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

Ministero della Transizione Ecologica

Ricerca di Sistema elettrico

Metodologie di calcolo del Fabbisogno energetico degli edifici: metodi dinamici. Sviluppo e applicazione di un codice di calcolo III

Mazzarella L., Alongi A., Angelotti A., Pasini M.

POLITECNICO
MILANO 1863

Report RdS/PTR2021/107

METODOLOGIE DI CALCOLO DEL FABBISOGNO ENERGETICO DEGLI EDIFICI: METODI DINAMICI. SVILUPPO E APPLICAZIONE DI UN CODICE DI CALCOLO III

Mazzarella L., Alongi A., Angelotti A., Pasini M. (Politecnico di Milano – Dipartimento di Energia)

Dicembre 2021

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dello Sviluppo Economico (oggi Ministero della Transizione Ecologica) - ENEA
Piano Triennale di Realizzazione 2019-2021 - III annualità
Obiettivo: *N. 1 - Tecnologie*
Progetto: *1.5 - Tecnologie, tecniche e materiali per l'efficienza energetica ed il risparmio di energia negli usi finali elettrici degli edifici nuovi ed esistenti*
Work package: *1 - Edifici ad alta efficienza energetica*
Linea di attività: *LA1.8 - Metodologie di calcolo del Fabbisogno energetico degli edifici: metodi dinamici. Sviluppo e applicazione di un codice di calcolo III*
Responsabile del Progetto: Giovanni Puglisi, ENEA
Responsabile del Work package: Domenico Iatauro, ENEA

Il presente documento descrive le attività di ricerca svolte all'interno dell'Accordo di collaborazione *"Metodologie di calcolo del Fabbisogno energetico degli edifici: metodi dinamici"*
Responsabile scientifico ENEA: Ing. Domenico Iatauro
Responsabile scientifico Politecnico di Milano – Dipartimento di Energia: Prof. Livio Mazzarella

Indice

Sommario

L'attività di ricerca esposta in questo rapporto rappresenta il lavoro svolto nel terzo anno del piano triennale di sviluppo di un nuovo codice di calcolo per la simulazione delle prestazioni termiche ed energetiche degli edifici, chiamato OpenBPS. Tale codice di calcolo è stato pensato per rispondere ai requisiti della nuova implementazione della direttiva EPBD Recast e successive modifiche. Per raggiungere i nuovi obiettivi di utilizzazione ottimale delle fonti di energia rinnovabili e dell'edificio a energia quasi zero (NZEB) diventa, infatti, ineluttabile l'impiego di modelli di calcolo che lavorino su base oraria e che siano estremamente flessibili per descrivere sistemi sempre più complessi e tra loro interagenti; tali programmi o codici di calcolo vengono identificati con l'acronimo BPSt (Building Performance Simulation tool).
La filosofia seguita nello sviluppo del suddetto codice, già ampiamente spiegata nei suoi aspetti tecnici nei precedenti rapporti relativi alla I e II annualità del progetto, è stata quella di produrre non un codice eseguibile dotato di interfaccia utente, ma delle librerie di calcolo che possano essere implementate all'interno di software tecnici di terze parti. Questo per evitare che, terminato il corrente progetto di ricerca, i suoi risultati non trovino un'applicazione diffusa e soprattutto una modalità di manutenzione e sviluppo continuativo che ne assicuri l'aggiornamento e la fruibilità nel tempo. L'attività del terzo anno, salvo alcuni ampliamenti del codice in seguito dettagliati, si è quindi concentrata, come da programma, sulla individuazione e definizione, per quanto possibile delle modalità di gestione futura delle librerie OpenBPS in un'ottica di Open Source, tale da consentire a quanti, esperti del settore, che vogliano e possano contribuire, di prestare la loro competenza in modo coordinato ed efficace. L'idea centrale dell'approccio Open Source è infatti quella basare la manutenzione e lo sviluppo di un codice tramite la creazione di una comunità di sviluppatori (Community), che si prenda carico, attraverso degli strumenti individuati e appositamente sviluppati, della gestione del codice di calcolo già prodotto arricchendolo di funzionalità e manutenendolo.
Nell'ambito presente ricerca si è quindi cercato di sviluppare una procedura codificata chiara e trasparente, nel seguito indicata come protocollo, che consenta di assicurare lo sviluppo continuativo ella libreria di calcolo per la simulazione termoenergetica degli edifici OpenBPS, nonché la sua manutenzione e l'aggiornamento, tramite il coinvolgimento di quanti, esperti del settore, vogliano e possano contribuire, prestando la loro competenza in modo coordinato ed efficace in un ambito "open source" o similare.
Il protocollo cerca quindi di definire le regole di sviluppo, documentazione e validazione del codice che viene reso disponibile dagli sviluppatori, e le modalità di verifica e validazione finale dei codici proposti prima della loro accettazione definitiva come componenti ufficiali del codice OpenBPS. Per questa seconda fase occorre individuare il criterio di selezione dei valutatori indipendenti che, sulla base della documentazione fornita dagli sviluppatori, operano una validazione terza del codice proposto. Le informazioni concernenti le validazioni, sia quelle effettuate dagli sviluppatori, che quelle effettuate dai valutatori terzi devono essere ben documentate e chiaramente associate a ciascun componente o modulo che sarà poi reso disponibile nel repository ufficiale del codice OpenBPS. Da qui la necessità di standardizzare questa procedura il più possibile e che questa standardizzazione sia prodotta da un soggetto specificatamente competente, sia per l'oggetto (calcolo dinamico delle prestazioni termoenergetiche degli edifici) sia per il metodo (Ente possibilmente normatore).
Per garantire la diffusione di una versione stabile del codice sorgente e/o eseguibile, specificatamente in termini di librerie dinamiche (dll), ma soprattutto per garantire che l'utente finale dei risultati dei calcoli eseguiti con il codice OpenBPS possa essere certo della qualità dello strumento utilizzato per produrli, occorre che la suddetta procedura identifichi una modalità di gestione del rilascio del codice e dei suoi aggiornamenti ed estensioni, che, a fianco della disponibilità del codice sorgente, garantisca che questo non sia stato

impropriamente modificato o alterato. Occorre certamente identificare anche un Ente che possa essere garante della corrispondenza del codice utilizzato da parti terze per produrre i risultati forniti all'utente finale con il codice ufficiale presente nel repository dell'Ente di gestione del progetto Open Source.
L'analisi degli obbietti preposti, tra cui uno sviluppo condiviso da parte di una comunità di sviluppatori indipendenti, e l'indagine condotta nell'ambito della produzione, manutenzione e sviluppo di software tecnico ha portato all'individuazioni di dieci azioni principali da intraprendere per la gestione futura di OpenBPS in una comunità di sviluppo; questi si possono sintetizzare in:

1. definizione di un Team di Gestione dell'intero progetto;
2. individuazione dell'oggetto su cui lavorare e del suo mercato di riferimento;
3. Individuazione della licenza o delle licenze associabili al progetto nel suo complesso e/o a parti dello stesso;
4. scelta della piattaforma di gestione del progetto;
5. individuazione delle caratteristiche di visibilità del repository (pubblico o privato);
6. creazione del repository, individuazione di un metodo di numerazione delle versioni successive, definizione delle regole con cui deve essere gestita la documentazione che deve accompagnare ogni singola parte del software sia per gli sviluppatori che per gli utenti;
7. assegnazione dei ruoli dei singoli attori e delle regole di ingaggio;
8. definizione delle modalità di verifica e accettazione delle modifiche ed integrazioni al codice sorgente;
9. definizione della modalità di compilazione dell'applicativo;
10. definizione delle modalità di verifica della conformità degli applicativi commerciali al codice sorgente originale.

Alcune di queste azioni, soprattutto le prime, richiedono decisioni puntuali che influenzano scelte o azioni successive in quanto ne determinano i gradi di libertà e conseguentemente influenzano le possibili utilizzazioni del software da parte del mercato.

La scelta principale, che di fatto instrada l'intero percorso di sviluppo del software e del protocollo per la sua gestione, è legata **all'individuazione del mercato di riferimento**. Gli elementi di scelta sono:

a. software per uso libero da parte del mercato per simulazioni in ambito di ricerca, universitario, sperimentazione;
b. software destinato a produrre output avente un ruolo legale, come ad esempio un calcolo di progettazione o un attestato di prestazione energetica;
c. una soluzione intermedia alle due.

La scelta tra a, b e c determina in cascata:

- la licenza associabile all'utilizzo finale del software;
- la piattaforma di riferimento per la gestione del processo di sviluppo del software e le modalità di gestione della stessa;
- la selezione degli sviluppatori e di coloro che aiuteranno a testare il software o sue parti.

La soluzione scelta influenza anche la futura scelta del livello di sicurezza e affidabilità della conformità dell'eseguibile o dei calcoli derivanti dalla sua applicazione al codice sorgente base.

Le principali figure e/o ruoli coinvolti, indipendentemente dal percorso scelto sono:

- team di gestione con ruoli manageriali sull'intero processo composto da un rappresentante ENEA e del Politecnico di Milano, possibilmente da un rappresentante del CTI e da figure tecniche quali: un

progettista software, un legale, uno o più esperti settoriali di modellazione energetica del sistema edificio impianto;

- Comunità di sviluppo composta da:
    - amministratore della comunità
    - amministratore informatico della comunità
    - esperti di modellazione dei singoli componenti del sistema edificio-impianto (indicativamente: involucro, climatizzazione invernale, climatizzazione estiva, automazione e controllo)
    - sviluppatori
    - esperti di settore per le fasi di test

Scenari possibili

Oltre l'approccio totalmente Open Source (sviluppo di un codice sorgente in comunità pubblica con licenza Open Source), i due scenari alternativi che si ritiene essere più sostenibili sono:

- sviluppo di un codice sorgente in comunità privata con licenza tale da consentire la distribuzione di un software eseguibile senza aver associato il sorgente. Validazione mediante chiave hash (SHA-256) sul software;
- sviluppo di un codice sorgente in comunità privata ma con licenza tale da consentirne l'utilizzo solo in un ambiente del tipo "web service". Validazione mediante chiave hash (SHA- 256) sull'input e output rilasciato all'utente remoto.

Azioni da intraprendere

Al termine del progetto occorrerà coordinare con le parti potenzialmente interessate allo sviluppo di OpenBPS e alla sua possibile applicazione a contesti anche regolamentati per legge, cioè gli sviluppatori da un lato (Politecnico di Milano), i finanziatori dall'altro (ENEA e MISE), eventuali associazioni tecnico scientifiche, come ad esempio AiCARR, IBPS-Italy, ecc., soggetti normatori come ad esempio il Comitato Termotecnico, una possibile linea di ulteriore sviluppo tramite l'identificazione del mercato di riferimento e quindi l'attuazione delle decisione che ne conseguono in cascata, come sopra evidenziato.

In assenza di tale identificazione e determinazione, lo sviluppo del codice OpenBPS e della relativa Community di sviluppo si muoverà nell'ottica puramente open source software per uso libero da parte del mercato per simulazioni in ambito di ricerca, universitario, sperimentazione.

Conclusioni

Al termine del terzo anno di attività del progetto di ricerca mirante a realizzare un codice di calcolo per la simulazione dinamica delle prestazioni energetiche del sistema edificio che, pur consentendo un'analisi dettagliata e di qualità (base di calcolo oraria e sub-oraria), sia utilizzabile da utenti con gradi diversi di competenza e che possa evitare una grande richiesta di dati all'utente, tramite l'impiego di informazioni provenienti direttamente dai costruttori di componenti, e che sia principalmente un insieme di librerie dinamiche fruibili da interfacce sviluppate da terzi, si può concludere che:

- il codice sviluppato il primo anno, riguardante la prestazione termo-energetica dell'involucro edilizio è stato esteso questo terzo anno includendo nel "nodo" aria il bilancio del vapore d'acqua, e quindi il calcolo del carico termico latente e del fabbisogno di energia per deumidificazione e umidificazione, presente quando si vuole controllare l'umidità dell'aria interna;

- a complemento dell'attività di sviluppo del codice, sempre nell'ottica di librerie dinamiche fruibili da terzi, sono stati sviluppati una serie di metodi per il calcolo delle proprietà termofisiche delle sostanze pure, per adesso limitate a acqua e aria secca, in funzione della temperatura e a pressione costante, e per la miscela aria umida, per la quale oltre le proprietà termofisiche, parametriche rispetto alla pressione, in funzione di ogni coppia possibile di coordinate psicrometriche, sono state implementate anche le principali trasformazioni della stessa (miscelazione adiabatica, deumidificazione, umidificazione, ecc.);
- una parte importante dell'attività del terzo anno è stata rivolta alla identificazione delle problematiche relative allo sviluppo efficiente e trasparente di quando fin qui prodotto nel futuro tramite la creazione di una comunità di sviluppatori indipendenti ma cooperanti in un'ottica Open Source; dei dieci punti chiave identificati solo alcuni hanno trovato una risposta immediata che si è consolidata tramite la progettazione e la realizzazione di una repository del codice e di un sito web del progetto; altri sono stati solo parzialmente risolti con proposte interlocutrici che ruotano principalmente su un quesito fondamentale, a cui si potrà dare risposta solo successivamente alla conclusione della presente ricerca mettendo intorno ad uno stesso tavoli i potenziali futuri attori interessati: qual è il mercato di riferimento di questo codice di calcolo, un Open Sorce o un Free-to-Use?
- il sito web sviluppato, disponibile su macchina virtuale e in files di backup, è attualmente implementato su un server del Politecnico di Milano, in versione demo, cioè senza la possibilità di scaricare gli allegati, documentazione e codice eseguibile demo, per renderlo visibile all'ENEA, all'indirizzo IP 131.175.28.249, e verrà reso completamente operativo, su tale host o su altro host eventualmente messo a disposizione da ENEA, solo dopo l'approvazione finale del progetto e con il nulla osta dell'ENEA;
- il repository del codice sorgente per lo sviluppo collaborativo e il controllo di versione tramite git, dotato di pagine Wiki e un forum per le discussioni tra sviluppatori, è disponibile su macchina virtuale e in files di backup ed è anch'esso attualmente implementato su un server del Politecnico di Milano, in modalità protetta, cioè senza la possibilità di visualizzazione e accesso a terzi che non possano entrare nella Intranet di Ateno, ciò ovviamente per i motivi citati in precedenza per il sito Web.

Introduzione

L'efficienza energetica degli edifici è importante sia dal punto di vista economico che ambientale ed è il punto focale della politica energetica e ambientale dell'Unione Europea, così come attuata tramite le due direttive EPBD e il loro aggiornamento [2,3,4], e della politica energetica italiana tramite il loro recepimento [5,6,7]. La tecnologia per pervenire a un notevole risparmio energetico nel settore dell'edilizia di fatto esiste già, ma tale potenziale risparmio non è ancora stato pienamente sfruttato, anche a causa di una prassi progettuale non più adeguata ai nuovi obiettivi e necessità. Infatti, per raggiungere gli obiettivi di efficienza posti delle nuove regole legislative, i progettisti hanno sempre più la necessità di utilizzare strumenti che siano adeguati all'analisi e la comprensione del complesso comportamento degli edifici, sia rispetto al fabbisogno energetico che al soddisfacimento e al mantenimento di altri requisiti prestazionali, quali il benessere termico, acustico e visivo. Fortunatamente, negli ultimi quarant'anni, per fornire una valutazione accurata e dettagliata della prestazione energica degli edifici e non solo, sono stati sviluppati, dapprima nell'ambito della ricerca, e, successivamente, in ambito commerciale, i Building Performance Simulation Tools (BPST) [8]. I BPST sono software che spaziano, dal punto di vista della complessità, dai semplici fogli di calcolo a strumenti di simulazione avanzati per applicazioni specifiche, e, dal punto di vista dell'integrazione, da strumenti che gestiscono un singolo aspetto della progettazione degli edifici a strumenti che ne integrano i molteplici aspetti. In modo esemplificativo l'evoluzione storica di tali strumenti può essere suddivisa in quattro generazioni. Gli strumenti di prima generazione si fondano su metodi semplificati che si ritrovano sui manuali, cioè su calcoli basati su formulazioni analitiche che comprendono molte ipotesi semplificative e quasi sempre sull'ipotesi di stazionarietà dei fenomeni considerati. Gli strumenti di seconda generazione si basano su metodi che introducono una modellazione semplificata, ancora analitica, della dinamica degli edifici. Gli strumenti attuali, di terza generazione, utilizzano invece metodi numerici e forniscono un'integrazione parziale dei diversi aspetti delle prestazioni degli edifici, ad esempio quelli energetici, illuminotecnici e acustici. La futura quarta generazione, in fase di sviluppo, tende ad una completa integrazione in relazione a diversi aspetti delle prestazioni dell'edificio, con nuovi sviluppi miranti alla realizzazione di interfacce utente che forniscono assistenza intelligente impiegando basi di dati di competenze specifiche, miranti al controllo della qualità delle applicazioni e alla formazione degli utenti. Limitandoci agli strumenti attualmente disponibili, questi possono quindi catturare le specificità della realtà molto meglio di strumenti precedenti (seconda e prima generazione), ma sono più complessi da utilizzare.

Gli strumenti per l'analisi delle prestazioni energetiche dell'edificio sono progettati e realizzati per prevedere il consumo annuo di energia dei sistemi impiantistici presenti. Basati su un sistema di equazioni che definisce le prestazioni termiche degli edifici e dei sistemi impiantistici considerati, questi strumenti eseguono simulazioni su base oraria o sub-oraria del comportamento dell'edificio, come si definisce oggi il sistema fabbricato-impianto, in funzione delle condizioni al contorno, delle strategie operative e delle azioni di controllo predefinite. Tali strumenti tra cui Carrier SAD, Trane TRACE 700, DOE-2, Equest , EnergyPlus, ESP-r, IDA ICE, TRNSYS, HVACSIM ,+, VA114, SIMBAD, IES, sono in genere utilizzati per calcolare e analizzare le prestazioni a pieno carico e a carico parziale, per analizzare la strategia di funzionamento del sistema, per confrontare le diverse alternative di progetto e così via. All'interno di questa categoria di strumenti si può isolare un sottogruppo che ha come specificità la capacità di analizzare correttamente l'impatto delle diverse tipologie e strategie di controllo, del quale fanno parte tra gli altri ESP-r, EnergyPlus, IDA ICE, TRNSYS. Gli strumenti di simulazione che appartengono a questo sottogruppo, dai quali ci si attende un'accurata valutazione delle prestazioni del sistema, devono essere in grado di trattare le deviazioni dal comportamento ideale che si verificano nei sistemi reali e di modellare realisticamente sia i controlli che la dinamica del

sistema impiantistico, nel seguito identificato dall'acronimo HVAC, il che però non è quasi mai verificato, giacché la quasi totalità dei componenti HVAC è modellata tramite equazioni stazionarie, che non descrivono sempre correttamente la dinamica del componente. L'unica possibilità di avere una buona corrispondenza tra realtà e simulazione è quella di rispettare le ipotesi alla base del modello di simulazione del fabbricato e dell'impianto e di procedere a una accurata calibrazione sui dati misurati. La prima condizione comporta la conoscenza dei modelli dei componenti e delle relative ipotesi semplificative, dei loro tempi caratteristici e quindi della determinazione del minimo passo temporale che comporti il soddisfacimento dell'ipotesi di stato quasi stazionario, che si ha quando le equazioni dello stato stazionario sono in grado di ben approssimare lo stato non stazionario; è evidente che ciò comporta una certa competenza specifica da parte dell'utente dello strumento di simulazione. La seconda condizione, in realtà necessaria solo se si opera su un edificio esistente ai fini di una diagnosi energetica, comporta conoscenze e competenze specifiche ancora maggiori. Come se ciò non bastasse, i diversi approcci di modellazione del sistema HVAC presenti nei diversi strumenti disponibili, quali quelli puramente concettuali, quelli basati sui componenti o su un approccio multidominio, richiedono diversi livelli di abilità da parte degli utenti, diverse risoluzioni temporali e/o spaziali nella modellazione e diversi livelli di capacità di personalizzazione. Un elevato livello di dettaglio nella rappresentazione del sistema richiede una maggiore conoscenza del sistema stesso, a causa del crescente numero di parametri necessari al modello per descriverlo e che spesso sono difficili da ottenere in quanto non vengono forniti dai produttori. Ciò comporta anche una maggiore richiesta computazionale, sia in tempo che in memoria, e un'analisi dei risultati più complicata, ergo maggiori costi. Di conseguenza, un buon modello deve essere il meno complesso possibile, ma tale da mantenere la sua validità in funzione degli obbiettivi della simulazione; infine, tale minima complessità del modello non è un parametro assoluto ma dipende da cosa si vuole ottenere.

Si deve poi considerare che al variare degli obiettivi della simulazione, quindi della complessità del modello, il costo, in termini di quantità di informazione, tempo necessario per costruirlo, tempo di calcolo e quantità di memoria impiegata, può superare il valore aggiunto prodotto per l'utente e tutto ciò va bilanciato con la necessità di avere un errore cioè una deviazione dei risultati del modello dai dati reali, che sia accettabile). L'errore in un modello che rispecchia fedelmente il sistema descritto è la somma di quello di astrazione, di quello nei dati di ingresso, e di quelli numerici; il primo è dovuto alle astrazioni apportate dalla modellazione a causa dell' incompletezza del modello rispetto al sistema fisico, il secondo è dovuto alle incertezze nei valori nei parametri di ingresso, che possono essere quantificate ottenendo la corrispondente incertezza del risultato del modello, la cosiddetta incertezza predittiva. L'errore di astrazione può dipendere dal livello di complessità, che diminuisce all'aumentare della complessità del modello, ma anche e soprattutto dall'inadeguatezza del modello, dovuta, ad esempio, all'impiego di un modello lineare in un sistema non lineare. Infine, gli errori numerici, che dipendono in parte dal passo spaziale di discretizzazione e in parte da quello temporale, possono essere controllati diminuendo il passo, il che aumenta il tempo di calcolo e l'uso di memoria. Come si può ben vedere, si è in presenza di una coperta corta che comporta una competenza particolare per un uso efficiente della simulazione dinamica.

In conclusione, l'applicazione della simulazione dinamica nella progettazione edilizia è a oggi problematica non solo perché gli strumenti di simulazione sono particolarmente complessi e molti progettisti edili non hanno familiarità con le loro proprietà e limitazioni, ma anche perché la stessa realizzazione del modello comporta competenze e conoscenze non usuali. Nella vita reale, la natura del processo di progettazione edile e la carenza di formazione nel settore della modellistica e della simulazione energetica hanno reso difficile sia per l'architetto che per l'ingegnere edile l'impiego efficiente e consistente di tali strumenti, comunque oggi indispensabili.

Come si può ovviare a tale oggettiva situazione di conflitto tra esigenze e competenze? Vi sono due vie che occorre percorrere in parallelo:

- sviluppare negli utilizzatori una migliore comprensione della modellazione e della simulazione energetica per consentire loro di valutare e mettere in pratica le tecniche per realizzare edifici ad alta efficienza energetica, tramite una formazione continue e mirata;

- rendere disponibili dei codici di simulazione dinamica del sistema edificio che, pur consentendo un'analisi dettagliata e di qualità (base di calcolo oraria e sub-oraria), siano utilizzabili da utenti con gradi diversi di competenza, facilmente integrabili in altri strumenti di progettazione, ad esempio nei BIM, e che possano evitare una grande richiesta di dati all'utente, tramite l'impiego di informazioni provenienti direttamente dai costruttori di componenti.

L'attività, di cui all'accordo di collaborazione tra Enea e Politecnico di Milano – Dipartimento di Energia, è proprio motivata e centrata sullo sviluppo di un codice di calcolo innovativo che risponda ai requisiti su citati, pur operando su base oraria o sub-oraria, e che sia in grado di analizzare in modo dettagliato i vari fenomeni termo-energetici che si attuano all'interno del sistema edificio.

Tale attività di ricerca e sviluppo del codice è articolata in tre fasi, corrispondenti a tre annualità, che affrontano rispettivamente,

- fase I:

  sviluppo e validazione del codice di calcolo relativo alla prestazione termo-energetica dell'involucro edilizio,

- fase II:

  lo sviluppo e la validazione dei codici di calcolo dei componenti impiantistici previsti nel pacchetto normativo sotto mandato della Commissione Europea per l'implementazione della EPBD recast e la loro integrazione nel processo di calcolo dinamico,

- fase III:

  definizione di una procedura codificata chiara e trasparente che consenta di assicurare, al termine del progetto, lo sviluppo continuativo, la manutenzione e l'aggiornamento del codice di calcolo, tramite il coinvolgimento di quanti, esperti del settore, vogliano e possano contribuire, prestando la loro competenza in modo coordinato ed efficace in un ambito "open source".

A margine di tale attività è poi prevista un'applicazione del codice sviluppato a dei test case forniti da Enea per consentire una comparazione di diversi metodi e modelli di calcolo, sviluppati sempre nell'ambito dello stesso piano triennale.

1 Libreria proprietà dei fluidi

1.1 Stato di riferimento standardizzato

Per la definizione delle caratteristiche termofisiche dei fluidi è necessario stabilire uno stato di riferimento che consenta un'attribuzione univoca ai potenziali e alle grandezze termodinamiche di interesse. Per una sostanza pura per la definizione di tale stato è sufficiente definire due sole grandezze di stato, la pressione p_0 e la temperatura T_0. Purtroppo, le condizioni di riferimento di temperatura e pressione standardizzate non sono univoche e, di seguito, sono riportate le più comuni:

- **STP - Standard Temperature and Pressure**

  definita dalla IUPAC (International Union of Pure and Applied Chemistry) come:

  - $T_{0;STP} = 273.15\ \mathrm{K} = 0\ ^\circ\mathrm{C}$
  - $p_{0;STP} = 1\ \mathrm{bar}$ (ma spesso si trova la vecchia definizione di $101\ 325\ \mathrm{Pa}$)
- **NTP - Normal Temperature and Pressure**
  - $T_{0;NTP} = 293.15\ \mathrm{K} = 20\ ^\circ\mathrm{C}$
  - $p_{0;NTP} = 101\ 325\ \mathrm{Pa}$
- **SATP - Standard Ambient Temperature and Pressure**
  - $T_{0;SATP} = 298.15\ \mathrm{K} = 25\ ^\circ\mathrm{C}$
  - $p_{0;SATP} = 101\ 325\ \mathrm{Pa}$

1.2 Aria atmosferica

Nel caso specifico dell'aria, considerata SECCA, cioè senza tener conto del contenuto di vapore d'acqua, l'ISO nel 1975 nella norma ISO 2533 [9], estesa nel 1977 [10], **specifica per l'aria dell'atmosfera**, definisce le seguenti condizioni di riferimento a livello medio del mare:

- **ISO - International Standard Atmosphere (ISA)**
  - $T_{0;ISA} = 288.15\ \mathrm{K} = 15\ ^\circ\mathrm{C}$
  - $p_{0;ISA} = 101\ 325\ \mathrm{Pa}$

per le quali deriva **a livello del mare e per aria secca**, avendo assunto

- $R = 8\ 314.32\ \mathrm{J/(kmol \cdot K)}$ — costante universale dei gas
- $\kappa = c_p/c_v = 1.4$ — rapporto tra capacità termiche specifiche a pressione e volume costante

e tramite l'applicazione della legge dei gasi ideali, le seguenti proprietà:

- $M_{DA} = 28.964\ 42\ \mathrm{kg/kmol}$ — massa molecolare aria secca
- $R_{DA} = 287.052\ 87\ \mathrm{J/(kg \cdot K)}$ — constante di gas ideale dell'aria secca
- $\rho_{0;ISA} = 1.2250\ \mathrm{kg/m^3}$ — massa volumica aria secca

- $c_{0;ISA} = 340.29\ \mathrm{m/s}$ velocità del suono
- $\mu_{0;ISA} = 1.7894 \cdot 10^{-6}\ \mathrm{Pa \cdot s}$ viscosità dinamica
- $\lambda_{0;ISA} = 25.343 \cdot 10^{-3}\ \mathrm{W/(m \cdot K)}$ conducibilità termica

Le proprietà derivate sono calcolate secondo le seguenti relazioni:

$$\rho_{DA} = \frac{p}{R_{DA} \cdot T} \tag{1.1}$$

$$c_{DA} = \sqrt{\kappa \cdot R_{DA} \cdot T} \tag{1.2}$$

$$\mu_{DA} = \frac{1.458 \cdot 10^{-6} \cdot T^{1.5}}{T + 110.4} \tag{1.3}$$

$$\lambda_{DA} = \frac{2.648\,151 \cdot 10^{-3} \cdot T^{1.5}}{T + [245.4 \cdot 10^{-(12/T)}]} \tag{1.4}$$

La norma ISO assume un profilo lineare della temperatura con la quota, per cui, limitandosi a 11.000 metri di quota massima, la distribuzione di temperatura, approssimando l'altezza geopotenziale con l'altitudine (errore inferiore allo 0.2% a 11 000 metri), risulta essere:

$$T_{DA}(h_{alt}) = T_{0;ISA} + \beta \cdot\ h_{alt} = 288.15 - 6.5 \cdot 10^{-3}\ h_{alt} \quad [\mathrm{K}] \tag{1.5}$$

con

β gradiente termico tra 0 e 11 000 m, pari a $-6.5 \cdot 10^{-3}$ K/m;

h_{alt} altitudine sopra il livello del mare, in [m].

Di conseguenza la distribuzione di pressione risulta essere (Appendice A):

$$p(h_{alt}) = p_{0;ISA} \cdot \left(T/T_{0;ISA}\right)^{-(g/\beta R_{DA})} = 101\,325 \cdot\ (1 - 2.5577 \cdot 10^{-5}\ h_{alt})^{5.25588} \tag{1.6}$$

per

g accelerazione gravitazionale pari a 9.80665 m/s^2.

Analogamente (Appendice A) si può derivare la distribuzione di massa volumica con la quota, tra 0 e 11 000 metri, risulta essere data da:

$$\rho(h_{alt}) = \rho_{0;ISA} \cdot \left(T/T_{0;ISA}\right)^{-(1+g/\beta R_{DA})} = 1.225 \cdot\ (1 - 2.5577 \cdot 10^{-5}\ h_{alt})^{4.25588} \tag{1.7}$$

1.2.1 Aria secca standard in aeraulica

La norma UNI EN ISO 5801 [11] riporta le caratteristiche dell'aria secca in condizioni standard per la definizione delle prestazioni dei ventilatori e quindi costituisce un riferimento standardizzato per tutti i sistemi aeraulici della climatizzazione e della ventilazione ambientale.

Condizioni standard adottate ➔ NTP: $T = 293.15\ \mathrm{K}$ (20 °C) e $p = 101\,325\ \mathrm{Pa}$

Le proprietà definite in condizioni **NTP** sono:

- $\kappa = c_p/c_v = 1.4$ rapporto tra capacità termiche specifiche a pressione e volume costante
- $R_{DA} = 287\ \ \mathrm{J/(kg\ K)}$ constante di gas ideale dell'aria secca
- $c_p = 1\,004.5\ \ \mathrm{J/(kg\ K)}$ capacità termica specifica a pressione costante
- $c_v = 717.5\ \ \mathrm{J/(kg\ K)}$ capacità termica specifica a pressione costante
- $\rho = 1.2\ \ \mathrm{kg/m^3}$ massa volumica aria secca
- $\mu = 10.06 \cdot 10^{-6}\ \ \mathrm{Pa \cdot s}$ viscosità dinamica
- $c = 343.8\ \ \mathrm{m/s}$ velocità del suono

Inoltre la stessa norma propone una correlazione per determinare la viscosità dell'aria secca in funzione della temperatura, nel campo da −20 °C a +100 °C, some segue:

$$\mu = (17.1 + 0.048 \cdot \theta) \cdot 10^{-6}\ \ \mathrm{Pa \cdot s} \tag{1.8}$$

con θ temperatura espressa in gradi celsius.

1.2.2 Aria umida standard in aeraulica

La stessa norma UNI EN ISO 5801 [11] riporta le caratteristiche dell'aria umida in **condizioni standard NTP** ($T = 293.15\ \mathrm{K}$ e $p = 101\,325\ \mathrm{Pa}$) aggiungendo un'umidità relativa standard e relative costanti di gas ideale, cioè:

- $\phi = 0.40$ umidità relativa standard
- $R_{MA} = 288\ \ \mathrm{J/(kg\ K)}$ constante di gas ideale dell'aria umida
- $R_{H_2O} = 461.5\ \ \mathrm{J/(kg\ K)}$ constante di gas ideale del vapore d'acqua

In queste condizioni la massa volumica dell'aria umida standard risulta essere

$$\rho_{MA} = \frac{101\,325}{288 \cdot 293.15} = 1.20035\ \ \mathrm{kg/m^3} \tag{1.9}$$

da cui segue che, in condizioni standard, si può assumere indifferentemente sia per aria secca che per aria umida il valore unico di 1.2 kg/m^3.

Inoltre la norma riporta una correlazione per il calcolo della pressione di vapore saturo tra $0 \leq \theta \leq 100$ °C, come segue:

$$p_{sat} = 610.8 + 44.442 \cdot \theta + 1.413\,3 \cdot \theta^2 + 0.0276\,8 \cdot \theta^3 + 2.55667 \cdot 10^{-4} \cdot \theta^4 + 2.891\,66 \cdot 10^{-6} \cdot \theta^5 \tag{1.10}$$

1.2.3 Aria umida standard: formulazione estesa

Nelle indicazioni della norma UNI EN ISO 5801 [11] mancano però sia la definizione delle grandezze che caratterizzano lo stato dell'aria umida, riportate comunque nella norma UNI EN ISO 9346 [12], che la definizione dello stato di riferimento per le proprietà che caratterizzano il suo stato energetico.

Nel corpo normativo si trovano solo indicazioni sparse su alcune delle proprietà caratteristiche dell'aria umida, come, nel caso della norma

- **UNI EN 16798-5-1** [13], che introduce l'entalpia di transizione di fase liquido saturo-vapore saturo come:

$\Delta h_{l\to v}^{(H_2O)} = 2448 \quad \mathrm{J/kg}$ (senza specificare a quali condizioni si ha tale valore)

e la distribuzione della massa volumica con la quota come

$$\rho_{DA} = \rho_{DA;st}\left(1 - \frac{0.00651 \cdot h_{alt}}{293}\right)^{4.255} \tag{1.11}$$

dove

h_{alt} altitudine sopra il livello del mare, in [m];

$\rho_{DA;st} = 1{,}204 \quad \mathrm{kg/m^3}$

NOTA: occorre notare che la (1.11) risulta essere in parte inconsistente rispetto alla definizione di proprietà dell'aria atmosferica definite nella norma ISO 2533 [9]; infatti ricavando da questa la distribuzione di massa volumica con la quota (Appendice A), equazione (1.7), si può vedere che la

- UNI EN 16798-5-1 adotta come stato di riferimento **NTP**: $T_{0;NTP} = 293.15 \ \mathrm{K}$ approssimato all'intero e un gradiente di temperatura verticale $\beta = 0.00651$ K/m, esponente 4.255;
- ISO 2533 si basa sull'**ISA**: $T_{0;ISA} = 288.15 \ \mathrm{K}$ con un gradiente di temperatura verticale $\beta = 0.0065$ K/m e un esponente pari a 4.25588;

differenze minimizzate (minori al 2%) dalla assunzione, quale massa volumica di riferimento, nella (1.11) della massa volumica a pressione atmosferica a $293 \ \mathrm{K}$, $\rho_{DA;st} = 1{,}204 \ \mathrm{kg/m^3}$, leggermente superiore al valore della UNI EN ISO 5801 a pari condizioni di riferimento (1.2 kg/m^3).

- **UNI EN ISO/DIS 8502-4** [14], che riporta la seguente relazione per la temperatura di rugiada, valida per $\theta > 0$,:

$$\theta_{dp} = 234.175 \cdot \frac{(234.175 + \theta)(\ln 0.01 + \ln \phi) + 17.080\,85 \cdot \theta}{234.175 \cdot 17.080\,85 - (234.175 + \theta)(\ln 0.01 + \ln \phi)} \tag{1.12}$$

- **UNI EN ISO 13788** [15], che riporta invece delle correlazioni per la determinazione della pressione di saturazione del vapore in funzione della temperatura e viceversa, la temperatura in funzione della pressione, cioè:

$$p_{sat} = 610.5 \cdot e^{\frac{17.269\,\theta}{237.3+\theta}} \quad \forall \ \theta \geq 0 \ ^\circ\mathrm{C} \tag{1.13}$$

$$p_{sat} = 610.5 \cdot e^{\frac{21.875\,\theta}{265.5+\theta}} \quad \forall \ \theta < 0 \ ^\circ\mathrm{C} \tag{1.14}$$

e

$$\theta_{sat} = \frac{237.3 \ \ln\left(\frac{p_v}{610.15}\right)}{17.269 - \ln\left(\frac{p_v}{610.5}\right)} \quad \forall \ p_v \geq 610.5 \ \mathrm{Pa} \tag{1.15}$$

$$\theta_{sat} = \frac{265.5 \ \ln\left(\frac{p_v}{610.5}\right)}{21.875 - \ln\left(\frac{p_v}{610.5}\right)} \quad \forall \ p_v < 610.5 \ \mathrm{Pa} \tag{1.16}$$

e inoltre riporta una costante dei gas ideale per il vapore d'acqua pari a

- $R_{H_2O} = 462 \ \mathrm{J/(kg\,K)}$ approssimando all'intero il valore della UNI EN ISO 5801.

- **UNI EN 15316-4-1** [**Error! Reference source not found.**], che riporta la seguente formula per il calcolo dell'entalpia di vaporizzazione dell'acqua:

$$\Delta h_{l\to v}^{(H_2O)} = 2500.6 - 2,.435 \cdot \theta \quad \text{J/kg} \tag{1.17}$$

con θ temperatura di transizione di fase in gradi celsius.

1.2.4 ASHRAE Fundamentals (2017)

La "standardizzazione" delle proprietà dell'aria secca e umida operata dall'ASHRAE sono riportate nel Handbook of Fundamentals [17].

Costanti

- $R = 8\,314.472$ J/(kmol K) costante universale gas ideale
- $M_{DA} = 28.966$ kg/kmol massa molecolare aria secca
- $R_{DA} = 287.042$ J/(kg K) constante di gas ideale dell'aria secca
- $M_{H_2O} = 18.015\,268$ kg/kmol massa molecolare vapore d'acqua
- $R_{H_2O} = 461.524$ J/(kg K) constante di gas ideale del vapore d'acqua
- $\varepsilon = M_{H_2O}/M_{DA} \cong 0.621\,945$ rapporto masse molari vapore d'acqua- aria secca
- $c_{p;DA} = 1\,006$ J/(kg K) capacità termica specifica a pressione costante aria secca
- $c_{p;lw} = 4\,186$ J/(kg K) capacità termica specifica a pressione costante acqua liquida
- $c_{p;H_2O} = 1\,860$ J/(kg K) capacità termica specifica a pressione costante vapor d'acqua
- $\Delta h_{l\to v}^{(H_2O)} = 2.501 \cdot 10^6$ J/kg entalpia di transizione di fase liquido saturo-vapore saturo

Stato di riferimento

- $p_0 = 101\,325$ Pa pressione di riferimento (~ atmosferica a quota del mare)
- $T_0 = 293.15$ K $= 0$ °C temperatura di riferimento

Variazione della pressione e temperatura dell'atmosferica con la quota:

- Atmosfera Standard U.S.

$$p = 101.325 \cdot (1 - 2.255\,77 \cdot 10^{-5} \cdot h_{alt})^{5.2559} \quad \text{kPa} \tag{1.18}$$

$$\theta = 15 - 0.0065 \cdot h_{alt} \quad °\text{C} \tag{1.19}$$

con h_{alt} altitudine sopra il livello del mare, in [m];

Pressione di saturazione del vapore p_{sat} in funzione della temperatura θ

Nel campo di temperatura da -100 a 0 °C:

$$\ln p_{sat} = C_1/T + C_2 + C_3 T + C_4 T^2 + C_5 T^3 + C_6 T^4 + C_7 \ln T \tag{1.20}$$

con $T = \theta + 273.15$ K e

$C_1 = -5.674\,535\,9$ E+03
$C_2 = 6.392\,524\,7$ E+00
$C_3 = -9.677\,843\,0$ E−03

$C_4 = 6.221\,570\,1\ \text{E}{-07}$
$C_5 = 2.074\,782\,5\ \text{E}{-09}$
$C_6 = -9.484\,024\,0\ \text{E}{-13}$
$C_7 = 4.163\,501\,9\ \text{E}{+00}$

Nel campo di temperatura da 0 °C a 200 °C:

$$\ln p_{sat} = C_8/T + C_9 + C_{10}T + C_{11}T^2 + C_{12}T^3 + C_{13}\ln T \tag{1.21}$$

con

$C_8 = -5.800\,220\,6\ \text{E}{+03}$
$C_9 = 1.391\,499\,3\ \text{E}{+00}$
$C_{10} = -4.864\,023\,9\ \text{E}{-02}$
$C_{11} = 4.176\,476\,8\ \text{E}{-05}$
$C_{12} = -1.445\,209\,3\ \text{E}{-08}$
$C_{13} = 6.545\,967\,3\ \text{E}{+00}$

Da cui

$$p_{sat} = e^{(\ln p_{sat})} \quad \text{[Pa]} \tag{1.22}$$

Temperatura di saturazione del vapore θ_{dp} in funzione della pressione parziale del vapore p_v

Nel campo di temperatura minore di 0 °C:

$$\theta_{dp} = 6.09 + 12.608\,\alpha + 0.4959\,\alpha^2 \tag{1.23}$$

e nel campo di temperatura da 0 °C a 93 °C:

$$\theta_{dp} = C_{14} + C_{15}\alpha + C_{16}\alpha^2 + C_{17}\alpha^3 + C_{18}(p_v/1000) \tag{1.24}$$

con

$\alpha = \ln(p_v/1000)$ con p_v in pascal e con

$C_{14} = 6.54$
$C_{15} = 14.526$
$C_{16} = 0.7389$
$C_{17} = 0.09486$
$C_{18} = 0.4569$

1.2.5 Grandezze psicrometriche

Basandosi sulla teoria delle miscele di gas ideali e, in particolare, sulla nozione di pressione parziale di un componente la miscela, che corrisponde a trasporre la composizione della miscela da frazione molare a scala delle pressioni, per l'aria umida considerata una miscela bicomponente si ha

$$p_v \equiv p_t \cdot \frac{N_{H_2O}}{N_{H_2O} + N_{DA}} \tag{1.25}$$

$$p_a \equiv p_t \cdot \frac{N_{DA}}{N_{H_2O} + N_{DA}} \tag{1.26}$$

con

p_v pressione parziale del vapore d'acqua, in [Pa]

p_a pressione parziale dell'aria secca, in [Pa]

p_t pressione termodinamica della miscela, in [Pa];

N_{DA} numero di moli di aria secca nella muscela, in [mol]

N_{H_2O} numero di moli di vapore d'acqua nella miscela, in[mol]

Le principali grandezze psicrometriche sono:

- **Umidità relativa ϕ**

$$\phi \equiv \frac{p_v}{p_{sat}(\theta)} = f(p_v, \theta) \quad [-] \tag{1.27}$$

con

p_v pressione parziale del vapore, in [Pa]

p_{sat} pressione di saturazine del vapore, in [Pa]

- **Umidità massica x**

$$x \equiv \frac{m_{H_2O}}{m_{DA}} = \frac{M_{H_2O}}{M_{DA}} \cdot \frac{p_v}{p_a} = \varepsilon \cdot \frac{p_v}{p_{atm} - p_v} = \varepsilon \cdot \frac{\phi \cdot p_{sat}(\theta)}{p_{atm} - \phi \cdot p_{sat}(\theta)} = f_x(\theta, \phi; p_{atm}) \quad \left[\frac{\mathrm{kg_{H_2O}}}{\mathrm{kg_{DA}}}\right] \tag{1.28}$$

con

p_{atm} pressione dell'atmosfera all quota considerata, in [Pa]

- **Entalpia specifica h_{MA}**

$$h_{MA} \equiv h_{DA}(\theta) + x \cdot h_{H_2O}(\theta, p_v) = f_h(\theta, p_v, x; p_{atm}) \quad \left[\frac{\mathrm{J}}{\mathrm{kg_{DA}}}\right] \tag{1.29}$$

con

h_{DA} entalpia specifica dell'aria secca, in [J/kg$_{DA}$]

h_{H_2O} entalpia specifica del vapore d'acqua, in [J/kg$_{H2O}$]

x umidità massica, in [kg$_{H2O}$/kg$_{DA}$].

