e la connessione tra Signal emesso e Slot associato fallisce se il tipo dei parametri non corrisponde. È inoltre possibile connettere molteplici Slot al medesimo Signal, ma anche più Signal al medesimo Slot, consentendo quindi una grande flessibilità.

Il meccanismo Signal-Slot è disponibile per tutti gli oggetti che estendono la classe *QObject*, classe base degli oggetti Qt, o le sue sottoclassi.

### User Interface

Qt mette a disposizione varie tecnologie per la realizzazione di interfacce utente.

**Qt Widgets** sono le tradizionali interfacce utente degli ambienti desktop. I widget si integrano perfettamente con le piattaforme desktop, fornendo un look'n'feel nativo su Windows, Linux e OSX, ma mal si prestano alla realizzazione di UI dinamiche e animate di comune uso su dispositivi mobili.

**Qt Quick** è un modulo che permette la creazione di interfacce utente reattive e fluide, fornendo un nuovo motore di rendering e definendo concetti quali transizioni e animazioni. QML (Qt Meta-Object Language) è il linguaggio alla base di Qt Quick che permette di costruire interfacce utente in modo dichiarativo.

**Qt WebKit** è di un web-engine utilizzato principalmente per integrare contenuti web dentro applicazioni Qt.

Tali tecnologie non sono alternative, infatti l'uso di una non esclude la possibilità di realizzare parte delle UI con una tecnologia differente, sebbene ognuna si presti meglio ad un determinato tipo di interfacce grafiche[22]. È lo sviluppatore che decide autonomamente in base alle sue esigenze o al tipo di applicazione che va a realizzare.

### Binding Qt per Python

Di solito le applicazioni che utilizzano Qt sono scritte in C++, tuttavia esistono binding per vari linguaggi di programmazione scritti da terze parti.

Esistono due differenti progetti che realizzano binding Qt per Python. Anche se il più utilizzato rimane **PyQt** di Riverbank Computing, dal 2009 Nokia, che aveva acquisito Trolltech, ha iniziato lo sviluppo di **PySide** che tuttora viene portato avanti da The Qt Company a causa di questioni legate al licensing[20].

# Bibliografia


[1] Arne Bröring et al. «New Generation Sensor Web Enablement». In: *Sensors* 11.3 (2011), p. 2652. ISSN: 1424-8220. DOI: 10.3390/s110302652.

[2] Xingchen Chu e Rajkumar Buyya. «Service Oriented Sensor Web». English. In: *Sensor Networks and Configuration*. A cura di Nitaigour P. Mahalik. Springer Berlin Heidelberg, 2007, pp. 51–74. ISBN: 978-3-540-37364-3. DOI: 10.1007/3-540-37366-7_3.

[3] Open Geospatial Consortium. «Sensor Web Enablement Architecture (Best Practice)». In: OGC SWE initiative (2008). A cura di Ingo Simonis. URL: http://portal.opengeospatial.org/files/?artifact_id=29405.

[4] Simon Cox. «Observations and Sampling». In: (2015). [Online; accessed 10-November-2015]. URL: https://www.seegrid.csiro.au/wiki/AppSchemas/ObservationsAndSampling.

[5] Simon Cox. *OGC Abstract Specification Geographic information - Observations and Measurements*.

[6] Jackson Shannon P. Delin Kevin A. e R.R. Some. «Sensor Webs». In: *NASA Tech Briefs* (23 1999), p. 80. URL: http://www.nasatech.com/Briefs/Oct99/NPO20616.html.

[7] Kevin A. Delin e Shannon P. Jackson. «Sensor web: A new instrument concept». In: *Symposium on Integrated Optics*. International Society for Optics e Photonics. 2001, pp. 1–9.

[8] Liping Di. «Geospatial sensor web and self-adaptive Earth predictive systems (SEPS)». In: *Proceedings of the Earth Science Technology Office (ESTO) Meeting, San Diego, USA*. 2007, pp. 1–4. URL: `http://www.esto.nasa.gov/sensorwebmeeting/papers/di.pdf`.

[9] *IEEE 1451.2-1997. IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Transducer to Microprocessor Communication Protocols & TEDS Formats*. standard IEEE. 1997.

[10] *JSON vs XML: some hard numbers about verbosity*. [Online; accessed 10-November-2015]. 2013. URL: `http://pragmateek.com/json-vs-xml-some-hard-numbers-about-verbosity/`.

[11] *Le stazioni di monitoraggio in laguna*. [Online; accessed 10-November-2015]. URL: `http://www.atlantedellalaguna.it/?q=node/121`.

[12] Dan Mandl. «Experimenting with sensor webs using Earth observing 1». In: *Proceedings of 2004 IEEE Aerospace Conference*. Vol. 1. IEEE. 2004.

[13] OpenHub. *The QGIS Open Source Project*. [Online; accessed 10-November-2015]. URL: `https://www.openhub.net/p/qgis`.

[14] QGIS Development Team. *QGIS 2.8 Geographic Information System API Documentation*. Open Source Geospatial Foundation. 2014. URL: `http://qgis.org/api/2.8/`.

[15] QGIS Development Team. *QGIS 2.8 Geographic Information System PyQGIS Developer Cookbook*. Open Source Geospatial Foundation. 2014.

URL: http://docs.qgis.org/2.8/en/docs/pyqgis_developer_cookbook/.

[16] QGIS Development Team. *QGIS 2.8 Geographic Information System User Guide*. Open Source Geospatial Foundation. 2014. URL: http://docs.qgis.org/2.8/en/docs/user_manual/.

[17] Cesare Rizzetto. «Ma che cos'è questo "caranto?"» In: (2004). [Online; accessed 10-November-2015]. URL: http://www.archeosub.it/articoli/geotec/caranto.htm.

[18] *Sensor Data Model for Imagery and Gridded Data*. ISO Standard. 2004.

[19] Gary Sherman. *QGIS Users Around the World*. [Online; accessed 10-November-2015]. 19 Dic. 2011. URL: http://spatialgalaxy.net/2011/12/19/qgis-users-around-the-world/.

[20] The Qt Company. *About PySide - Qt Wiki*. [Online; accessed 10-November-2015]. 2014. URL: http://wiki.qt.io/About_PySide.

[21] The Qt Company. *Signals and Slots - Qt 5.5 Documentation*. [Online; accessed 10-November-2015]. 2015. URL: http://doc.qt.io/qt-5/signalsandslots.html.

[22] The Qt Company. *User Interfaces - Qt 5.5 Documentation*. [Online; accessed 10-November-2015]. 2015. URL: http://doc.qt.io/qt-5/topics-ui.html.

[23] *UNI 4546:1984. Termini e definizioni fondamentali - Misure e misurazioni*. norma UNI. 1984.

[24] CORILA Consorzio per il cordinamento delle ricerche inerenti al sistema lagunare di Venezia. *La laguna di Venezia*. [Online; accessed 10-November-2015]. 2014. URL: http://www.corila.it/?q=node/104&language=it.