- **Temperatura a bulbo umido θ_{wb}**

è definita come la temperatura di saturazione di una t**rasformazione adiabatica di saturazione** con iniezione di acqua liquida alla stessa temperatura che si ha al termine della saturazione, cioè dalla risoluzione dell'equazione implicita in θ_{wb}

$$\theta_{wb} \equiv \theta_{sat}(\theta_{db}, x) \;\rightarrow\; h_{MA}(\theta_{db}, x) + [x_{sat}(\theta_{wb}) - x] \cdot h_{lw}(\theta_{wb}) = h_{MA;sat}(\theta_{wb}) \tag{1.30}$$

con

h_{lw} entalpia specifica dell'acqua liquida

- **Temperatura di rugiada θ_{dp}**

è definita come la temperatura di saturazione che si raggiunge a pressione totale costante con una **trasformazione di raffreddamento a umidità massica costante**, risoluzione dell'equazione implicita in θ_{dp}

$$\theta_{dp} \equiv \theta_{sat}(\theta_{db}, x) \;\rightarrow\; x_{sat}(\theta_{sat}) = x = \varepsilon \frac{p_{sat}(\theta_{dp})}{p_{atm} - p_{sat}(\theta_{dp})} \tag{1.31}$$

da cui

$$\theta_{dp} \equiv \theta_{sat}(\theta_{dp}, x) \;\rightarrow\; p_{sat}(\theta_{db}) = p_{atm}\,\frac{x}{\varepsilon + x} \;\rightarrow\; \theta_{db} = f_{dp}\left(p_{atm}\,\frac{x}{\varepsilon + x}\right) = f_{dp}(x; p_{atm}) \tag{1.32}$$

- **Volume specifico v_{MA}**

$$v_{MA} \equiv \frac{V}{m_{DA}} = \frac{R_{DA}T}{p_{atm}}\left(1 + \frac{x}{\varepsilon}\right) = f_v(\theta, x; p_{atm}) \qquad \left[\frac{\mathrm{m}^3}{\mathrm{kg_{DA}}}\right] \tag{1.33}$$

- **Massa volumica ρ_{MA}**

$$\rho_{MA} \equiv \frac{m_{DA} + m_{H_2O}}{V} = \frac{1}{v_{MA}}(1 + x) = f_\rho(\theta, x; p_{atm}) \qquad \left[\frac{\mathrm{kg_{MA}}}{\mathrm{m}^3}\right] \tag{1.34}$$

1.2.6 Calcolo semplificato dell'entalpia specifica dell'aria umida in aeraulica

Mentre l'aria secca può essere considerata, nel capo di temperature e pressioni caratteristiche dell'aeraulica (pressione atmosferica e temperature tra -45 e + 80 °C), con ottima approssimazione un gas ideale con capacità termiche specifiche costanti, per il vapore d'acqua tale approssimazione è più problematica: infatti, occorre tenere conto della transizione di stato liquido-vapore e dal comportamento del vapore in vicinanza della curva limite che risente in modo significativo dell'effetto di gas reale. Per minimizzare tale effetto e poter determinare una relazione semplificata ma sufficientemente accurata per il calcolo dell'entalpia specifica del vapore d'acqua si è operato nel seguente modo:

- **stato di riferimento per la determinazione dell'entalpia specifica del vapore d'acqua**:
  - punto triplo: $T_0 = 273.16\ \mathrm{K}$, $p_0 = 611.7\ \mathrm{Pa}$ ➔ $h_{H_2O;0} \equiv h_{H_2O;liq.sat} = 0$

  entalpia specifica del liquido saturo al punto triplo, assunto come riferimento, è posta uguale a 0;

  NOTA: nelle tabelle del vapore considerato gas reale cosi come calcolate dalla IAPWS (*International Association for the Properties of Water and Steam*) [18] al punto triplo, assunto come riferimento sempre lo stato del liquido saturo, viene considerata nulla l'energia interna specifica e non l'entalpia specifica, che risulta essere pari a 0.611 872 J/kg.

  la transizione di fase liquido vapore viene assunta avvenire sempre alle condizioni del punto triplo e quindi presa costante e pari a:

  - $\Delta h_{l\rightarrow v;0}^{(H_2O)} \equiv \Delta h_{l\rightarrow v}^{(H_2O)}(T_0) = const;$

  il vapore d'acqua saturo alle condizioni del punto triplo e in prossimità della curva limite alla pressione del punto triplo viene considerato con buona approssimazione un gas ideale con capacità termiche specifiche costanti:

  - $c_{p;H_2O} = const$

  l'entalpia specifica dell'acqua può essere quindi determinata con la seguente equazione approssimante:

$$h_{H_2O} = \Delta h_{l\rightarrow v;0}^{(H_2O)} + c_{p;H_2O}\cdot(T - T_0) \cong \Delta h_{l\rightarrow v;0}^{(H_2O)} + c_{p;H_2O}\cdot\theta \tag{1.35}$$

con $\begin{cases} T_0 = 273.16\ \mathrm{K} \\ p_0 = 611.7\ \mathrm{Pa} \end{cases}$ ➔ $(T - T_0) \cong (T - 273.15) = \theta$

- **stato di riferimento per la determinazione dell'entalpia specifica dell'aria secca considerata gas ideale**:
  - temperatura: $T_0 = 273.15\ \mathrm{K}$, ➔ $h_{DA;0} \equiv 0$

l'entalpia specifica dell'acqua può essere quindi determinata con la seguente equazione:

$$h_{DA} = c_{p;DA} \cdot (T - T_0) = c_{p;DA} \cdot (T - 273.15) = c_{p;DA} \cdot \theta \tag{1.36}$$

L'equazione generale (1.29) che definisce l'entalpia specifica dell'aria umida può essere quindi riscritta come:

$$h_{MA} \equiv h_{DA} + x \cdot h_{H_2O} = c_{p;DA} \cdot \theta + x\left[\Delta h_{l\to v}^{(H_2O)} + c_{p;H_2O} \cdot \theta\right] \qquad \left[\frac{\mathrm{J}}{\mathrm{kg_{DA}}}\right] \tag{1.37}$$

Con le approssimazioni introdotte l'entalpia specifica dell'aria umida non è più una funzione di tre variabili (temperatura pressione e umidità massica) ma si riduce ad una funzione di due variabile, parametrica rispetto alla pressione atmosferica; cioè:

$$h_{MA}(\theta, p_v, x; p_{atm}) \quad \Longrightarrow \quad h_{MA}(\theta, x; p_{atm}) \tag{1.38}$$

1.2.7 Calcolo standardizzato semplificato delle grandezze psicrometriche

Per il calcolo delle proprietà termodinamiche dell'aria umida, approccio semplificato, si assumono i seguenti valori per le costanti fisiche di base:

Costanti

- $\boldsymbol{R = 8\,314.462\,6(18)\ \mathrm{J/(kmol\ K)}}$ costante universale gas ideale, da [22] e non da [18][1]
- $\boldsymbol{M_{DA} = 28.965\,46\ \mathrm{kg/kmol}}$ massa molecolare aria secca, da [20] e [21]
- $\boldsymbol{R_{DA} = 287.047\ \ \mathrm{J/(kg\ K)}}$ constante di gas ideale dell'aria secca, ricalcolata e approssimata a 6 cifre significative
- $\boldsymbol{M_{H_2O} = 18.015\,268\ \ \mathrm{kg/kmol}}$ massa molecolare vapore d'acqua, da [18]
- $\boldsymbol{R_{H_2O} = 461.523\ \ \mathrm{J/(kg\ K)}}$ constante di gas ideale del vapore d'acqua, ricalcolata e approssimata a 6 cifre significative
- $\boldsymbol{\varepsilon = M_{H_2O}/M_{DA} \cong 0.621\,957}$ rapporto masse molari vapore d'acqua- aria secca, ricalcolato tramite le masse molari riportate con 6 c.s.
- $\boldsymbol{c_{p;DA} = 1\,006\ \ \mathrm{J/(kg\ K)}}$ capacità termica specifica a pressione costante aria secca, approssimata all'intero da [23]
- $\boldsymbol{c_{p;lw} = 4\,186\ \mathrm{J/(kg\ K)}}$ capacità termica specifica dell'acqua liquida a 1 bar (valore medio tra 20 e 90 °C) da [19]
- $\boldsymbol{c_{p;H_2O} = 1\,888\ \ \mathrm{J/(kg\ K)}}$ capacità termica specifica a pressione costante del vapore d'acqua saturo al punto triplo, approssimata all'intero, da [19]
- $\boldsymbol{\Delta h_{l\to v;0}^{(H_2O)} = 2.500\,91 \cdot 10^6 \quad \mathrm{J/kg}}$ entalpia di transizione di fase liquido saturo-vapore saturo, al punto triplo da [19],

Stato di riferimento: STP (Standard Temperature and Pressure)

- $p_0 = p_{0;STP} = 101\,325\ \mathrm{Pa}$ pressione di riferimento (~ atmosferica a quota del mare)

[1] Viene utilizzato il valore raccomandato dal CODATA 2018 [21] di 8.314 462 618 J/(mol K) invece del valore di 8.314 371 J/(mol K) utilizzato per la costruzione le tabelle internazionali del vapore d'acqua IAPWS [17]; data la significativa approssimazione utilizzata per determinare l'entalpia specifica del vapore d'acqua non ha molto senso imporre tale consistenza.

- $T_0 = T_{0;STP} = 273.15\ \text{K} = 0\ °\text{C}$ temperatura di riferimento

Variazione della pressione, massa volumica e temperatura dell'aria secca atmosferica con la quota:

per quote comprese tra 0 e 11 000 m si utilizzano le equazioni (1.5), (1.6) e (1.7) ricavate dalla norma ISO 2533 [9], che vengono però, per consistenza, traslate a $T_0 = T_{0;STP} = 273.15\ \text{K}$, cioè:

$$T_{atm;0}(h_{alt}) = 273.15 - 6.5 \cdot 10^{-3}\ h_{alt} \quad [\text{K}] \tag{1.39}$$

$$p_{atm;0}(h_{alt}) = 101\ 325 \cdot\ (1 - 1.74193 \cdot 10^{-5}\ h_{alt})^{5.25588} \tag{1.40}$$

$$\rho_{atm;0}(h_{alt}) = 1.2923 \cdot\ (1 - 1.74193 \cdot 10^{-5}\ h_{alt})^{4.25588} \tag{1.41}$$

per $\rho_{atm;0}(0) = R_{DA}T_{0;STP}/p_{atm;0} = 1.2923$ kg/m³.

Di conseguenza la temperatura in quota rispetto ad una generica temperatura a livello del mare è data da

$$T_{atm}(h_{alt}) = T_{atm;0}(h_{alt}) + T_{atm}(0) - 273.15 = T_{atm}(0) - 6.5 \cdot 10^{-3}\ h_{alt} \quad [\text{K}] \tag{1.42}$$

Mentre per pressione e massa volumica non vi è una semplice traslazione, ma occorre di volta in volta ricalcolare il coefficiente angolare a moltiplicatore della quota e per la massa volumica anche il valore a quota zero, cioè:

$$p_{atm}(h_{alt}) = 101\ 325 \cdot \left(1 - \frac{0.0065}{T_{atm}(0)}\ h_{alt}\right)^{5.25588} \tag{1.43}$$

$$\rho_{atm}(h_{alt}) = \frac{352.991}{T_{atm}(0)} \cdot \left(1 - \frac{0.0065}{T_{atm}(0)}\ h_{alt}\right)^{4.25588} \tag{1.44}$$

1.2.7.1 Pressione di saturazione per il vapore d'acqua, funzione di θ

Limitando il campo di applicazione le correlazione della UNI EN ISO 13788 [15] presentano uno scostamento massimo rispetto alla tabelle del vapore [19] dello 0.25% tra 0 e 80 °C e dell' 1% tra 0 e -30 °C. Si è preferito scegliere questa formulazione rispetto ad altre leggermente più accurate in tale campo di temperature, per la forma direttamente invertibile e per la velocità di calcolo, essendo comunque una trattazione semplificata dell'aria umida; quindi

$$p_{sat} = 610.5 \cdot e^{\frac{17.269\ \theta}{237.3+\theta}} \quad \forall\ \ \theta \geq 0\ °\text{C} \tag{1.45}$$

$$p_{sat} = 610.5 \cdot e^{\frac{21.875\ \theta}{265.5+\theta}} \quad \forall\ \ \theta < 0\ °\text{C} \tag{1.46}$$

1.2.7.2 Temperatura di saturazione per il vapore d'acqua, funzione di p_v

Limitando il campo di applicazione le correlazioni della UNI EN ISO 13788 [15] presentano uno scostamento massimo rispetto alle tabelle del vapore [19] dello 0.2% tra 0 e 100 °C e dello 0.42% tra 0 e -40 °C; quindi per i motivi di cui la punto precedente si adottano se seguenti correlazioni:

$$\theta_{sat} = \frac{237.3\ \ln\left(\frac{p_v}{610.5}\right)}{17.269 - \ln\left(\frac{p_v}{610.5}\right)} \qquad \forall\ \ p_v \geq 610.5\ \text{Pa} \tag{1.47}$$

$$\theta_{sat} = \frac{265.5\ \ln\left(\frac{p_v}{610.5}\right)}{21.875 - \ln\left(\frac{p_v}{610.5}\right)} \qquad \forall\ \ p_v < 610.5\ \text{Pa} \tag{1.48}$$

1.2.7.3 Umidità massica in funzione di θ e ϕ (parametrica rispetto a p_{atm})

Per la (1.28) e le costanti fisiche adottate si ha:

$$x = \varepsilon \cdot \frac{\phi \cdot p_{sat}(\theta)}{p_{atm} - \phi \cdot p_{sat}(\theta)} = 0.621\,957 \cdot \frac{\phi \cdot p_{sat}(\theta)}{p_{atm} - \phi \cdot p_{sat}(\theta)} \qquad \left[\frac{\text{kg}_{\text{H}_2\text{O}}}{\text{kg}_{\text{DA}}}\right] \tag{1.49}$$

1.2.7.4 Entalpia specifica dell'aria umida in funzione di θ e x

Per la (1.37) e le costanti fisiche adottate si ha:

$$h_{MA} = c_{p;DA} \cdot \theta + x\left[\Delta h_{l\to v}^{(H_2O)} + c_{p;H_2O} \cdot \theta\right] = 1.006 \cdot \theta + x \cdot [2501 + 1.888 \cdot \theta] \qquad \left[\frac{\text{kJ}}{\text{kg}_{\text{DA}}}\right] \tag{1.50}$$

1.2.7.5 Entalpia specifica dell'acqua liquida in funzione di θ

Assumendo costante nel campo di interesse la capacità termica specifica a pressione costante dell'acqua liquida e preso come riferimento quello STP, l'entalpia specifica dell'acqua liquida piò essere calcolata avendo preso $h_{lw;0} = 0$ per $T_0 = 273.15$ K, come:

$$h_{lw} = c_{p;lw} \cdot (T - T_0) = 4.1086 \cdot \theta \qquad \left[\frac{\text{kJ}}{\text{kg}_{H_2O}}\right] \tag{1.51}$$

1.2.7.6 Temperatura di rugiada in funzione di x (parametrica rispetto a p_{atm})

Dalla (1.32) per

$$p_{sat}(\theta_{dp}) = p_{atm}\ \frac{x}{\varepsilon + x} \tag{1.52}$$

si ricava

$$\theta_{dp} = \frac{237.3\ \ln\left(\frac{p_{atm}}{610.5} \cdot \frac{x}{0.621\,957 + x}\right)}{17.269 - \ln\left(\frac{p_{atm}}{610.5} \cdot \frac{x}{0.621\,957 + x}\right)} \qquad \forall\ \ p_v \geq 610.5\ \text{Pa} \tag{1.53}$$

$$\theta_{dp} = \frac{265.5\ \ln\left(\frac{p_{atm}}{610.5} \cdot \frac{x}{0.621\,957 + x}\right)}{21.875 - \ln\left(\frac{p_{atm}}{610.5} \cdot \frac{x}{0.621\,957 + x}\right)} \qquad \forall\ \ p_v < 610.5\ \text{Pa} \tag{1.54}$$

1.2.7.7 Temperatura di bulbo umido in funzione di θ e x (parametrica rispetto a p_{atm})

La temperatura di bulbo umido è definita dalla relazione (1.30), che per la (1.50) può essere riscritta come:

$$\theta_{wb} = \frac{h_{MA}(\theta_{db}, x) - \boldsymbol{x_{sat}}(\boldsymbol{\theta_{wb}}) \cdot \Delta h_{l\to v}^{(H_2O)_0}}{c_{p;DA} + \boldsymbol{x_{sat}}(\boldsymbol{\theta_{wb}}) \cdot c_{p;H_2O} - (x_{sat} - x) c_{p;lw}} \tag{1.55}$$

con

$$\boldsymbol{x_{sat}}(\boldsymbol{\theta_{wb}}) = 0.621\,957 \cdot \frac{p_{sat}\big((\boldsymbol{\theta_{wb}})\big)}{p_{atm} - p_{sat}\big((\boldsymbol{\theta_{wb}})\big)} \tag{1.56}$$

cioè, si ha una relazione implicita che va risolta numericamente per iterazione tra una prima stima di θ_{wb} con la (1.55) per $x_{sat} = x$ e poi il suo ricalcolo avendo calcolato la $x_{sat}(\theta_{wb})$ con la (1.56).

In pratica, è però più comodo operare in modo diverso, esplicitando la (1.55) rispetto al termine noto x umidità massica dell'aria soggetta alla trasformazione di saturazione adiabatica, ottenendo quella che vien chiamata l'equazione dello psicrometro, che per la (1.50) risulta essere:

$$x = \frac{c_{p;DA}(\boldsymbol{\theta_{wb}} - \theta_{db}) + \boldsymbol{x_{sat}}(\boldsymbol{\theta_{wb}}) \left[\Delta h_{l\to v}^{(H_2O)_0} + \left(c_{p;H_2O} - c_{p;lw}\right) \cdot \boldsymbol{\theta_{wb}}\right]}{\Delta h_{l\to v}^{(H_2O)_0} + c_{p;H_2O} \cdot \theta_{db} - c_{p;lw}\boldsymbol{\theta_{wb}}} \tag{1.57}$$

La (1.57) consente un più chiaro controllo del processo iterativo consentendo di stimare l'errore di convergenza direttamente con il dato assegnato, cioè

$$err = |x_{stima} - x| = \tag{1.58}$$

dove x_{stima} è calcolato con la (1.57) per i valori stimati di θ_{wb}.

1.2.7.8 Volume specifico in funzione di θ e x (parametrica rispetto a p_{atm})

Il volume specifico si determina tramite la (1.33) come

$$v_{MA} = \frac{R_{DA}T}{p_{atm}}\left(1 + \frac{x}{\varepsilon}\right) = \frac{287.047 \cdot (\theta + 273.15)}{p_{atm}}\left(1 + \frac{x}{0.621\,957}\right) \quad \left[\frac{\mathrm{m}^3}{\mathrm{kg_{DA}}}\right] \tag{1.59}$$

1.2.7.9 Massa volumica in funzione di θ e x (parametrica rispetto a p_{atm})

La massa volumica si determina tramite la (1.34) come

$$\rho_{MA} = \frac{1}{v_{MA}}(1 + x) = \frac{p_{atm}\varepsilon(1 + x)}{R_{DA}T(\varepsilon + x)} = \frac{0.621\,957 \cdot p_{atm} \cdot (1 + x)}{287.047 \cdot (\theta + 273.15) \cdot (0.621\,957 + x)} \quad \left[\frac{\mathrm{kg_{MA}}}{\mathrm{m}^3}\right] \tag{1.60}$$

1.2.7.10 Proprietà termodinamiche dell'aria umida funzioni di θ e ϕ (parametriche rispetto a p_{atm})

La (1.49) consente di determinare l'umidità massica x in funzione di θ e ϕ, e quindi di trasformare tutte le precedenti relazioni di calcolo delle proprietà termodinamiche dell'aria umida funzioni di θ e x in funzioni di θ e ϕ.

1.2.7.11 Proprietà termodinamiche dell'aria umida funzioni di θ_{db} e θ_{wb} (parametriche rispetto a p_{atm})

La (1.57) consente di determinare l'umidità massica x in funzione di θ_{db} e θ_{wb}, e quindi di trasformare tutte le precedenti relazioni di calcolo delle proprietà termodinamiche dell'aria umida funzioni di θ e x in funzioni di θ_{db} e θ_{wb}.

1.2.7.12 Proprietà termodinamiche dell'aria umida funzioni di h_{MA} e x (parametriche rispetto a p_{atm})

Per la determinazione delle altre proprietà occorre determinare la temperatura dalla coppia h_{MA} e x in modo data potere utilizzare le precedenti relazioni di calcolo delle proprietà termodinamiche dell'aria umida funzioni di θ e x.

Dalla (1.50) si ha la soluzione esplicitando direttamente la temperatura come

$$\theta = \frac{h_{MA} - x \cdot \Delta h_{l\to v}^{(H_2O)}}{c_{p;DA} + x \cdot c_{p;H_2O}} \tag{1.61}$$

1.2.7.13 Proprietà termodinamiche dell'aria umida funzioni di h_{MA} e ϕ (parametriche rispetto a p_{atm})

Per la determinazione delle altre proprietà occorre determinare la temperatura dalla coppia h_{MA} e ϕ in modo data potere utilizzare le precedenti relazioni di calcolo delle proprietà termodinamiche dell'aria umida funzioni di θ e ϕ.

Per ricavare la temperatura occorre risolvere numericamente rispetto alla temperatura θ l'equazione trascendente che si ottiene sostituendo la (1.49) nella (1.50) , cioè:

$$h_{MA} = c_{p;DA} \cdot \theta + \left[\Delta h_{l\to v}^{(H_2O)} + c_{p;H_2O} \cdot \theta\right] \cdot \varepsilon \cdot \frac{\phi \cdot p_{sat}(\theta)}{p_{atm} - \phi \cdot p_{sat}(\theta)} \tag{1.62}$$

1.2.8 Calcolo dello stato finale di alcune trasformazioni dell'aria umida

È utile introdurre nella libreria dell'aria umida alcuni delle trasformazioni termodinamiche che vengono realizzate nei sistemi di condizionamento dell'aria (trasformazioni isobare con temperatura e umidità massica variabili).

1.2.8.1 Trasformazione di miscelazione isobara adiabatica

La trasformazione di miscelazione isobara adiabatica è la più semplice delle trasformazioni a cui può essere soggetta un'aria umida in impianto d climatizzazione. Per due correnti d'aria di portate massiche d'aria secca $\dot{m}_1$ e $\dot{m}_2$, con riferimento a Figura 1, lo stato finale della miscela è dato da:

$$h_{MA;M} = \frac{\dot{m}_1}{\dot{m}_1 + \dot{m}_2} h_{MA;1} + \frac{\dot{m}_2}{\dot{m}_1 + \dot{m}_2} h_{MA;2} \tag{1.63}$$

$$x_M = \frac{\dot{m}_1}{\dot{m}_1 + \dot{m}_2} x_1 + \frac{\dot{m}_2}{\dot{m}_1 + \dot{m}_2} x_2 \tag{1.64}$$

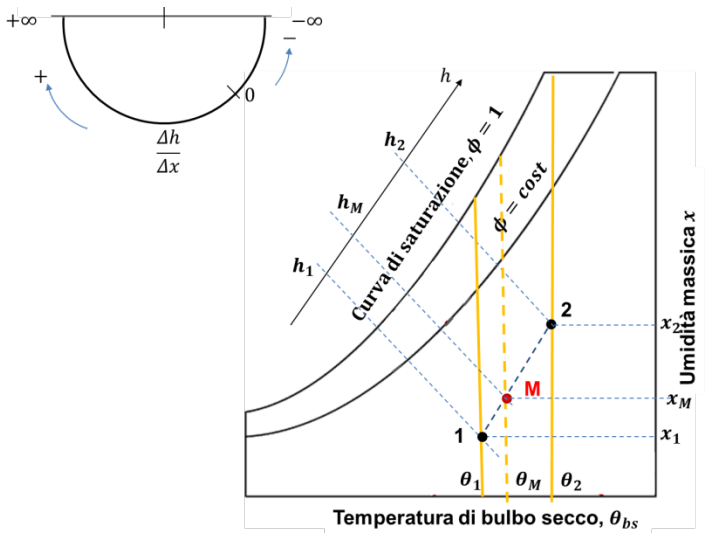


Figura 1 – Trasformazione miscelazione adiabatica di due arie umide

Se vi sono più di due flussi d'aria umida che si miscelano le (1.69)e (1.70) possono essere generalizzato come:

$$h_{MA;M} = \sum_{i=1}^{N} \frac{\dot{m}_i}{\dot{m}_{tot}} h_{MA;i} \tag{1.65}$$

$$x_M = \sum_{i=1}^{N} \frac{\dot{m}_i}{\dot{m}_{tot}} x_i \tag{1.66}$$

con

$$\dot{m}_{tot} = \sum_{i=1}^{N} \dot{m}_i \tag{1.67}$$

dove N è il numero di flussi d'aria. Nota $h_{MA;M}$ si può determinare la temperatura dello stato finale come:

$$\theta_2 = \frac{h_{MA;2} - x_2 \cdot \Delta h_{l\to v}^{(H_2O)_0}}{c_{p;DA} + x_2 \cdot c_{p;H_2O}} \tag{1.68}$$

1.2.8.2 È Trasformazione isobara di umidificazione e di saturazione adiabatica con acqua liquida

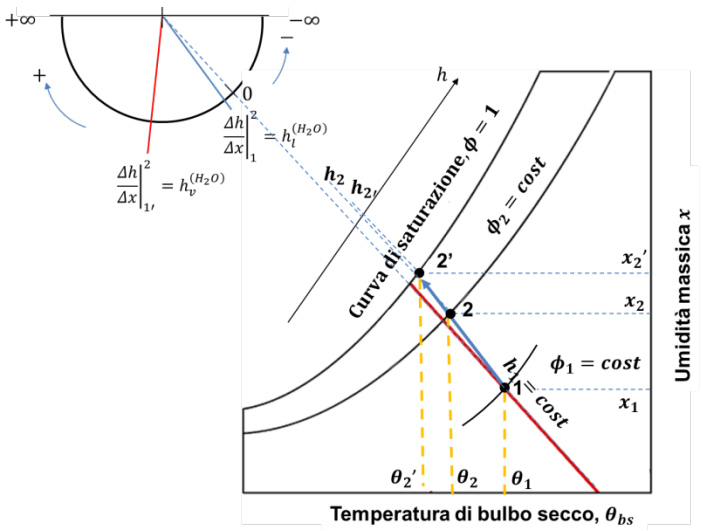


Figura 2 – Trasformazione di saturazione adiabatica di un'aria umida

Nella trasformazione di umidificazione dell'aria tramite saturatore adiabatico, cioè con nebulizzazione di acqua liquida anche in eccesso rispetto quella teoricamente necessaria per ottenere la completa saturazione dell'aria, lo stato finale è definito, con riferimento a Figura 2, dall'equazione:

$$h_{MA;2} = h_{MA;1} + \epsilon_s[x_{2'} - x_1] \cdot h_{lw}(\theta_{lw}) \tag{1.69}$$

dove

$\epsilon_s = \frac{x_2 - x_1}{x_{2'} - x_1}$ efficienza di saturazione, grandezza caratteristica del singolo apparato

$h_{lw}(\theta_{lw})$ entalpia specifica dell'acqua liquida nebulizzata, funzione della sua temperatura θ_{lw}.

Lo stato finale è determinato dell'intersezione della retta di umidificazione con la curva di saturazione e, successivamente dall'applicazione dell'efficienza di saturazione.

Per la completa saturazione si ha:

$$x_{2'} = x_{sat}(\boldsymbol{\theta_{2'}}) = \frac{h_{MA;1} - c_{p;DA}\boldsymbol{\theta_{2'}} - c_{p;lw} \cdot \theta_{lw}}{\Delta h_{l\to v}^{(H_2O)_0} + c_{p;H_2O} \cdot \boldsymbol{\theta_{2'}} - c_{p;lw}\theta_{lw}} \tag{1.70}$$

che insieme con la

$$x_{sat}(\boldsymbol{\theta_{2'}}) = 0.621\,957 \cdot \frac{p_{sat}(\boldsymbol{\theta_{2'}})}{p_{atm} - p_{sat}(\boldsymbol{\theta_{2'}})} \tag{1.71}$$

rappresenta l'equazione implicita in $(\boldsymbol{\theta_{2'}})$, che consente di determinare lo stato finale di saturazione. Risulta però più efficiente determinare la soluzione mettendo in evidenza il termine noto, l'entalpia specifica dello stato iniziale $h_{MA;1}$, come:

$$h_{MA;1} = c_{p;DA}\boldsymbol{\theta_{2'}} + x_{sat}(\boldsymbol{\theta_{2'}}) \cdot \left[\Delta h_{l\to v}^{(H_2O)_0} + c_{p;H_2O} \cdot \boldsymbol{\theta_{2'}}\right] - [x_{sat}(\boldsymbol{\theta_{2'}}) - x_1] \cdot c_{p;lw}\theta_{lw} \tag{1.72}$$

così da poter definire l'errore di convergenza sul dato iniziale come:

$$err = \left|h_{MA;1;stima} - h_{MA;1}\right| \tag{1.73}$$

Nota la temperatura di saturazione $\theta_{2'}$ è nota tramite la (1.71) l'umidità massica a saturazione, $x_{sat}(\theta_{2'})$, e quindi, tramite la (1.51), l'entalpia specifica di saturazione, $h_{MA;2'}$.

Note le proprietà nelle condizioni di saturazione si utilizza il dato relativo allo specifico apparato, l'efficienza di saturazione ϵ_s, per determinare lo stato finale della trasformazione di umidificazione, cioè

$$x_2 = x_1 + \epsilon_s \cdot [x_{sat}(\theta_{2'}) - x_1] \tag{1.74}$$

e quindi, essendo nota per la (1.69) l'entalpia specifica $h_{MA;2}$, si può determinare la temperatura dello stato finale come:

$$\theta_2 = \frac{h_{MA;2} - x_2 \cdot \Delta h_{l\to v}^{(H_2O)_0}}{c_{p;DA} + x_2 \cdot c_{p;H_2O}} \tag{1.75}$$

Quindi lo stato finale $\{\theta_2, x_2\}$ può essere visto come una funzione a due valori dello stato iniziale $\{\theta_1, x_1\}$, parametrica rispetto all'efficienza di saturazione e alla pressione ambientale, cioè:

$$\{\theta_2; x_2\} = f_{hum}(\{\theta_1, x_1\};\ \epsilon_s;\ p_{atm}) \tag{1.76}$$

1.2.8.3 Trasformazione isobara di umidificazione adiabatica con acqua liquida o vapore

Nel caso di una trasformazione adiabatica che non preveda l'impiego del saturatore adiabatico, il livello di umidificazione desiderato è ottenuto tramite controllo retroattivo della l'umidità relativa ottenuta a termine della trasformazione per modulazione della quantità d'acqua nebulizzata o di vapore iniettato.

In questo caso, la trasformazione contenuta nella libreria assume come dati noti le condizioni iniziali dell'aria umida, la quantità di acqua iniettata e il suo stato termodinamico, determinando così il relativo stato finale, che dovrà poi essere soggetto alla verifica del sistema di controllo adottato nell'apparato di umidificazione.

Si ha quindi

$$h_{MA;2} = h_{MA;1} + (x_2 - x_1) \cdot h_{w;in} \tag{1.77}$$

con

$$x_2 = x_1 + \dot{m}_{w;in}/\dot{m}_{DA} \tag{1.78}$$

da cui

$$\theta_2 = \frac{h_{MA;1} - x_2 \cdot \Delta h_{l\to v}^{(H_2O)_0} + (x_2 - x_1) \cdot h_{w;in}}{c_{p;DA} + x_2 \cdot c_{p;H_2O}} \tag{1.79}$$

dove

$h_{w;in}$ è l'entalpia specifica dell'acqua introdotta, liquida o vapore;

$\dot{m}_{w;in}$ è la portata d'acqua introdotta nell'ipotesi che sia tutta assorbita dall'aria;

$\dot{m}_{DA}$ è la portata dell'aria umida trattata.

1.2.8.4 Trasformazione di raffreddamento con deumidificazione

Una trasformazione caratteristica degli impianti di climatizzazione è quella di raffreddamento con contemporanea deumidificazione dell'aria trattata. In questo caso lo stato finale dell'aria trattata dipende, a

parità di stato iniziale, dalle caratteristiche dell'apparato che la tratta, solitamente una batteria alettata, che sono definite tramite la sua efficienza, ϵ_c, o il by-pass factor BF, che sono legati dalla equazione:

$$\epsilon_c = 1 - BF \tag{1.80}$$

dove il fattore di by-pass è definito, con riferimento a Figura 3, come

$$BF = \frac{h_{MA;2} - h_{MA;R}}{h_{MA;1} - h_{MA;R}} = \frac{x_2 - x_R}{x_1 - x_R} \tag{1.81}$$

Lo stato finale non si troverà mai sulla curva di saturazione, essendo non realizzabile un'efficienza unitaria, ma, con riferimento Figura 3, nel punto (2) sulla retta congiungente il punto di rugiada della batteria (R) e lo stato iniziale dell'aria trattata (1).

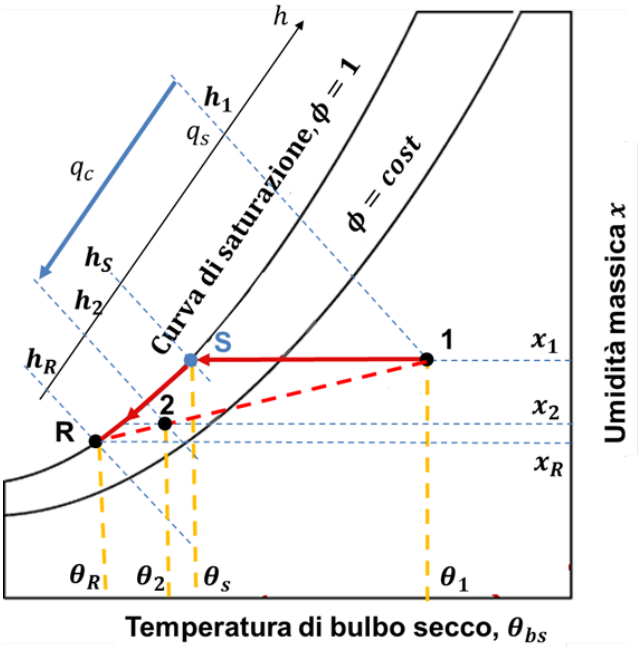


Figura 3 – Trasformazione raffreddamento con deumidificazione di un'aria umida

Si possono verificare due diverse condizioni:

a) si vuole simulare il comportamento di una batteria alettata, e quindi è nota l'efficienza della batteria alettata ϵ_c , cioè il by-pass factor BF, e la temperatura a cui si fa funzionare, cioè la temperatura di rugiada dell'apparato, θ_R;

b) vi vuole dimensionare il sistema e quindi è nota la temperatura di rugiada dell'apparato, θ_R, e le condizioni di umidità massica dello stato finale desiderato, x_2.

Nel caso a) si determina l'umidità massica nelle condizioni del punto di rugiada dell'apparato come

$$x_R = x_{sat}(\theta_R) = 0.621\,957 \cdot \frac{p_{sat}(\theta_R)}{p_{atm} - p_{sat}(\theta_R)} \tag{1.82}$$

Nota la temperatura θ_R e l'umidità massica x_R si può determinate l'entalpia specifica $h_{MA;R}$ tramite la (1.50) e quindi tramite la (1.81) sia l'umidità massiche che l'entalpia specifica dello stato finale, come:

$$x_2 = BF \cdot x_1 + (1 - BF) \cdot x_R \tag{1.83}$$

$$h_{MA;2} = BF \cdot h_{MA;1} + (1 - BF) \cdot h_{MA;R} \tag{1.84}$$

e dalla seconda la temperatura come:

$$\theta_2 = \frac{h_{MA;2} - x_2 \cdot \Delta h_{l\to v}^{(H_2O)_0}}{c_{p;DA} + x_2 \cdot c_{p;H_2O}} \tag{1.85}$$

Nel caso b) essendo nota (θ_R) e il fattore di by-pass dell'apparato è incognito, **si sta dimensionando il sistema** e quindi è nota almeno l'umidità massica nella condizione finale, cioè x_2, e quindi il by-pass factor della batteria è dato da:

$$BF = \frac{x_2 - x_{sat}(\theta_R)}{x_1 - x_{sat}(\theta_R)} \tag{1.86}$$

Nella libreria è implementata la sola condizione a) che consiste in una funzione a due valori dello stato iniziale $\{\theta_1, x_1\}$ e della temperatura di rugiada dell'apparatoθ_R, parametrica rispetto fattore di by-pass e alla pressione ambientale, cioè:

$$\{\theta_2; x_2\} = f_{dehum}(\{\theta_1, x_1\}, \theta_R;\ BF;\ p_{atm}) \tag{1.87}$$

1.2.9 Acqua

Per la definizione delle caratteristiche termofisiche dell'acqua nei suoi vari stati di aggregazione (solido, liquido e aeriforme) si fa principalmente riferimento alla "International Association for the Properties of Water and Steam", una associazione internazionale senza scopo di lucro di organizzazioni nazionali che si occupano delle proprietà dell'acqua e del vapore, in particolare proprietà termofisiche, linee guida sulla chimica dei cicli e altri aspetti del vapore ad alta temperatura, dell'acqua e delle miscele acquose rilevanti per i cicli di energia termica e altre applicazioni industriali e scientifiche.

1.2.9.1 Quantità termofisiche di riferimento

Le principali quantità termofisiche di riferimento sono [24]:

Massa Molare:

- $M = 518.015\,268\ \ \mathrm{g/mol}$ massa molare

Costante Specifica di Gas Ideale:

- $R = 0.461\,518\,05\ \mathrm{kJ/(kg\,K)}$ costante dei gas ideale specifica per l'acqua

Costanti Critiche

- $T_c = 647.096\ \ \mathrm{K}$ temperatura del punto critico
- $p_c = 22.064\ \ \mathrm{MPa}$ pressione del punto critico
- $\rho_c = 322\ \ \mathrm{kg/m^3}$ massa volumica del punto critico

Punto Triplo

- $T_t = 273.16\ \ \mathrm{K}$ temperatura
- $p_t = 611.657\ \ \mathrm{Pa}$ pressione

- $\rho_t' = 999.79 \ \ \mathrm{kg/m^3}$ massa volumica del liquido al punto triplo
- $\rho_t'' = 0.004\,854\,58 \ \ \mathrm{kg/m^3}$ massa volumica del vapore al punto triplo

Punto di Ebollizione Normale (a pressione atmosferica)

- $T_b = 373.124 \ \ \mathrm{K}$ temperatura
- $p_b = 0.101\,325 \ \ \mathrm{MPa}$ pressione
- $\rho_b' = 958.367 \ \ \mathrm{kg/m^3}$ massa volumica del liquido
- $\rho_{tb}'' = 0.597\,657 \ \ \mathrm{kg/m^3}$ massa volumica del vapore

Stato di riferimento: Stato liquido al punto triplo

- $u_t' = 0 \ \ \mathrm{J/kg}$ energia interna specifica del liquido
- $s_t' = 0 \ \ \mathrm{J/(kg \cdot K)}$ entropia specifica del liquido
- $h_t' = 0.611\,782 \ \ \mathrm{J/kg}$ entalpia specifica del liquido

Proprietà a pressione atmosferica: $p_0 = 0.101\,325 \ \mathrm{MPa}$

- $\rho_{max} = 999.974\,95 \ \ \mathrm{kg/m^3}$ massa volumica massima dell'acqua a $\theta \cong 4 \ ^\circ\mathrm{C}$
- $\mu_0 = \ 1.0016 \ \ \mathrm{mPa \cdot s}$ viscosità dinamica a $T_0 = 273.15 \ \ \mathrm{K}$

Momento di Dipolo:

- $\mathbf{p} = 6.1875(3) \cdot 10^{-30} \ \ \mathrm{C \cdot m}$ momento di dipolo elettrico

Polarizzabilità statica totale media del dipolo

- $\alpha = \ 1.49(1) \cdot 10^{-24} \ \ \mathrm{cm^3}$

2 Bilancio entalpico per il calcolo dell'umidità dell'aria

2.1 Bilancio energetico e di massa

Il metodo di calcolo dello stato termodinamico della zona termica dell'edificio in OpenBPS è basato su un'equazione di bilancio entalpico integrale, in cui l'aria della zona viene modellata in base all'ipotesi di miscelamento perfetto, ovvero con l'assunzione di proprietà uniformi in tutto lo spazio della zona. La struttura orientata agli oggetti di OpenBPS consente ovviamente calcoli più dettagliati del semplice impiego del modello basato sui coefficienti di scambio termico convettivo per descrivere i processi convettivi, come i metodi CFD o zonali, che possono essere aggiunti in futuro sviluppo. Nello sviluppo iniziale del codice di calcolo è stata implementata solo la parte "sensibile" del bilancio entalpico integrale, che costituisce la cosiddetta equazione differenziale ordinaria del "nodo aria", qui semplicemente definita oggetto `Air-Node`.

L'oggetto `Air-Node` gestisce i vari flussi di massa come l'aria di ventilazione, l'aria di espulsione e le infiltrazioni, oltre allo scambio termico convettivo con le superfici della stanza (pareti, finestre, soffitti e pavimenti), assunte avere temperatura superficiale uniforme. Tiene conto della capacità termica dell'aria ambiente e valuta direttamente i guadagni termici convettivi dovuti a persone e apparecchiature. Lo scambio termico attraverso ciascun componente dalla struttura dell'edificio determina poi le temperature delle superfici della zona utilizzate nell'oggetto `Air-Node` per calcolare lo scambio termico convettivo.

In questo terzo anno della ricerca si è ritenuto fondamentare estendere le capacità del codice passando da una classe (oggetto quando istanziato) `Air-Node` in grado di operare solo sulla determinazione della temperatura e quindi dei carichi termici e energia "sensibile" (aria secca) a una classe estesa `Air-Node` che invece consideri l'aria umida, cioè una miscela di aria secca e vapore d'acqua, e quindi in grado di operare anche sull'umidità e sulla determinazione dei carichi e energia "latente".

2.1.1 Equazione di bilancio entalpico per l'aria umida

L'equazione che determina lo stato energetico di un'aria umida in un volume (zona termica) è, considerate accettabili le **ipotesi** di **dissipazioni interne nell'aria dovute ai gradienti di velocità trascurabili e processi quasi ovunque isobari**, è data da [25]:

$$\frac{dH^{(AU)}(t)}{dt} = \sum_{j=1}^{N_{in/out}} \Phi_{H,j}^{(ad,AU)} + \sum_{k=1}^{N_{\sigma}} \Phi_{H,k}^{(H_2O)} + \sum_{i=1}^{N_{ss}} \Phi_{T,i} \tag{2.1}$$

dove

$H^{(AU)}(t) = \int_V \rho^{(AS)} h^{(AU)} dV$ entalpia dell'ara umida nel volume V;

$\Phi_{H,j}^{(ad,AU)} = -\int_{S_j} \rho^{(AS)} h^{(AU)} \vec{v} \cdot \vec{n}\, dS$ flusso entalpico avvettivo di aria umida attraverso la superficie *j*;

$\Phi_{H,k}^{(H_2O)} = \dot{M}_{H_2O,k} h_k^{(H_2O)}$ flusso entalpico dovuto a transizione liquido vapore e viceversa;

$\Phi_{T,i}$ flusso termico attraverso le superfici di contorno solide.

Utilizzando il teorema della media integrale e considerando trascurabile la covarianza delle grandezze in gioco, la (2.1) si può riscrivere come:

$$\langle h^{(AU)}\rangle_V V \frac{d\langle \rho^{(AS)}\rangle_V}{dt} + V\langle \rho^{(AS)}\rangle_V \frac{d}{dt}\langle h^{(AU)}\rangle_V = \sum_{j=1}^{N_{in}} \dot{m}_{in_j}^{(AS)} \langle h^{(AU)}\rangle_{S_j} - \sum_{j=1}^{N_{out}} \dot{m}_{out_j}^{(AS)} \langle h^{(AU)}\rangle_{S_j} + \sum_{k=1}^{N_\sigma} \dot{M}_{H_2O,k} h_k^{(H_2O)} + \sum_{i=1}^{N_{ss}} \Phi_{T,i} \quad (2.2)$$

dove

$\langle h^{(AU)}\rangle_V$ entalpia specifica dell'aria umida, valore medio di volume;

$\langle h^{(AU)}\rangle_{S_j}$ entalpia specifica dell'aria umida, valore medio di superficie di ingresso o uscita;

$h_k^{(H_2O)}$ entalpia specifica dell'acqua immessa ($\dot{M}_{H_2O,k} > 0$) o estratta ($\dot{M}_{H_2O,k} < 0$) da volume *V*;

$\langle \rho^{(AS)}\rangle_V$ massa volumica dell'aria secca, valore medio di volume (non necessariamente costante);

$\dot{m}_{in_j}^{(AS)}$ portata d'aria secca in ingresso dalla superficie S_j;

$\dot{m}_{out_j}^{(AS)}$ portata d'aria secca in uscita dalla superficie S_j;

$\dot{M}_{H_2O,k}$ portata d'acqua immessa o estratta dal volume d'aria *V*;

Applicando **l'ipotesi** di **miscelamento perfetto**, $\langle h^{(AU)}\rangle_{S_{j,out}} \equiv \langle h^{(AU)}\rangle_V \quad \forall\, j$, per cui si ha:

$$\sum_{j=1}^{N_{out}} \dot{m}_{out_j}^{(AS)} \langle h^{(AU)}\rangle_{S_j} = \langle h^{(AU)}\rangle_V \sum_{j=1}^{N_{out}} \dot{m}_{out_j}^{(AS)} \quad (2.3)$$

e utilizzando il **principio di conservazione della massa** del componente aria secco in forma integrale, espresso come:

$$\sum_{j=1}^{N_{out}} \dot{m}_{out_j}^{(AS)} = \sum_{j=1}^{N_{in}} \dot{m}_{in_j}^{(AS)} - V\frac{d\langle \rho^{(AS)}\rangle_V}{dt} \quad (2.4)$$

si può eliminare dall'equazione (2.2) la derivata temporale del valor medio di volume della massa volumica dell'aria secca **mantenendo l'ipotesi di comprimibilità della miscela aria umida**, ottenendo:

$$V\langle \rho^{(AS)}\rangle_V \frac{d}{dt}\langle h^{(AU)}\rangle_V = \sum_{j=1}^{N_{in}} \dot{m}_{in_j}^{(AS)} \left(\langle h^{(AU)}\rangle_{S_j} - \langle h^{(AU)}\rangle_V\right) + \sum_{k=1}^{N_\sigma} \dot{M}_{H_2O,k} h_k^{(H_2O)} + \sum_{i=1}^{N_{ss}} \Phi_{T,i} \quad (2.5)$$

Introducendo l'equazione costitutiva entalpica per l'aria umida,

$$h^{(AU)}(\theta, x) = h^{(AS)} + x\, h^{(H_2O)} = c_p^{(AS)}\theta + x\left[\Delta h_{l\to v}^{(H_2O)_0} + c_p^{(H_2O)}\theta\right] \quad (2.6)$$

e considerando:

- trascurabile la covarianza del prodotto $\theta \cdot x$,
- il flusso termico tra aria umida e superfici solide puramente convettivo e pari a:

$$\Phi_{T,i} = A_i \langle h_{cv} \rangle_{S_i} \left(\theta_{s_i} - \langle\theta\rangle_V\right) \tag{2.7}$$

- l'entalpia specifica dell'acqua introdotta/estratta esprimibile come differenza al suo valore medio di volume, cioè:

$$h_k^{(H_2O)} = \left\langle h_v^{(H_2O)} \right\rangle_V + \left[h_k^{(H_2O)} - \left\langle h_v^{(H_2O)} \right\rangle_V\right] = h_v^{(H_2O)}\{\langle\theta\rangle_V\} + \Delta h_k^{(H_2O)}(\theta_k) \tag{2.8}$$

la (2.5) si può riscrivere come:

$$\begin{aligned} &V\left\langle \rho^{(AS)} \right\rangle_V c_p^{(AU)}\{\langle x\rangle_V\}\frac{d\langle\theta\rangle_V}{dt} + h_v^{(H_2O)}\{\langle\theta\rangle_V\}\left[V\left\langle \rho^{(AS)} \right\rangle_V \frac{d\langle x\rangle_V}{dt}\right] = \\ &\sum_{j=1}^{N_{in}} \dot{m}_{in_j}^{(AS)} c_p^{(AU)} \left\{\langle x\rangle_{S_j}\right\}\left(\langle\theta\rangle_{S_j} - \langle\theta\rangle_V\right) + \sum_{i=1}^{N_{ss}} A_i \langle h_{cv}\rangle_{S_i}\left(\theta_{s_i} - \langle\theta\rangle_V\right) + \\ &h_v^{(H_2O)}\{\langle\theta\rangle_V\}\left[\sum_{j=1}^{N_{in}} \dot{m}_{in_j}^{(AS)}\left(\langle x\rangle_{S_j} - \langle x\rangle_V\right) + \sum_{k=1}^{N_\sigma} \dot{M}_{H_2O,k}\right] + \sum_{k=1}^{N_\sigma} \dot{M}_{H_2O,k}\, \Delta h_k^{(H_2O)}(\theta_k) \end{aligned} \tag{2.9}$$

con

- $\Delta h_k^{(H_2O)}(\theta_k) < 0$ se si introduce acqua liquida che vaporizza o estrae vapore condensato;
- $\Delta h_k^{(H_2O)}(\theta_k) \lessgtr 0$ se si introduce o estrae direttamente vapore d'acqua con $\theta_k \lessgtr \langle\theta\rangle_V$.

La deviazione entalpica per introduzione/estrazione di vapore d'acqua è pari a

$$\Delta h_k^{(H_2O)}(\theta_k) = c_p^{(H_2O)} \cdot (\theta_k - \langle\theta\rangle_V) \tag{2.10}$$

cioè corrisponde ad un semplice surriscaldamento o sottoraffreddamento del vapore introdotto/estratto rispetto a valore medio di volume; mentre se si considera acqua liquida si ha

$$\Delta h_k^{(H_2O)}(\theta_k) = c_l^{(H_2O)} \cdot \theta_k - c_p^{(H_2O)}\langle\theta\rangle_V - \Delta h_{l\to v}^{(H_2O)_0} \tag{2.11}$$

Generalizzando, si può scrivere la deviazione entalpica come costituita da due parti:

$$\begin{gathered} \Delta h_k^{(H_2O)}(\theta_k, y) = \left(c_{y,k}^{(H_2O)} \cdot \theta_k - c_p^{(H_2O)}\langle\theta\rangle_V\right) - \left(\delta_k \cdot \Delta h_{l\to v}^{(H_2O)_0}\right) \\ if\{y = p \to \delta_k = 1\}\, elseif\{y = l \to \delta_k = 0\} \end{gathered} \tag{2.12}$$

cioè:

$$\Delta h_k^{(H_2O)}(\theta_k, y) = \Delta h_{k,s}^{(H_2O)}(\theta_k, y) - \left(\delta_k \cdot \Delta h_{l\to v}^{(H_2O)_0}\right) \tag{2.13}$$

L'equazione di bilancio entalpico (2.9) può quindi essere separata in due distinte equazioni differenziali ordinarie, una funzione esplicita solo della variabile temperatura media di volume, la seconda funzione esplicita della sola umidità massica media di volume; tali equazioni risultano solo accoppiate indirettamente tramite la dipendenza dei rispettivi termini di accumulo, dove nel primo caso la capacità termica specifica dell'aria umida dipende dal valore medio di volume dell'umidità massica, mentre nel secondo calo l'entalpia specifica del vapore d'acqua è valutata alla temperatura media di volume dell'aria umida. Tali equazioni prendono convenzionalmente rispettivamente il nome di equazione di **bilancio entalpico sensibile**, equazione (2.14),

$$V\langle\rho^{(AS)}\rangle_V c_p^{(AU)}\{\langle x\rangle_V\}\frac{d\langle\theta\rangle_V}{dt}$$
$$= \sum_{j=1}^{N_{in}} \dot{m}_{in_j}^{(AS)} c_p^{(AU)} \left\{\langle x\rangle_{S_j}\right\}\left(\langle\theta\rangle_{S_j} - \langle\theta\rangle_V\right) + \sum_{i=1}^{N_{ss}} A_i\langle h_{cv}\rangle_{S_i}\left(\theta_{S_i} - \langle\theta\rangle_V\right) \quad (2.14)$$
$$+ \sum_{k=1}^{N_\sigma} \dot{M}_{H_2O,k}\,\Delta h_{k,s}^{(H_2O)}(\theta_k, y)$$

e di **bilancio entalpico latente**, equazione (2.15):

$$h_v^{(H_2O)}\{\langle\theta\rangle_V\}\left[V\langle\rho^{(AS)}\rangle_V \frac{d\langle x\rangle_V}{dt}\right] =$$
$$h_v^{(H_2O)}\{\langle\theta\rangle_V\}\left[\sum_{j=1}^{N_{in}} \dot{m}_{in_j}^{(AS)}\left(\langle x\rangle_{S_j} - \langle x\rangle_V\right) + \sum_{k=1}^{N_\sigma}\dot{M}_{H_2O,k}\right] - \Delta h_{l\to v}^{(H_2O)_0}\sum_{k=1}^{N_\sigma}\dot{M}_{H_2O,k}\,\delta_k \quad (2.15)$$

Nell'equazione (2.15) i termini tra parentesi quadre rappresentano i termini del bilancio di massa del vapore d'acqua contenuto nell'aria umida che occupa il volume *V*. Si può notare quindi come il bilancio entalpico latente coincida con il bilancio di massa del vapore moltiplicato l'entalpia specifica del vapore, valutata alla temperatura media di volume, a meno della sola deviazione entalpica "latente", che risulta sulla se si inietta o sottrae direttamente solo vapore.

2.1.2 Equazioni di bilancio entalpico sensibile e latente

Per l'integrazione delle due equazioni di bilancio entalpico, conviene operare una separazione dei vari termini in funzione dei diversi "sistemi" interagenti con l'aria; questi sono:

- *ENV*: le partizioni del fabbricato che identificano il volume della zona (pareti interne, esterne, solai, e le aperture in queste presenti);
- *SYS*: i sistemi impiantistici per la climatizzazione ambientale presenti nella zona;
- *P&C*: le persone e le cose che occupano la zona.

2.1.2.1 Equazione di bilancio entalpico sensibile

Posto per i termini avvettivi, il numero totale di sezioni di ingresso, N_{in}, pari a

$$N_{in} = N_{in}^{(ENV)} + N_{in}^{(SYS)} + N_{in}^{(P\&C)} \quad (2.16)$$

la sommatoria dei **flussi avvettivi** può essere riscritta come:

$$\sum_{j=1}^{N_{in}}\left[\dot{m}_{in_j}^{(AS)} c_p^{(AU)}\left\{\langle x\rangle_{S_j}\right\}\left(\langle\theta\rangle_{S_j} - \langle\theta\rangle_V\right)\right] = \sum_{j=1}^{N_{in}^{(ENV)}}[\,] + \sum_{j=1}^{N_{in}^{(SYS)}}[\,] + \sum_{j=1}^{N_{in}^{(P\&C)}}[\,]$$
$$= \sum_{j=1}^{N_{in}^{(ENV)}} \dot{m}_{in_j}^{(AS)} c_p^{(AU)}\left\{\langle x\rangle_{S_j}\right\}\left(\langle\theta\rangle_{S_j} - \langle\theta\rangle_V\right) + \dot{Q}_{SYS}^{(AD,S)} + \dot{Q}_{P\&C}^{(AD,S)} \quad (2.17)$$

con

$\dot{Q}_{SYS}^{(AD,S)}$ termine sorgente advettivo sensibile impianto;

$\dot{Q}_{P\&C}^{(AD,S)}$ termine sorgente advettivo sensibile persone e cose

Posto per i termini convettivi, il numero totale di superfici di scambio termico, N_{ss}, pari a

$$N_{ss} = N_{ss}^{(ENV)} + N_{ss}^{(SYS)} + N_{ss}^{(P\&C)} \tag{2.18}$$

la sommatoria dei **flussi convettivi** può essere riscritta come:

$$\begin{aligned}\sum_{i=1}^{N_{ss}} A_i \langle h_{cv} \rangle_{S_i} \left(\theta_{s_i} - \langle\theta\rangle_V\right) &= \sum_{i=1}^{N_{ss}^{(ENV)}} [\,] + \sum_{i=1}^{N_{ss}^{(SYS)}} [\,] + \sum_{i=1}^{N_{ss}^{(P\&C)}} [\,] \\ &= \sum_{i=1}^{N_{ss}^{(ENV)}} A_i \langle h_{cv} \rangle_{S_i} \left(\theta_{s_i} - \langle\theta\rangle_V\right) + \dot{Q}_{SYS}^{(CV,S)} + \dot{Q}_{P\&C}^{(CV,S)}\end{aligned} \tag{2.19}$$

con

$\dot{Q}_{SYS}^{(CV,S)}$ termine sorgente convettivo sensibile impianto;

$\dot{Q}_{P\&C}^{(CV,S)}$ termine sorgente convettivo sensibile persone e cose

Posto per i termini di deviazione entalpica, il numero totale sorgenti o pozzi, N_σ, pari a

$$N_\sigma = N_\sigma^{(ENV)} + N_\sigma^{(SYS)} + N_\sigma^{(P\&C)} \tag{2.20}$$

la sommatoria dei **flussi dovuti al contributo sensibile per introduzione/estrazione d'acqua** può essere riscritta come:

$$\begin{aligned}\sum_{k=1}^{N_\sigma} \left[\dot{M}_{H_2O,k} \Delta h_{k,s}^{(H_2O)}(\theta_k, y)\right] &= \sum_{k=1}^{N_\sigma^{(ENV)}} [\,] + \sum_{k=1}^{N_\sigma^{(SYS)}} [\,] + \sum_{k=1}^{N_\sigma^{(P\&C)}} [\,] \\ &= \sum_{k=1}^{N_\sigma^{(ENV)}} \dot{M}_{H_2O,k} \Delta h_{k,s}^{(H_2O)}(\theta_k, y) + \dot{Q}_{SYS}^{(MS,S)} + \dot{Q}_{P\&C}^{(MS,S)}\end{aligned} \tag{2.21}$$

con

$\dot{Q}_{SYS}^{(MS,S)}$ termine sorgente sensibile dovuto a introduzione/estrazione di acqua da parte dell'impianto;

$\dot{Q}_{P\&C}^{(MS,S)}$ termine sorgente sensibile dovuto a introduzione/estrazione di acqua di persone e cose.

L'equazione di bilancio entalpico sensibile può quindi essere riscritta mettendo in evidenza solo le interazioni che l'aria ha con l'involucro edilizio, raggruppando invece le interazioni con gli altri sistemi (impianto e persone e cose) nei termini sorgente/pozzo $\dot{Q}$; cioè:

$$\begin{aligned}&V\langle\rho^{(AS)}\rangle_V c_p^{(AU)} \{\langle x\rangle_V\} \frac{d\langle\theta\rangle_V}{dt} \\ &= \sum_{j=1}^{N_{in}^{(ENV)}} \dot{m}_{in_j}^{(AS)} c_p^{(AU)} \left\{\langle x\rangle_{S_j}\right\} \left(\langle\theta\rangle_{S_j} - \langle\theta\rangle_V\right) + \sum_{i=1}^{N_{ss}^{(ENV)}} A_i \langle h_{cv} \rangle_{S_i} \left(\theta_{s_i} - \langle\theta\rangle_V\right) \\ &+ \sum_{k=1}^{N_\sigma^{(ENV)}} \dot{M}_{H_2O,k} \Delta h_{k,s}^{(H_2O)}(\theta_k, y) + \dot{Q}_{SYS}^{(S)} + \dot{Q}_{P\&C}^{(S)}\end{aligned} \tag{2.22}$$

avendo posto

$$\dot{Q}_{SYS}^{(S)} = \dot{Q}_{SYS}^{(AD,S)} + \dot{Q}_{SYS}^{(CV,S)} + \dot{Q}_{SYS}^{(MS,S)} \tag{2.23}$$

$$\dot{Q}_{P\&C}^{(S)} = \dot{Q}_{P\&C}^{(AD,S)} + \dot{Q}_{P\&C}^{(CV,S)} + \dot{Q}_{P\&C}^{(MS,S)} \tag{2.24}$$

2.1.2.2 Equazione di bilancio entalpico latente

Usando per i termini avvettivi la suddivisione definita dalla (2.16) si ha che la sommatoria dei **flussi avvettivi** può essere riscritta come:

$$\begin{aligned} h_v^{(H_2O)}\{\langle\theta\rangle_V\}\sum_{j=1}^{N_{in}} \dot{m}_{in_j}^{(AS)}\left(\langle x\rangle_{S_j} - \langle x\rangle_V\right) &= \sum_{j=1}^{N_{in}^{(ENV)}}[\,] + \sum_{j=1}^{N_{in}^{(SYS)}}[\,] + \sum_{j=1}^{N_{in}^{(P\&C)}}[\,] \\ = h_v^{(H_2O)}\{\langle\theta\rangle_V\}\sum_{j=1}^{N_{in}^{(ENV)}} \dot{m}_{in_j}^{(AS)}\left(\langle x\rangle_{S_j} - \langle x\rangle_V\right) &+ \dot{Q}_{SYS}^{(AD,L)} + \dot{Q}_{P\&C}^{(AD,L)} \end{aligned} \tag{2.25}$$

con

$\dot{Q}_{SYS}^{(AD,L)}$ termine sorgente avvettivo latente impianto;

$\dot{Q}_{P\&C}^{(AD,L)}$ termine sorgente avvettivo latente persone e cose

Usando, per i termini di introduzione/estrazione d'acqua con entalpia specifica pari a quella media di volume, la suddivisione definita dalla (2.20) si ha che la sommatoria dei **flussi dovuti al contributo latente per introduzione/estrazione d'acqua** può essere riscritta come:

$$\begin{aligned} h_v^{(H_2O)}\{\langle\theta\rangle_V\}\sum_{k=1}^{N_\sigma} \dot{M}_{H_2O,k} &= \sum_{k=1}^{N_\sigma^{(ENV)}}[\,] + \sum_{k=1}^{N_\sigma^{(SYS)}}[\,] + \sum_{k=1}^{N_\sigma^{(P\&C)}}[\,] \\ = h_v^{(H_2O)}\{\langle\theta\rangle_V\}\sum_{k=1}^{N_\sigma^{(ENV)}} \dot{M}_{H_2O,k} &+ \dot{Q}_{SYS}^{(MS,L)} + \dot{Q}_{P\&C}^{(MS,L)} \end{aligned} \tag{2.26}$$

con

$\dot{Q}_{SYS}^{(MS,L)}$ termine sorgente latente dovuto a introduzione/estrazione d'acqua con entalpia specifica pari a quella media di volume da parte dell'impianto;

$\dot{Q}_{P\&C}^{(MS,L)}$ termine sorgente latente dovuto a introduzione/estrazione d'acqua con entalpia specifica pari a quella media di volume da parte persone e cose.

Usando, per i termini di deviazione entalpica legata all'introduzione/estrazione d'acqua, sempre la suddivisione definita dalla (2.20) si ha che la sommatoria dei **flussi dovuti alla deviazione entalpica sensibile** può essere riscritta come:

$$\Delta h_{l\to v}^{(H_2O)_0}\sum_{k=1}^{N_\sigma}\dot{M}_{H_2O,k}\,\delta_k = \sum_{k=1}^{N_\sigma^{(ENV)}}[\,] + \sum_{k=1}^{N_\sigma^{(SYS)}}[\,] + \sum_{k=1}^{N_\sigma^{(P\&C)}}[\,]$$
$$= \Delta h_{l\to v}^{(H_2O)_0}\sum_{k=1}^{N_\sigma^{(ENV)}}\dot{M}_{H_2O,k}\delta_k + \dot{Q}_{SYS}^{(\Delta H,L)} + \dot{Q}_{P\&C}^{(\Delta H,L)} \tag{2.27}$$

con

$\dot{Q}_{SYS}^{(\Delta H,L)}$ termine sorgente latente dovuto alla deviazione entalpica legata all'introduzione/estrazione d'acqua da parte dell'impianto impianto;

$\dot{Q}_{P\&C}^{(\Delta H,L)}$ termine sorgente latente dovuto alla deviazione entalpica legata all'introduzione/estrazione d'acqua da parte di persone e cose

L'equazione di bilancio entalpico latente può quindi essere riscritta mettendo in evidenza solo le interazioni che l'aria interna ha con l'aria esterna tramite l'involucro edilizio, raggruppando invece le interazioni con gli altri sistemi (impianto e persone e cose) nei termini sorgente/pozzo $\dot{Q}$; cioè:

$$h_v^{(H_2O)}\{\langle\theta\rangle_V\}\left[V\langle\rho^{(AS)}\rangle_V\frac{d\langle x\rangle_V}{dt}\right] = \dot{Q}_{SYS}^{(L)} + \dot{Q}_{P\&C}^{(L)}$$
$$h_v^{(H_2O)}\{\langle\theta\rangle_V\}\left[\sum_{j=1}^{N_{in}^{(ENV)}}\dot{m}_{in_j}^{(AS)}\left(\langle x\rangle_{S_j} - \langle x\rangle_V\right) + \sum_{k=1}^{N_\sigma^{(ENV)}}\dot{M}_{H_2O,k}\right] - \Delta h_{l\to v}^{(H_2O)_0}\sum_{k=1}^{N_\sigma^{(ENV)}}\dot{M}_{H_2O,k}\,\delta_k \tag{2.28}$$

avendo posto

$$\dot{Q}_{SYS}^{(L)} = \dot{Q}_{SYS}^{(AD,L)} + \dot{Q}_{SYS}^{(MS,L)} + \dot{Q}_{SYS}^{(\Delta H,L)} \tag{2.29}$$

$$\dot{Q}_{P\&C}^{(L)} = \dot{Q}_{P\&C}^{(AD,L)} + \dot{Q}_{P\&C}^{(MS,L)} + \dot{Q}_{P\&C}^{(\Delta H,L)} \tag{2.30}$$

2.1.3 Equazioni di bilancio termico e di umidità massica per il nodo aria

Ponendo per facilità di lettura

$\langle\theta\rangle_V = \theta_{ai}$ $\quad\langle\theta\rangle_{S_j} = \theta_{a,in_j}$ $\quad\langle x\rangle_V = x_i$ $\quad\langle x\rangle_{S_j} = x_{in_j}$

$\langle\rho^{(AS)}\rangle_V = \rho_a$ $\quad c_p^{(AU)}\{\langle x\rangle_{S_j}\} = c_{p,a}$ $\quad\langle h_{cv}\rangle_{S_i} = h_{cv_i}$

$V\langle\rho^{(AS)}\rangle_V c_p^{(AU)}\{\langle x\rangle_V\} = C_a$ $\quad\dot{m}_{in_j}^{(AS)} = \dot{m}_{a,in_j}$ $\quad h_v^{(H_2O)} = h_v$

e nell'ipotesi di considerare trascurabile:

- **l'evaporazione dell'acqua di costruzione dalle strutture;**
- **la diffusione del vapore attraverso le strutture**

e quindi di trascurare i termini:

- $\sum_{k=1}^{N_\sigma^{(ENV)}}\dot{M}_{H_2O,k}\Delta h_{k,s}^{(H_2O)}(\theta_k, y)$ nell'equazione di bilancio entalpico sensibile;

- $-\Delta h_{l\to v}^{(H_2O)_0} \sum_{k=1}^{N_\sigma^{(ENV)}} \dot{M}_{H_2O,k}\,\delta_k$ nell'equazione di bilancio entalpico latente;

le due equazioni di bilancio entalpico possono essere riscritte come:

$$C_a \frac{d\theta_{ai}}{dt} = \sum_{j=1}^{N_{in}^{(ENV)}} \dot{m}_{a,in_j} c_{p,a} \left(\theta_{a,in_j} - \theta_{ai}\right) + \sum_{i=1}^{N_{ss}^{(ENV)}} A_i h_{cv_i} \left(\theta_{s_i} - \theta_{ai}\right) + \dot{Q}_{SYS}^{(S)} + \dot{Q}_{P\&C}^{(S)} \tag{2.31}$$

che viene anche detta **equazione di bilancio termico dell'aria**, e

$$V \rho_a h_v \frac{dx_i}{dt} = h_v \sum_{j=1}^{N_{in}^{(ENV)}} \dot{m}_{a,in_j} \left(x_{in_j} - x_i\right) + \dot{Q}_{SYS}^{(L)} + \dot{Q}_{P\&C}^{(L)} \tag{2.32}$$

Dividendo la (2.32) per l'entalpia specifica (media di volume) del vapore d'acqua contenuto nell'aria umida della zona, si ha ricava l'equazione differenziale ordinaria nella massa volumica (media di volume dell'ambiente), cioè l'**equazione di bilancio igrometrico dell'aria**:

$$V \rho_a \frac{dx_i}{dt} = \sum_{j=1}^{N_{in}^{(ENV)}} \dot{m}_{a,in_j} \left(x_{in_j} - x_i\right) + \dot{M}_{SYS}^{(H_2O)} + \dot{M}_{P\&C}^{(H_2O)} \tag{2.33}$$

avendo posto

- $\dot{Q}_{SYS}^{(L)}/h_v \equiv \dot{M}_{SYS}^{(H_2O)} \gtrless 0$ portata equivalente di vapore introdotta/estratta dall'impianto;
- $\dot{Q}_{P\&C}^{(L)}/h_v \equiv \dot{M}_{P\&C}^{(H_2O)} \gtrless 0$ portata equivalente di vapore introdotta/estratta da persone e cose.

La prima, l'equazione di bilancio termico, era stata già descritta e implementata all'inizio dell'attività di ricerca (vedasi [26]) ma indicando la temperatura dell'aria con il simbolo T, normalmente utilizzato per indicare la temperatura in kelvin. Risulta invece adesso evidente che, per le convenzioni adottate nel definire l'entalpia specifica dell'aria umida, la temperatura che compare nell'equazione di bilancio termico deve essere, per consistenza formale, espressa in gradi celsius, cioè simbolo θ, come riportato nella (2.31).

2.2 ODE umidità per il nodo aria

L'equazione (2.33) si può riscrivere come:

$$\frac{dx_i}{dt} + \frac{1}{V\rho_a}\left(\sum_{j=1}^{N_{in}^{(ENV)}} \dot{m}_{a,in_j}\right) x_i = \frac{1}{V\rho_a}\left(\sum_{j=1}^{N_{in}^{(ENV)}} \dot{m}_{a,in_j} x_{in_j} + \dot{M}_{SYS}^{(H_2O)} + \dot{M}_{P\&C}^{(H_2O)}\right) \tag{2.34}$$

che può essere meglio dettagliata sostituendo ai termini sorgete/pozzo di portata d'acqua equivalente le loro definizioni date in precedenza. Separando il termine avvettivo dovuto alle infiltrazioni dirette di aria esterna dai termini di interscambio tra zone limitrofe nella (2.33), cioè $N_{in}^{(ENV)} = ex + N_{coupl}$ si ottiene:

$$\frac{dx_i}{dt} + \frac{1}{V\rho_a}\left(\sum_{j=1}^{N_{coupl}} \dot{m}_{a,in_j} + \dot{m}_{inf}\right) x_i = \frac{1}{V\rho_a}\left(\sum_{j=1}^{N_{coupl}} \dot{m}_{a,in_j} x_{z_j} + \dot{m}_{inf} x_e + \dot{M}_{SYS}^{(H_2O)} + \dot{M}_{P\&C}^{(H_2O)}\right) \tag{2.35}$$

dove

$\dot{m}_{inf}$ portata massica di aria secca esterna di infiltrazione, in [kg$^{(AS)}$/s];
x_e umidità massica dell'aria esterna, in [kg$^{(H2O)}$/ kg$^{(AS)}$];
$\dot{m}_{inf}$ portata massica di aria secca esterna di interscambio con zone limitrofe, in [kg$^{(AS)}$/s];
x_{z_j} umidità massica dell'aria entrante da zone limitrofe, in [kg$^{(H2O)}$/ kg$^{(AS)}$];

Posto

$$A(t) = \frac{1}{V\rho_a}\left(\sum_{j=1}^{N_{coupl}} \dot{m}_{a,in_j} + \dot{m}_{inf}\right) \tag{2.36}$$

$$f(t) = \frac{1}{V\rho_a}\left(\sum_{j=1}^{N_{coupl}} \dot{m}_{a,in_j} x_{z_j} + \dot{m}_{inf} x_e + \dot{M}_{SYS}^{(H_2O)} + \dot{M}_{P\&C}^{(H_2O)}\right) \tag{2.37}$$

la (2.35) si può scrivere come:

$$\frac{dx_i}{dt} + A(t)\, x_i = f(t) \tag{2.38}$$

equazione differenziale ordinaria lineare del primo ordine non omogenea, il cui integrale generale ha la seguente forma:

$$x_i(t) = e^{-\int A(t)dt} \cdot \int f(t) e^{\int A(t)dt}\, dt \tag{2.39}$$

Nei seguenti paragrafi sono riportate alcune soluzioni analitiche e numeriche che si possono ottenere sotto specifiche ipotesi semplificative.

2.2.1 Soluzione analitica con parametri costanti

La soluzione analitica con parametri costanti del bilancio termico è stata già riportata in [26] e quindi viene di seguito riportata la soluzione, simile, per il bilancio igrometrico.

Nell'**ipotesi**:

ρ_a , $\dot{m}_{a,in_j}$, x_{z_j} , $\dot{m}_{inf}$, x_e , $\dot{M}_{SYS}^{(H_2O)}$, $\dot{M}_{P\&C}^{(H_2O)}$ $= costanti$ all'interno dell'intervallo di integrazione

⇨ $\boldsymbol{A(t) = \bar{A} = cost}$; $\boldsymbol{f(t) = \bar{f} = cost}$

la (2.39) si riduce a:

$$x_i(t) = e^{-\bar{A}\,t} \cdot \left[\bar{f} \int e^{\bar{A}\,t}\, dt + C_1\right] = e^{-\bar{A}\,t} \cdot \left[\frac{\bar{f}}{\bar{A}} e^{\bar{A}\,t} + C_1\right] = \frac{\bar{f}}{\bar{A}} + C_1 e^{-\bar{A}\,t} \tag{2.40}$$

Per determinare la costante C_1 poniamo la condizione

$$x_i(t_1) = \frac{\bar{f}}{\bar{A}} + C_1 e^{-\bar{A}\,t_1} \quad \Longrightarrow \quad C_1 = x_i(t_1) e^{\bar{A}\,t_1} - \frac{\bar{f}}{\bar{A}} e^{\bar{A}\,t_1} \tag{2.41}$$

da cui

$$x_i(t) = \frac{\bar{f}}{\bar{A}}\left(1 - e^{-\bar{A}\,(t-t_1)}\right) + x_i(t_1)e^{-\bar{A}\,(t-t_1)} \tag{2.42}$$

Infine se consideriamo il tempo $t_2 = t_1 + \Delta t$, l'umidità massica dopo un intervallo temporale Δt è pari a:

$$x_i(t_2) = \frac{\bar{f}}{\bar{A}} + \left(x_i(t_1) - \frac{\bar{f}}{\bar{A}}\right) \cdot e^{-\bar{A}\Delta t} \tag{2.43}$$

NOTA: la (2.43) consente di determinare analiticamente la variazione di umidità massica del nodo aria qualora si considerino costanti tutti i parametri dell'equazione di bilancio igrometrico all'interno dell'intervallo di integrazione Δt. È però possibile variare tali paramenti nel successivo intervallo di integrazione, ottenendo così un'approssimazione integrale a gradini (rispetto ai parametri).

2.2.2 Soluzione analitica con parametri variabili linearmente

Nel seguito è riportata la soluzione analitica nell'ipotesi di coefficienti costanti e umidità massiche e portate d'acqua variabili nel tempo linearmente:

Nell'**ipotesi**:

ρ_a , $\dot{m}_{a,in_j}$, x_{z_j} , $\dot{m}_{inf}$, x_e , $\dot{M}_{SYS}^{(H_2O)}$, $\dot{M}_{P\&C}^{(H_2O)} = costanti$ all'interno dell'intervallo di integrazione

⇨ $\boldsymbol{A(t) = \bar{A} = cost}$; $\boldsymbol{f(t) = \bar{f} = cost}$

la (2.39) si riduce a:

ρ_a , $\dot{m}_{a,in_j}$, $\dot{m}_{inf}$, $= costanti$ aall'interno dell'intervallo di integrazione

x_{z_j} x_e , $\dot{M}_{SYS}^{(H_2O)}$, $\dot{M}_{P\&C}^{(H_2O)}$ funzioni lineari del tempo all'interno dell'intervallo di integrazione

⇨ $\boldsymbol{A(t) = \bar{A} = cost}$; $\boldsymbol{f(t) = f(t_1) + \frac{f(t_2)-f(t_1)}{t_2-t_1}(t - t_1) = a + b(t - t_1)}$

la (2.39) si riduce a:

$$x_i(t) = e^{-\bar{A}\,t} \cdot \left[\int f(t)e^{\bar{A}\,t}\,dt + C_1\right] = e^{-\bar{A}\,t} \cdot \left[\frac{a\bar{A} + b[\bar{A}(t - t_1) - 1]}{\bar{A}^2}e^{\bar{A}\,t} + C_1\right] \tag{2.44}$$

Per determinare la costante C_1poniamo la condizione

$$x_i(t_1) = \frac{a\bar{A} - b}{\bar{A}^2} + C_1 e^{-\bar{A}\,t_1} \quad \Longrightarrow \quad C_1 = x_i(t_1)e^{\bar{A}\,t_1} - \frac{a\bar{A} - b}{\bar{A}^2}e^{\bar{A}\,t_1} \tag{2.45}$$

da cui

$$x_i(t) = \frac{a\bar{A} - b}{\bar{A}^2}\left(1 - e^{-\bar{A}\,(t-t_1)}\right) + \frac{b}{\bar{A}}(t - t_1) + x_i(t_1)e^{-\bar{A}\,(t-t_1)} \tag{2.46}$$

Infine se consideriamo il tempo $t_2 = t_1 + \Delta t$, l'umidità massica dopo un intervallo temporale Δt è pari a:

$$x_i(t_2) = \frac{a\bar{A} - b}{\bar{A}^2} + \frac{b}{\bar{A}}\Delta t + \left(x_i(t_1) - \frac{a\bar{A} - b}{\bar{A}^2}\right) \cdot e^{-\bar{A}\Delta t} \tag{2.47}$$

NOTA: la (2.47) consente di determinare analiticamente la variazione di temperatura del nodo aria qualora si considerino costanti i coefficienti e variabile linearmente la forzante dell'equazione di bilancio termico all'interno dell'intervallo di integrazione Δt, consentendo una migliore approssimazione rispetto al caso di parametri costanti.

2.2.3 Integrazione numerica

Con riferimento a quanto già riportato nel paragrafo 4.8.2 di [25], nel seguito verranno sviluppati diversi schemi di integrazione numerica per il bilancio igrometrico.

2.2.3.1 Metodo di Eulero

Il metodo di Eulero per l'integrazione di un'equazione differenziale lineare ordinaria del primo ordine consiste nell'approssimare la derivata prima nel tempo con il suo rapporto incrementale, il che equivale ad approssimarla con una differenza in avanti, cioè:

$$\frac{dx_i}{dt} \cong \frac{x_i(t + \Delta t) - x_i(t)}{\Delta t} \tag{2.48}$$

da cui, sostituendo nella (2.33) e introducendo la variabile discreta x_i^τ e l'indice temporale adimensionale τ ,tale che $t = \tau \cdot \Delta t$, si ottiene:

$$V\rho_a \frac{x_i^{\tau+1} - x_i^\tau}{\Delta t} = \sum_{j=1}^{N_{coupl}} \dot{m}_j^\tau (x_{z,j}^\tau - x_i^\tau) + \dot{m}_{inf}^\tau (x_e^\tau - x_i^\tau) + \dot{M}_{P\&C}^\tau + \dot{M}_{sys}^\tau \tag{2.49}$$

Da cui si ottiene un'equazione algebrica esplicita:

$$x_i^{\tau+1} = x_i^\tau + \frac{\Delta t}{V\rho_a} \sum_{j=1}^{N_{coupl}} \dot{m}_j^\tau (x_{z,j}^\tau - x_i^\tau) + \frac{\Delta t}{V\rho_a} \dot{m}_{inf}^\tau (x_e^\tau - x_i^\tau) + \frac{\Delta t}{V\rho_a} \left(\dot{M}_{P\&C}^\tau + \dot{M}_{sys}^\tau\right) \tag{2.50}$$

che può essere direttamente utilizzata in tale forma per l'integrazione numerica.

NOTA: Per semplificare la scrittura e migliorare la leggibilità, nelle formule presenti sono state utilizzate le seguenti posizioni, che verranno adottate anche nel seguito:

$\dot{M}_{SYS}^{(H_2O)}(t) \equiv \dot{M}_{sys}^\tau$, $\dot{M}_{P\&C}^{(H_2O)}(t) \equiv \dot{M}_{P\&C}^\tau$ e $\dot{m}_{a,in_j}(t) = \dot{m}_j^{\tau+1}$

2.2.3.2 Metodo implicito

Il metodo implicito nasce dalla costatazione che il metodo di Eulero di fatto coincide con l'approssimazione della derivata temporale ordinaria con una differenza finita in avanti, e consiste nell'utilizzare per tale approssimazione una differenza indietro, cioè:

$$\frac{dx_i}{dt} \cong \frac{x_i(t) - x_i(t-\Delta t)}{\Delta t} \tag{2.51}$$

da cui, sostituendo nella (2.33) e introducendo la variabile discreta x_i^τ e l'indice temporale adimensionale τ ,tale che $t = \tau \cdot \Delta t$, si ottiene con una tralazione unitaria del tempo:

$$V\rho_a \frac{x_i^{\tau+1} - x_i^\tau}{\Delta t} = \sum_{j=1}^{N_{coupl}} \dot{m}_j^{\tau+1}(x_{z,j}^{\tau+1} - x_i^{\tau+1}) + \dot{m}_{inf}^{\tau+1}\left(x_e^{\tau+1} - x_i^{\tau+1}\right) + \dot{M}_{P\&C}^{\tau+1} + \dot{M}_{sys}^{\tau+1} \tag{2.52}$$

Da cui si ottiene la seguente equazione algebrica:

$$x_i^{\tau+1}\left[1 - \frac{\Delta t}{V\rho_a}\left(\sum_{j=1}^{N_{coupl}} \dot{m}_j^{\tau+1} + \dot{m}_{inf}^{\tau+1}\right)\right] = x_i^\tau + \frac{\Delta t}{V\rho_a}\left(\sum_{j=1}^{N_{coupl}} \dot{m}_j^{\tau+1} x_{z,j}^{\tau+1} + \dot{m}_{inf}^{\tau+1} x_e^{\tau+1} + \dot{M}_{P\&C}^{\tau+1} + \dot{M}_{sys}^{\tau+1}\right) \tag{2.53}$$

E posto

$$D^{\tau+1} = 1 - \frac{\Delta t}{V\rho_a}\left(\sum_{j=1}^{N_{coupl}} \dot{m}_j^{\tau+1} + \dot{m}_{inf}^{\tau+1}\right) \tag{2.54}$$

la (2.53) può essere riscritta come:

$$x_i^{\tau+1} = \frac{1}{D^{\tau+1}} x_i^\tau + \frac{\Delta t}{D^{\tau+1} C} \cdot \left(\sum_{j=1}^{N_{coupl}} \dot{m}_j^{\tau+1} x_{z,j}^{\tau+1} + \dot{m}_{inf}^{\tau+1} x_e^{\tau+1} + \dot{M}_{p\&f}^{\tau+1} + \dot{M}_{sys}^{\tau+1}\right) \tag{2.55}$$

La (2.55) rappresenta lo schema implicito dell'equazione del nodo aria, che necessita per la sua utilizzazione della conoscenza delle grandezze al passo temporale attuale $\tau + 1$ e non il loro valore pregresso (all'istante τ), ad esclusione ovviamente del valore pregresso dell'umidità massica dell'aria ambiente.

La formulazione implicita viene solitamente impiegata quando si utilizza lo schema completamente implicito per la soluzione del bilancio termico del nodo aria, soluzione adottata quando impiegata anche nella soluzione dalla conduzione termica nelle pareti. In tal caso è conveniente raggruppare le equazioni del nodo aria e le equazioni matriciali di tutte le pareti in un unico sistema algebrico, costruendo una matrice unica che contenga tali equazioni e tutte quelle delle pareti.

2.2.3.3 Metodo del trapezio (Crank-Nicolson): estensione al Theta Method

Ricordando quanto già espresso al paragrafo 4.8.2 in [25], cioè che il problema di Cauchy associato ad una ODE del primo ordine, espresso come:

$$\begin{cases} y'(t) = f(t, y(t)) & t \in I \\ y(t_0) = y_0 \end{cases} \tag{2.56}$$

ha, per una funzione continua in $\overline{t_0\ t}$, come soluzione la forma generale

$$y(t) - y_0 = \int_{t_0}^{t} y'(\tau)d\tau = \int_{t_0}^{t} f(\tau, y(\tau))d\tau \tag{2.57}$$

il metodo del trapezio, detto anche di Crank-Nicolson, è uno dei metodi numerici di integrazione di ODE più utilizzato, e consiste nell'approssimare l'integrale a secondo membro della (2.57) con la formula del trapezio (sviluppata per il calcolo approssimato delle aree), ottenendo il seguente schema numerico:

$$y_{n+1} - y_n = \frac{\Delta t}{2}[f(t_n) + f(t_{n+1})] \tag{2.58}$$

con

Δt intervallo di discretizzazione del dominio temporale;

t_n tempo discretizzato, $t_n = n \cdot \Delta t$

n indice intero avanzamento temporale, $n \in \mathbb{N}$;

$f(t_n)$ funzione integranda valutata al tempo discreto t_n.

Il bilancio igrometrico del nodo aria, definito dalla (2.33), può essere riscritto come:

$$V\rho_a \frac{dx_i}{dt} = \sum_{j=1}^{N_{coupl}} \dot{m}_j(x_{z,j} - x_i) + \dot{m}_{inf}(x_e - x_i) + \dot{M}_{tot}^{(H_2O)} \tag{2.59}$$

dove si sono esplicitati i flussi avvettivi di infiltrazione, $\dot{m}_{inf}$, e si è compattato in un unico termine sorgente, $\dot{M}_{tot}^{(H_2O)}$, il contributo delle sorgenti/pozzi di massa d'acqua dovuti a persone e cose, $\dot{M}_{SYS}^{(H_2O)} e\ \dot{M}_{P\&C}^{(H_2O)}$.

Applicando alla (2.59) lo schema della (2.58) e introducendo la variabile discreta x_i^{τ}e l'ndice temporale adimesionale τ ,tale che $t = \tau \cdot \Delta t$, si ottiene:

$$\begin{aligned} V\rho_a \left[\frac{dx_i}{dt}\right]^{\frac{t_{i+1}+t_i}{2}} &= V\rho_a \frac{x_i^{\tau+1} - x_i^{\tau}}{\Delta t} \\ &= \frac{1}{2}\left[\sum_{j=1}^{N_{coupl}} \dot{m}_j\left(x_{z,j} - x_i\right) + \dot{m}_{inf}(x_e - x_i) + \dot{M}_{tot}^{(H_2O)}\right]^{\tau+1} \\ &+ \frac{1}{2}\left[\sum_{j=1}^{N_{coupl}} \dot{m}_j(x_{z,j} - x_i) + \dot{m}_{inf}(x_e - x_i) + \dot{M}_{tot}^{(H_2O)}\right]^{\tau} \end{aligned} \tag{2.60}$$

Possiamo generalizzare la forma espressa dalla (2.59), sostituendo ai pesi ½ il generico parametro θ, compreso tra 0 e 1, ottenendo:

$$V\rho_a \frac{x_i^{\tau+1} - x_i^{\tau}}{\Delta t} = \theta \left[\sum_{j=1}^{N_{coupl}} \dot{m}_j (x_{z,j} - x_i) + \dot{m}_{inf}(x_e - x_i) + \dot{M}_{tot}^{(H_2O)} \right]^{\tau+1} + (1-\theta) \left[\sum_{j=1}^{N_{coupl}} \dot{m}_j (x_{z,j} - x_i) + \dot{m}_{inf}(x_e - x_i) + \dot{M}_{tot}^{(H_2O)} \right]^{\tau} \quad (2.61)$$

Esplicitando le variabili si ha:

$$\left[\frac{V\rho_a}{\Delta t} + \theta \left(\sum_{j=1}^{N_{coupl}} \dot{m}_j + \dot{m}_{inf} \right)^{\tau+1} \right] x_i^{\tau+1} = \left[\frac{V\rho_a}{\Delta t} - (1-\theta) \left(\sum_{j=1}^{N_{coupl}} \dot{m}_j + \dot{m}_{inf} \right)^{\tau+1} \right] x_i^{\tau} + \theta \left[\sum_{j=1}^{N_{coupl}} \dot{m}_j x_{z,j} + \dot{m}_{inf} x_e + \dot{M}_{tot}^{(H_2O)} \right]^{\tau+1} + (1-\theta) \left[\sum_{j=1}^{N_{coupl}} \dot{m}_j x_{z,j} + \dot{m}_{inf} x_e + \dot{M}_{tot}^{(H_2O)} \right]^{\tau} \quad (2.62)$$

Posto

$$D^{\tau+1} = \left[\frac{V\rho_a}{\Delta t} + \theta \left(\sum_{j=1}^{N_{coupl}} \dot{m}_j + \dot{m}_{inf} \right)^{\tau+1} \right] \quad (2.63)$$

la (2.62) si può riscrivere come

$$x_i^{\tau+1} = \frac{1}{D^{\tau+1}} \left[\frac{V\rho_a}{\Delta t} - (1-\theta) \left(\sum_{j=1}^{N_{coupl}} \dot{m}_j + \dot{m}_{inf} \right)^{\tau+1} \right] \cdot x_i^{\tau} + \frac{\theta}{D^{\tau+1}} \left[\sum_{j=1}^{N_{coupl}} \dot{m}_j x_{z,j} + \dot{m}_{inf} x_e + \dot{M}_{tot}^{(H_2O)} \right]^{\tau+1} + \frac{1-\theta}{D^{\tau+1}} \left[\sum_{j=1}^{N_{coupl}} \dot{m}_j x_{z,j} + \dot{m}_{inf} x_e + \dot{M}_{tot}^{(H_2O)} \right]^{\tau} \quad (2.64)$$

Si può derivare anche una formulazione alternativa, moltiplicando la (2.61) per $\Delta t/(V\rho_a)$, ottenendo:

$$x_i^{\tau+1} = x_i^{\tau} + \theta \frac{\Delta t}{V\rho_a}\left[\sum_{j=1}^{N_{coupl}} \dot{m}_j\left(x_{z,j} - x_i\right) + \dot{m}_{inf}(x_e - x_i) + \dot{M}_{tot}^{(H_2O)}\right]^{\tau+1} + (1-\theta)\frac{\Delta t}{V\rho_a}\left[\sum_{j=1}^{N_{coupl}} \dot{m}_j\left(x_{z,j} - x_i\right) + \dot{m}_{inf}(x_e - x_i) + \dot{M}_{tot}^{(H_2O)}\right]^{\tau} \tag{2.65}$$

e raccogliendo solo rispetto a $x_i^{\tau+1}$ si ottiene:

$$x_i^{\tau+1}\left(1 + \theta \frac{\Delta t}{V\rho_a}\left[\sum_{j=1}^{N_{coupl}} \dot{m}_j + \dot{m}_{inf}\right]^{\tau+1}\right) = = x_i^{\tau} + \theta \frac{\Delta t}{V\rho_a}\left[\sum_{j=1}^{N_{coupl}} \dot{m}_j x_{z,j} + \dot{m}_{inf} x_e + \dot{M}_{tot}^{(H_2O)}\right]^{\tau+1} + (1-\theta)\frac{\Delta t}{V\rho_a}\left[\sum_{j=1}^{N_{coupl}} \dot{m}_j\left(x_{z,j} - x_i\right) + \dot{m}_{inf}(x_e - x_i) + \dot{M}_{tot}^{(H_2O)}\right]^{\tau} \tag{2.66}$$

Posto

$$D^{\tau+1} = \left(1 + \theta \frac{\Delta t}{V\rho_a}\left[\sum_{j=1}^{N_{coupl}} \dot{m}_j + \dot{m}_{inf}\right]^{\tau+1}\right) \tag{2.67}$$

dalla (2.66) si ricava

$$x_i^{\tau+1} = x_i^{\tau} + \theta \frac{\Delta t}{D^{\tau+1} V\rho_a}\left[\sum_{j=1}^{N_{coupl}} \dot{m}_j x_{z,j} + \dot{m}_{inf} x_e + \dot{M}_{tot}^{(H_2O)}\right]^{\tau+1} + (1-\theta)\frac{\Delta t}{D^{\tau+1} V\rho_a}\left[\sum_{j=1}^{N_{coupl}} \dot{m}_j\left(x_{z,j} - x_i\right) + \dot{m}_{inf}(x_e - x_i) + \dot{M}_{tot}^{(H_2O)}\right]^{\tau} \tag{2.68}$$

3 Codice sorgente

3.1 Ambiente di sviluppo

Il progetto informatico è stato sviluppato in C# all'interno di Visual Studio 2019 (Figura 4).

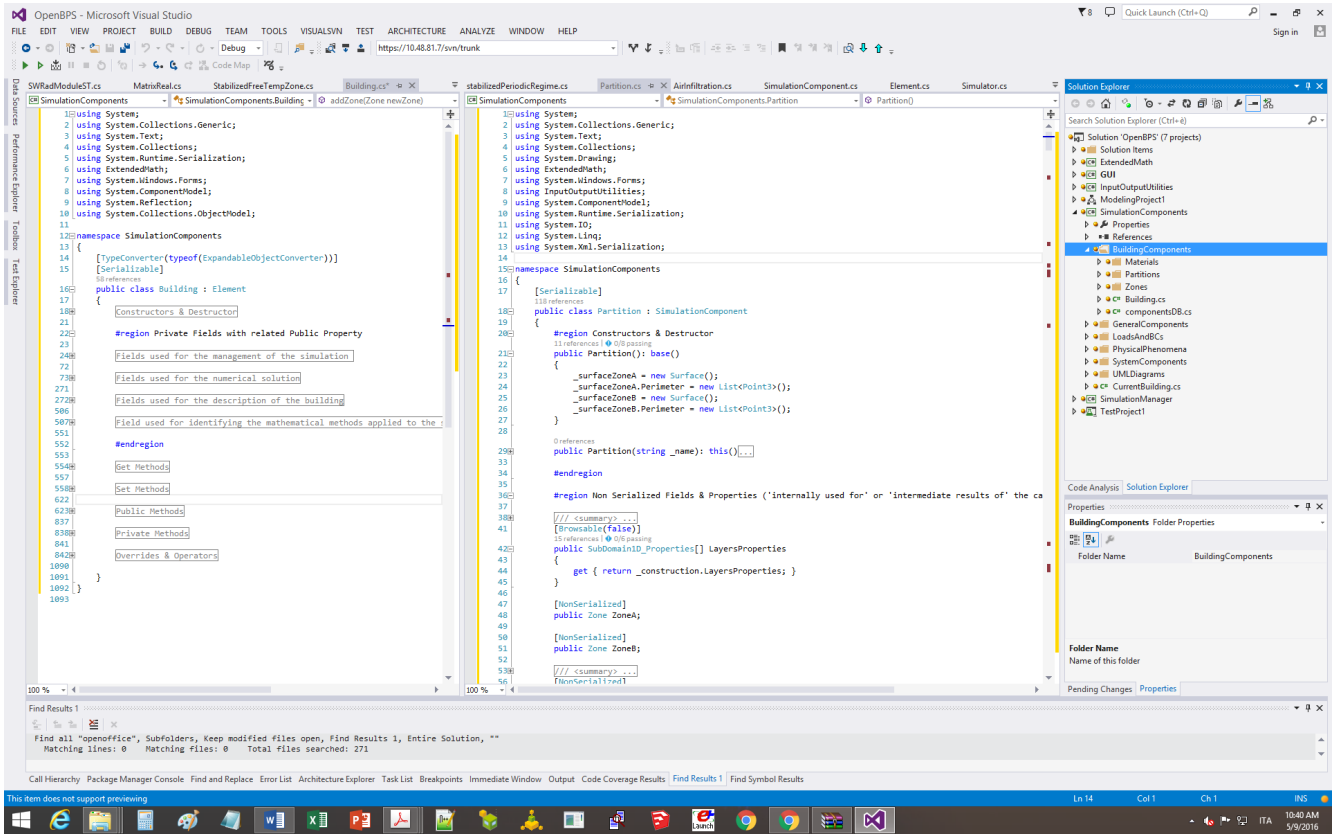


Figura 4 - Ambiente di sviluppo e sotto-strutturazione in "Progetti" della "Soluzione" informatica:

L'ambiente di sviluppo prevede la creazione di una "Solution", o Soluzione, contenente gli elementi necessari per l'implementazione di un applicativo informatico. Tali elementi comprendono uno o più progetti, file e metadati per la definizione di tale Soluzione nel suo complesso. In particolare i "progetti" sono utilizzati per gestire logicamente, costruire ed eseguire elementi di natura diversa, sia dal punto di vista informatico che strutturale, necessari all'implementazione di un programma.

Dei progetti inclusi nella nostra Soluzione, uno darà luogo ad un programma eseguibile (exe), altri quattro genereranno ciascuno una libreria dinamica (DLL), mentre gli ultimi due, di supporto allo sviluppo, genereranno "warning" o "report di test".

In particolare, come è possibile vedere nella struttura ad albero presente sulla destra (Figura 4) e riportata con maggior dettaglio in Figura 5, abbiamo:

1. un applicativo eseguibile, che costituisce l'interfaccia grafica utente, chiamato **GUI**;
2. quattro progetti di libreria dinamica (dll), identificati come:

a. ExtendedMath, contenente gli algoritmi di calcolo usati per la simulazione;

b. SimulationManager, contenente l'implementazione del gestore della simulazione;

c. SimulationComponents, contenente la descrizione dei componenti costituenti il sistema edificio;

d. InputOutputUtilities, contenente gli strumenti per la gestione degli input e degli output;

3. un progetto per la modellazione dell'applicativo: ModelingProject1;
4. un progetto di test dell'applicativo: TestProject1.

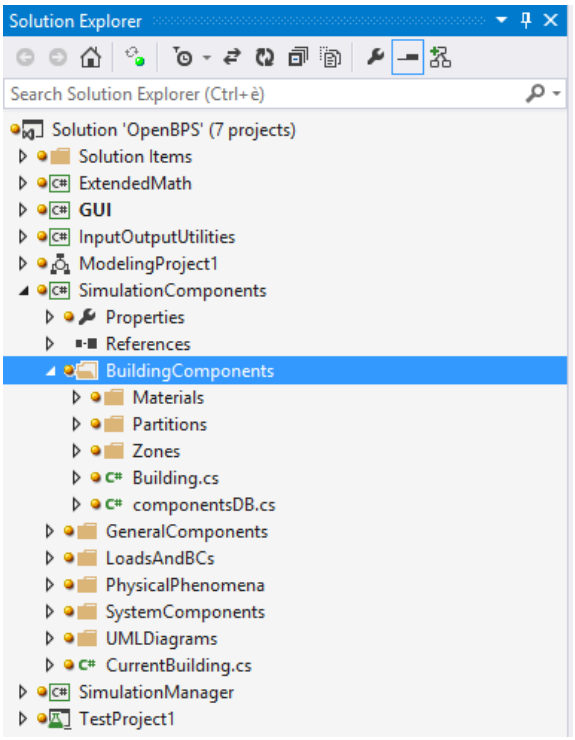


Figura 5 - Struttura Soluzione Informatica

Al momento è stato sviluppato un progetto eseguibile (GUI), all'interno del quale è implementata l'interfaccia grafica (Figura 6) per l'uso del kernel di calcolo.

Tale interfaccia consente di importare file di input di EnergyPlus, con alcune limitazioni, o file creati con il plug-in di TRNSYS17 o di OpenStudio in SketchUp, sempre con alcune limitazioni.
È possibile al suo interno usare funzioni di drag-and-drop per assegnare il solutore alle diverse partizioni, o la tipologia di costruzione (strati) delle diverse costruzioni (opache o trasparenti), richiamando le tipologie di costruzione create in precedenza e salvate in un database di progetto.

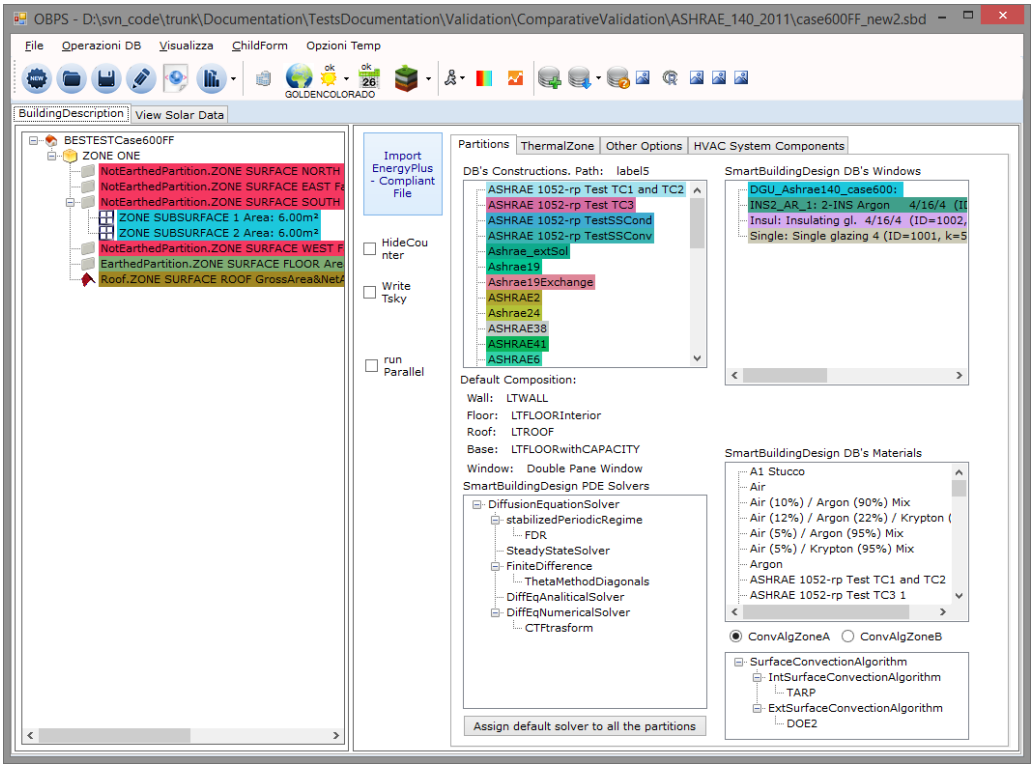


Figura 6 - Interfaccia grafica utente: GUI

Sono stati implementati anche altri supporti per la definizione dei dati di input, come una mappa geografica, che utilizza una libreria legata a GoogleMap, per definire le coordinate della località in cui è ambientata la simulazione (Figura 7).

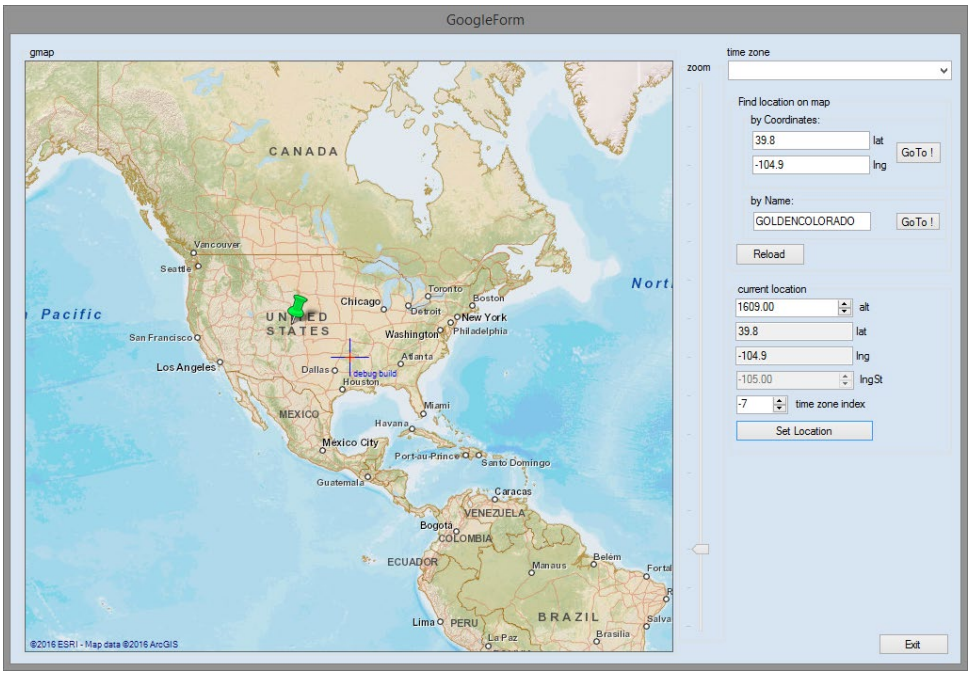


Figura 7 - Strumenti di supporto alla definizione dei dati di input

Se riprendiamo invece le librerie di componente (Figura 5), troviamo, ad esempio, all'interno della libreria "SimulationComponent", le strutture per la simulazione degli elementi che costituiscono l'edificio, fra i quali citiamo i più importanti, come:

1. i componenti dell'edificio "BuildingComponents":
   a. partizioni interne ed esterne, opache e trasparenti
   b. i volumi d'aria
   c. i materiali
2. i componenti "LoadsAndBCs", usati per definire i carichi e le condizioni al contorno dei vari componenti costituenti l'edificio, quali schedule, infiltrazioni, apporti interni, ecc.
3. i componenti dell'impianto "SystemComponents".

La filosofia di programmazione scelta per l'implementazione dei modelli descrittivo, matematico e numerico del sistema edificio è quella tipica della programmazione ad oggetti.
La struttura ad oggetti implementata, consente, fra le altre cose, una più agevole:

1. suddivisione delle responsabilità inerenti alle diverse parti di codice;
2. navigazione/comprensione del codice sviluppato;
3. riutilizzo/estensione/manutenzione del codice sviluppato.

La suddivisione delle responsabilità all'interno di un modello complesso, è di fondamentale importanza per la facile identificazione di "chi fa cosa". È infatti importante che ciascuna struttura di codice, racchiusa all'interno di una "classe", sia identificabile con una "funzionalità" specifica, in modo da essere facilmente identificabile ed il più facilmente possibile "sostituibile/modificabile", senza la necessità di agire su più parti di codice.

Questo consente una facile comprensione, identificazione e manutenzione di quanto sviluppato o ancora da sviluppare.

Un esempio applicativo di tale suddivisione delle responsabilità è visibile dal diagramma di classe di una partizione, come riportato in Figura 8, laddove le frecce uscenti dalla "classe" "Partition" indicano che la partizione "possiede" dei componenti che si occupano di gestire i diversi aspetti descrittivi o numerici funzionali alla sua simulazione.

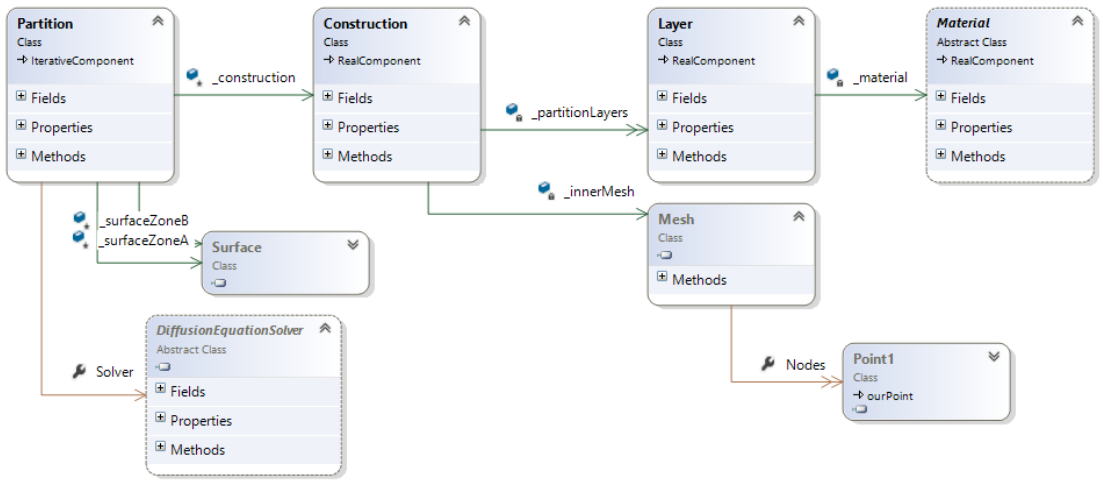


Figura 8 -Struttura Partizione

La partizione ha infatti la responsabilità di contenere al suo interno informazioni relative alla sua collocazione spaziale, alle parti da cui è composta ed al solutore numerico che si intende usare per la sua simulazione dinamica. È responsabilità del solutore numerico quella di eseguire il calcolo numerico necessario alla sua simulazione e della "Construction", che può essere la stessa per diverse partizioni, quella di gestire i diversi strati e materiali di cui è composta la partizione.

Oltre all'interfaccia grafica ed alle librerie di componenti o strumenti di sviluppo, sono presenti nella Soluzione due progetti di supporto allo sviluppo: il progetto di modellazione e quello di test.

Tali progetti sono rispettivamente funzionali:

1. alla definizione delle relazioni implementate ed implementabili, secondo quanto definito dalle strategie di sviluppo
2. alla fase di test del codice.

Infatti tramite il progetto di modellazione "ModellingProject1" è possibile definire quali interrelazioni fra le diverse librerie siano consentite e quali non lo siano, laddove tale identificazione è stata formalizzata allo scopo di realizzare una migliore strutturazione del codice (Figura 9).

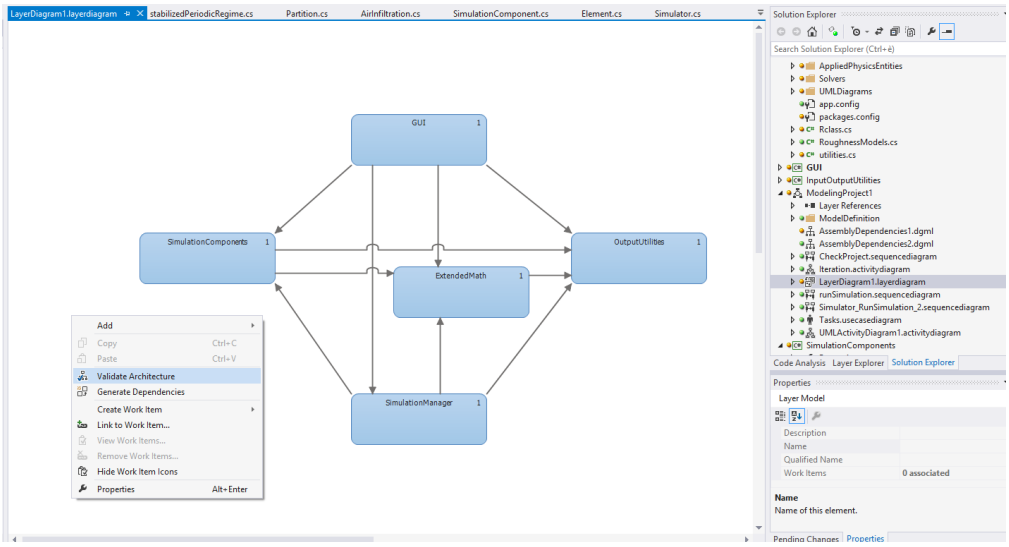


Figura 9 - Validazione architetturale

Se qualche sviluppatore dovesse implementare delle interdipendenze che non rispettano tali prescrizioni, il progetto di modellazione bloccherebbe la compilazione del progetto.

Il progetto di test, "TestProject1", è invece funzionale a eseguire in automatico diversi test di parti di codice per i quali siano stati definiti input e output attesi. Questo progetto è utile ad un continuo controllo delle funzionalità del codice, in quando, in seguito a modifiche, i test definiti al suo interno devono essere ancora verificati.

3.2 Le librerie dinamiche

Come più volte specificato, l'obbiettivo di questa ricerca non è quella di produrre un software direttamente utilizzabile dall'utente finale, ma delle librerie dinamiche che possano essere integrate in interfacce proprietarie di software terzi, che in tal modo possono aggiungere alle loro funzionalità quella della simulazione dinamica delle prestazioni degli edifici.

Il codice sviluppato si articola, come già detto, in cinque progetti che corrispondono a quattro librerie dinamiche, già elencate e che verranno nel seguito brevemente descritte.

Ad ogni libreria si è cercato di far corrispondere uno specifico e diverso spazio di programmazione, identificato dal *namespace*, ma questo principio non è stato sempre attuato e sarà il principale obbiettivo della prima revisione del codice.

3.2.1 ExtendedMath

La libreria **ExtendedMath** contiene gli algoritmi di calcolo usati per la simulazione. Questi sono raggruppati in tre divere cartelle in funzione della loro specifica natura, ad esclusione delle utilità (metodi di servizio) che risiedono nel file *utility.cs* posto nella radice del progetto. Tutte le classi e strutture e i loro relativi metodi e proprietà appartengono al ***namespace*** **ExtendedMath.**

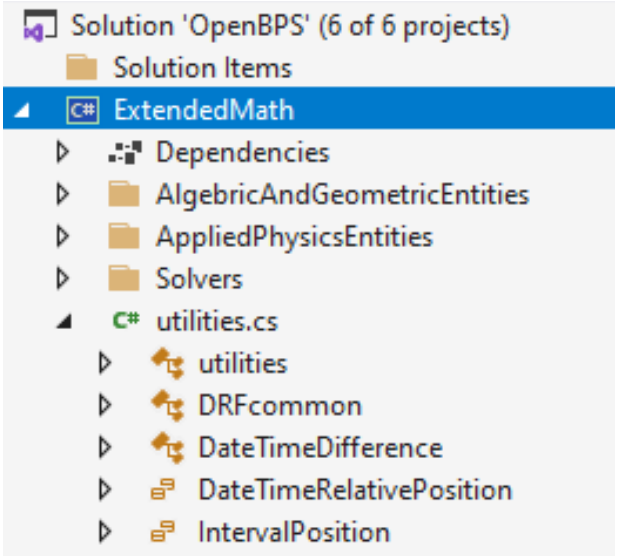


Figura 10 – Struttura della libreria ExtendedMath

Le "*utilità*", come evincibile da Figura 10 , sono costituite da tre classi e due enumeratori:

Classi:

- **utilities**: contenete 5 metodi per il confronto e la manipolazione, trasformazione di date;
- **DRFcommon**: definisce una classe statica per la generazione dei campi di variabili necessari per la ricerca di radici di una funzione, utilizzato nel metodo per la determinazione delle funzioni di trasferimento per la conduzione;
- **DateTimeDifference**: definisce una classe pubblica per poter operare la differenza tra date.

Enumeratori:

- **DateTimeRelativePosition**: ritorna un indice numerico per Before, Equal, Later
- **IntervalPosition**: ritorna un indice numerico per Start, Mid, End

Come riportato in Figura 10 , le tre cartelle specializzate sono: *AglebricAndGeometricEntities*, *AppiedPhysicsENtities* e *Solvers*.

3.2.1.1 AglebricAndGeometricEntities

La cartella *AglebricAndGeometricEntities* contiene, come evincibile da Figura 11, le classi con i relativi metodi e le strutture necessarie per la descrizione e manipolazione numerica e geometrica del sistema edificio.
In particolare nei file contenuti si hanno le seguenti **classi**:

- **Complex**: questa classe definisce le variabili complesse e la loro algebra, introducendo gli operatori complessi di somma, sottrazione, moltiplicazione, divisione, elevazione a potenza, esponenziale, logaritmo, valore assoluto, radice quadrata, parte reale e immaginaria, le funzioni trigonometriche e iperboliche e le loro inverse, e altro;

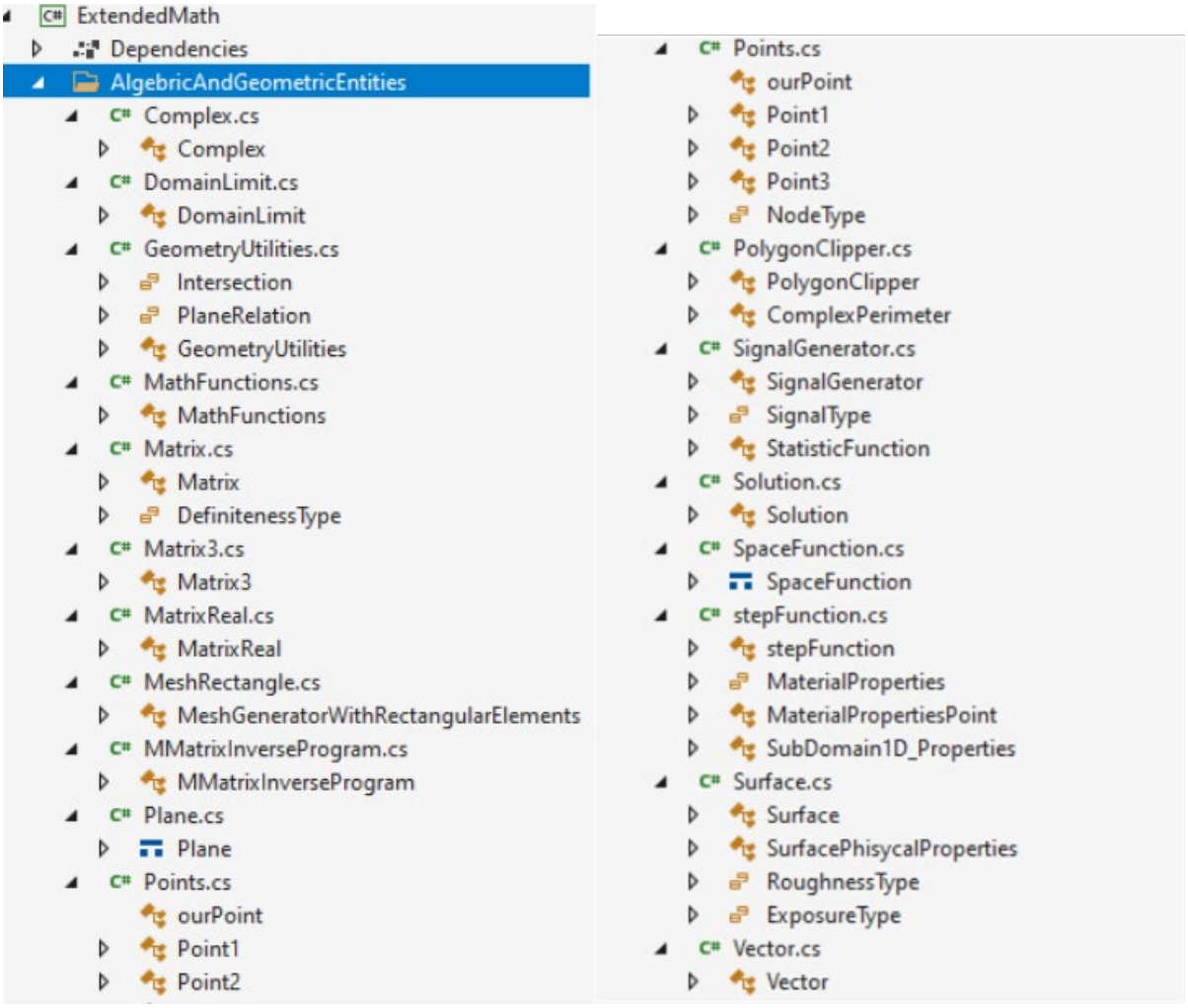


Figura 11 – Struttura della cartella *AglebricAndGeometricEntities*

- **DomainLimit**: questa classe definisce una variabile dominio monodimensionale, identificata da nome, estremo inferiore e superiore;
- **GeometryUtilities**: questa classe definisce una serie di metodi per la determinazione delle coordinate geometriche della proiezione di un punto su un piano, delle coordinate di un piano perpendicolare, le intersezioni tra linee o piani e linee e piani, il calcolo della distanza tra punti, il calcolo dell'area di una superficie e la lunghezza di una linea, la verifica che un punto sia interno ad una determinata figura geometrica o no; la classe fa uso di due enumeratori:
  - **Intersection**: None, Tangent, Intersection, Containment;
  - **PlaneRelation**: Perpendicular, Parallel, Other
- **MathFunctions:** questa classe definisce una serie di metodi per l'interpolazione lienale monodimensionale e bidimensionale;
- **Matrix:** questa classe definisce vettori e matrici a 2 dimensioni sia reali che complesse, anche particolari come la matrice di Hilbert, Delta di Kronecker, identità, zero, a scacchiera, tridiagonali, casuali, casuali zero-uno, e tutta una serie metodi per l'implementazione di operatori algebrici per il calcolo matriciale, trasposizione, fattorizzazione, decomposizione, inversione di matrice e risoluzione di sistemi lineari, e altro; la classe fa uso dell'enumeratore:
  - **DefinitenessType**: PositiveDefinite, PositiveSemidefinite, NegativeDefinite, NegativeSemidefinite, Indefinite;
- **Matrix3:** questa classe definisce matrici a 3 dimensioni e vettori multipli necessari per le rappresentazioni di spazi geometrici tridimensionali, vendo definiti i relativi operatori algebrici di

addizione, sottrazione e moltiplicazione e una serie di metodi per la moltiplicazione vettoriale, la variazione di scala, la rotazione rispetto ad un asse, la prospettiva, e altro;

- **MatrixReal:** questa classe è sostanzialmente identica alla classe Matrix, con l'unica differenza che è specializzata per matrici reali e quindi, in tal caso, più efficiente della classe più generale;
- **MeshGeneratorWithRectangularElements**: classe che contiene dei metodi per generare una mesh all'interno di un rettangolo;
- **MMatrixInverseProgram**: questa classe definisce un metodo che implementa un particolare algoritmo (decomposizione di Crout) per l'inversione di matrice;
- **Plane**: non è una classe ma una struttura che serve per generare delle variabili **Piano**, che contengono tutti i parametri geometrici che lo indentificano nello spazio, oltre dei metodi che consentono di determinare la distanza di un punto dal piano, sia in modulo che con segno (rispetto al sistema di riferimento), e altro;
- **Point1**: questa classe definisce il punto monodimensionale, cioè su una linea, e fornisce un metodo per determinare la sua distanza da un secondo punto;
- **Point2**: questa classe definisce il punto bidimensionale, cioè su un piano, e fornisce un metodo per determinare la sua distanza da un secondo punto;
- **Point3**: questa classe definisce il punto tridimensionale, cioè nello spazio, e fornisce un metodo per determinare la sua distanza da un secondo punto e uno per applicargli una trasformazione definita da una matrice di trasformazione tridimensionale, oltre gli operatori algebrici di somma e sottrazione vettoriale;
- **PolygonClipper**: questa classe definisce una serie di metodi necessari per determinare le proiezioni secondo una direzione predefinita di una figura geometrica (ad esempio il rettandolo di una finestra) su una o più altre superfici comunque disposte nello spazio (serve per determinare le aree illuminate dalla radiazione solare diretta);
- **SignalGenerator**: questa classe contiene dei metodi per generare quattro semplici forme d'onda periodiche tra cui seno, quadrato, triangolo e dente di sega e altre, utili per definire forzanti periodiche; la classe fa uso dell'enumeratore:
    - **SignalType**: Sine, Cosine, Square, Triangle, Sawtooth, Pulse, WhiteNoise // random between -1 and 1 //, GaussNoise // random between -1 and 1 with normal distribution //, DigitalNoise, UserDefined, // user defined between -1 and 1 //, PositiveSine // negative values disregarded//;
- **StatisticFunction**: questa classe contiene dei metodi per calcolare la media, la deviazione standard, la varianza, la distribuzione normale standard e la funzione di distribuzione normale standard cumulativa inversa;
- **Solution**: questa classe introduce la variabile *solution,* che include oltre che il risultato di un calcolo, il timestamp, l'identificativo, le eventuali coordinate del punto, e dei metodi per aggiungere una linea di testo ad un file (in genere utilizzate per archiviare i risultati dei calcoli); nel caso in cui il file non esistesse, viene creato, altrimenti la linea di testo viene aggiunta in coda al file/stringa
- **SpaceFunction**: questa è una struttura che contiene due metodi, uno per definire una funzione discreta nello spazio (insieme di punti ordinato) e l'altra per interpolare linearmente tra due punti di tale funzione discreta;
- **stepFunction**: questa classe contiene il metodo per assegnare una determinata proprietà termofisica ad uno specifico strato di materiale; fa uso dell'enumeratore:
    - **MaterialProperties:** Conductivity, ThermalResistance, VolumetricCapacity, Diffusivity;

  e si appoggia alle classi **MaterialPropertiesPoint** e **SubDomain1D_Properties** per funzionare;
- **MaterialPropertiesPoint:** questa classe definisce la variabile "punto materiale" associando ad una coordinata spaziale le proprietà termofisiche dello strato a cui appartiene;
- **SubDomain1D_Properties:** questa classe consente di determinare una conduttività dipendente dalla temperatura tramite la formula

$$\lambda(T) = \lambda_0 \cdot e^{a\cdot(T-T_0)}$$

oltre che una resistenza termica (assegnata la distanza internodale), e la diffusività termica, sempre in funzione indiretta della temperatura;

- **Surface**: questa classe definisce tutte le proprietà di una superficie, sia geometriche che fisiche, come la planarità, la posizione nello spazio del suo vettore normale, la posizione dei vertici, il perimetro e l'area della superficie, l'angolo azimutale e zenitale, l'inclinazione sull'orizzonte, se è parzialmente o totalmente trasparente assegnando perimetro e area della parte trasparente, se è parzialmente o totalmente ombreggiata assegnando perimetro e area della parte soleggiata, la rugosità della superficie, l'esposizione al vento, velocità e direzione relativa del vento, se vi sono dei flussi imposti di radiazione ad orde corte e/ onde lunghe e/o convettivi i loro valori; inoltre dispone di metodi per il calcolo dell'area e l'imposizione o l'annullamento dei flussi imposti descritti in precedenza; la classe fa uso di due enumeratori:
    - **RoughnessType**: VeryRough, Rough, MediumRough, MediumSmooth, Smooth, VerySmooth;
    - **ExposureType**: notSet = 0, exposedwall = 1, semiexposedwall = 2, shelteredwall = 3, exposedlongwall = 4, exposedshortwall= 5 ,semiexposedlongwall = 6, semiexposedshortwall = 7, shelteredlongwall = 8, shelteredshortwall= 9,exposedroof10 = 10, semiexposedroof10 = 11, exposedlongroof10 = 12, semiexposedlongroof10 = 13, shelteredlongroof10 = 14, exposedroof10_30 = 15, semiexposedroof10_30 = 16, shelteredroof10_30= 17, exposedlongroof10_30= 18, semiexposedlongroof10_30 = 19, shelteredlongroof10_30= 20, exposedroof30 = 21, semiexposedroof30 = 22, shelteredroof10 = 23, shelteredroof30 = 24, exposedlongroof30 = 25, semiexposedlongroof30 = 26, shelteredlongroof30 = 27, exposedroofnopitch = 28, semiexposedroofnopitch = 29, shelteredroofnopitch = 30;
- **SurfacePhisycalProperties**: questa classe definisce le proprietà radiative di una superficie, quali l'emissività all'infrarosso e per le onde corte il coefficiente di assorbimento e di riflessione, e dei metodi di copia e verifica delle proprietà;
- **Vector**: questa classe definisce i vettori nello spazio tramite coordinate cartesiane, gli operatori algebrici tra vettori (addizione, sottrazione, prodotto interno, prodotto vettoriale e prodotto scalare-vettore, divisione scalare-vettore), la norma di un vettore e il suo vettore quadratico, il prodotto interno di tre vettori, l'angolo tra due vettori o tra due versori (più veloce), i due operatori logici di uguaglianza e differenza;

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.1.2 AppiedPhysicsEntities

La cartella *AppiedPhysicsEntities* contiene, come evincibile da Figura 12, le classi con i relativi metodi e le strutture necessarie per la descrizione e manipolazione numerica di alcuni fenomeni termofisici di base che possono esser generalizzati.
In particolare nei file contenuti si hanno le seguenti **classi**:

- **BBViewFactor**: questa classe definisce i fattori di forma per radiazione diffusa tra due superfici, detti anche fattori di forma per corpo nero (BB), introducendo una serie di metodi per calcolarli, in forma esatta o in modo numerico, a seconda della loro regolarità geometrica, del fatto che siano parzialmente non visibile l'una all'altra, ecc.;
- **BCFunction**: questa classe definisce le condizioni al contorno da assegnare ad una generica superfici note a priori e fornite dalla classe **Signal;** questa classe non viene utilizzata nella simulazione, poiché la funzione della condizione al contorno solitamente non è nota; è stata implementata solo per testare il comportamento dei metodi computazionali implementati;

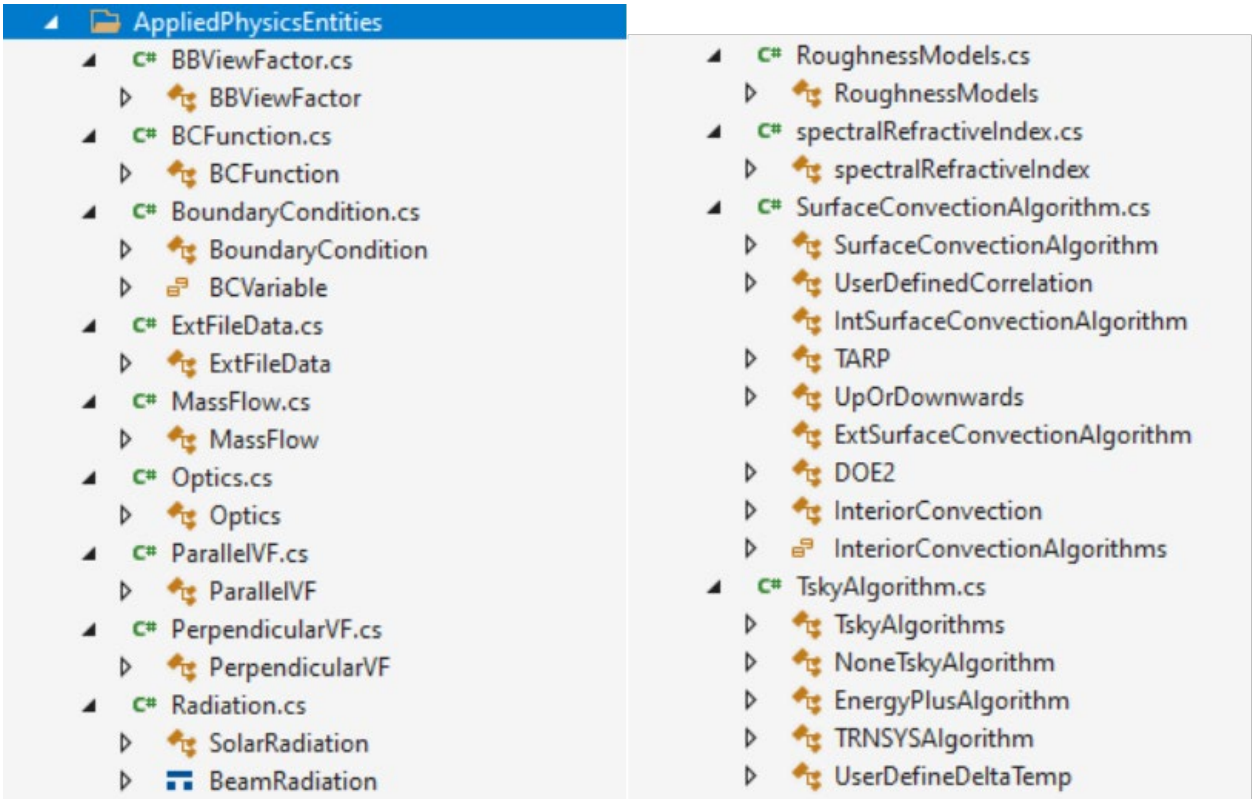


Figura 12 – Struttura della cartella *AppiedPhysicsEntities*

- **BoundaryCondition**: questa classe definisce la tipologia della condizione al contorno per una data superficie piana localizzata sul contorno di un problema conduttivo, il valore della sua grandezza caratteristica; la classe fa uso dell'enumeratore:
  - **BCVariable**: TSup (1° tipo), Flux (2° tipo), Top (3° tipo), Comb (2° + 3* tipo), NotDefined;
- **ExtFileData**: questa classe definisce dei metodi per la lettura di file contenenti dei profili temporali di dati, in cui la rima colonna contiene il timestamp;
- **MassFlow**: questa classe definisce le proprietà di un flusso di massa avvettivo monofase, in termini di portata massica, temperatura e capacità termica specifica del fluido;
- **Optics**: questa classe statica definisce un metodo per l'estrazione dell'indice di rifrazione complesso dai file di Optics;
- **ParallelVF**: questa classe contiene il metodo per il calcolo analitico del fattore di forma per radiazione diffusa tra due superfici finite parallele;
- **PerpendicularVF**: questa classe contiene il metodo per il calcolo analitico del fattore di forma per radiazione diffusa tra due superfici finite perpendicolari;
- **SolarRadiation**: questa classe contiene la definizione della radiazione solare con tutte le sue componenti, diretta, diffusa, totale, diretta orizzontale, diffusa orizzontale, totale orizzontale, collimata, angoli d'incidenza e coseno dell'angolo d'incidenza, oltre che il timestamp;
- **BeamRadiation**: questa struttura contiene la definizione della radiazione collimata (bim) sia scalare che vettoriale e l'angolo d'incidenza rispetto alla normale della superficie a cui si associa;
- **RoughnessModels**: questa classe statica contiene un metodo per la determinazione del coefficiente di rugosità di una superficie in funzione della sua definizione qualitativa: VeryRough, Rough, MediumRough, MediumSmooth, Smooth, VerySmooth; contien anche un metodo per l'attribuzione del rapporto di riflessione speculare;
- **spectralRefractiveIndex**: questa struttura contiene la definizione dell'indice di rifrazione spettrale complesso e la sua relativa lunghetta d'onda;
- **SurfaceConvectionAlgorithm**: questa struttura contiene degli algoritmi per la determinazione dei coefficienti di scambio termico superficiale convettivi per superfici comunque orientate in

convezione naturale, interne ed esterne, e in convezione forzata le esterne in presenza di vento, oltre che il coefficiente di scambio termico interno in convezione forzata in tubazioni e canali, per questi ultimi la classe fa uso dell'enumeratore:
    - **InteriorConvectionAlgorithms**: Dittus, Gnielinski;
- **TskyAlgorithms:** questa struttura contiene degli algoritmi per la determinazione della temperatura equivalente del cielo per la determinazione degli scambi termici radiativi, tramite le correlazioni implementate in EnergyPlus e in TRNSYS;

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.1.3 Solvers

La cartella *Solvers* contiene, come evincibile da Figura 13, le classi con i relativi metodi e le strutture necessarie con gli algoritmi di soluzione numerica di problemi termofisici specifici, come la conduzione termica monodimensionale in pareti multistrato, la radiazione termica in ambienti grigi, ecc..

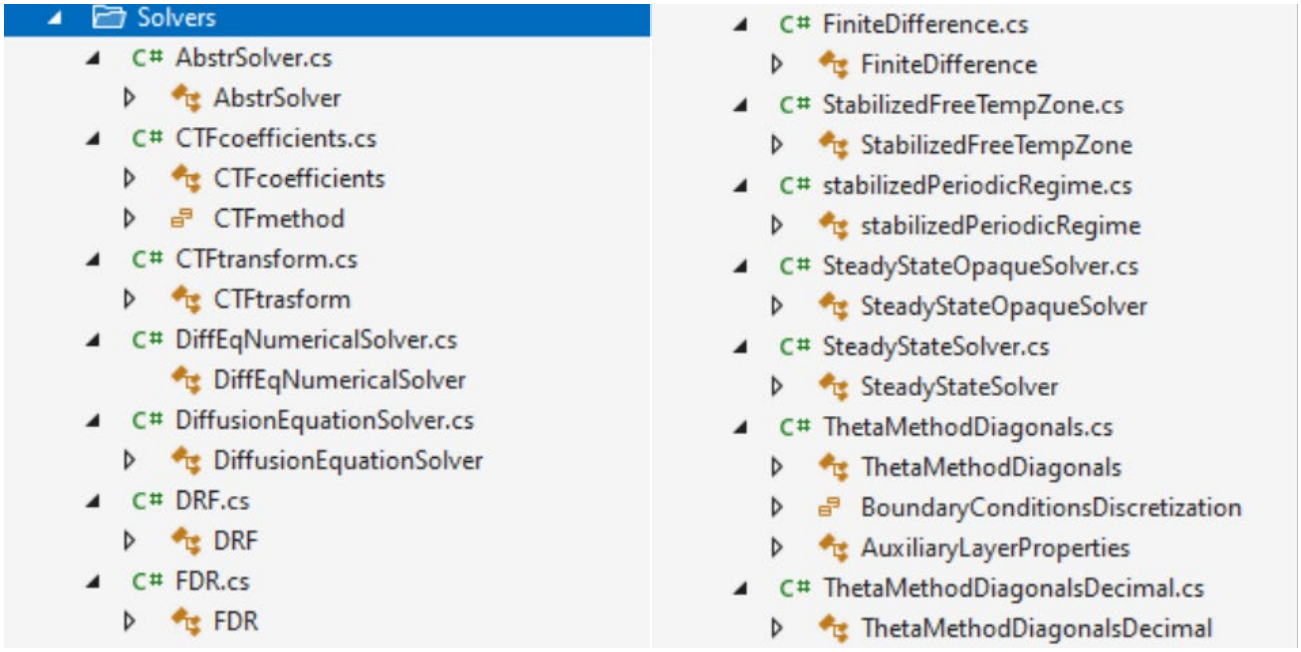


Figura 13 – Struttura della cartella *Solvers*

In particolare nei file contenuti si hanno le seguenti **classi**:

- **AbstrSolver**: questa è una classe astratta da cui ereditano tutti i diversi solutori presenti in questa cartella, che definisce i metodo fondamentali comuni dei solutori, quali: Load(), Check(), CheckAfterInit(), calculation(), timeStepConverged(), DeepCopy() per il DiffusionEquationSolver;
- **CTFcoefficients**: questa classe definisce l'insieme dei parametri che costituiscono le funzioni di trasferimento per la conduzione termica in pareti piane multistrato; tale classe fa uso dell'enumeratore:
    - **CTFmethod**: SS (State-Space method), DRF (Direct Root Finding method), FDR (Frequency Domain Regression method), NoOne;
- **CTFtrasform**: questa classe, che eredita dalla classe **DiffEqNumericalSolver**, definisce i metodi per dei flussi termici in funzione del tempo e delle condizioni al contorno assegnate, utilizzano il metodo delle funzioni di trasferimento per la conduzione termica in pareti piane multistrato;
- **DiffEqNumericalSolver:** questa classe eredita totalmente dalla classe **DiffusionEquationSolver**;

- **DiffusionEquationSolver**: questa classe definisce e gestisce il vettore dei tempi e assegna le condizioni al contorno per la risoluzione dell'equazione della conduzione termica, qualsiasi sia il metodo adottato;
- **DRF**: questa classe definisce e determina i coefficienti delle funzioni di trasferimento per la conduzione termica monodimensionale con il metodo DRF (Direct Root Finding method) originale di Mitalas;
- **FDR**: questa classe definisce e determina i coefficienti delle funzioni di trasferimento per la conduzione termica monodimensionale con il metodo FDR (Frequency Domain Regression method);
- **FiniteDifference**: questa classe definisce due metodi che ritornano il numero di Fourier e di Biot per un assegnato strato e superficie di una parete multistrato;
- **StabilizedFreeTempZone**: questa classe definisce e implementa i metodi necessari per applicare ad un dato modulo di edificio il calcolo in regime armonico della temperatura del nodo aria in regime libero, risolvendo sempre in regime armonico la conduzione nelle partizioni multistrato che lo costituiscono;
- **stabilizedPeriodicRegime**: questa classe definisce e implementa i metodi necessari per il calcolo della conduzione termica in regime periodico in pareti multistrato, ritornano non solo i flussi termici scambiati alle interfacce ma anche le distribuzioni di temperatura all'interno della parete per poter eseguire test di confronto sull'accuratezza di altri solutori numerici (funzioni di trasferimento, differenze finite, ecc.);
- **SteadyStateOpaqueSolver**: questa classe definisce e implementa i metodi necessari per il calcolo della conduzione termica in regime stazionario in pareti opache alla radiazione solare, in funzioni delle condizioni al contorno assegnate;
- **SteadyStateSolver**: questa classe definisce le proprietà necessarie per il calcolo della conduzione termica in regime stazionario monodimensionale in pareti piane;
- **ThetaMethodDiagonals**: questa classe, che eredita dalla classe **FiniteDifference**, definisce i metodi necessari per il calcolo della conduzione termica in regime variabile con le differenze finite (Theta-Method) per pareti piane in ipotesi di flusso monodimensionale, utilizzando la formulazione tridiagonale e la decomposizione LU con l'algoritmo di Thomas; tale classe fa uso dell'enumeratore:
  - **BoundaryConditionsDiscretization**: FDFirstOrder, FDsecondOrderUncentered, FV;
- **AuxiliaryLayerProperties**: classe ausiliaria alla classe **ThetaMethodDiagonals** che definisce la posizione dei nodi e la resistenza termica di strato;

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.2 SimulationManager

La libreria **SimulationManager** contiene l'implementazione del gestore della simulazione e, come evincibile da Figura 14 , è costituita da tre classi. Tutte le classi e strutture e i loro relativi metodi e proprietà appartengono al ***namespace*** **SimulationManager.**

Classi:

- **DataAnalysis**: obsoleta e non più utilizzata (resta per possibili sviluppi futuri);
- **Simulator**: questa classe pubblica definisce i metodi di gestione del processo di simulazione, tra cui:
  - **Initialize()**: l'inizializzatore della simulazione;
  - **setStartingAndEndingTime**(): fissa la data temporale di inizio e fine della simulazione;
  - **setSimulationDataAndBCs**(): verifica la presenza del file climatico, inizializza il calcolatore della radiazione solare e il gestore dei dati climatici;

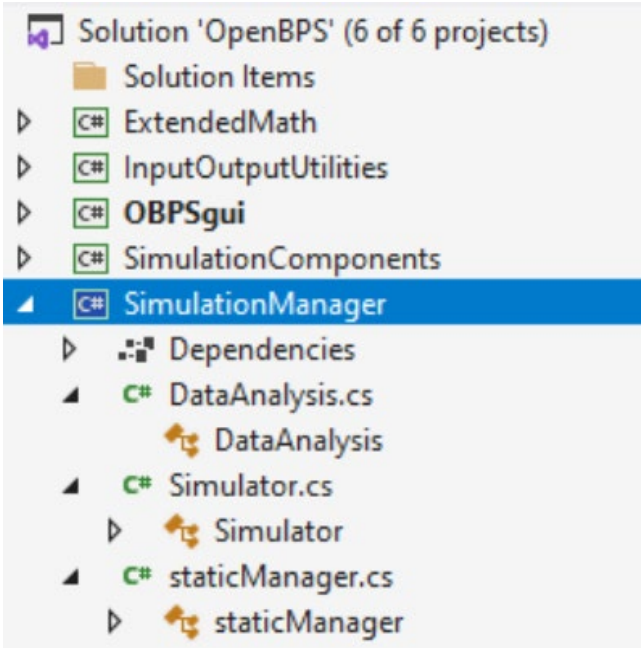


Figura 14 – Struttura della cartella *SimulationManager*

- **FindMinTimeStep**(): determina il minimo intervallo temporale di integrazione tra tutti quelli esposti dai vari componenti del sistema edificio;
- **SetMinTimeStep**(): fissa il minimo intervallo di integrazione;
- **ChangeTimeStep**: modificatore del passo temporale di integrazione;
- **setThermalBridges**(): aggiunge oi coefficiente di ponte termico al bilancio energetico del modulo di edificio per il calcolo diretto stazionario del loro contributo;
- **CreateIterativeComponentsList**(): determina quali componenti sono presenti e definisce la coda delle chiamate per gestire eventuali iterazioni;
- **CalculateSolarRadiationOnAsurface**(): determina la radiazione solare incidente su una determinata superficie;
- **RunSimulation**(): lancia la simulazione per tutto il periodo temporale definito predefinendo se il calcolo dei vari componenti il sistema edificio (fabbricato + sistemi tecnici) deve essere seriali o parallelizzato; l'esecuzione avviene , chiamando il metodo **AdvanceSimulationsTime**();
- **AdvanceSimulationsTime**(): esegue la simulazione del sistema edificio eseguendo, se richiesto, il calcolo dei componenti in parallelo in funzione del numero di processori presenti, o in serie su un singolo processore, gestisce le interazioni tra i componenti e le eventuali iterazioni per assicurare la convergenza della soluzione;

- **staticManager**: è la classe che attualmente gestisce l'interazione con la GUI di sviluppo, verificando tramite il metodo **CheckProject**() che il progetto definito dall'utente sia eseguibile (tutto quello che serve è stato specificato) e lanciando la simulazione tramite il metodo **StartSimulation**().

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.3 SimulationComponents

La libreria **SimulationComponents** contiene la descrizione dei componenti costituenti il sistema edificio, sia dell'involucro sia degli impianti tecnici. Questi sono raggruppati in sei divere cartelle in funzione della loro

specifica natura, ad esclusione dell'interfaccia di Input/Output che risiede nel file *IOutpt.cs* posta nella radice del progetto.

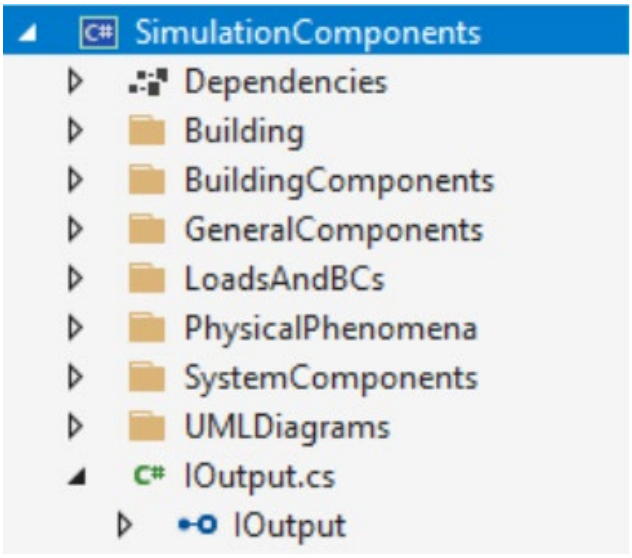


Figura 15 – Struttura della libreria SimulationComponents

L'interfaccia IOutput appartiene al *namespace* **SimulationComponents** e contiene quattro metodi:

- `GetOutputProperties()`: lista del tipo **PropertyInfo**;
- `SetOutput()`:;
- `GetResults()`: array reale;
- `ClearOutput()`:;

Come riportato in Figura 15, le sei cartelle specializzate sono: *Building*, *BuildingComponents*, *GeneralComponents*, *LoadsAndBCs*, *PhisicalPhenomena* e *SystemComponents*.

3.2.3.1 Building

La cartella *Building* contiene, come evincibile da Figura 16, le classi con i relativi metodi e le strutture necessarie per la descrizione e simulazione del sistema edificio, quale componente base contenitore. Tutte le classi e strutture e i loro relativi metodi e proprietà presenti nella cartella *Building* appartengono al ***namespace*** **SimulationComponents.**

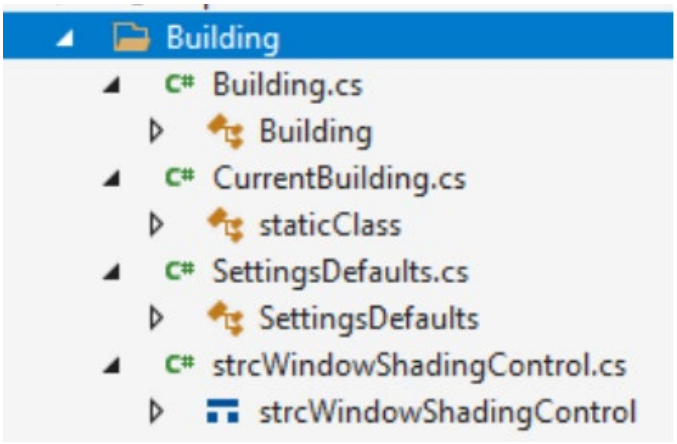


Figura 16 – Struttura della cartella *Building*

In particolare nei file contenuti si hanno le seguenti **classi**:

- **Building**: questa classe, che eredita dalla superclasse **Element**, è la classe fondamentale del codice di calcolo, in quanto classe contenitore che contiene:
    - **i campi necessari alla gestione della simulazione:** la suddivisione in zone, necessaria per la definizione dei campi di output della simulazione e allocazione delle risorse necessarie; data inizio e fine simulazione; tipo di tempo utilizzato
    - **i campi necessari alla soluzione numerica:** i parametri necessari per la verifica della convergenza su temperature e flussi termici, i parametri necessari per l'approssimazione numerica (geometrica, algebrica, per l'integrazione numerica), i parametri di controllo dei flussi d'informazione (verbose), i parametri che disattivano alcune funzioni di calcolo (semplificatori); i parametri che definisco dei limiti di calcolo (numero massimo di inter-riflessioni della radiazione solare diretta in uno spazio grigio chiuso, radiazione solare minima diretta e diffusa trasmessa dalle finestrature, ecc.); i solver default se non prescelti dall'utente, i valore id inizializzazione di tutte le variabili di stato (temperature, ecc.), il numero giorni da utilizzare per l'inizializzazione della simulazione;
    - **i campi necessari alla descrizione dell'edificio**: la posizione spaziale dell'edificio rispetto all'azimut del sito, l'associazione ai dati climatici del sito in esame, il calendario della simulazione, la presenza di ombreggiamenti, la tipologia del terreno a contatto con l'edificio, la descrizione geometrica, le caratteristiche termofisiche dei componenti d'involucro, le finestrature, i ponti termici, le infiltrazioni d'aria e i carichi termici interni, i profili d'uso dell'edificio;
    - **i campi necessari alla identificazione dei metodi matematici applicati al calcolo della prestazione di componenti dell'edificio:** la scelta dei modelli matematico/numerici per la soluzione dei problemi termofisici, se si usano o meno coefficienti di scambio termico variabili o costanti, temperature dell'aria o temperatura operante come variabile controllata, i set point del controllo dagli ambienti, la definizione del metodo di calcolo della radiazione scambiata tra superfici interne (coefficienti di scambio termico superficiale radiativi costanti, variabili con le temperature superficiali, calcolo con i fattori di forma e di radiazione mutua); , la definizione del metodo di calcolo della radiazione scambiata tra superfici esterne e ambiente circostante (coefficienti di scambio termico superficiale radiativi costanti, variabili con le temperature superficiali, calcolo con i fattori di forma e di radiazione mutua)
    - **i campi necessari alla descrizione degli impianti HVAC:** l'attribuzione dei terminali d'impianto agli ambienti, l'attribuzione dei generatori da utilizzare, la tipologia di controllo e i suoi parametri, la definizione del metodo di calcolo delle perdite di distribuzione, se l'impianto sere solo per riscaldare o solo per raffrescare o per entrambe le funzioni, la definizione per zona dei periodi di riscaldamento e raffrescamento tramite il metodo **setHVACSystemEnergyRequired**();
    - **i metodi per la gestione del processo di partizionamento dell'edificio:**
        - metodi `get` per la verifica esistenza coefficienti di assorbimento e emissione per le superfici delle partizioni (obsoleto, non più usato);
        - metodi per la gestione delle zone: `addZone()`, `CheckZoneLists()`;
        - metodi per la gestione delle partizioni: `AddPartition()`, `DeletePartition()`, `ModifyPartition()`, `FindFatherWithName()`, `FindPartitionWithIDint()`, `AssignAndSetPartitions()`, `AssignPartitionsToZones()`;
        - metodi per la verifica di consistenza dei dati: `Check()`, `CheckExternalConditions()`;
        - metodo per la generazione e attribuzione delle matrici di fattori di forma alle zone: `setViewFactorMatrices()`;
        - metodo Load() che è il metodo principale che viene utilizzato, in fase di istanziazione, per attribuire gli eventuali flussi d'aria interzona, e creare le partizione, attribuire

loro i dati, assegnare le partizioni alle zone, identificare le partizioni interne, attribuire eventuali masse interne alle zone;

- **CurrentBuilding**: classe statica di tipo **Building** che conterrà, una volta istanziata, tutti i dati della simulazione;
- **SettingsDefaults**: questa classe contiene i metodi impiegati per definire tutti i dati di default quando si crea l'oggetto **CurrentBuilding**:
    - `setAllDefaults()` metodo che chiama tutti i vari metodi specializzati seguenti auto esplicativi:
        - `setDeafultMultipleReflectionSettings();`
        - `setDefaultPartitionSolverAndGround();`
        - `setDefaultConvectionAlgorithms();`
        - `setDeafultAirHeatBalance();`
        - `setDeafultTskyAlgorithm();`
        - `setDefaultConvergenceControls();`
    - `setDefaultPartitionSolver()` chiamato da `setDefaultPartitionSolverAndGround();`
    - `setDefaultCalendar()` (obsoleto, usato solo per testing);
- **strcWindowShadingControls**: struttura che consente di associare ad ogni finestra di ogni zona un profilo di ombreggiamento da utilizzare durante la simulazione.

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.3.2 BuildingComponents

La cartella *BuildingComponents* contiene, come evincibile da Figura 17, le classi con i relativi metodi e le strutture necessarie per la descrizione e simulazione dei singoli componenti del fabbricato, che, ad esclusione di tre classi di carattere generale, sono raggruppati in tre cartelle in ordine di granularità: *Materials*, *Partitions* e *Zones*. Tutte le classi e strutture e i loro relativi metodi e proprietà presenti nella cartella *BuildingComponents* e sottocartelle appartengono al ***namespace*** **SimulationComponents.**

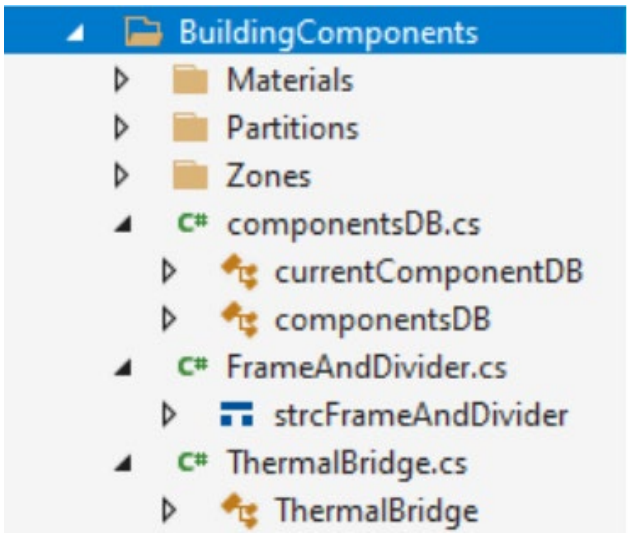


Figura 17 – Struttura della cartella *BuildingComponents*

Le tre **classi di carattere generale** sono:

- **componentsDB**: classe che eredita dalla superclasse **Element** e che istanzia la classe **Construction** associando ad una generica partizione il numero di strati componenti e i relativi materiali con le loro specifiche proprietà;
- **currentComponentDB**: classe statica di tipo **componentsDB** che conterrà, una volta istanziata, tutti i dati dello specifico componente;
- **ThermalBridge**: classe che definisce l'allocazione dei coefficienti di ponte termico lineare alle diverse zone interessate;

E anche presente la **struttura**:

- **strcFrameAndDivider**: struttura che contiene tutti i dati geometrici e termofisici relativi ai telai di elementi trasparenti, compresi il numero di divisori orizzontali e verticali, che possono avere proprietà diverse da quelle del telaio perimetrale, contiene anche le caratteristiche dei distanziatori se si ha un doppio o triplo vetro.

3.2.3.2.1 Materials

La cartella *Materials* contiene, come evincibile da Figura 18, le classi con i relativi metodi che descrivono le proprietà termofisiche dei materiali omogenei solidi, liquidi e gassosi, le miscele gassose.

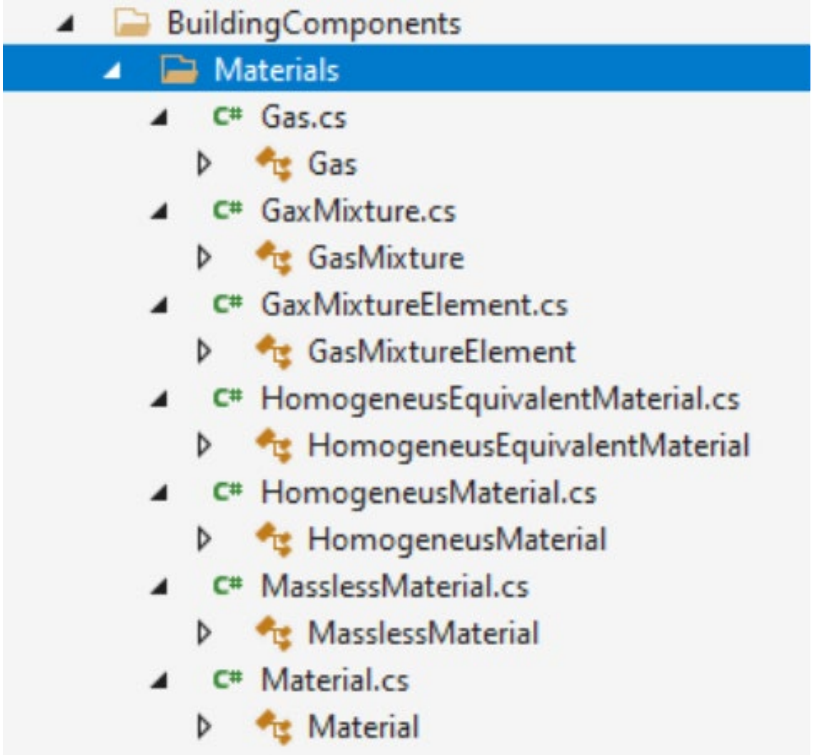


Figura 18 – Struttura della sottocartella *Materials*

In particolare nei file contenuti nella cartella si hanno le seguenti **classi**:

- **Material**: classe astratta che eredita dalla superclasse **Element** e che, a sua volta, se ereditata, assegna alla classe ereditante:
  - **Group:** una categoria;
  - **ConductivityTemperatureDependency**: eventuali parametri per una conduttività termica dipendente dalla temperatura;
  - **IsMassless**: booleano che identifica un materiale con capacità termica specifica trascurabile;
  - **i metodi**:

    - `CheckSimilarity()` che confronta due istanze della classe **Material** verificando se hanno gli stessi valori per ogni proprietà;
    - `Equals()` che verifica se un determinato oggetto è un'istanza della classe **Material**;
    - `ToString()` che ritorna il campo letterale **Name** ereditato dalla superclasse **Element**;

- **Gas**: classe che eredita dalla classe **Material** e che definisce le proprietà termofisiche di un gas, quali la massa e il peso molare, la capacità termica specifica a pressione costante, il rapporto tra le capacità termiche specifiche (indice della adiabatica), la conduttività termica e la viscosità dinamica; fornisce inoltre i metodi per la determinazione delle proprietà derivate utilizzando l'equazione dei gas ideali in condizioni standard (0 °C e 1 atm) quali massa volumica, viscosità cinematica, diffusività termica e l'estrazione delle proprietà base da un oggetto **Gas**; dispone inoltre di un metodo `CheckSimilarity()` per la verifica che due oggetti Gas siano uguali o no in ogni proprietà;
- **GasMixture**: classe che eredita dalla classe **Material** e che consente di determinare le proprietà di una miscela di gas ideali a partire dai suoi componenti note le frazioni molari, e che dispone inoltre dei metodi ereditati dalla classe **Material**;
- **GasMixtureElement**: classe che definisce i parametri `InnerGas` (identificativo del singolo elemento) e `GasMolarFraction` usati dalla classe **GasMixture**;
- **HomogenousMaterial**: classe che eredita dalla classe **Material** e che definisce le proprietà termofisiche di un solido omogeneo, quali la massa volumica, la conduttività termica e la capacità termica specifica fornisce inoltre il metodo per la determinazione della diffusività termica come proprietà derivata e un metodo per assegnare in fase di istanziazione un nome e un gruppo allo specifico oggetto **HomogenousMaterial**, oltra ai metodi ereditati dalla classe **Material**;
- **HomogenousEquivalentMaterial**: classe che eredita dalla classe **HomogenousMaterial** e che determina la conduttività termica equivalente di un materiale a partire dalla resistenza termica areica di un campione di spessore noto;
- **MasslessMaterial**: classe che eredita dalla classe **Material** e che definisce come unica proprietà del materiale la resistenza termica senza specificarne lo spessore.

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.3.2.2 Partitions

La cartella *Partitions* contiene, come evincibile da Figura 19, le classi con i relativi metodi che descrivono e determinano le prestazione dei componenti l'involucro edilizio, quali pareti, solai, tetti, finestre, ecc..

In particolare nei file contenuti nella cartella si hanno le seguenti **classi**:

- **Partition**: classe che eredita dalle classi **SimulationComponent** e **IOutput** e che serve a creare, una volta istanziata, gli oggetti partizione che rappresentano gli elementi costitutivi del fabbricato dell'edificio; la classe definisce tutte le proprietà sia geometriche che termofisiche dell'elemento costruttivo, facendo uso della classe **Construction,** e attribuisce tale elemento alla zona se elemento di involucro o alle zone se elemento interno, facendo uso della classe **Zone**; inoltre attribuisce alle sue superfici le condizioni al contorno specificate e i relativi coefficienti di scambio termico superficiale; in particolare questa classe eredita da **SimulationComponent** il vettore `variables[20]` per memorizzare nell'oggetto **Partition** le condizioni al contorno che lo interessano, cioè:
    - `variables[0]` contiene la temperatura dell'aria della zona lato A, TairA;

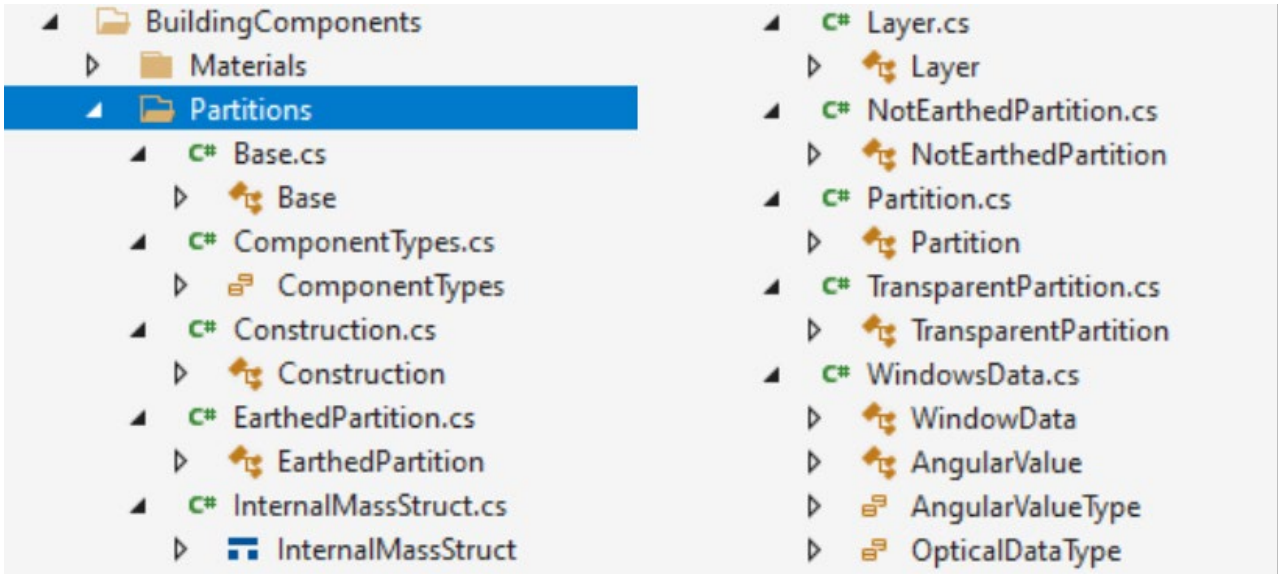


Figura 19 – Struttura della sottocartella *Partitions*

- `variables[1]` contiene la temperatura dell'aria della zona lato B, TairB;
- `variables[2]` contiene la temperatura operante della zona lato A, TopA;
- `variables[3]` contiene la temperatura operante della zona lato B, TopB;
- `variables[4]` contiene il coefficiente di scambio termico superficiale radiativo lato A, hrd_A;
- `variables[5]` contiene il coefficiente di scambio termico superficiale radiativo lato B, hrd_B;
- `variables[6]` contiene il coefficiente di scambio termico superficiale convettivo lato A, hrd_A;
- `variables[7]` contiene il coefficiente di scambio termico superficiale convettivo lato B, hrd_B;
- `variables[8]` contiene il flusso termico superficiale areico scambiato al lato A, `DensityFluxA` (positivo se diretto da A a B);
- `variables[9]` contiene il flusso termico superficiale areico scambiato al lato B, `DensityFluxB` (positivo se diretto da A a B);
- `variables[10]` contiene il coefficiente di scambio termico superficiale radiativo rispetto al cielo, hrdSky;

i metodi più importanti di questa classe sono:

- `Initialize()`: chiamato in fase di istanziazione dell'oggetto partizione assegna a questo tutte le proprietà in funzione delle scelte dell'utente (tipo di solver, tipo di condizioni al contorno ecc.);
- `isPartOfUncontrolledZone()`: verifica se la partizione è interna ad una zona non climatizzata;
- `TempTrasmittance()`: calcola la trasmittanza termica della partizione;
- `SetZoneAInfo()`: assegna le corrette condizioni al contorno al lato A della partizione;
- `SetZoneBInfo()`: assegna le corrette condizioni al contorno al lato B della partizione;

- `setDRFCeoff()`: richiama il generatore dei coefficienti delle funzioni di trasferimento per la conduzione (se selezionato come solver) e, una volta determinati, li memorizza nell'oggetto partione;
- `SetOptTimeStep()`: determina time step interno ottimale per la specifica partizione;
- `Calculate()`: esegue ogniqualvolta invocato dal simulation manager il calcolo della prestazione della partizione nel time step di simulazione, assegnando le condizioni al contorno attuali, chiamando il solver definito e verificando la convergenza della soluzione numerica (se necessario);
- `SetTopA()`: calcola la temperatura operante della zona A in funzione della temperatura dell'aria e media radiante;
- `SetTopB()`: calcola la temperatura operante della zona B in funzione della temperatura dell'aria e media radiante;
- `TimeConverged()`: verifica la convergenza della soluzione numerica la termine del time step di simulazione (usato per problemi non lineari);

corredati da un insieme di metodi per l'estrazione dei risultai dettagliati dall'oggetto (ad esempio la distribuzione interna di temperatura);

- **Base**: classe che eredita dalla classe **Partition** aggiunge a questa la definizione dei parametri che servono per descrivere l'interazione di un pavimento su terreno con quest'ultimo, come la conduttività termica del terreno, l'area in pianta e il perimetro;
- **EarthedPartition**: classe che eredita dalla classe **Partition** aggiunge a questa la definizione dei parametri che servono per descrivere l'interazione di una partizione quando posta contro il terreno, quali l'affondamento rispetto al piano di campagna, e inizializza la condizione al contorno con la temperatura del terreno;
- **NotEarthedPartition**: classe che eredita dalla classe **Partition** e che definisce la presenza o meno di aperture opache e/o trasparenti nella partizione, tramite metodi che consentono di aggiungere tali elementi all'oggetto quando istanziato, di determinare in tal caso l'area netta della partizione, di assegnare la presenza o meno di ombre e la loro geometria, di assegnare i parametri che ne determinano l'esposizione;
- **TransparentPartition**: classe che eredita dalla classe **NotEarthedPartition** e che attribuisce all'oggetto partizione trasparente tutte le proprietà necessarie per determinare, a partire dalla caratterizzazione dei suoi componenti (caratteristiche geometriche, termofisiche e ottiche dei pannelli trasparenti, del gas contenuto nelle intercapedini, del telaio e dei distanziatori e delle loro geometrie) e dalla presenza di sistemi schermanti, la rete elettrica equivalente (ricalcolata ad ogni intervallo temporale essendo il problema non lineare: resistenza dell'intercapedine varia con la temperatura) che consente di determinare il flusso termico trasmesso dovuto sia alla differenza di temperature sia alla radiazione solare assorbita, sia proveniente dall'esterno che dall'interno; in particolare questa classe eredita da **Parition** il vettore `variables[20]` per memorizzare nell'oggetto **TransparentPartition** le condizioni al contorno aggiuntive che lo interessano, cioè:
  - `variables[11]` contiene la trasmittanza radiativa attuale, `CurrentAngTramisttance;`
  - `variables[12]` contiene la assorbanza angolare della radiazione diretta attuale, `CurrentOutAngDirAbs;`
  - `variables[13]` contiene l'area attualmente illuminata dal sole, `CurrentSunlitarea;`
  - `variables[14]` contiene l'irradianza solare diretta incidente attuale, `CurrentExtIncBeamRad;`
  - `variables[15]` contiene l'irradianza solare diffusa incidente attuale, `CurrentExtIncDiffRad;`

  - `variables[16]` contiene l'irradianza solare normale diretta trasmessa attuale, `CurrentExtTrasmDirectNormIrradiance;`
  - `variables[17]` contiene il flusso solare diretto trasmesso attuale, `CurrentExtTrasmDirSolarFlux;`
  - `variables[18]` contiene l'irradianza solare diffusa trasmessa attuale, `CurrentExtTrasmDiffRad;`

- **Construction**: classe che eredita dalla classe **Element** e che costruisce la struttura di una partizione ordinando gli strati definiti dalla classe Layer dalla lato A al lato B, distinguendo tra strati solidi e intercapedini e assegnando alle due superfici, A e B, le loro specifiche proprietà, determinando la conduttanza termica e la capacità termica areica; se gli strati sono trasparenti include le proprietà definite nella classe **WindowData** e associa all'oggetto, tramite il metodo `GetAngValue()` tutte le proprietà ottiche necessarie in funzione dell'angolo di incidenza della radiazione; la classe include inoltre un metodo per definire una ripartizione spaziale ottimale di ogni strato per l'impiego di metodi numerici discreti (differenze finite o volumi finiti) e la costante temporale minima (strato più critico) del sistema;
- **Layer**: classe che eredita dalla classe **Element** e che associa un materiale al generico strato, definito da uno spessore, attribuendogli le proprietà termofisiche del materiale e calcolandone la resistenza termica areica; inoltre definisce se lo staro è interno a una partizione o superficiale, nel qual caso assegna anche le proprietà della superficie;
- **WindowData**: questa classe definisce le proprietà addizionali che deve avere una partizione trasparente alla radiazione solare e luminosa, in funzione del tipo di dati da utilizzare, così come definiti dall'enumeratore **OpticalDataType**, oltre che i dati relativi al telaio e ai distanziatori, come le loro aree superficiali le trasmittanze termiche, i coefficienti di assorbimento della radiazione, ecc.; per le laste trasparenti, se del caso, assegna tutte le proprietà ottiche e radiative in funzione dell'angolo d'incidenza della radiazione tramite la classe **AngularValue**;
- **AngularValue**: questa classe definisce le proprietà ottiche e radiative in funzione dell'angolo d'incidenza della radiazione per un sistema trasparente composto fino a un massimo di 6 lastre (mentre trasmittanza e riflettanza sono uniche e riferite all'intero sistema, i coefficienti di assorbimento devono esser specifici al singolo strato);

dalla **struttura**:

- **InternalMassStruct**:

dai seguenti **enumeratori**:

- **ComponentTypes**: Window, Door, Roof, VertPart, HorPart;
- **AngularValueType**: Tsol, AbsRadFromA, AbsRadFromB, AbsTotRadFromA, AbsTotRadFromB, Rfsol, Rbsol, Tvis, Rfvis, Rbvis, SHGC
- **OpticalDataType**: SpectralAverage, Spectral, BSDF, AngularSpectralAverage

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.3.2.3 Zones

La cartella *Zones* contiene, come evincibile da Figura 20, le classi con i relativi metodi che descrivono e determinano le prestazioni della singola zona dell'edificio, risolvendo i bilanci energetici e di massa per il nodo aria e i bilanci radiativi tre le superfici della zona.

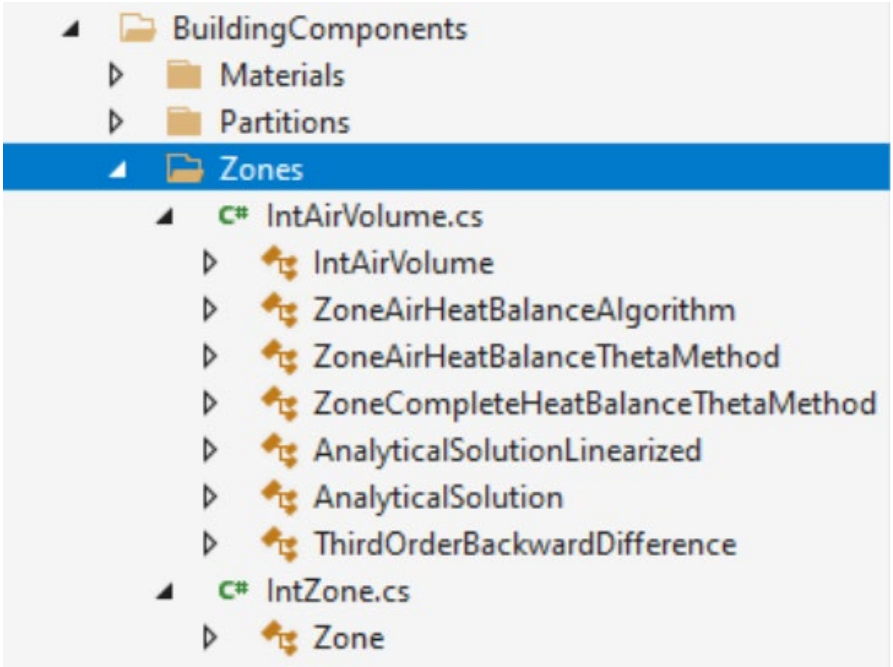


Figura 20 – Struttura della sottocartella *Partitions*

In particolare nei file contenuti nella cartella si hanno le seguenti **classi**:

- **Zone**: classe che eredita dalla classe **Element** e che definisce i parametri che caratterizzano una specifica zona in cui è suddiviso l'edificio, utilizzando la classe **IntAirVolume** per quanto riguarda tutte le proprietà e i metodi relativi alla risoluzione del bilancio energetico e di massa integrali per il volume di controllo associato alla zona, a cui aggiunge altre proprietà quali il numero e tipo di partizioni presenti con tutte le loro caratteristiche (esterne, interne, con o senza aperture trasparenti/opache, ecc.), quali il numero e tipo di emettitori del sistema HVAC eventualmente presenti nella zona e la quota di potenza scambiata per radiazione, il tipo numero e potenza di eventuali sorgenti interne, una capacità termica per unità di pavimento definibile per tener conto di eventuale mobilia o altro presente nella zona, le matrici di trasmissione e assorbimento per il trattamento delle riflessioni radiative multiple in un ambiente grigio, se si vuole fare il calcolo dettagliato dello scambio termico radiativo, il fattore di rilassamento per la gestione della convergenza, un termostato e un umidostato per il controllo della temperatura e dell'umidità dell'aria; questa classe è poi dotata di metodi che calcolano il volume componendo geometricamente le partizioni, fissano i profili dei guadagni interni, calcolano le nuove temperature superficiali dovute al solo scambio termico radiativo da utilizzare come condizioni al contorno per la risoluzione del bilancio energetico sul nodo aria; calcolano i coefficienti di scambio termico superficiale in funzioni del tipo di condizione al contorno assegnata e della scelta o meno di linearizzare lo scambio termico radiativo, verificano la consistenza dei dati che definiscono la zona stessa;
- **IntAirVolume**: classe che eredita dalle classi **SimulationComponent** e **IOutput** e che serve a definire tutti i parametri che servono per la risoluzione del bilancio energetico e di massa integrali per il volume di controllo associato alla zona contenete aria umida, cioè le proprietà termofisiche dell'aria umida contenuta, compreso una capacità termica modificata per tener conto di eventuale mobilia o altri oggetti e se si usano proprietà costanti o variabili i funzione di temperatura e umidità, i valori dei set point di temperatura e umidità per il controllo climatico, la definizione di quale solutore va impiegato per integrare l'equazione di bilancio energetico e di massa, se la zona è servita o meno da un sistema HVAC e quindi è termicamente o igrotermicamente controllata, se si sta calcolando il carico termico ideale o eseguendo una simulazione con un sistema di controllo con isteresi e potenza di impianto finita; inoltre la classe è dotata di metodi che oltre che consentire l'attribuzione delle proprietà e delle condizioni al contorno, consentono di determinare l'evoluzione temporale di

temperatura e umidità dell'aria interna e determinare i flussi energetici scambiati nelle loro varie componenti (convettiva, avvettiva, sensibile, latente, ecc.); infine questa classe eredita da **SimulationComponent** il vettore `variables[]` (che viene ridimensionato a 34) per memorizzare nell'oggetto **IntAirVolume** le quantità del bilancio energetico che interessa esporre in output, cioè:

- `variables[0]` contiene la temperatura della zona (aria o operante), IntAirTemp;
- `variables[1]` contiene l'umidità massica dell'aria della zona, `AbsHumidity`;
- `variables[2]` contiene l'umidità relativa dell'aria della zona, `RH`;
- `variables[3]` contiene la pressione dell'aria della zona, `Pressure`;
- `variables[4]` contiene la somma delle portate massiche d'aria in ingresso alla zona, `Sum_MassRate`;
- `variables[5]` contiene la somma delle portate di capacità termica dell'aria in ingresso alla zona, `Sum_MassRateTimesCp`;
- `variables[6]` la somma delle portate di capacità termica per la temperatura dell'aria in ingresso alla zona, `Sum_MassRateTimesCpTimesTemp`;
- `variables[7]` contiene il flusso termico totale relativo ai guadagni interni, `TotalIntGainsFlux`;
- `variables[8]` contiene il flusso termico totale relativo ai guadagni interni per radiazione solare diffusa ridistribuita usando i fattori di radiazione mutua, `TotalDiffSolarGainsFlux`;
- `variables[9]` contiene il flusso termico totale relativo ai guadagni interni per radiazione solare diretta ridistribuita dopo la prima incidenza tramite i fattori di radiazione mutua, `TotalBeamSolarGainsFlux`;
- `variables[10]` contiene il flusso termico totale relativo ai guadagni interni per radiazione solare diffusa nell'ipotesi di distribuzione omogenea, `TotalExtDiffSolarGainsFlux`;
- `variables[11]` contiene il flusso termico totale relativo ai guadagni interni per radiazione solare diretta nell'ipotesi di distribuzione omogenea, `TotalExtBeamSolarGainsFlux`;
- `variables[12]` contiene la potenza termica (con segno) richiesta al sistema HVAC, `HVACSystemPowerRequired`;
- `variables[13]` contiene l'energia termica richiesta dalla zona al sistema HVAC, `HVACSystemEnergyRequired`;
- `variables[14]` contiene la potenza erogata dal sistema HVAC alla zona, `HVACSystemPower`;
- `variables[15]` contiene l'energia erogata dal sistema HVAC alla zona, `HVACSystemEnergy`;
- `variables[16]` contiene la somma dei prodotti area della superfice per coefficiente di scambio termico superficiale convettivo, `Sum_AiHcvi`;
- `variables[17]` contiene la somma dei prodotti area della superfice per coefficiente di scambio termico superficiale convettivo per temperatura delle superfici, `Sum_AiHcviTi`;
- `variables[18]` contiene la potenza termica per riscaldamento richiesta al sistema HVAC, `HVACSystemHeatingPowerRequired`;
- `variables[19]` contiene la potenza termica per raffrescamento richiesta al sistema HVAC, `HVACSystemCoolingPowerRequired`;
- `variables[20]` contiene la temperatura media radiante della zona, `Tmr`;
- `variables[21]` contiene la temperatura operante della zona, `Top`;

- `variables[22]` contiene la somma dei coefficienti di globali di scambio termico per ponte termico, `Sum_Htbi`;
- `variables[23]` contiene la somma dei prodotti coefficienti di globali di scambio termico per ponte termico per la temperatura interna, `Sum_HtbiTi`;
- `variables[24]` contiene la potenza richiesta dalla zona per effetto di un surriscaldamento o sottoraffreddamento, `HVACSystemOverHeatingOrColling`;
- `variables[25]` contiene la massa volumica attuale della zona, `CurrentDryAirDensity`;
- `variables[26]` contiene la capacità termica specifica dell'aria umida attuale della zona, `CurrentMoistAirCp`;
- `variables[27]` contiene la portata massica di vapore richiesta al sistema HVAC per umidificare, `HVACSystemWaterRequired`;
- `variables[28]` contiene la portata massica di vapore associata ai carichi latenti, `TotalLatentGainsMassFlow`;
- `variables[29]` contiene la temperatura interna (aria o operante) della zona con impianto fermo o senza impianto, `IntAirTempNoHVAC`;
- `variables[30]` contiene l'umidità massica dell'aria della zona con impianto fermo o senza impianto, `AbsHumidityNoHVA`;
- `variables[31]` contiene la temperatura operante della zona con impianto fermo o senza impianto, `TopNoHVAC`;
- `variables[32]` contiene l'energia richiesta all'impianto HVAC per compensare i carichi latenti, `HVACSystemLatentEnergyRequired`;
- `variables[33]` contiene i carichi latenti totali calcolati come portata per differenza di entalpia specifica, `TotalLatentGainsMassFlowDotEnthalpyDiff`;

i **metodi** più importanti di questa classe sono:

- `Initialize()`: questo metodo inizializza l'oggetto una volta istanziata la classe attribuendogli tutti i valore delle proprietà che lo definiscono, compresi i parametri di controllo della simulazione, definisce e alloca l'area di memoria necessaria per l'archiviazione dei valori richiesti in output, tramite un insieme di metodi come `AddConvFlux()`, `AddDiffSolarFlux()`, ecc., e in particolare consente di trasformare l'equazione di bilancio entalpico sensibile per il nodo aria nell'equazione di bilancio termico per l'ambiente, sostituendo alla variabile temperature dell'aria la variabile temperatura operativa e coefficienti di scambio termico superficiale misti convettivo/radiativi (assumendo come input la temperature media radiante e i coefficienti radiativi forniti dalla classe chiamante **Zone**), utile per il calcolo del carico termico ideale, più che per le simulazioni;
- `Calculate()`: metodo che, quando invocato, esegue le integrazioni del bilancio energetico e di massa per il passo di simulazione corrente attribuendo alle condizioni al contorno i valori attuali;
- `TimeConverged()`: metodo che verifica il raggiungimento della convergenza numerica nei problemi non lineari;

- **ZoneAirHeatBalanceAlgorithm**: questa classe definisce i metodi virtuali principali comuni a i diversi risolutori applicabili all'equazione differenziale ordinaria che descrive il volume di controllo contenente aria umida; in particolare:
  - `Initialize()`**:** inizializza i diversi risolutori in base alle proprie esigenze;

  - `calculation()`: esegue l'integrazione al dato passo temporale
  - `timeConverged()`: verifica il raggiungimento della convergenza numerica;
- **ZoneAirHeatBalanceThetaMethod**: questa classe, **limitata al solo bilancio termico**, implementa i metodi della classe **ZoneAirHeatBalanceAlgorithm** specializzandoli tramite *override* degli stessi e aggiungendo un metodo specializzato per l'attuazione del controllo della temperatura e dell'umidità; l'integrazione numerica avviene l'approssimazione dell'equazione differenziale con due equazioni alle differenze una in avanti e una indietro nel tempo, pesata dal parametro theta (0-1);
- **ZoneCompleteHeatBalanceThetaMethod**: questa classe è un'estensione della precedente e effettua contestualmente l'integrazione sia del bilancio energetico (bilancio entalpico dell'aria umida) sia del bilancio di massa del vapore; implementa i metodi della classe **ZoneAirHeatBalanceAlgorithm** specializzandoli tramite *override* degli stessi e utilizzando per il controllo della temperatura e dell'umidità il metodo definito nella classe precedente;
- **AnayticalSolution**: questa classe definisce un metodo che opera l'integrazione analitica dell'equazione differenziale ordinaria rispetto al tempo nell'ipotesi di coefficienti e forzante costanti nell'intervallo di integrazione; attualmente è specializzata rispetto al solo bilancio termico e non considera la potenza termica erogata dall'impianto;
- **AnayticalSolutionLinearized**: questa classe definisce un metodo che opera l'integrazione analitica dell'equazione differenziale ordinaria rispetto al tempo nell'ipotesi coefficienti costanti e di variazione lineare nel tempo della forzante nell'intervallo di integrazione; attualmente è specializzata rispetto al solo bilancio termico e non considera la potenza termica erogata dall'impianto all'inizio dell'intervallo ma solo alla fine;
- **ThirdOrderBackwardDifference**: questa classe definisce un metodo che opera l'integrazione numerica dell'equazione differenziale ordinaria rispetto al tempo utilizzando uno schema alle differenze finite del terzo ordine all'indietro, su quattro punti; attualmente è specializzata rispetto al solo bilancio termico e non considera la potenza termica erogata dall'impianto.

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.3.3 GeneralComponents

La cartella *GeneralComponents* contiene, come evincibile da Figura 21, tre classi di carattere generale che definiscono alcuni campi di carattere generale che vengono condivisi tra più classi, come dei parametri identificati dello oggetto generato da una classe istanziata e un'area di memorizzazione di quantità specifiche necessarie ad alcune classi.

Tutte le classi e strutture e i loro relativi metodi e proprietà presenti nella cartella *GeneralComponents* appartengono al ***namespace*** **SimulationComponents.**

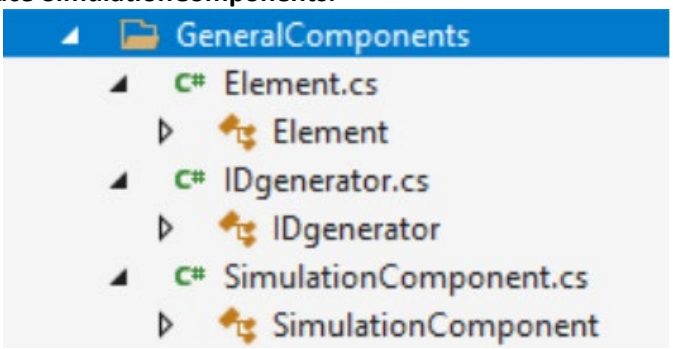


Figura 21 – Struttura della cartella *GeneralComponents*

In particolare nei file contenuti nella cartella si hanno le seguenti **classi**:

- **Element**: questa è una superclasse generica che, se ereditata, assegna alla classe ereditante:
    - **Name:** un nome;
    - **IDn**: un identificativo letterale;
    - **IDint**: un identificativo numerico;
    - **MessageForTheUser**: un eventuale messaggio per l'interfaccia utente;
    - i seguenti metodi:
        - `setIDs()` che utilizza la classe **IDgenerator** per generare l'identificativo numerico;
        - `ResetIDint()` per assegnare un nuovo identificativo numerico;
        - `CalculateLog()` e `CalculateLogVirtual()` per calcolare il logaritmo naturale (obsoleti, usati solo nel progetto di testing TestProjectNew):
        - `ShowMessageForTheUser()` che mostra il contenuto del messaggio per l'utente in una MessageBox; (da chiamare solo dall'interfaccia utente);
        - `Check()` che ritorna un booleano dopo aver attribuito l'identificativo numerico (metodo virtuale, cioè che può essere modificato dalla classe ereditante);
        - `Load()` metodo virtuale (cioè che può essere modificato dalla classe ereditante) vuoto;
        - `WriteEndOfSimulationMessage()` metodo virtuale (cioè che può essere modificato dalla classe ereditante) vuoto;
        - `ToString()` che ritorna il campo letterale **Name**;

  Questa classe viene ereditata dalle classi: **Building, Zone, Construction, Layer, Material, ThermalBridge, Ground, AirInfiltration, IntergalGain** e **SimulationComponents**.
- **IDgenerator**: classe che contiene il metodo `GetNewID()` che attribuisce un numero intero identificativo verificando che non sia stato già assegnato ad una zona o a una partizione;
- **SimulationComponent**: classe che eredita dalla classe **Element** e che definisce il vettore `variables[]`, dimensionato nel metodo virtuale `Initialize` [20], che servirà nella classi che a loro volta la ereditano, per memorizzare nell'oggetto le o condizioni al contorno che lo interessano (classi **Partition** e **TransparentPartition**) o i valori delle quantità rilevanti del bilancio energetico e di massa che vi vuole esporre all'output (in questo caso, classe **IntAirVolume**, tramite l'override del metodo `Initialize` viene dimesionato [34]).

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.3.4 LoadsAndBCs

La cartella *LoadsAndBCs* contiene, come evincibile da Figura 22, le classi, con i relativi metodi, necessarie per la descrizione di quelle che vengono sinteticamente qui chiamate condizioni al contorno per il bilancio termoigrometrico dell'ambiente, cioè la definizione dei profili orari dei carichi termici, delle infiltrazioni d'aria, della radiazione solare, dell'occupazione, ecc. Tutte le classi e strutture e i loro relativi metodi e proprietà presenti nella cartella *GeneralComponents* e sottocatella *Schedule* appartengono al ***namespace*** **SimulationComponents.**

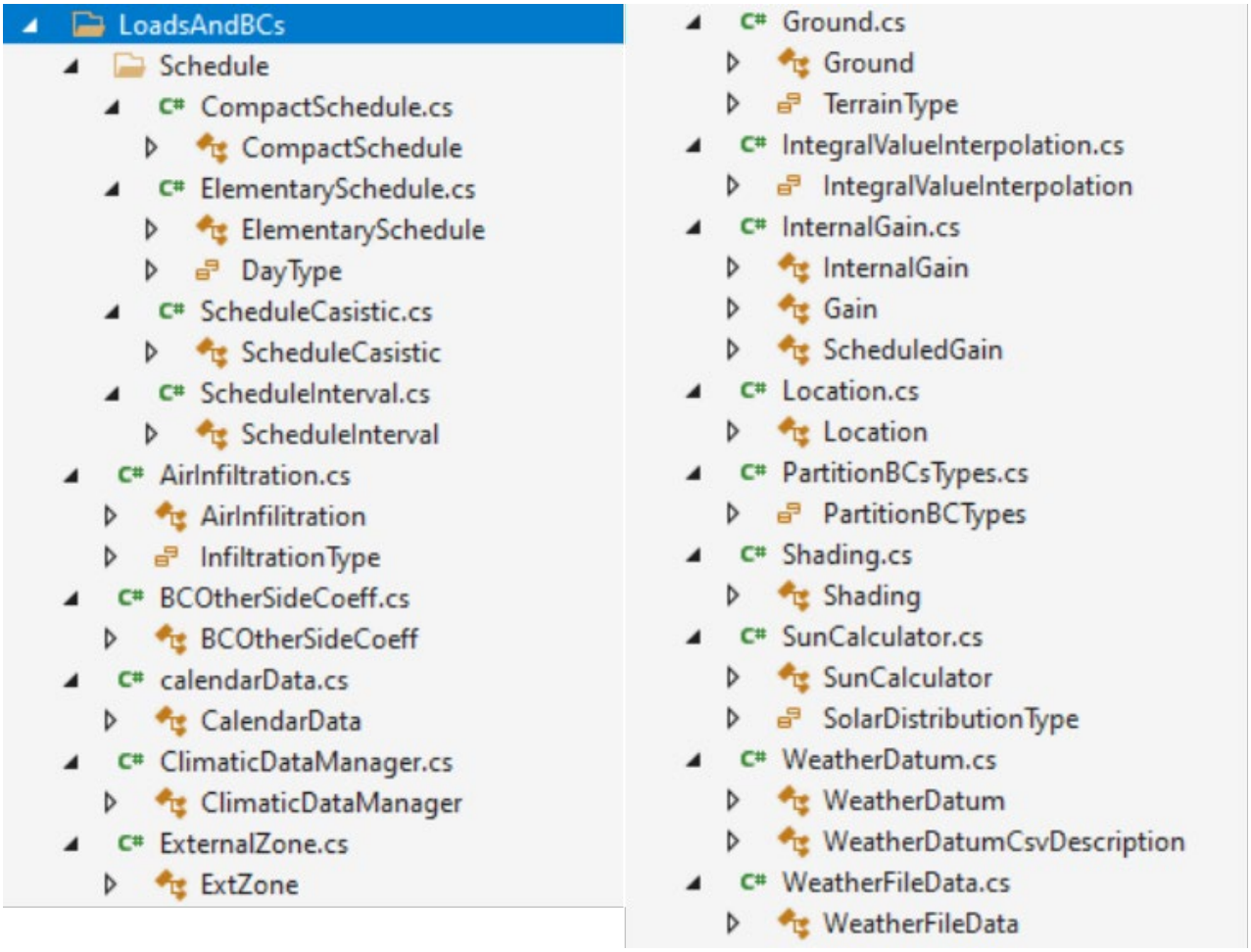


Figura 22 – Struttura della cartella *LoadsAndBCs*

In particolare nei file contenuti nella cartella si hanno le seguenti **classi**:

- **AirInfilitration** : classe che eredita dalla superclasse **Element** e che definisce e attribuisce i profili orari di portate d'aria esterna di infiltrazione in termini di volumi/ora, classe che utilizza **l'enumeratore**:
  - **InfiltrationType**: CubicMeterPerSecond, CubicMeterPerSecondPerArea, CubicMeterPerSecondPerVolume, AirChangePerHour, KgPerSecond;
- **BCOtherSideCoeff**: classe che eventualmente attribuisce dei profili orari di coefficienti di scambio termico superficiale esterni a scopi comparativi;
- **CalendarData**: classe che definisce il calendario della simulazione tramite i profili di accensione e spegnimento dei sistemi impiantistici, gironi di vacanza e l'eventuale commutazione riscaldamento/raffrescamento;
- **ClimaticDataManager**: classe che gestisce i dati climatici associati al sito dell'edifico, verificando la "granularità" temporale dei dati forniti e confrontandola con intervallo di interazione utilizzato per quindi gestire nel modo più opportuno l'interpolazione dei dati stessi secondo richiesta facendo uso dell'enumeratore **IntegralValueInterpolation**; inoltre la classe, tramite un metodo specifico, verifica la consistenza del dato di radiazione solare confrontandolo con la massima possibile (l'extraterrestre), eventualmente correggendola e segnalando l'inconveniente;
- **ExtZone**: classe che definisce, aggiorna e calcola le condizioni al contorno legate ai dati climatici per le zone che affacciano sull'ambiente esterno, calcolando in particolare la temperatura equivalente del cielo per gli scambi termici radiativi ad onde lunghe;

- **Ground**: classe che eredita dalla superclasse **Element** e che definisce tutte le proprietà termofisiche (conduttività e diffusività termica) del terreno circostante l'edifico, tra cui anche il suo coefficiente di albedo, per attribuirgli una temperatura, costante o variabile in funzione delle scelte dell'utenza; inoltre la classe attribuisce al "terreno" circostante un'orografia territoriale tramite **l'enumeratore**:
    - **TerrainType**: Country, Suburbs, City, Ocean, Urban;
- **InternalGain** : questo metodo, che eredita dalla superclasse **Element**, definisce, tramite l'impiego del metodo **Gain**, i profili orari dei guadagni interni e ne attribuisce i valori orari tramite dei specifici metodi *get_variabile* in termini di produzione di vapore e di portata entalpica latente, sorgenti termiche di natura convettiva, sorgenti termiche di natura radiativa;
- **Gain**: questo metodo definisce il generico profilo associandogli un identificatore, un booleano che specifica se il valore attribuito è costante nel tempo o variabile, il nome dell'eventuale file che lo riporta, la data corrispondente all'inizio dell'attribuzione del valore riportato e quella corrispondente al suo termine, un dato addizionale che complementa il dato principale (ad esempio il dato principale è la portata di vapore d'acqua e il dato secondario è l'entalpia specifica di tale vapore);
- **ScheduledGain**: questa classe, utilizzata dalla classe **intZone**, serve per istanziare la struttura dei profili di guadagni interni tramite la definizione di tre campi: fazione di latente, frazione di sensibile e frazione di convettivo, che servono per separare un unico profilo in termini di guadagno interno tote nelle tre componenti citate;
- **Location**: questa classe definisce e riporta tutti i parametri caratteristici del sito dell'edificio, cioè, nome, latitudine, longitudine, altitudine, fuso orario di appartenenza;
- **Shading**: questa classe, che eredita dalla superclasse **SimulationComponent** ed è utilizzata dalla classe **Building**, definisce il profilo orario di ombreggiamento di una determinata superficie associando un profilo contente la distribuzione oraria di ombreggiamento a una superficie di quello specifico componente dell'edificio;
- **SunCalculator**: classe che definisce e espone tutti i metodi necessari per il calcolo dell'irradianza solare su una superficie comunque orientata e tutti i parametri ad essa associati (angolo di incidenza, ecc.), avendo prima, tramite il metodo `SetSunCalculator()`, caricato in opportuna variabile tuti i valori del file climatico associato alla località ed avendo poi, se del caso, rielaborato dati di radiazione disponibili in funzione dei modelli prescelti (ad esempio cielo isotropo, cielo anisotropo, ecc.), inoltre consente di calcolare l'irradianza solare a ciel sereno tramite le trasmittanze a radiazione diretta e diffusa del cielo; questa classe fa uso dell**'enumeratore**
    - **SolarDistributionType**: MinimalShadowing, FullExterior, FullInteriorAndExterior, FullExteriorWithReflections, FullInteriorAndExteriorWithReflections;
- **WeatherDatum**: classe che definisce la struttura di tutti i possibili campi della variabile dei dati climatici e che verrà impiegata nella simulazione per definire le condizioni al contorno climatiche;
- **WeatherDatumCsvDescription**: classe che associa alla struttura di tutti i possibili campi della variabile dei dati climatici eventualmente presenti in un file climatico di formato non standard, che impieghi come separatore di campo la virgola (.csv), la presenza o meno di quel dato;
- **WeatherFileData**: classe che definisce la struttura e i campi che potrebbe contenere il file dei dati climatici e una serie di descrittori che specificano il file di origine, il suo formato, se sono presenti i minuti o se occorre aggiungerli al timestamp che caratterizza il dato, se è presente l'irradianza diretta normale o no, se è presente la radiazione diffusa o no, se sono presenti sia la radiazione diffusa e la totale sull'orizzontale oltre che la diretta, quale utilizzare tra le due per evitare inconsistenze durante il calcolo, quale tipo di modello di calcolo dell'angol d'incidenza utilizzare tar quelli disponibili, e alto; questa classe fa uso dell'enumeratore **IntegralValueInterpolation.**

In particolare nei file contenuti nella cartella si hanno i seguenti **enumeratori**:

- **IntegralValueInterpolation**: Default, LinearInterpForIntegrals, LinearInterpAvgAtmosphericTransmittance, TRNSYSMethod;
- **PartitionBCTypes**: Outdoors, Ground, Tfixed, Zone, Adiabatic, ExternalComponent, IntZoneSymmetry

Il progetto contiene inoltre una cartella **Schedule** che a sua volta contiene dei metodi che definiscono diverse tipologie di profili, nello specifico:

- **CompactSchedule**: classe che definisce il generico profilo come una lista ordinata di valori;
- **ElementarySchedule**: classe che fa uso della classe **ScheduleCasistic** per definire contestualmente più profili e che fa uso dell'enumeratore:
  - **DayType**: Weekdays, Weekends, Holidays, AllDays, AllOtherDays, WinterDesignDay, SummerDesignDay;
- **ScheduleCasistic**: classe che definisce contestualmente tre diversi profili, uno associato al giorno della settimana tramite l'enumeratore di sistema DayOfWeek, un altro associato al tipo di giorno definito dall'enumeratore **DayType**, un ultimo associato alla durata del valore assegnato nel giorno (max 24 ore) tramite la classe **ScheduleInterval**;
- **ScheduleInterval**: classe che definisce per un dato giorno la durata di un valore assegnato all'interno delle 24 ore.

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.3.5 PhysicalPhenomena

La cartella *PhysicalPhenomena* contiene, come evincibile da Figura 23, le classi, con i relativi metodi, necessarie per la descrizione.

Tutte le classi e strutture e i loro relativi metodi e proprietà presenti nella cartella *PhysicalPhenomena Schedule* appartengono al ***namespace*** **SimulationComponents,** salvo la classe **StandardFlowComponent (**in via di sviluppo**)** che appartiene al suo sottospazio dal *namespace* **SimulationComponents.Physical Phenomena.**

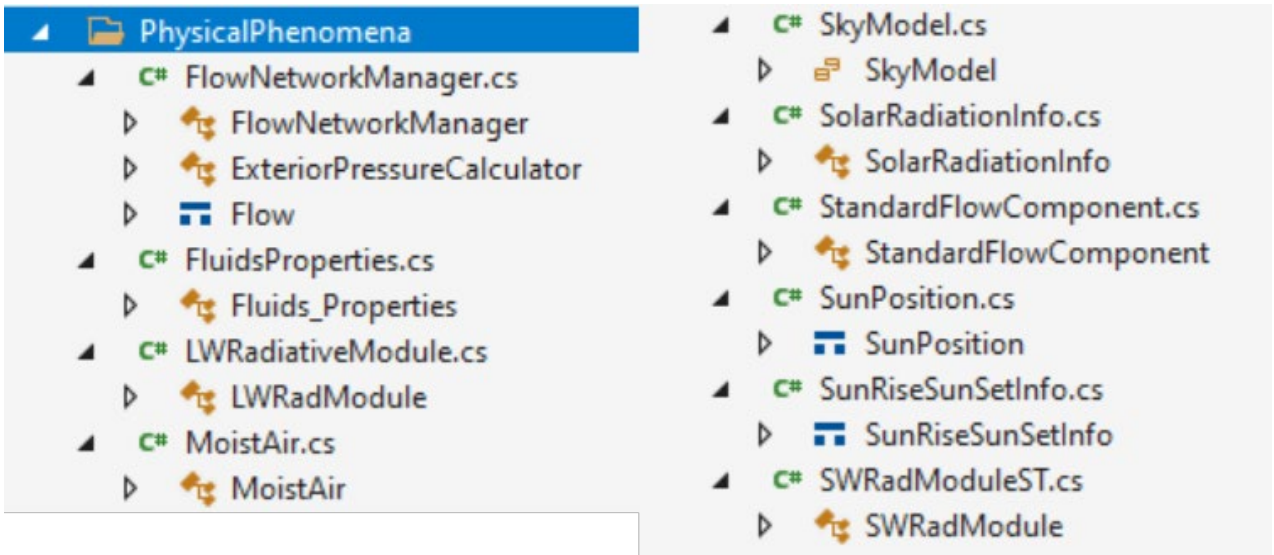


Figura 23 – Struttura della cartella *PhysicalPhenomena*

In particolare nei file contenuti nella cartella si hanno le seguenti **classi**:

- **FlowNetworkManager**: classe, che eredita dalla superclasse **SimulationComponent**, che gestisce la definizione e la risoluzione di una rete elettrica equivalente per la determinazione dei flussi d'aria

entranti e uscenti da una zona, sia verso l'esterno che verso zone limitrofe, in funzione della permeabilità all'aria dei vari elementi della zona (attualmente in via di sviluppo);

- **ExteriorPressureCalculator**: classe che definisce i metodi per il calcolo delle differenze di pressione (pressioni relative) agenti sull'esterno dell'edificio, che inducono i moti d'aria all'interno dello stesso in funzione della sua permeabilità, dovute sia all'azine del vento sia all'effetto della differenza di temperatura tra ambiente esterno e interno; (attualmente in via di sviluppo);
- **Fluids_Properties**: classe che contiene i metodi che ritornano le proprietà termofisiche di sostanze pure in funzione della temperatura e della pressione (per la sola fase aeriforme), che sono identificate tramite sue sottoclassi, cioè classi ereditanti; quelle attualmente implementate sono:
    - **Water**: classe contenente i metodi che ritornano le proprietà massa volumica, viscosità dinamica e capacità termica specifica a pressione costante, del liquido e del vapore *in condizioni di saturazione* in funzione della temperatura, proprietà ottenibili anche tramite i metodi:
        - `Get_liquid_water_property()` e **l'enumeratore LiquidProperty** (Density, DynamicViscosity, ThermalCapacityByMass);
        - `Get_water_vapour_property()` e **l'enumeratore VapourProperty** (Density, DynamicViscosity, ThermalCapacityByMass);

      inoltre contiene il metodo `Psat()` che ritorna la pressione di saturazione del vapore;
    - **DryAir**: classe contenente i metodi che ritornano le proprietà massa volumica, viscosità dinamica, capacità termica specifica e conduttività termica dell'aria secca, in funzione della temperatura a pressione standard (atm), proprietà ottenibili anche tramite il metodo:
        - `Get_dry_air_property_ATMP()` **e l'enumeratore DryAirProperty** (Density, DynamicViscosity, ThermalCapacityByMass, `ThermalConductivity`) ;

      N.B.: la densità è ricavata dalla legge dei gas ideali e quindi è anche funzione della temperatura, se il suo metodo è richiamato direttamente;
    - **Substance_ID**: **enumeratore** dei tipo di sostanze pure disponibili in libreria; (Water, DryAir, tc) (non ancora implementato)
- **LWRadModule**: classe che definisce le matrici di zona dei fattori di radiazione mutua `GBViewFactors` (onde lunghe), estraendo le informazioni necessarie dagli oggetti **Zone** (emissività); per tali calcoli, effettuati invocando il metodo `CalculateMatrix()`, è prima determinata la matrice dei fattori di forma per radiazione isotropa, `BBMatrix`, utilizzando i metodi definiti nella classe **BBViewFactor** della libreria **ExtendeMath**; i valori dei fattori di radiazione mutua relativi a due specifiche superfici di una zona sono poi estraibili tramite il metodo `GetGBViewFactor()`; inoltre la classe fornisce un metodo semplificato (media pesata sulle aree) per la distribuzione spaziale e quindi di attribuzione alle singole superficie dei guadagni interni dovuti a sorgenti di radiazione ad onde lunghe assegnati dall'utente tramite un profilo temporale, `SetLWScheduledGains()`; infine, tramite il metodo `calculation( )` determina la temperatura media radiante lineare della zona;
- **MoistAir**: classe contenente i metodi che ritornano le proprietà dell'aria umida considerata una miscela di gas ideali, aria secca e vapore d'acqua; il metodo implementa la struttura **State**, che definisce per ogni stato termodinamico le seguenti proprietà:
    - DryBulbTemperature;
    - WetBulbTemperature;
    - RelativeHumidity;
    - HumidityRatio;
    - Enthalpy;
    - WaterPartialPressure;
    - SpecificVolume;
    - Density;

  implementa quindi una serie di metodi che consento di ricavare, fissata la pressione, tutte queste proprietà in funzione di due proprietà note, in particolare:
    - temperatura a bulbo secco e umidità relativa;

  - temperatura a bulbo secco e umidità massica;
  - temperatura a bulbo secco e temperatura bulbo umido;
  - entalpia specifica e umidità massica;
  - entalpia specifica e temperatura a bulbo secco;
  - entalpia specifica e umidità relativa;

  e dei metodi per il calcolo di:

  - capacità termica specifica in funzione dell'umidità massica;
  - temperatura di rugiada in funzione dell'umidità massica;
  - pressione di saturazione in funzione della temperatura;
  - temperatura di saturazione in funzione della pressione;
  - portata massica in funzione della portata volumica, temperatura e l'umidità relativa;
  - portata massica in funzione della portata volumica, temperatura e l'umidità massica;
  - portata massica del componente aria secca in funzione della portata volumica, temperatura e l'umidità massica;
  - portata massica del componente aria secca in funzione della portata volumica, temperatura e l'umidità relativa;

  infine implementa dei metodi che descrivono le trasformazioni elementari dell'aria umida, quali:

  - miscelazione di due arie umide;
  - umidificazione adiabatica;
  - saturazione adiabatica;
  - deumidificazione;

- **SolarRadiationInfo**: questa classe definisce le proprietà associate alla radiazione solare, quali l'angolo orario all'inizio, medio e alla fine dell'intervallo temporale corrispondente all'intervallo di simulazione, il coseno dell'angolo zenitale riferito all'angolo orario medio, l'irradianza solare media globale sull'orizzontale, diretta normale e diffusa sull'orizzontale, il numero di ore di soleggiamento nel giorno considerato;
- **StandardFlowComponent**: classe che implementa i metodi per determina i flussi d'aria di infiltrazione e interzona (in via di sviluppo);
- **SWRadModule**: classe che definisce le matrici di zona dei fattori di riflessione e assorbimento per la radiazione solare diffusa (onde corte), estraendo le informazioni necessarie dagli oggetti **Zone** (coefficiente di riflessione e di assorbimento radiazione solare); in particolare i metodi:
  - `Initialize()`: questo metodo invoca il metodo `CalculateMutualReflectionMatrix()`, che in realtà calcola due diverse matrici, `MultipleReflectionsAbsMatrix` (che definisce quanto viene assorbito dalla superficie j della radiazione proveniente dalla superficie i) e `multipleReflectionsTransMatrix` (che definisce quanto viene trasmesso fuori dalla zona dalla superficie j della radiazione proveniente da i) (matrici definite nella classe **InZone**) utilizzando la matrice dei fattori di forma, `BBMatrix`, determinata in precedenza utilizzando i metodi definiti nella classe **BBViewFactor** della libreria **ExtendeMath**; lo stesso metodo inoltre verifica l'esistenza o meno di ombreggiamenti sulle superfici esterne trasparenti esposte alla radiazione solare, determinando l'area ombreggiata e quindi quella soleggiata;
  - `calculation()`: calcola il flusso di radiazione solare entrate dalle superfici trasparenti, sia diretto che diffuso, tramite i metodi `GetTransmittedDirSWRad ()` e `GetTransmittedDiffSWRad()` tenendo conto di eventuali ombreggiamenti, e li assegna alle singole superfici interne della zona tramite i metodi `assingDirSWFlux()` e `assignDiffSWFlux()`;
  - `assingDirSWFlux()`: assegna ad ogni superficie interna il flusso radiativo incidente dovuto alle riflessioni interne speculari per la radiazione diretta, tramite il metodo `ProjectRay()`);
  - `assignDiffSWFlux()`: assegna ad ogni superficie interna il flusso radiativo incidente dovuto alle riflessioni interne diffuse per la radiazione diffusa, tramite i fattori di riflessione usando il metodo `GetMultipleReflectionsFactor()`;

In particolare nei file contenuti nella cartella si hanno le seguenti **strutture**:

- **Flow**: struttura che definisce le proprietà di un flusso nella classe **FlowNetworkManager** , in termini di codice identificativo, portata massica, temperatura e umidità massica;
- **SunPosition**: struttura che definisce i parametri che determinano la posizione del sole nel cielo, quali tempo legale, tempo locale e tempo solare, tempo medio siderale locale (LMST), tempo medio siderale di Greenwich (GMST), data nel calendario giuliano, angolo orario, anglo zenitale e azimutale, angolo di declinazione, correzione del tempo per oscillazione asse terrestre (EQtime), seno e coseno dell'angolo di declinazione, elevazione solare, rapporto n.del giorno/n.totale giorni nell'anno (FrYear), parametro per lo spostamento di un giorno di FrYear (dipende dalla formula usata nel metodo **SunCalculator**), irradianza solare extraterrestre, vettore radiazione diretta nornale, versore posizione del sole;
- **SunRiseSunSetInfo**: struttura che definisce l'angolo di alba e tramonto associato al tempo locale.

In particolare nei file contenuti nella cartella si hanno i seguenti **enumeratori**:

- **SkyModel**: Isotropic, AnisotropicPerez1999, AnisotropicPerezEplus;

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.3.6 SystemComponents

La cartella *SystemComponents* contiene, come evincibile da Figura 24, le classi con i relativi metodi e le strutture necessarie per la descrizione e simulazione dei singoli componenti degli impianti tecnici, che, ad esclusione di alcune classi, strutture e enumeratori di carattere generale contenute nel file *SystemComponent.cs*, sono raggruppati in due cartelle: *Controllers* e *HVAC*.

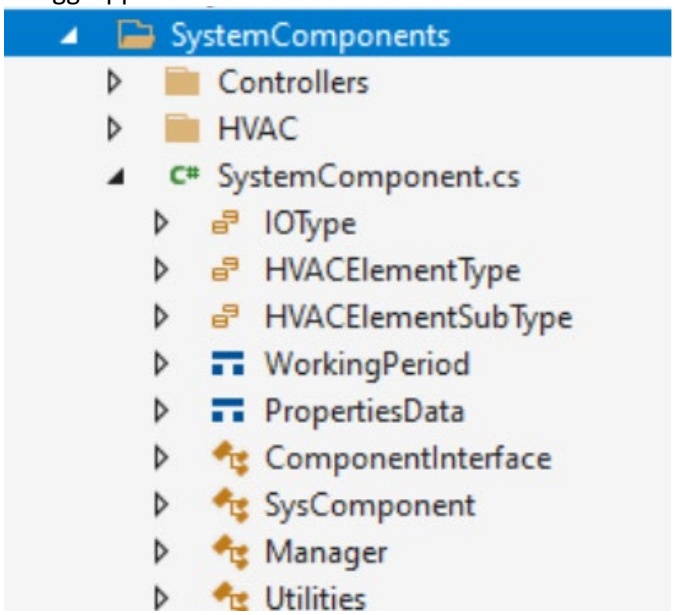


Figura 24 – Struttura della cartella *SystemComponents*

Non tutte le classi e strutture e i loro relativi metodi e proprietà presenti nella cartella *SystemComponents* e nelle sue sottocartelle appartengono al ***namespace* SimulationComponents,** ma alcune classi appartiene dei suoi sottospazi dal *namespace* **SimulationComponents.xxx.yyy.**

Le **classi** di carattere generale appartengono al ***namespace*** **SimulationComponents** e sono:

- **SysComponent**: classe astratta che definisce alcune proprietà generiche comuni a tutti i tipi di componenti di sistema, quali il nome identificativo del componente (Id), la zona a cui è associato il componente (`MyZone`, `MyZOneName`), degli indicatori di modalità di funzionamento (`OpModeIndex`, `OnOffIndex`), un booleano (`HasConverged`), che verifica e indica il raggiungimento della convergenza interna, un booleano (`PrintAllOutput`) per dire all'oggetto del componente se rendere disponibili o meno le degli output dettagliati per il suo debug; implementa inoltre degli indicatori facoltativi che specificano se il componente necessita di potenza elettrica (`ElPowerAbsorbedIndex`) e se emette radiazione termica (`LWFluxIndex`); questa classe virtuale soprattutto implementa un'istanza della classe **ComponentInterface** (vedasi dopo) e alcuni metodi virtuali generali, quali:
  - `Initialize()`: metodo virtuale, che può essere modificato dal singolo componente, che provvede, in fase di istanziazione dell'oggetto del componente, di assegnarli tutte le proprietà necessarie, effettuare tutti i precalcoli necessari al funzionamento dell'oggetto durante la simulazione e allocare delle aree di moria accessorie, se del caso ;
  - `Calculate()`: metodo virtuale, che può essere modificato dal singolo componente, che provvede, in fase di simulazione, a leggere i dati dall'input, tramite il metodo `ReadInput()` o `ReadConstantInput()`, inizializzando se del caso il vettore dei dati di input al passo temporale precedente, e ad eseguire i metodi specifici della classe dell'oggetto che ne determinano la prestazione e quindi l'output (questa parte è presente negli override del metodo riportati nelle classi specifiche al componente);
  - `TimeConverged()`: metodo virtuale, che può essere modificato dal singolo componente, che provvede, in fase di simulazione, al salvataggio dell'input corrente al raggiungimento della convergenza;;
  - `StoreIterationValues`: metodo virtuale, che può essere modificato dal singolo componente, che provvede, in fase di simulazione, al salvataggio dell'output corrente al raggiungimento della convergenza;
- **ComponentInterface**: classe che definisce la struttura dei dati che il singolo componente, che può essere del tutto autonomo, scambia con il codice chiamante, cioè con l'oggetto **Simulator** o più specificatamente con l'oggetto **HAVCSystem**; tale classe contiene delle proprietà (INPUT, OUTPUT, PARAMETERS, INFO, DATA e STORAGE) che sono istanze delle sue sottoclassi e delle sue strutture; le **sottoclassi** sono:
  - **Input**: la classe definisce i seguenti **campi**:
    - `OPERATING_MODE`: un intero pari a 0 = HEATING; 1 = COOLING;
    - `Inputs[]`: un array che contiene valori di input richiesti dal modello,
    - `InputsQualifier`: un array che contiene un indice intero qualifica il tipo di input tramite **l'enumeratore** pubblico **IOType**,
    - `InitialIputValues[]`: un array che contiene i valori di inizializzazione dei valori di input,
    - `Connections[,]`: array bidimensionale contenete il numero ID del componente collegato e l'ID di uscita a cui è collegato;

    e le seguenti **proprietà**:
    - `InputPairDescription[]`: array stringa che contiene un descrittore testuale dell'oggetto che riceve l'input, un descrittore testuale dell'oggetto e dell'output di quell'oggetto da cui si riceve l'input;
    - `InputPairRef[]`: array *Tuple* che contiene l'intero identificativo dell'oggetto che riceve l'input, l'oggetto che fornisce l'input, l'intero identificativo dell'output dell'oggetto che fornisce l'input;
    - `InputWithoutRef[]`: array *Tuple* che contiene in ogni elemento un valore intero e uno reale, che corrispondono al ID dell'input e al valore assegnatogli costante per tutta la simulazione;
  - **Info**: la classe definisce i seguenti **campi**:

- `ComponentName`: nome del componente;
- `ModelName`: nome del modello implementato per la descrizione del componente e il calcolo delle sue prestazioni;
- `ModelID`: identificatore numerico del modello;
- `SteadyState`: booleano che indica se il modello è stato sviluppato sotto l'approssimazione dello stato quasi stazionario;
- `CharacteristicTime`: tempo caratteristico del componente;
- `SwitchableState`: booleano che indica se il modello implementa sia l'approssimazione dello stato stazionario sia equazioni evolutive che tengono conto del tempo caratteristico del componente, e se quindi si può utilizzare sia l'uno che l'altro;
- `PredefinedFlowDirection`: booleano che indica se il flusso di massa in quello che viene definito un ingresso del componente è predefinito (e quindi sempre definito positivo);
- `FlowDirectionByPotentialDifference`: booleano che indica se il flusso di massa in quello che viene definito un ingresso del componente è governato dalle differenze di pressione e quindi possono invertirsi (l'ingresso può diventare un'uscita);
- `SwitchableFlowDirection`: booleano che indica se il modello consente sia un flusso di massa imposto che governato dalle differenza di pressione;
- `NumberOfInputs`: numero massimo di input richiesti dal modello (qualcuno potrebbe essere opzionale);
- `NumberOfOutputs`: numero di output forniti dal modello;
- `NumberOfParameterValue`: numero massimo di parametri di tipo numerico richiesti dal modello (qualcuno potrebbe essere opzionale);
- `NumberOfParameterTypes`: numero massimo di parametri di tipo letterale (stringa) richiesti dal modello (qualcuno potrebbe essere opzionale);
- `PropertiesData`: booleano che indica se il modello richiede altri dati oltre quelli forniti dai parametri;
- `NumberOfDataValues`: numero di dati caratteristici numerici che esprimono le ulteriori proprietà del componente;
- `NumberOfDataInfo`: numero di dati caratteristici testuali che esprimono le ulteriori proprietà del componente;
- `LibraryRequided`: booleano che indica se gli ulteriori dati caratteristici devono essere prelevati da una libreria binaria dinamica (.dll);
- `LibraryPath`: percorso nel file system che individua la localizzazione della libreria;
- `EquipmentName`: nome del componente i cui dati risiedono nella libreria;
- `QualifiedEquipmentName`: nome qualificante (nome interno) della componete i cui dati risiedono nella libreria;

e le seguenti **proprietà**:

- `MyType`: enumeratore **HVACElementType**;
- `WorkingPeriods[]`: [0] = fromMonth, [1] = fromDay, [2] = toMonth , = toDay, [4] = mode;

le **strutture** sono:

- **Output**: che contiene tre campi:
  - `Outputs[]`: array dei valori di output forniti dal modello;
  - `OutputsQualifier[]`: array di indici interi che qualificano il tipo di output tramite l'enumeratore pubblico **IOType**;
  - `Warnings`: una stringa che contiene messaggi di avviso/errore prodotti dal modello del componente durante la sua esecuzione;
- **Parameters**: che contiene tre proprietà:
  - `ParamValue[]`: array dei valori numerici dei parametri richiesti dal modello;

        - `ParamType[]: []`: array di indici interi che qualificano il tipo di parametri richiesto tramite l'enumeratore pubblico **IOType**;
        - `ParamUnit[]`: array dei delle unità di misura dei valori numerici dei parametri richiesti dal modello;
    - **StoredData**: che contiene cinque campi:
        - `StoredDataNumber`: numero di dati da conservare per tutta la durata della simulazione;
        - `TemporaryStoredDataNumber`: numero di dati da conservare solo temporaneamente**;**
        - `PermanentStoredValues[]`: valori dei dati da conservare per tutta la durata della simulazione;
        - `TemporaryStoredValues[]`: valori dei dati da conservare solo temporaneamente**;**
        - `PreviousIterationValues[]`: valori dei dati all'iterazione precedente;

  Le proprietà della classe **ComponentInterface** sono quindi definite come:
    - **INPUT**: membro di tipo **Input**;
    - **OUTPUT**: membro di tipo **Output**;
    - **PARAMETERS**: membro di tipo **Parameters**;
    - **INFO**: membro di tipo **Info**;
    - **DATA**: membro di tipo **PropertiesData**;
    - **STORAGE**: membro di tipo **StoredData**;
- **Manager**: classe che definisce i metodi per la gestione dell'esecuzione dei componenti durante il run-time collegando gli input richiesti di ciascun componente con i relativi output degli altri componenti e l'eventuale parallelismo nel calcolo; la classe inoltre implementa un metodo per la conversione dell'identificatore numerico dello stato del componente in una stringa di testo qualificante tramite **l'enumeratore**:
    - **ContrAction**: OFF = 0, ON = 1, Initialization = 2, PartiallyON = 3, VectorTempGraterThanZoneAirTemp = 4, VectorTempLowerThanZoneAirTemp = 5;
- **Utilities**: classe che contiene il metodo `InitializeArray<T>(int length)` che consente di inizializzare un vettore di oggetti di dimensione a piacere in modo dinamico, e il metodo `GetInstance(string FullyQualifiedName)` che consente di identificare e quindi poi utilizzare una specifica istanza di un oggetto .

Le **strutture** di carattere generale appartengono al ***namespace*** **SimulationComponents** e sono:

- **WorkingPeriod**: struttura che definisce il mese di inizio e il mese di fine del periodo di predisposizione al funzionamento di un componente e la modalità di funzionamento per quei componenti che possono lavorare sia in riscaldamento (=0) che in raffrescamento (=1); la struttura dispone di un metodo che consente di modificare dinamicamente tali proprietà;
- **PropertiesData**: struttura che mette a disposizione dei componenti di sistema un'area di memoria dove immagazzinare tutti quei dati caratteristici dello specifico componente necessari alla sua caratterizzazione quando istanziato in oggetto; tale area è suddivisa in due array monodimensionali `DataInfo[]`, contenete stringhe, e `DataValues[]`, contente valori numerici, dimensionati in fase di istanziazione dell'oggetto, in funzione di parametri `DataInfoSize` e `DataValuesSize`.

Gli **enumeratori** di carattere generale appartengono al ***namespace*** **SimulationComponents** e sono:

- **IOType**: Mass = 1, Lenght, Force, Time, Temperature, Pressure, ElectricCurrent, ElectricVoltage, Energy, Power, Velocity, MassFlow, ThermalCapacityFlow, HumidityRatio, MassRatio, Area, Volume, MassDensity, SpecificEnergy, SpecificHeatCapacity, SpecificVolume, Dimesionless, AmbTemperature, SPTemperature, AmbMassFlow, ElectricPower, LWFlux, OperatingMode, OnOffSwitch;
- **HVACElementType**: Emitter, Generator, Pipe, Pump, ThreeWayValve, Splitter, Mixer;
- **HVACElementSubType**: Radiator, RadiantPanel, FanCoil, Boiler, HeatPump.

3.2.3.6.1 Controllers

La cartella *Controllers* contiene, come evincibile da Figura 25, le classi con i relativi metodi necessarie per la descrizione di sistemi di controllo e regolazione degli impianti.

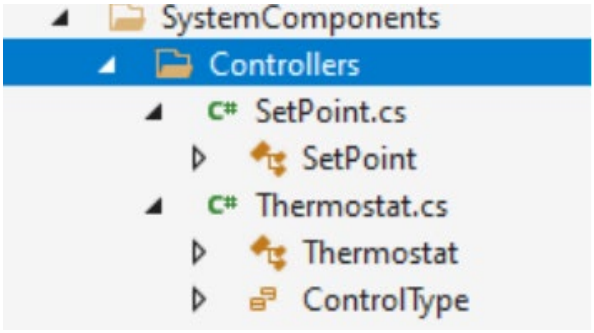


Figura 25 – Struttura della cartella *Controllers*

In particolare nei file contenuti nella cartella si hanno le seguenti **classi**:

- **SetPoint**: classe, appartenente al *namespace* **SimulationComponents**, che definisce le caratteristiche di un set point tramite le **proprietà**:
  - `From`: data e tempo di attivazione del set point;
  - `To`: data e tempo di disattivazione del set point;
  - `Value`: valore del set point;
  - `Range`: banda morta o ampiezza dell'isteresi del set point;
- **Thermostat**: classe, appartenente al *namespace* **SimulationComponents.SystemComponents .HVAC,** che definisce un controllore termostatico o igrostatico tramite le **proprietà**:
  - `Name`: nome identificativo del controllore;
  - `MyType`: tipologia del controllore identificata tramite l'enumeratore **ControlType**;
  - `SetPointSchedNames[]`: i nomi identificativi dei profili orari di valori di set point da applicare al controllore;
  - `SetPointsSchedules[]`: array dei profili orari di valori di set point da applicare al controllore;
  - `ConstantSetPoints[]`: array di valori costanti nel tempo di set point;

il seguente l'**enumeratore**:

- **ControlType**: thermostat = 0, humidostat = 1.

e i seguenti **metodi**:

- `Check()`: verifica la consistenza dei dati di definizione del controllore;
- `getRiscValue()`: fornisce per un dato controllore alla data e tempo attuale il valore del set point di riscaldamento;
- `getRaffrValue()`: fornisce per un dato controllore alla data e tempo attuale il valore del set point di raffrescamento.

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.3.6.2 HVAC

La cartella *HVAC* contiene, come evincibile da Figura 26, le classi con i relativi metodi necessarie per la descrizione e simulazione dei componenti del sistema impiantistico per la climatizzazione ambientale e la ventilazione (HVAC), che, ad esclusione di alcune classi, strutture e enumeratori di carattere generale contenute nel file *HVACcomponent.cs*, sono raggruppati in due cartelle: Emitters e HeatGenerators.

Tutte le classi, con i relativi metodi e proprietà, e gli enumeratori presenti nella radice della cartella *HVAC,* cioè nel file *HVACcomponent.cs*, appartengono al sottospazio definito dal *namespace* **SimulationComponents.SystemComponents.HVAC**.

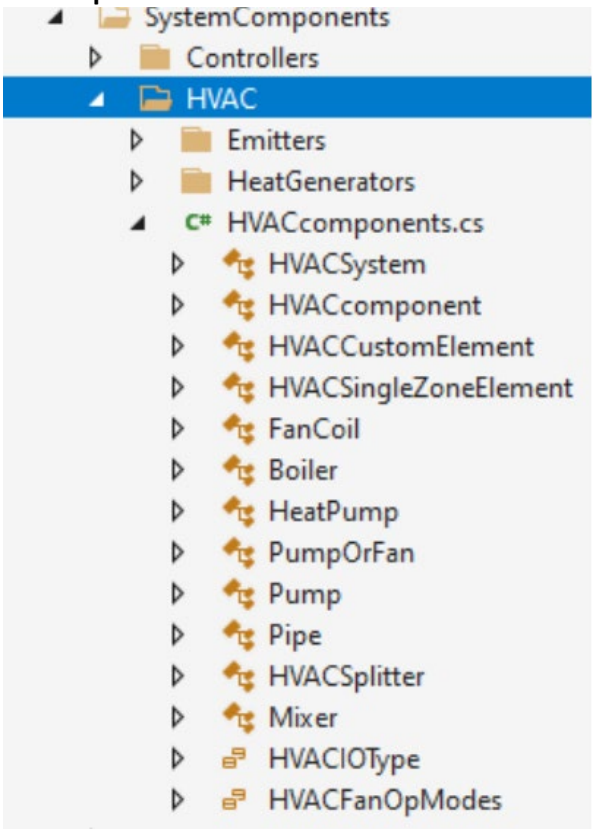


Figura 26 – Struttura della cartella *HVAC*

Le **classi** di carattere generale sono:

- **HVACSystem**: classe che eredita dalla superclasse **SimulationComponent** e che definisce le caratteristiche del sistema di climatizzazione nel suo complesso tramite le sue proprietà e i metodi implementati; in particolare le **proprietà** sono:
  - `Components`: contiene la lista degli oggetti dei componenti il sistema del tipo **HVACSingleZoneElement**;
  - `SysComponents`: contiene la lista degli oggetti dei componenti il sistema del tipo **SysComponent**;
  - `MyZones[]`: array contenete gli oggetti di tipo **Zone**, cioè le zone in cui è suddiviso l'edificio;
  - `MyGenerators[]`: array contenete gli oggetti di tipo **HVACcomponent** ma della sottoclasse generatori;
  - `MyEmitters[]`: array contenete gli oggetti di tipo **HVACcomponent** ma della sottoclasse emettitori;
  - `PrintEmittersOutput`: booleano per il debug degli emettitori;
  - `UseMasterControlOnGenerators`: booleano che indica se c'è o no un controllore master che gestisce il controllo interno dei generatori;
  - `UseMasterControlOnEmitters`: booleano che indica se c'è o no un controllore master che gestisce il controllo interno degli emettitori;

- `OPERATING_MODE`: definisce la modalità di funzionamento, 0= HEATING; 1 =COOLING;;
- `WorkingPeriodsNumber`: numero dei diversi periodi di funzionamento dell'impianto;
- `WorkingPeriods`: lista di vettori di interi che definiscono il mese e il giorno di inizio e fine del periodo di funzionamento, oltre che il modo (riscaldamento o raffrescamento);
- `ResetWorkingPeriodNumber`: booleano, che controlla la modifica del numero dei periodi di funzionamento;
- `OutputCounter`: contatore della quantità di output prodotti dal sistema;
- `ElPower`: potenza elettrica complessivamente richiesta dai componenti del sistema;

i **metodi** sono:

- `Initialize()`: metodo che chiama il metodo generale virtuale per l'inizializzazione di componenti iterativi `SimulationComponents.Initialize()` e successivamente i metodi di inizializzazione per tutti i componenti presenti nella lista `Components` e `SysComponents`; infine definisce la modalità di funzionamento iniziale chiamando il metodo privato `setOperatingMode()`;
- `Check()`: metodo che verifica la consistenza dei dati e delle connessioni;
- `Calculate()`: metodo che esegue i calcoli chiamando prima il metodo generale virtuale per il calcolo `SimulationComponents.Calculate()` e successivamente i metodi `Calculate()` dei singoli component costituenti il sistema, e gestisce l'eventuale processo iterativo per la determinazione della prestazione dell'impianto conseguenti alle azioni del sisyema di controllo;
- `TimeConverged()`: metodo gestiste gli output e gli eventuali salvataggi degli stati interni quando la convergenza è stata raggiunta;
- `SetSimplifiedSplittes()`: metodo per la modifica dinamica delle caratteristice dello splitter.
- `TryToMatchComponents()`: metodo di verifica delle connessioni tra componenti usato dal GUI e che quindi va rimosso dalla libreria;

- **HVACcomponent**: classe astratta che eredita dalla superclasse **SimulationComponent** e che definisce le caratteristiche comuni ai componenti del sistema HVAC, caratteristiche specializzabili tramite *override* dall'implementazione in uno specifico componente; in particolare le **proprietà** sono:
  - `Id`: identificatore alfanumerico del componente;
  - `ModelId`: identificatore alfanumerico del modello del componente;
  - `MyType`: tipologia del componente definita tramite l'enumeratore pubblico **HVACElementType**;
  - `Model`: informazioni addizionali sul componente fornite dall'stanza della classe **HVACCustomElement**;
  - `Multiplier`: eventuale riduttore della portata massica in ingresso al componente (divisore fisso);
  - `ModulateMultiplier`: booleano, se True, il riduttore della portata massica in ingresso al componente è determinato dal rapporto tra `TempMult` e il divisore fisso (`Multiplier`);
  - `Initialization`: booleano che identifica la fase di inizializzazione del componente;
  - `Theta`: parametro per l'integrazione con il theta-method (0-1);
  - `Power_switch_OFF`: booleano che controlla l'attivazione/disattivazione del componete;
  - `TempMult`: rapporto tra la potenza richiesta e la potenza fornita;
  - `Partializable`: booleano, indica se il componente è parzializzabile nella sua prestazione o no;
  - `OPERATING_MODE`: definisce la modalità di funzionamento, 0= HEATING; 1 =COOLING;
  - `WorkingPeriods`: lista di tipi **WorkingPeriod,** struttura definita tra le strutture di carattere generale della libreria;
  - `ControlledByMaster`: booleano che indica se il sistema di controllo interno del componente è a sua volta controllato da un master control;

- `AddWorkingPeriods`: booleano che controlla l'aggiunta di `WorkingPeriods` nella lista che li contiene;
- `Params[]`: riporta i valori dei parametri numerici richiesti dal modello del componente considerato;
- `InputNumber`: numero degli input richiesti dal modello del componente;
- `OutputNumber`: numero degli output forniti dal modello del componente;
- `Input[]`: array dei valori degli input richiesti dal modello del componente;
- `Output[]`: array dei valori degli output forniti dal modello del componente;
- `InputType[]`: array delle tipologie degli input richiesti dal modello del componente;
- `OutputType[]`: array delle tipologie degli output forniti dal modello del componente;
- `INTERNAL_STATES[]`: vettore delle proprietà di stato interne del componete;
- `PrevTimeStepINTERNAL_STATES[]`: vettore delle proprietà di stato interne del componete al passo temporale precedente;
- `MaxIterationThrottling`: numero massimo di variazioni di stato consentite al sistema di controllo all'interno di un passo di integrazione temporale durante processi iterativi alla ricerca della convergenza;
- `InputPairCsv[]`: array testuale che riporta le connessioni tra gli input del componente corrente e gli output degli altri componenti a questo collegati;
- `InputPairRef[]`: array *Tuple* che riporta le connessioni tra gli input del componente corrente e gli output degli altri componenti a questo collegati;
- `HasConverged`: booleano, che indica se si è raggiunta la convergenza all'interno di un processo iterativo;

i **metodi** sono:

- `Initialize()`: metodo che chiama il metodo generale virtuale per l'inizializzazione di componenti iterativi `SimulationComponents.Initialize()` ;
- `Check()`: metodo che verifica la consistenza dei dati e delle connessioni;
- `IsInsideWorkingPeriod()`: metodo per la verifica se il componente deve essere simulato o no (se lo specifico intervallo temporale di integrazione è all'interno on del periodo di attivazione dello specifico componente);
- `ReadInput()`: metodo astratto che viene definito successivamente nella classe specifica del componente con un override;
- `SetDefaultInputOutputTypes()`: metodo astratto che viene definito successivamente nella classe specifica del componente con un override;
- `SetObjectCoupling()`: metodo astratto che viene definito successivamente nella classe specifica del componente con un override;
- `checkTrottoling()`: metodo che tiene sotto controllo le oscillazioni (variazioni di stato) del controllore durante i processi iterartivi ;
- `TimeConverged()`: metodo gestiste gli output e gli eventuali salvataggi degli stati interni quando la convergenza è stata raggiunta;

- **HVACCustomElement**: questa classe definisce le proprietà fondamentali per includere nel simulatore un componente non previsto a livello di modelli e non fornito come oggetto di una libreria dinamica; in questo caso la prestazione del componente è descritta da una mappa di prestazione fornita dall'utente; in particolare le **proprietà** sono:
  - `ModelId`: identificatore alfanumerico del modello del componente;
  - `MyType`: tipologia del componente definita tramite l'enumeratore pubblico **HVACElementType**;
  - `MySubType`: tipologia secondaria del componente definita tramite l'enumeratore pubblico **HVACElementSubType**;
  - `Params[]`: array contenete i valori numerici dei parametri necessari per l'impiego del componente;

  - `MultyParams`: proprietà definita tramite la struttura pubblica **SpaceFunction,** definita nella libreria **ExtendedMath**, che implementa i metodi per l'interpolazione lineare;
  - `MyTableData`: proprietà che contiene i dati di prestazione tabulati organizzati secondo la struttura pubblica **TableData**;
- **HVACSingleZoneElement**: classe che eredita dalla classe **HVACcomponent** e che provvede ad associare al singolo componente le seguenti **proprietà**:
  - `MyZoneName`: nome della zona che si associa;
  - `MyZone`: oggetto **Zone** della zona associata;
  - `Multipliers`: moltiplicatori da utilizzare quando nella zona associata ci sono più componenti in parallelo uguali al componente corrente;
  - `MaxTimeStepStandBy`: indicatore del numero di passi di integrazione temporale il componente resta in standby prima di riattivarsi per avvenuta richiesta del sistema di controllo;

  e i seguenti **metodi**:
  - `Check()`: verifica la consistenza dei dati;
  - `Calculate()`: in override al metodo generale, chiama il metodo `ReadInput()` e istanzia il vettore che contiene l'output;
  - `ReadInput()`: in override al metodo generale, associa gli input del componente corrente agli output dei componenti a questo collegati;
  - `setMultiplier()`: attiva il moltiplicatore;
  - `SetDefaultInputOutputTypes()`: attualmente non usato;
  - `SetObjectCoupling()`: istanzia un array stringa per una successiva verifica degli accoppiamenti tra input del componente attuale e gli output dei componenti a questo collegati ;
  - `storeControlActionAndStopPumps`: ;
- **FanCoil**: classe che eredita dalla classe **HVACSingleZoneElement** e che implementa in override la specializzazione per i tipi FanCoil dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Initialize()`, `SetDefaultInputOutputType()`, `Check()`, `Calculate()` e `TimeConverged()`;
- **Boiler**: classe che eredita dalla classe **HVACSingleZoneElement** e che implementa in override la specializzazione per i tipi Boiler dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Initialize()`, `Check()`, `Calculate()` e `TimeConverged()`;
- **HeatPump**: classe che eredita dalla classe **HVACSingleZoneElement** e che implementa in override la specializzazione per i tipi Boiler dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Initialize()`, `Check()`, `Calculate()` e `TimeConverged()`;
- **PumpOrFan**: classe che eredita dalla classe **HVACSingleZoneElement** e che implementa in override la specializzazione per i tipi PumpOrFan dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Check()` e `Calculate()`;
- **Pump**: classe che eredita dalla classe **HVACSingleZoneElement** e che implementa in override la specializzazione per i tipi Pump dei metodi generali definite nella classe **HVACcomponent**, nello specifico il solo metodo `Calculate()`;
- **Pipe**: classe che eredita dalla classe **HVACSingleZoneElement** e che descrive le prestazioni in regime stazionario o variabile di tubazioni o canali;

  le **proprietà** specifiche necessarie al modello sono poi:
  - `PlugFlowModel`: booleano che seleziona o meno il modello "plug flow";
  - `FrontPropagationMethod`: booleano che seleziona o meno il modello di "propagazione del fronte";
  - `Bounded`: booleano che viene utilizzato, esclusivamente nei test, per attivare o disattivare la verificare se la portata è tale che il fluido in ingresso nel passo temporale di integrazione ha percorso un tragitto maggiore della lunghezza del tratto considerato di tubazione o canale;

la classe implementa in override la specializzazione per i tipi Pipe dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Initialize()`, `Calculate()`, `Check` e `TimeConverged()`, un metodo specifico pubblico e diversi metodi privati che implementano i tre modelli della tubazione/condotta (volume di controllo, plug flow e di propagazione del fronte); nello specifico:

- `Pipe()`: definisce il tipo di componente come Pipe e definisce i tipi degli input e degli output invocando il metodo generale, modificato localmente tramite override, `SetDefaultInputOutputTypes()`;
- `initializePlugFlow()`: metodo che inizializza l'algoritmo del "plug flow";
- `plugFlow()`:metodo che calcola la prestazione della tubazione (temperatura di uscita del fluido) con l'algoritmo del "plug flow" invocando più volte il metodo `privato calcPlugSegment()` che a sua volta invoca il metodo privato `SolveDiffEq()`;
- `controlVolumeCalc()`: metodo che calcola la prestazione della tubazione (temperatura di uscita del fluido) con l'algoritmo dei volumi finiti (di controllo);
- `getAdvectionTemp()`: metodo privato che interpola/estrapola esponenzialmente, tramite il metodo privato `esponentialInterp()`, le temperature di fluido e parete interna per determinare la temperatura del fronte avvettivo basandosi sulla distribuzione spaziale ottenuta al passo temporale precedente;
- `calAirConvCoef()`: metodo privato per assegnare un coefficiente di scambio termico superficiale convettivo esterno equivalente, che tenga conto anche dell'isolamento esterno della tubazione/condotto;
- `setInternalConvectionCoefficient()`: metodo privato per assegnare il coefficiente di scambio termico superficiale convettivo interno in funzione delle geometria, delle proprietà e stato di moto del fluido;

- **HVACSplitter**: classe che eredita dalla classe **HVACSingleZoneElement** e che implementa un separatore adiabatico con 1 ingresso e 2 uscite tramite una lista di oggetti HVACSingleZoneElement di tipo HVACSplitter; questa classe implementa in override la specializzazione per i tipi HVACSplitter dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Initialize()`, `Check()`, `Calculate()` e `TimeConverged()`; inoltre implementa un metodo (`addComponentToGroup()`) che genera la lista di oggetti HVACSplitter, che però attualmente è chiamato dalla GUI, e quindi è una procedura da modificare per eliminare le interazioni con l'interfaccia.
- **Mixer**: classe che eredita dalla classe **HVACSingleZoneElement** e che implementa un miscelatore adiabatico con 2 ingressi e 1 uscita tramite una lista di oggetti HVACSingleZoneElement di tipo Mixer (unica proprietà della classe); questa classe implementa in override la specializzazione per i tipi Mixer dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Check()` e `Calculate()`; inoltre implementa un metodo (`setGroup()`) che genera la lista di oggetti Mixer, che però attualmente è chiamato dalla GUI, e quindi è una procedura da modificare per eliminare le interazioni con l'interfaccia.

Gli **enumeratori** di carattere generale sono:

- **HVACIOType**: GasTemp, FluidTemp, GasMassFlux, FluidMassFlux, Flux, Param;
- **HVACFanOpModes**: ContFanCycCoil, CycFanCycCoil.

3.2.3.6.2.1 Emitters

La cartella *Emitters* contiene, come evincibile da Figura 27, le classi con i relativi metodi necessarie per la descrizione e simulazione degli emettitori.

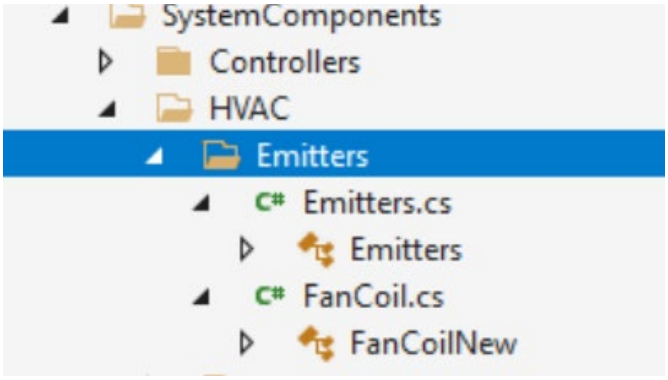


Figura 27 – Struttura della cartella *Emitters*

In particolare nei file contenuti nella cartella si hanno le seguenti **classi**:

- **Emitters**: questa classe appartiene al sottospazio definito dal *namespace* **SimulationComponents.SystemComponents** e implementa in override la specializzazione per i tipi Emitters (corpi scaldanti) dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Initialize()`, `Check()`, `Calculate()` e `TimeConverged()`;
- **FanCoilNew**: questa classe appartiene al sottospazio definito dal *namespace* **SystemComponents** e implementa in override la specializzazione per i tipi FanCoilNew (ventilconvettori) dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Initialize()`, `Check()`, `Calculate()` e `TimeConverged()`.

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.3.6.2.2 HeatGenerators

La cartella *HeatGenerators* contiene, come evincibile da Figura 28, le classi con i relativi metodi necessarie per la descrizione e simulazione dei generatori termici.

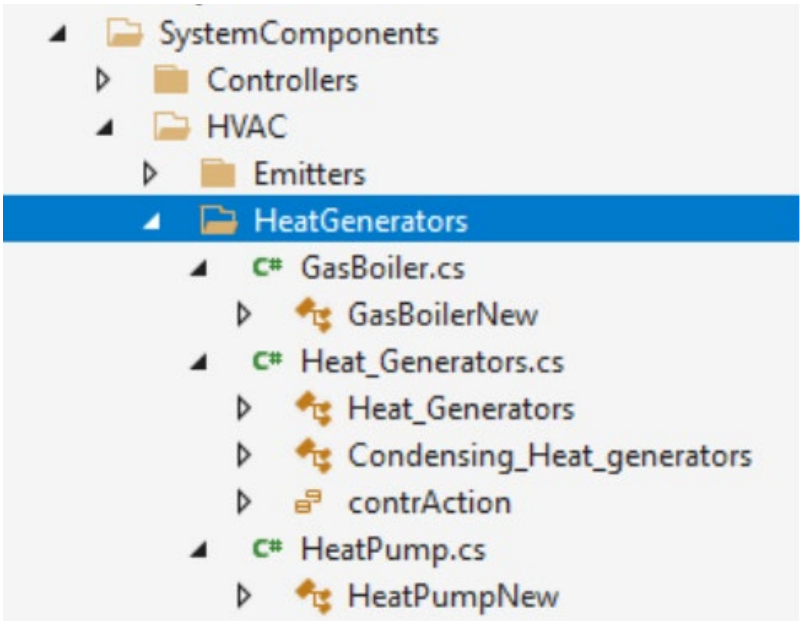


Figura 28 – Struttura della cartella *HeatGenerators*

In particolare nei file contenuti nella cartella si hanno le seguenti **classi**:

- **GasBoiler**: questa classe appartiene al sottospazio definito dal *namespace* **SystemComponents** e implementa in override la specializzazione per i tipi GasBoiler (caldaie a gas) dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Initialize()`, `Check()`, `Calculate()` e `TimeConverged()`;
- **HeatGenerators**: questa classe appartiene al sottospazio definito dal *namespace* **SystemComponents.SystemComponents** e implementa in override la specializzazione per i tipi - HeatGenerators (caldaie) dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Initialize()`, `Check()`, `Calculate()` e `TimeConverged()`;
- **Condensing_Heat_generators**: questa classe appartiene al sottospazio definito dal *namespace* **SystemComponents.SystemComponents** e implementa in override la specializzazione per i tipi - Condensing_Heat_generators (caldaie a condensazione) dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Initialize()`, `Check()`, `Calculate()` e `TimeConverged()`;
- **HeatPumps**: questa classe appartiene al sottospazio definito dal *namespace* **SystemComponents** e e implementa in override la specializzazione per i tipi HeatPumps (pompe di calore) dei metodi generali definite nella classe **HVACcomponent**, nello specifico i metodi `Initialize()`, `Check()`, `Calculate()` e `TimeConverged()`;

Si ha inoltre il seguente **enumeratore**:

- **contrAction**: OFF = 0, ON = 1, Stopped_T_returnDistrtib_OutWorkingTempRange, Stopped_T_environment_OutWorkingTempRange, VectorTempGraterThanZoneAirTemp, VectorTempLowerThanZoneAirTemp, InrangeControlActivated, ZoneSetPoitFullfilled, Initialization, PartiallyON = 9, MaxBoilerTempertureReached, MinAllowedWorkingTempDiff.

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

3.2.4 InputOutputUtlities

La libreria **InputOutputUtlities** contiene gli strumenti per la gestione degli input e degli output e, come evincibile Figura 29, è costituita da sette classi. Tutte le classi e i metodi e i loro relativi metodi e proprietà appartengono allo spazio identificato dal ***namespace*** **InputOutputUtlities.**

Figura 29 – Struttura della cartella *InputOutputUtlities*

Classi:

- **AttributeIOIndex**: definisce solamente un indice IO per la classe astratta di sistema **Attrìbute**;
- **Converters**: questa classe implementa dei metodi per particolari conversioni di dati:
  - **ToDataTable**(*ArrayList*): converte un *ArrayList* in una tabella bidimensionale;
  - **ToString**(): converte un *ArrayList* in una stringa di testo dove i dati sono sparati da un simbolo di tabulazione;
  - **SetValuesToObject**: cerca di modificare dei dati in una classe istanziata (oggetto);
- **ExecutableManager**: manda in esecuzione applicazioni esterne via linea di comando della console;
- **LoggingHelper**: classe che implementa un metodo che genera la scrittura di un log di eventi;
- **OutputSet**: classe che definisce l'output (outputset) come una struttura contenente un nome che identifica il tipo di output e due liste di stringhe, una con gli oggetti dei quello specifico, l'altra con le proprietà di tali oggetti;
- **OutputResults**: questa classe implementa i metodi da utilizzare per scrivere delle stringhe in un file, per verificare l'esistenza o creare la cartella di default per l'archiviazione dei risultati, scrivere i risultati in un file testuale con formati differenti, compreso il CSV;
- **SerializationUtils**: questa classe implementa i metodi da utilizzare leggere un file binario contenente degli oggetti (classi istanziate).

I dettagli di tali classi e strutture sono direttamente ricavabili dal codice sorgente presente nel repository del progetto

4 Protocollo per lo sviluppo, la validazione e la gestione del codice di calcolo OpenBPS

4.1 Introduzione

Nell'ambito presente ricerca si è cercato di sviluppare una procedura codificata chiara e trasparente che consenta di assicurare lo sviluppo continuativo di un codice di calcolo di simulazione termoenergetica degli edifici nonché la sua manutenzione e l'aggiornamento, tramite il coinvolgimento di quanti, esperti del settore, vogliano e possano contribuire, prestando la loro competenza in modo coordinato ed efficace in un ambito "open source".

Con codice di calcolo si intende il codice sorgente in C-sharp di un qualsiasi componete del sistema edificio-impianto integrabile nel codice OpenBPS per estenderne le sue funzionalità. Con documentazione si intende l'insieme dei documenti, in formato digitale (word, pdf, excel, etc.) necessari alla descrizione dei supporti teorici e degli algoritmi impiegati nella realizzazione del codice proposto. Con validazione si intende la verifica della qualità dei risultati ottenibili con il suddetto codice rispetto o risultati sperimentali, anche riportati nella letteratura tecnica, o rispetto a soluzioni analitiche per casi applicativi semplici o semplificati., stanti a dimostrare la validità del codice proposto.

Il protocollo deve quindi definire le regole di sviluppo, documentazione e validazione del codice che viene reso disponibile dagli sviluppatori e contenere le regole di verifica e validazione finale dei codici proposti prima della loro accettazione definitiva come componenti ufficiali del codice OpenBPS. Per questa seconda fase occorre individuare il criterio di selezione dei valutatori indipendenti che, sulla base della documentazione fornita dagli sviluppatori, operano una validazione terza del codice proposto. Le informazioni concernenti le validazioni, sia quelle effettuate dagli sviluppatori, che quelle effettuate dai valutatori terzi devono essere ben documentate e chiaramente associate a ciascun componente o modulo che sarà poi reso disponibile nel repository ufficiale del codice OpenBPS. Da qui la necessità di standardizzare questa procedura il più possibile e che questa standardizzazione sia prodotta da un soggetto specificatamente competente, sia per l'oggetto (calcolo dinamico delle prestazioni termoenergetiche degli edifici) sia per il metodo (Ente possibilmente normatore).

Per garantire la diffusione di una versione stabile del codice sorgente e/o eseguibile, specificatamente in termini di librerie dinamiche (dll), ma soprattutto per garantire che l'utente finale dei risultati dei calcoli eseguiti con il codice OpenBPS possa essere certo della qualità dello strumento utilizzato per produrli, occorre che la suddetta procedura identifichi una modalità di gestione del rilascio del codice e dei suoi aggiornamenti ed estensioni, che, a fianco della disponibilità del codice sorgente, garantisca che questo non sia stato impropriamente modificato o alterato. Occorre probabilmente identificare anche un Ente che possa essere garante della corrispondenza del codice utilizzato da parti terze per produrre i risultati forniti all'utente finale con il codice ufficiale presente nel repository dell'Ente di gestione del progetto Open Source

4.2 Considerazioni generali

L'obbiettivo di arrivare a creare una comunità di sviluppatori nell'ambito Open Source ha comportato un importante approfondimento, di seguito sintetizzato nei suoi elementi principali, relativamente a concetti propri delle comunità open source e dei requisiti che dovrebbe possedere un software eventualmente open

source che, come richiesto, dovrebbe però garantire in modo inequivocabile, in ultima analisi, la corrispondenza del risultato dei calcoli eseguiti dagli utenti con i calcoli che il codice sorgente originale esegue. Si ritiene quindi utile, se non addirittura necessario premettere una serie di valutazioni e relative considerazioni che, pur sembrando fuori tema, in realtà servono per consolidare l'approccio operativo che seguirà più avanti nel presente rapporto.

4.2.1 Sul concetto di software open-source

Il termine Open Source, originariamente Open Source Software (OSS), trova nel concetto di "*accessibile pubblicamente*" uno dei suoi requisiti fondamentali. In un contesto di globalizzazione spinta, come quello in cui l'intero sistema mondiale è attualmente confluito grazie ad internet e ai social media, il termine OSS è pertanto associabile alla possibilità che chiunque lo desideri possa accedervi ed avere un ruolo nella sua gestione e manutenzione. È questo un aspetto non trascurabile in quanto l'eventuale limitazione di accesso a parte della comunità internet porterebbe a critiche, secondo alcuni anche significative, verso la comunità che chiameremo per semplicità "selezionata".

Concetto base di un OSS è infatti quello che chiunque può vederlo, modificarlo e distribuirlo secondo le proprie necessità. L'OSS, come chiarisce ad esempio la RedHat, multinazionale impegnata nello sviluppo di OSS, viene "*sviluppato tramite un approccio decentralizzato e collaborativo, che si basa sulla peer review,"* con il supporto di professionisti IT e tecnici noti, *"e sul lavoro della community*".

Questo, è evidente, rende decisamente interessante l'approccio OSS, ma al contempo porta con sé un problema di fondo che rende la soluzione open source di complessa, se non difficile attuazione qualora si voglia applicare ad un software che ha la necessità di garantire la qualità e adeguatezza del risultato finale dei suoi calcoli.

Pensando ad OpenBPS nella sua configurazione OSS, infatti, uno degli aspetti più critici che si ritiene importante evidenziare è legato all'utenza finale. Prosegue infatti RedHat: "*Il software open source viene rilasciato con una tipologia specifica di licenza che rende il codice sorgente legalmente fruibile dagli utenti finali". Indicativamente, ma su questo è necessario un ulteriore approfondimento legale, le licenze devono avere carattere internazionale e il software, per essere un OSS, deve essere "disponibile in forma di codice sorgente senza costi aggiuntivi; l'utente potrà quindi visualizzare il codice del software e modificarlo in base alle sue esigenze" e ancora "essere reimpiegato in nuovi software; chiunque potrà quindi usarlo per creare e distribuire nuovi programmi*".

Un altro aspetto da considerare, connesso direttamente con quanto esplicitato sopra, è legato alla gratuità o meno dell'OSS. Come ribadisce RedHat "*il fatto che un software sia open source non significa necessariamente che il software eseguibile sia ceduto gratuitamente, ma implica la disponibilità gratuita del codice sorgente*". Ciò impone una riflessione circa i gradi di libertà che si potranno avere nella progettazione del sistema oggetto della presente relazione. In sintesi, il codice sorgente deve essere disponibile liberamente e gratuitamente, mentre il codice eseguibile, per l'elaborazione del quale sono necessarie competenze tecniche ed informatiche, potrebbe non esserlo. Applicando questo concetto ad OpenBPS, si potrebbe concludere che coloro che utilizzeranno il codice sorgente e ne creeranno l'eseguibile e le relative interfacce potranno porre sul mercato software brandizzati, così come potrebbe essere prodotto dalla comunità "*selezionata*" un eseguibile che a quel punto non necessariamente dovrebbe essere gratuito.

A voler approfondire il tema, si rischierebbe di sconfinare un po' troppo nell'etica. Vale comunque la pena allargare il discorso al cosiddetto "*software libero*" chiarendone la differenza con il software open source.

Afferma la già citata e più nota enciclopedia libera del web: "*Il primo [n.d.r. software libero], nato agli inizi degli anni ottanta, indica software la cui licenza soddisfa una definizione in quattro punti prevalentemente basata su concetti etici, quali la possibilità di studiare, di aiutare il prossimo, di favorire l'informatica; il secondo* [n.d.r. software open source], *nato alla fine degli anni novanta si riferisce* […] *ad una serie di 10 punti pratici che definiscono quali criteri legali debba soddisfare una licenza per essere considerata effettivamente libera, ovvero, con il nuovo termine, open source*".

I requisiti del **Software Libero** sono traducibili nelle seguenti quattro libertà (come definite da www.gnu.org):

1. Libertà di eseguire il programma come si desidera, per qualsiasi scopo.
2. Libertà di studiare come funziona il programma e di modificarlo in modo da adattarlo alle proprie necessità. L'accesso al codice sorgente ne è un prerequisito.
3. Libertà di ridistribuire copie in modo da aiutare gli altri.
4. Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti da voi apportati (e le vostre versioni modificate in genere), in modo tale che tutta la comunità ne tragga beneficio. L'accesso al codice sorgente ne è un prerequisito.

mentre i principi base dell'**open source** sono:

1. Ridistribuzione libera. La licenza non può impedire ad alcuna parte in causa la vendita o la cessione del software. Chiunque deve poter fare tutte le copie che vuole, venderle o cederle, e non deve pagare nessuno per poter fare ciò.
2. Codice sorgente. Il programma deve includere il codice sorgente. Codice deliberatamente offuscato non è ammesso. Questo in quanto il codice sorgente è necessario per modificare o riparare un programma.
3. Opere derivate. La licenza deve permettere modifiche e opere derivate e deve consentire la loro distribuzione sotto i medesimi termini della licenza del software originale, in quanto il software serve a poco se non si può modificare per fare la manutenzione, ad esempio, per la correzione di errori o il porting su altri sistemi operativi.
4. Integrità del codice sorgente dell'autore. La licenza può proibire che il codice sorgente venga distribuito in forma modificata solo se la licenza permette la distribuzione di pezze ("patch file") con il codice sorgente allo scopo di migliorare il programma al momento della costruzione.
5. Nessuna discriminazione contro persone o gruppi. La licenza deve essere applicabile per tutti, senza alcuna discriminazione per quanto nobile possa essere l'obiettivo della discriminazione. Ad esempio, non si può negare la licenza d'uso neanche a forze di polizia di regimi dittatoriali.
6. Nessuna discriminazione di settori. Analogamente alla condizione precedente, questa impedisce che si possa negare la licenza d'uso in determinati settori, per quanto questi possano essere deplorevoli. Non si può dunque impedire l'uso di tale software per produrre armi chimiche o altri strumenti di distruzione di massa.
7. Distribuzione della licenza. I diritti relativi al programma devono applicarsi a tutti coloro ai quali il programma sia ridistribuito, senza necessità di esecuzione di una licenza aggiuntiva.
8. La licenza non dev'essere specifica a un prodotto. I diritti relativi a un programma non devono dipendere dall'essere il programma parte di una particolare distribuzione di software.
9. La licenza non deve contaminare altro software. La licenza non deve porre restrizioni ad altro software che sia distribuito insieme a quello licenziato.
10. La licenza deve essere tecnologicamente neutra. Nessuna clausola della licenza deve essere proclamata su alcuna singola tecnologia o stile di interfaccia.

Un ultimo accenno, forse banale, è utile farlo relativamente alla differenza tra **software proprietario** e **software open o libero**. Il primo non consente l'accesso al codice sorgente. Un esempio conosciuto relativo a software conosciuti, come riportato da Wikipedia è costituito ad esempio da Adobe Photoshop o Microsoft Windows che sono software proprietari, mentre Kernel Linux è un software libero (Licenza GNU GPL – vedere oltre) e Google Chromium (su cui si basano i browser Chrome o Microsoft Edge) è un OSS. Passare quindi da una configurazione ad un'altra, in termini di diritti e doveri, comporta la individuazione di specifiche licenze.

Utile per riassumere gran parte dei concetti sintetizzati sopra è l'immagine seguente.

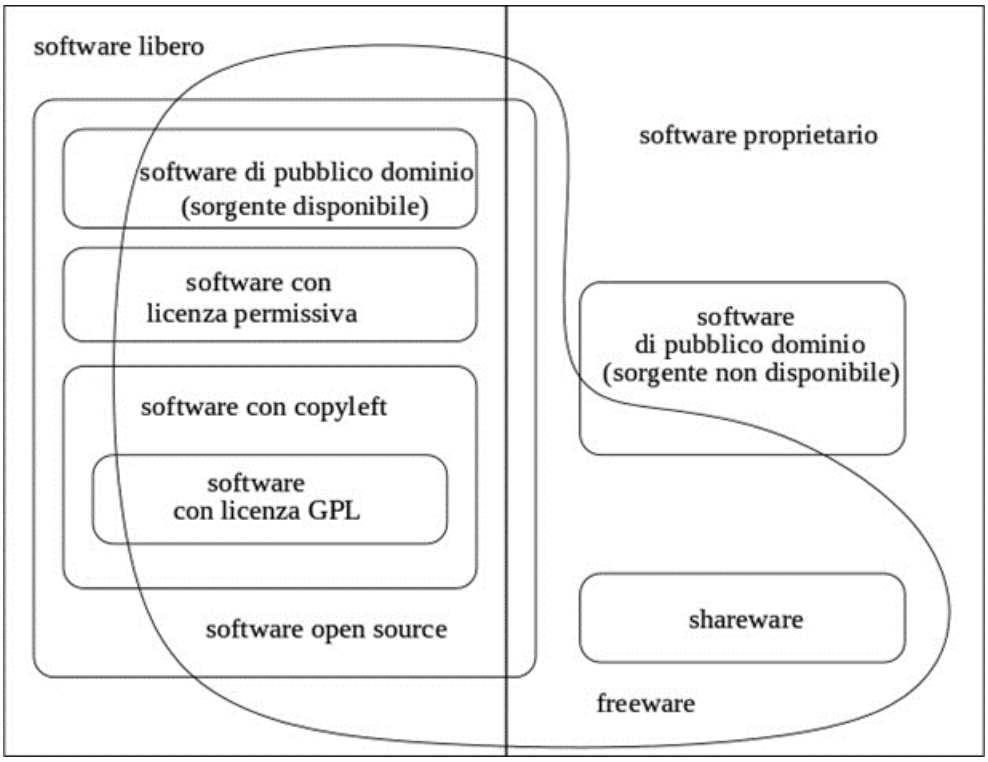


Figura 30 - Schema riassuntivo dei vari livelli di libertà dei software (www.gnu.org)

Una prima conclusione a cui si arriva è che il concetto di open source prevederebbe una serie di requisiti che in prima analisi sembrerebbero non potersi applicare pienamente ad OpenBPS e che in ogni caso è necessario acquisire il parere di un legale per le implicazioni che la distribuzione più o meno libera del codice sorgente potrebbe avere nei confronti dei componenti della community coinvolta e negli utenti finali. Alcuni di questi aspetti sono però oggetto di un ulteriore approfondimento che ci porta al complesso mondo delle licenze, meglio dettagliato al paragrafo seguente

Sulla base di quanto sopra, si potrebbe tentare anche di individuare una motivazione per la quale ha senso attivare un processo di gestione di un software OSS. Viene in aiuto il sito GitHub con alcune considerazioni in merito e spiega che ci sono molte ragioni per le quali un programmatore desidera o ritiene utile andare in questa direzione:

- **collaborazione estesa**:

  un progetto Open Source può accettare input da chiunque nel mondo senza alcuna limitazione.

- **possibilità di utilizzi differenziati**:

  un progetto Open Source può essere utilizzato da chiunque per qualunque scopo.

- **trasparenza**:

  chiunque può mettere la testa nel progetto e individuare errori e malfunzionamenti. La trasparenza in taluni contesti può essere elemento discriminate per l'utilizzabilità o meno del software.

Da questi tre punti, ad esempio, emerge come il secondo possa essere particolarmente critico se applicato al caso di studio di OpenBPS, in quanto ci potrebbe essere il concreto rischio che si moltiplichino algoritmi paralleli che confondono il mercato, ma si tornerà su questo aspetto più avanti nel testo.

Da non trascurare, inoltre, un aspetto particolare che caratterizza OpenBPS e che in qualche modo può influenzare in modo significativo l'evoluzione del protocollo di cui al presente rapporto: il fatto che il motore

primo del software è un soggetto universitario, pubblico, che ha elaborato il nucleo iniziale del codice sorgente, grazie a contributi pubblici. A tal fine, può essere utile ricordare che il settore è in parte disciplinato dal "Codice dell'amministrazione digitale" di cui al D.Lgs. n. 82/2005 e s.m.i.

Un ulteriore elemento che vale la pena richiamare a conclusione di questo capitolo è relativo alla possibilità o meno di vendere un OSS o i suoi derivati. In altre parole, l'approccio OSS significa necessariamente che lo stesso software debba essere gratuito? La risposta, che fornisce ad esempio GitHub, è che la definizione di Open Source non comprende il concetto di "*free of charge*", ma quest'ultimo è diretta conseguenza del fatto che un software con i requisiti Open Source è più logico che sia gratuito altrimenti nessuno parteciperà alla comunità. Ma è anche vero, continua GitHub, che esistono varie soluzioni per vendere parti del software. Basta scegliere la licenza adeguata e capire a cosa applicarla.

4.2.2 Le licenze

Il tema delle licenze diventa fondamentale nel contesto del presente studio in quanto la scelta della corretta "*protezione*" del codice sorgente OpenBPS e dei relativi applicativi è direttamente legata alle modalità di gestione della comunità interessata. In sintesi, la licenza determina le possibili opzioni che si potranno attuare durante la manutenzione, l'utilizzo e la distribuzione dello stesso codice sorgente, degli applicativi o delle librerie.

Il concetto di base è che ogni software deve essere corredato di licenza. Questa generalmente costituisce un file "txt" distribuito o reso disponibile con il software e che non può essere modificato e nemmeno tradotto (tranne specifica volontà del suo autore) per evitare utilizzi impropri dello stesso. In sintesi, vale solo quello emesso dall'autore.

La licenza più conosciuta per un OSS è la **General Public License** "**GPL**" che richiede, come criterio base, che tutti gli elaborati che ne sono coperti vengano accompagnati necessariamente dal testo della stessa licenza.

La GPL si basa sui seguenti assunti:

- non esiste alcun vincolo di modifica fino a quando il prodotto che ne deriva è gestito all'interno dell'ambito in cui avviene la modifica stessa. In altre parole, se colui che ha modificato il sorgente utilizza il nuovo sorgente per propri applicativi interni, anche in un contesto commerciale, non c'è nessun obbligo di rendere disponibile il codice sorgente degli applicativi.
- se si rende disponibile a terzi, gratuitamente o a titolo oneroso, l'applicativo derivato da un codice sorgente OSS modificato, è necessario fornire anche il codice sorgente modificato. La licenza applicabile sarebbe sempre la GPL e non potrebbe essere più restrittiva, pertanto, anche il nuovo codice sorgente si configurerebbe come OSS GPL indipendentemente dall'entità delle modifiche.
- non è possibile realizzare un programma non OSS, quindi "*closed*" utilizzando librerie tutelate da GPL. Il programma che utilizza quelle librerie deve necessariamente avere una licenza GPL.

Per gestire quest'ultimo punto è stata creata la licenza **Lesser GPL** (**LGPL**), che consente un grado di riservatezza superiore per alcune parti (generalmente minori) del software. È il caso, ad esempio, delle cosiddette librerie statiche, cioè di quelle sezioni dell'OSS pensate per essere copiate in un software non OSS. Le librerie statiche si differenziano dalle librerie dinamiche, o a collegamento dinamico (Dynamic-Link Library) laddove queste ultime sono pensate come elemento esterno, indipendente, che può essere richiamato dall'eseguibile in questione.

La scelta di utilizzare una LGPL dovrebbe essere in capo al titolare dell'attività di programmazione che valuterà l'opportunità o meno di consentire questa facoltà in funzione di quanto ritiene utile poter concedere gradi di riservatezza aggiuntivi a chi altrimenti non utilizzerebbe quelle librerie perché troppo libere.

Quindi, mentre una libreria statica nel momento in cui viene copiata nel sorgente derivato da un OSS sotto GPL per potersi svincolare del sorgente base deve essere coperta da LGPL, le librerie dinamiche, ad esempio

le note ".dll" di Windows o similari sono generalmente utilizzabili in Non OSS in quanto si presuppone che non siano derivate dall'originale potendone garantire la fedeltà al sorgente base, indipendentemente dalla licenza associata (GPL o LGPL)

La GPL impone inoltre che ogni versione derivata abbia il nome dell'autore come elemento dello stesso codice. Questo ovviamente ha una ricaduta sulla titolarità del codice derivato che formalmente appartiene da quel momento in avanti a colui che l'ha modificato.

Solo a titolo di esempio si riporta un estratto di Wikipedia che descrive una delle più note licenze GPL. "*Per quanto riguarda il free software una delle licenze utilizzata è la GNU General Public License (GPL). Questa licenza […] mira a rispettare i principi precedentemente esposti, introducendo il copyleft per garantirli. Con il termine copyleft si intende quel meccanismo per cui dopo un'elaborazione di un software libero è vietato ridistribuirlo come software proprietario ed è inoltre necessario segnalare le modifiche attuate all'autore del software. La motivazione è ovvia: alcuni potrebbero approfittarne, spezzando così la catena creativa che ha dato vita al progetto. Questa licenza permette agli autori di mettere la propria proprietà intellettuale al servizio della comunità*".

Esistono poi licenze specifiche per la pubblica amministrazione quando opera su software elaborati internamente o comunque OSS. Si ricorda ad esempio la licenza EUPL, basata sulla GPL, ma adattata dall'Unione Europea per specifiche applicazioni nel campo delle PA. È quindi oltremodo importante fare riferimento a figure legali esperte in materia.

Un ultimo cenno è utile venga fatto al cosiddetto concetto "***Copyleft***" che caratterizza il mondo degli F/OSS. Si riprende per semplicità, quanto definito da Wikipedia in materia.

L'espressione copyleft (talvolta indicata in italiano con "permesso d'autore"), indica un modello di gestione dei diritti d'autore basato su un sistema di licenze attraverso le quali l'autore (in quanto detentore originario dei diritti sull'opera) indica ai fruitori dell'opera che essa può essere utilizzata, diffusa e spesso anche modificata liberamente, pur nel rispetto di alcune condizioni essenziali. Il copyleft può essere applicato ad una moltitudine di opere, che spaziano dal software alle opere letterarie, dai video alle opere musicali, dalle banche dati alle fotografie. L'utilizzo commerciale di lavori copyleft differisce da quello dei lavori coperti da diritti di proprietà intellettuale. Tale utilizzo può includere anche l'aggirare la licenza acquisendo conoscenza del lavoro, o del modello di servizio di un lavoro copyleft. Generalmente ci si attende che i profitti finanziari di un business "*copyleft*" siano inferiori di quelli generati da un business che utilizza lavori proprietari. Ditte con prodotti proprietari possono far soldi con vendite esclusive, dal possesso esclusivo o trasferito, e lucrare sulle cause per i diritti di una creazione.

Il copyleft è l'abbreviazione di un meccanismo legale che assicura che i prodotti derivati da un lavoro coperto da licenza rimangano liberi (cosa che non è obbligatoria in un approccio "*open source*"). Se il concessionario di un lavoro coperto da copyleft distribuisce dei lavori derivati che non sono coperti dalla stessa (o in alcuni casi da una simile) licenza copyleft, allora dovrà affrontare delle conseguenze legali: per molti lavori in copyleft questo perlomeno implica che alcune condizioni della licenza cessino, lasciando il (precedente) concessionario senza il permesso di copiare e/o distribuire e/o mostrare pubblicamente e/o preparare prodotti derivati dal software, etc.

Nuovi modelli di business possono avvantaggiarsi delle particolarità dei lavori copyleft, ad esempio permettendo a programmatori volontari e a organizzazioni di sentirsi coinvolti e contribuire allo sviluppo; inoltre, il "*far parte della comunità*" aiuta a mantenere l'idea che ci si "*possa fidare*" di un'opera anche molto complessa, la cui creazione viene divisa e verificata dalla comunità nel suo complesso.

Un elenco e qualche approfondimento sulle licenze disponibili attualmente è riportato nell' Appendice B .

4.2.3 Sulle modalità di condivisione del codice sorgente

Indipendentemente dall'approccio che si ritiene utile seguire per lo sviluppo del software e dei gradi di libertà che si vuole dare allo stesso attraverso una specifica licenza, un passaggio obbligato è l'individuazione dello

strumento e della relativa piattaforma da utilizzare per la gestione della comunità che elaborerà il codice di calcolo, dando per consolidato l'approccio di costituire la citata comunità di sviluppatori, aperta o chiusa che sia.

Infatti, per consentire l'elaborazione di un software per il quale si prevede una continua manutenzione ed integrazione è necessario attuare il cosiddetto "*controllo* (di) *versione*" inteso come quella serie di azioni necessarie per gestire versioni multiple dell'elaborato.

Come chiarisce Wikipedia "*il controllo versione è qualunque pratica che tiene traccia e permette di controllare i cambiamenti al codice sorgente prodotti da ciascun sviluppatore, condividendone allo stesso tempo la versione più aggiornata o modificata da ciascuno mostrando dunque in tempi brevi lo stato di avanzamento del lavoro di sviluppo*".

Questo approccio è utilizzato comunemente, ad esempio, anche per i più noti file elaborati con Microsoft Word o Google Documents e condivisi tra più persone o elaborati dallo stesso utente che vuole distinguere versioni successive del proprio lavoro. Allo stesso modo, ad esempio, è gestita la cronologia di Wikipedia.

Da questo punto di vista, uno dei sistemi controllo versione (VCS) più diffuso ed utilizzato è basato su Git, ma oltre ad esso si rilevano SNV, Mercurial, Bazaar, Darcs, ecc.

Su di essi vengono poi ad integrarsi delle soluzioni applicative che consentono di ospitare (repository) il codice sorgente e di gestire il suo sviluppo collaborativo controllandone le varie versioni e gli accessi nonché gli accessi. Tra le soluzioni applicative basate sul sistema Git, si ricordano GitHub e Apache Allura.

Il principio su cui si basa il controllo versione è da un lato l'identificazione univoca di ogni nuova versione e dell'autore coinvolto e dall'altro la possibilità di evitare che due sviluppatori lavorino sulla stessa parte di codice senza che si riesca poi a ricostruire versioni ed autori. La condivisione, infatti, genera dei conflitti che devono essere gestiti o all'origine, evitando ad esempio che due sviluppatori possano accadere creando dei blocchi (lock) alle sezioni in lavorazione, o ex post chiedendo ad uno dei due sviluppatori di risolvere in conflitto qualora questo si verifichi. L'efficacia e l'oggettività della prima soluzione rispetto all'altra è evidente.

È utile evidenziare, anche se trattasi di un concetto evidente, che il ricorso ad una piattaforma di condivisione è essenziale per consentire di lavorare a più utenti da remoto e da macchine diverse. Mentre è superflua nel caso il sorgente sia disponibile su una singola macchina e gli accessi allo stesso codice siano gestiti in locale sulla stessa macchina.

Tornando ai sistemi di controllo di versione si possono citare sistemi locali, centralizzati e distribuiti.

I primi (VCS) agiscono su una sola macchina con tutte le correlate limitazioni, da un lato, e facilità di controlli dall'altro.

I secondi (CVCS) invece prevedono l'esistenza di un server centrale che colloquia con le macchine esterne su cui gli sviluppatori lavorano. Questa modalità offre vari vantaggi rispetto ai sistemi locali ma presenta anche qualche criticità. A livello amministrativo è facile controllare gli accessi e il numero di operatori che possono entrare nella comunità è elevato. Il server centralizzato, così come nei sistemi locali, diventa il collo di bottiglia sul quale è necessario focalizzare l'attenzione per garantire che il lavoro sia accessibile quando necessario e che vengano fatte le necessarie copie di backup.

Il terzo livello è costituito sistemi Distribuiti (DVCS) come il Git e gli altri sistemi citati sopra. Con l'approccio distribuito, ogni singolo client provvedere a creare una copia identica di quanto presente nel repository, comprensiva di tutte le versioni. Questa soluzione protegge da eventuali problemi che dovessero bloccare o rendere inaccessibile il server e/o uno o più client, purché ovviamente ne rimanga almeno uno funzionante ed accessibile; ma consente anche un elevato livello di interazione tra programmatori che potrebbero lavorare a gruppi su versioni o parti di codice sorgente diverse simultaneamente. Oltre a questi aspetti, l'approccio DVCS presenta vari vantaggi quali il fatto che il singolo programmatore opera localmente su ogni singolo file e questo determina una velocità di azione significativa rispetto ai sistemi CVCS.

Qualche ulteriore approfondimento è utile farlo pensando direttamente ad una delle piattaforme citate, Git, ritenendola adeguata alle necessità di cui al presente studio1. Oltre ai requisiti citati sopra si rileva che Git ha un elevato livello di integrità. In altre parole, come si rileva dal sito https://git- scm.com "*qualsiasi cosa in Git è controllata, tramite checksum, prima di essere salvata ed è referenziata da un checksum. Questo significa che è impossibile cambiare il contenuto di qualsiasi file o directory senza che Git lo sappia. Questa è una funzionalità interna di basso livello di Git ed è intrinseca della sua filosofia. Non può capitare che delle informazioni in transito si perdano o che un file si corrompa senza che Git non sia in grado di accorgersene. Il meccanismo che Git usa per fare questo checksum è un hash chiamato SHA-1. Si tratta di una stringa di 40-caratteri, composta da caratteri esadecimali (0–9 ed a–f) e calcolata in base al contenuto di file o della struttura della directory in Git. Un hash SHA-1 assomiglia a qualcosa come*:

24b9da6552252987aa493b52f8696cd6d3b00373

Vedrai questi hash dappertutto in Git perché li usa tantissimo. Infatti, Git salva qualsiasi cosa nel suo database, e il riferimento ad un file non è basato sul nome del file, ma sull'hash del suo contenuto.".

In queste poche righe compare per la prima volta un elemento importante: una sorta di identificativo univoco o chiave che identifica l'oggetto da cui è stata generata. Questo si chiama genericamente Hash e la sua declinazione nel caso specifico è lo SHA-1, ma ne esistono differenti forme e versioni.

Il concetto di firma digitale, o meglio e più estesamente, il concetto di validazione o protezione di un oggetto informatico è abbastanza complesso e in certi ambiti potrebbe sfociare in approcci tipici dei registri distribuiti e delle blockchain. Questa non è la sede adatta a sviluppare in modo approfondito la tematica; è utile, però, richiamare alcuni concetti, seppure parziali e non esaustivi, di garanzia di contenuti applicata al software o all'output di un software.

È elemento noto che quando non si desidera che soggetti terzi accedano ad un dato informatico è opportuno procedere con la cosiddetta crittografia del dato. Questa è una delle più diffuse e antiche modalità di protezione delle informazioni; tra le più conosciute si ricordano le codifiche utilizzate per rendere incomprensibili i messaggi militari durante i conflitti mondiali. L'evoluzione della tecnologia ha portato oggi ad algoritmi complessi (chiavi) che consentono di rendere illeggibili le informazioni trasmesse per chi non ha la chiave giusta, di identificare il mittente o il creatore dell'informazione, di identificare e/o autorizzare il destinatario a leggere l'informazione, di garantire che l'informazione abbia determinate caratteristiche. Chiavi pubbliche o private, crittografia simmetrica o asimmetrica, sono i termini forse più banali che consentono di inquadrare meglio questo ambito. La differenza e la complessità di una modalità di crittografia rispetto all'altra determinano il livello di affidabilità con cui si riesce a garantire ciò che si desidera garantire. Ad esempio, la crittografia simmetrica è una modalità di crittografia che prevede che sia il mittente che il destinatario abbiano la stessa chiave (privata). Se un soggetto terzo si impossessa della chiave, la crittografia perde il suo valore aggiunto. La tecnologia asimmetrica si basa invece su un sistema costituito da una chiave pubblica che tutti possono conoscere e da una chiave privata che conosce solo chi è autorizzato ad usarla. Chi invia il dato lo codifica con la chiave pubblica, chi lo decodifica utilizza una chiave privata che nemmeno il mittente conosce. Il livello di sicurezza del dato è quindi molto elevato.

Un sistema crittografico utilizzato generalmente per proteggere o identificare degli oggetti informatici è costituito dalle cosiddette funzioni Hash. Secondo Wikipedia "la funzione di hash o funzione hash produce una sequenza di bit, detta digest, (o una stringa) strettamente correlata con i dati in ingresso". Continua Wikipedia "La lunghezza dei valori di hash varia a seconda degli algoritmi utilizzati. Il valore più comunemente adottato è di 128 bit, che offre una buona affidabilità in uno spazio relativamente ridotto. Tuttavia, va registrata la possibilità d'uso di hash di dimensione maggiore (SHA, ad esempio, può anche fornire stringhe di 224, 256, 384 e 512 bit) e minore (che però è fortemente sconsigliato).

Le funzioni hash svolgono un ruolo essenziale nella crittografia: sono utili per verificare l'integrità di un messaggio, poiché l'esecuzione dell'algoritmo su un testo anche minimamente modificato fornisce un message digest completamente differente rispetto a quello calcolato sul testo originale, rivelando la tentata modifica.

Le funzioni hash possono offrire diversi livelli di sicurezza (debole o forte) in funzione di quanto sia facile "computazionalmente" fare in modo che un'informazione diversa da quella originale dia un hash uguale. Ma al di là dei termini, la sicurezza di una chiave hash è elevata.

In generale si ritiene che una funzione hash possa avere i seguenti aspetti positivi: non richiede una elevata capacità e potenza di calcolo; è di lunghezza fissa (i bit di cui sopra) indipendentemente dalla mole di dati in ingresso; una qualunque, anche minima, modifica del dato di input nella funzione determina una chiave hash diversa; non è possibile da un hash risalire ai dati che l'hanno creata.

Riassumendo, si potrebbe affermare che una chiave hash è una sintesi fedele, facile da ricavare e avente lunghezza predefinita di un qualunque insieme di dati (oggetto informatico).

Tra le più note funzioni hash si citano ad esempio le MD5, le RIPEMD, le SHA. E una delle più usate, che a seguito di approfondimento, sembrerebbe essere applicabile anche nel contesto del protocollo di cui al presente studio è la SHA-256 che offre un livello di sicurezza (resistenza alle collisioni – cioè alla possibilità di trovare due oggetti che diano la stessa chiave) elevato.

Dopo la necessaria, per quanto si vedrà più avanti, digressione sulla crittografia, è opportuno ritornare alla piattaforma tipo GitHub, analizzandone gli aspetti più significativi correlati al presente lavoro. Come funziona una piattaforma per la gestione di un OSS?

Tutto parte da un luogo/spazio (repository) in cui è caricato il codice sorgente originale e che può avere dimensioni variabili in funzione delle ambizioni di progetto. Il repository può essere pubblico, nel più classico degli approcci Open Source oppure privato. Questo è il primo elemento su cui è necessario prendere una decisione, come verrà ribadito successivamente, perché ovviamente influenza le caratteristiche della comunità che andrà a lavorare sul codice sorgente. Successivamente, una volta acquisito che una piattaforma di questo tipo deve consentire il controllo delle versioni secondo quanto chiarito sopra, il concetto base dei sistemi più diffusi è quello di operare attraverso le cosiddette ramificazioni (*branching* o *forking)* e le fusioni (*merging*). In sostanza uno sviluppatore della comunità duplica la parte di codice sorgente su cui vuole lavorare e apporta tutte le modifiche che ritiene. Una volta che la parte di codice modificata è pronta, può essere fusa con il sorgente principale facendone richiesta (*pull request*) all'amministratore competente ovviamente dopo che la modifica è stata testata e se ne è verificata la funzionalità e adeguatezza. Ogni passaggio è chiaramente monitorabile, annullabile e le precedenti versioni ripristinabili.

Il meccanismo è quindi relativamente semplice, fatti salvi ovviamente tutti gli aspetti tecnici che dovranno essere successivamente approfonditi e condivisi con la comunità selezionata per supportare lo sviluppo e la manutenzione del progetto.

Nell'Appendice C è riportato qualche ulteriore elemento utile a individuare la piattaforma più adeguata al progetto di cui al presente rapporto.

4.3 Protocollo di sviluppo

Lo sviluppo di un software, indipendentemente dall'uso di specifiche piattaforme di condivisione, prevede varie fasi, alcune delle quali banali, ma la cui gestione aiuta a impostare correttamente un processo che dovrà durare nel tempo. Ricostruendo un possibile percorso decisionale sulla base di varie fonti internet e di bibliografia, si ritiene utile evidenziare i seguenti passaggi, non necessariamente da applicare nel rigoroso ordine in cui sono elencati:

1. Definizione di un Team di Gestione dell'intero progetto
2. Individuazione dell'oggetto su cui lavorare e del suo mercato di riferimento
3. Individuazione della licenza o delle licenze associabili al progetto nel suo complesso e/o a parti dello stesso
4. Scelta della piattaforma di gestione del progetto

5. Individuazione delle caratteristiche di visibilità del repository (pubblico o privato)
6. Creazione del repository, individuazione di un metodo di numerazione delle versioni successive, definizione delle regole con cui deve essere gestita la documentazione che deve accompagnare ogni singola parte del software sia per gli sviluppatori che per gli utenti
7. Assegnazione dei ruoli dei singoli attori e delle regole di ingaggio
8. Definizione delle modalità di verifica e accettazione delle modifiche ed integrazioni al codice sorgente
9. Definizione della modalità di compilazione dell'applicativo
10. Definizione delle modalità di verifica della conformità degli applicativi commerciali al codice sorgente originale

Di seguito alcune considerazioni relativamente ad ognuna delle voci precedenti.

4.3.1 Definizione di un Team di Gestione dell'intero progetto

Primo passo fondamentale dello sviluppo del progetto di cui al presente rapporto, è la costituzione di un gruppo di persone che garantisca adeguata capacità gestionale e decisionale a supporto della progettazione dell'intero percorso e del suo successivo mantenimento nel tempo. Si tratta di un team che ha il compito di mettere a terra il progetto, farlo partire e supervisionarne l'evoluzione da un punto di vista generale ed in relazione al contesto generale che si andrà via via mutando.

Più avanti vengono definite alcune figure chiave che costituiranno l'ossatura della comunità di sviluppo; queste non necessariamente devono essere tutte coloro che hanno inizialmente progettato il sistema, anche se, ovviamente è preferibile che in un caso come questo, non venga perso il background che ha portato a progettare e a gestire i primi passi all'interno percorso.

Il Team di Gestione si ritiene debba necessariamente essere composto da una persona di riferimento per ognuno degli enti che sta collaborando a far partire l'iniziativa, quindi Politecnico di Milano e l'ENEA, ed è auspicabile che possa prendervi parte anche un esponente dell'Ente Normatore italiano, UNI, oggi rappresentato nello specifico settore dal Comitato Termotecnico Italiano (CTI). A questi va certamente affiancato un esperto informatico con adeguato know-how sulla progettazione dei software e sulle comunità di sviluppo, che aiuterà a progettare nel migliore dei modi tutto il sistema e a indirizzare, grazie a conoscenza più approfondite di quelle ricavabili dal presente rapporto, le scelte di seguito evidenziate. Un legale, magari solo per la fase di avvio, per l'approfondimento delle problematiche inerenti alle licenze, potrebbe completare il team.

Da valutare, in seconda battuta, se il Team di Gestione debba essere integrato permanentemente o solo in determinati contesti con figure di spicco dell'attuale settore della modellazione dell'edificio, ad esempio con ruoli ed esperienza simili a quelli propri di coloro che oggi partecipano alla Cabina di Regia CTI per il mandato M/480 sulle norme tecniche a supporto dell'EPBD. Questo consentirebbe di migliorare la supervisione dell'intero progetto garantendo adeguata competenza del Team Gestionale in ognuno dei settori di rilievo del sistema oggetto di modellazione. Gli stessi potrebbero poi essere figure anche più operative come si vedrà più avanti.

Il Team di Gestione deve:

- individuare tra i suoi componenti una figura centrale a cui compete la direzione e gestione operativa del team,
- individuare i ruoli dei singoli componenti, se del caso, e conseguentemente attribuire ad essi eventuali deleghe con le connesse specifiche operative,

- dotarsi di regole operative e decisionali (modalità di voto in caso di mancanza di consenso unanime su determinati aspetti, modalità di circolazione della documentazione, di convocazione delle riunioni, ecc.)
- individuare le figure necessarie alla corretta gestione della piattaforma di sviluppo del software prendendo spunto da quanto descritto nel presente documento,
- predisporre le "*call for expert*" per il reclutamento degli sviluppatori,
- condividere le regole di ingaggio per gli esperti esterni alla comunità che svolgeranno i test applicativi del software e delle relative versioni.

4.3.2 Individuazione dell'oggetto su cui lavorare e del suo mercato di riferimento

Nel contesto specifico in cui si sviluppa OpenBPS, è evidente che l'oggetto su cui lavorare è già definito a priori e addirittura ne è già stato costruito un codice sorgente base. Quello che manca al momento della redazione del presente rapporto è un quadro complessivo di quale dovrà essere il risultato finale e soprattutto il contesto in cui il motore di calcolo proprio di OpenBPS andrà ad operare. Il tema non è banale perché da questo dipendono ulteriori decisioni a cascata.

Sul tavolo al momento vi è solamente un "core" costituito da un codice sorgente con funzionalità limitate e applicabili solo a determinati oggetti e l'idea di fondo di sviluppare un software in grado di modellizzare il sistema edificio-impianto. Manca ancora una visione complessiva consolidata di quale sarà la funzione finale del software e gli utenti dello stesso. Questo determina il primo grado di libertà, ampissimo a questo punto del percorso logico che stiamo sviluppando, che influenza la maggior parte delle scelte successive. Ad esempio, un vero e proprio OSS si ritiene non possa costituire la base per un applicativo al quale sarà richiesto di definire dei valori, ad esempio, di prestazione energetica di un edificio sui quali si basano certificati con valore legale come gli Attestati di Prestazione Energetica. La libera diffusione del codice sorgente, anche se da un certo punto in avanti dovesse essere regolamentata con specifiche licenze ad esempio su librerie applicative, potrebbe determinare una eccessiva circolazione dei codici di calcolo e di potenziali varianti al di fuori dei sistemi di controllo adottati all'interno del protocollo OpenBPS. Diverso invece è il discorso se il software lo si indirizzasse verso un libero uso per finalità di studio e di sviluppo di sempre nuove funzionalità. Le variazioni sul tema tra i due casi estremi citati sono comunque numerose.

L'individuazione del mercato di riferimento e conseguentemente la scelta tra una delle due opzioni relativamente alla diffusione del software finale o l'individuazione di una soluzione intermedia che permetta, ad esempio, una maggior libertà fino ad un certo punto dello sviluppo del codice sorgente, è una milestone di progetto che andrebbe risolta con adeguata lungimiranza.

È inoltre necessario, come specificato più avanti individuare una modalità di validazione del codice sorgente originale, in quanto, seppure elaborato in un contesto la cui competenza è fuori discussione, le possibili destinazioni d'uso del software finale richiedono un estremo rigore nella verifica di ogni suo singolo componente.

4.3.3 Individuazione della licenza o delle licenze associabili al progetto nel suo complesso e/o a parti dello stesso

Direttamente connessa al punto precedente è la scelta delle licenze associabili ad OpenBPS. Come evidenziato nei capitoli introduttivi esistono decine di possibili variazioni sul tema e la scelta dell'una o dell'altra dipende dalle caratteristiche che si vogliono associare al prodotto del lavoro comunitario. La possibilità di produrre, da un certo punto in avanti della vita del software, degli applicativi per i quali è necessario garantire la totale affidabilità dei calcoli impone ad esempio di effettuare scelte diverse circa sia le modalità di sviluppo sia le licenze associabili. Si richiama ad esempio l'attenzione sulle modalità di verifica della conformità dell'eventuale applicativo commerciale al codice sorgente depositato nella repository

originale. Una eccessiva apertura in fase di programmazione del codice sorgente potrebbe comportare una altrettanto libera gestione degli applicativi derivati con l'impossibilità, di fatto, di costruire un processo sufficientemente affidabile di validazione del risultato finale di calcolo derivante dall'utilizzo dell'applicativo. In sintesi, una elevata libertà iniziale potrebbe pregiudicare la qualità del risultato finale.

Parlando di licenze, emerge evidente anche la necessità di coinvolgere un legale esperto in materia chepossa supportare con adeguata competenza le scelte che si andranno a fare.
Inoltre, non marginale è la lungimiranza con la quale si dovrebbe già oggi individuare il grado di affidabilità che dovranno avere i risultati dei calcoli derivanti dall'applicazione del software.
Quest'ultima è pertanto un'ulteriore milestone di progetto che influenza l'intero protocollo oggetto del presente rapporto e conseguentemente anche la selezione delle licenze associabili.
Alla luce di quanto sopra, però, vista la complessità del tema "licenze" e delle implicazioni legali adesso connesse, non è possibile che nel presente documento venga proposta una scelta definitiva.
Solo a titolo di esempio e coerentemente con il punto precedente viene quindi proposta la licenza Pubblica dell'Unione Europea (EUPL) versione 1.2.

4.3.4 Scelta della piattaforma di gestione del progetto

La scelta di una piattaforma ovviamente aiuta a gestire alcuni degli aspetti gestionali descritti sopra, quelli più pratici ed applicativi.

Da questo punto di vista esistono differenti soluzioni, come ad esempio le già citate GitHub o Apache Allura, ma in questa fase non si ritiene necessario approfondirne ulteriormente le caratteristiche, ritenendole tutte paragonabili e partendo dal concetto che spesso la preferenza viene accordata ad una piattaforma piuttosto che all'altra in funzione dell'esperienza pregressa di chi dovrà decidere in merito. Ovviamente alcune piattaforme hanno un costo, anche se di poche centinaia di euro all'anno, mentre altre sono gratuite. Un confronto tra alcune di esse è riportato in Appendice C, come estratto dal sito internet di Allura. Qui si vuole solo evidenziare che le piattaforme differiscono per parametri quali:

- licenza associata al motore (GPL, BSD, Apache, MIT, ecc.)
- linguaggio (PhP, Python, Ruby, Java, Perl, ecc.)
- database (Mongo, Any SQL, My SQL, PostgreSQL, ecc.)
- sistema Git
- possibilità di gestire richieste di Fork o Merge,
- ticket tracker
- Wiki e Forum

La scelta della piattaforma è legata a decisioni pregresse, come spiegato nei punti precedenti e non può trascurare il concetto che un vero OSS deve concettualmente consentire la completa libertà di ingresso di potenziali collaboratori nella comunità di sviluppo. Da questo punto di vista varie piattaforme di gestione lasciano la libertà di creare repository private o pubbliche.

Ecco, quindi, che anche in questa terza fase del processo di sviluppo del protocollo è necessaria una decisione in merito al grado di libertà che si vuole dare allo stesso.

La piattaforma scelta per la fase di sviluppo del presente progetto fino al suo termine è stata Allura della Apache Foundation, essendo essa stessa un progetto open source ed essendo completamente gratuita. Di contro questa piattaforma necessita di una implementazione su un server dedicato, non essendo previsto un servizio di hosting, come ad esempio per GitHub o GitLab. Si ritiene quindi che la scelta fatta a questo stadio di sviluppo del protocollo può ovviamente essere rivista sulla base dell'esperienza delle figure che verranno via via coinvolte e delle decisioni che verranno prese in linea con le criticità espresse nel presente documento.

4.3.5 Individuazione delle caratteristiche di visibilità del repository (pubblico o privato)

Quanto citato in precedenza aiuta a indirizzare una possibile scelta relativamente al presente anello del protocollo per la gestione di OpenBPS.

Indicativamente, sulla base di quanto acquisito fino ad ora, si ritiene che l'accesso riservato ad una comunità selezionata di sviluppatori, attraverso un repository privato, sia la soluzione migliore.

Questo garantirebbe soprattutto un maggior controllo sulla competenza tecnica dei singoli attori oltre che un miglior controllo sulle differenti versioni che gli sviluppatori potrebbero gestire. Una decisione in tal senso è legata sostanzialmente alla scelta, fatta sulla base dei punti precedenti, circa la licenza che a questo punto imporrà alla comunità di sviluppo determinati gradi di libertà.

4.3.6 Creazione del repository, individuazione di un metodo di numerazione delle versioni successive, definizione delle regole con cui deve essere gestita la documentazione che deve accompagnare ogni singola parte del software sia per gli sviluppatori che per gli utenti

Si tratta di azioni pratiche che necessitano però di una certa preparazione e capacità, eventualmente basata su esperienza pregressa, di prevedere come sarà l'evoluzione del lavoro di sviluppo. Il passaggio più banale dei tre citati è quello relativo alla creazione della "cartella" in cui andrà posto il codice sorgente iniziale. La creazione di un repository prevede anche la possibilità di procedere con i cosiddetti "fork" ossia delle azioni di trascrizione dell'intero codice in un altro repository che da quel momento assume nuova vita propria e indipendente nel nuovo ramo. Questa evenienza è praticabile, ad esempio, in funzione dei diritti di lettura/scrittura, assegnati a ciascun sviluppatore oppure in funzione del grado di sviluppo del codice, che da un certo punto in avanti potrebbe essere particolarmente complesso, da rendere più semplice la sua copia fisica in altra cartella per poter lavorare più liberamente su alcune sue parti.

Similmente facile è l'individuazione di una metodologia di numerazione delle versioni. Chi scrive ritiene che si tratti di decisioni non particolarmente critiche e complesse, se all'interno del team decisionale sarà presente una figura con comprovata esperienza nel settore, ma è importante che ad esempio la numerazione abbia già in sé tutti i gradi di libertà che si ritiene possano servire nel corso dello sviluppo del codice. L'aspetto forse più importante è il terzo elemento, legato alla documentazione a supporto. Sinteticamente si tratta di stabilire in anticipo le regole con cui ogni sviluppatore deve documentare il suo lavoro e che dovrà consentire a chiunque prenda in mano l'elaborato, di comprendere senza ulteriori spiegazioni cosa è stato fatto, l'approccio seguito, i risultati, le modifiche inserite, ecc.

Spesso, infatti, si sottovaluta la difficoltà che un soggetto terzo potrebbe incontrare nel prendere in mano e gestire un elaborato, soprattutto in tema di software, sviluppato da altri. E' quindi importante che nella progettazione delle modalità di gestione della comunità si ponga attenzione a questi dettagli, soprattutto considerata la volontà iniziale di avere una elevata compartecipazione allo sviluppo del software. Esempi di documentazione sono i classici file "*Readme*" che possono avere contenuti più o meno complessi a seconda delle regole che ci si è dati.

4.3.7 Assegnazione dei ruoli dei singoli attori e delle regole di ingaggio

Partendo dall'ipotesi che il percorso logico che ha portato fino a questo punto sia stato improntato a definire una comunità privata o comunque non completamente free, si pone come strategico il ruolo e i gradi di libertà che ogni sviluppatore dovrà avere e soprattutto come selezionare gli stessi. Ma gli sviluppatori non sono i soli attori, in quanto la comunità deve avere alcune figure/ruoli chiave, che possono eventualmente coincidere in un'unica persona, quali ad esempio:

- un amministratore: gestisce gli accessi e gli account della piattaforma, verifica il corretto operato dei singoli, controlla l'allineamento della documentazione alle regole definite per la comunità, gestisce l'accesso ai forum, ecc. Questa figura si dovrebbe anche relazionare con il Team di Gestione.

- un esperto del linguaggio di programmazione utilizzato per la scrittura dell'intero codice sorgente, che dovrà, soprattutto in accordo con l'esperto di modellazione energetica, garantire un certo grado di linearità e coerenza allo sviluppo del software, fornire il necessario supporto informatico ai singoli sviluppatori, ecc. Tale figura dovrebbe seguire l'intero sviluppo del software, interloquendo frequentemente con l'esperto di modellazione in presenza di criticità significative nella trascrizione delle formule di fisica tecnica in linguaggio informatico (ad esempio per la connessione tra moduli diversi, per l'individuazione di input e output adeguati, ecc.),
- un esperto di modellazione energetica dell'edificio: supervisiona l'evoluzione del software nel suo complesso dirigendo, se del caso e se necessario, le azioni dei singoli in modo da consentire un'evoluzione organica di ogni singola parte del software, aprendo eventuali nuovi filoni di sviluppo, gestendo almeno un forum generale legato all'intero software, creando le connessioni tra sviluppatori quando se ne ravvisa la necessità o l'utilità, ecc. Questa figura si dovrebbe anche relazionare con il Team di Gestione.
- un esperto per ogni macrosettore di modellazione del sistema edificio-impianto: involucro, climatizzazione invernale, climatizzazione estiva ed eventualmente di sistemi di automazione e controllo che coordini, in stretta collaborazione con l'esperto di cui al punto precedente, il singolo ambito di competenza. Come anticipato sopra queste figure potrebbero coincidere con gli esperti di settore che supportano il Team di Gestione.

La selezione degli sviluppatori è un altro punto nodale del presente rapporto e costituisce milestone di progetto.

La modellazione di parti specifiche di un sistema edificio-impianto è aspetto complesso, ma alla portata probabilmente di molti esperti disponibili oggi nel contesto della simulazione energetica e quindi, ad esempio tra i soci di due potenziali partner nella prosecuzione dello sviluppo quali AiCARR e IBPSA-Italy, e/o anche tra i componenti dei gruppi di lavoro del CTI. Leggermente più di nicchia invece è il livello di conoscenza dell'intero edificio o meglio dell'intero sistema edificio-impianti e delle relazioni che lo stesso ha con l'ambiente circostante, ma sicuramente presente tra i soggetti coinvolti in questa attività di ricerca, ENEA e Politecnico di Milano. Per questa ragione è bene muoversi verso l'individuazione di una persona di riferimento per la visione globale del progetto e di svariati esperti per lo sviluppo di singole parti di codice con esperienza e competenza più o meno settoriale, verticale e di nicchia.

In funzione della complessità del software, è opportuno pensare ad una struttura piramidale in cui pochi sono i responsabili che delegano determinate azioni e ruoli nella comunità.

In prima battuta, senza rendere la struttura della comunità troppo complessa, si ritiene ipotizzabile la creazione di due livelli di sviluppatori. Un primo livello proveniente dal contesto universitario impegnato in prima persona nello sviluppo di parti del codice sorgente e delle relative librerie. Un secondo livello utile e fondamentale per testare i singoli moduli o parti più estese di software, mediante l'applicazione, se del caso, anche di casi studio specifici.

La selezione pertanto dovrebbe tenere conto di alcuni elementi base, che potrebbero essere valutati dal Team di Gestione in prima istanza, e successivamente dai responsabili di settore. Indicativamente come anticipato sopra si ritiene che gli sviluppatori debbano fare riferimento ad un ambito universitario, onde evitare possibili criticità legate all'inserimento nell'organico della comunità privata soggetti che per ragioni commerciali avrebbero evidenti interessi, seppure leciti. Questo però non escluderebbe da processo di sviluppo di OpenBPS di operatori privati e software house commerciali, in quanto potenziali destinatari e utilizzatori del prodotto finito.

Una possibile modalità di selezione di questa figure potrebbe essere attuata mediante una "call for expert" da lanciare in ambito universitario per il primo livello e un reclutamento di operatori interessati sfruttando ad esempio collaborazioni già in essere con gli enti intestatari del presente rapporto (ENEA e Politecnico di Milano) o tavoli di lavoro esistenti, come il gruppo consultivo Software House del CTI, che raggruppa ad oggi

la maggior parte delle case editrici impegnate nello sviluppo di software commerciali per il calcolo delle prestazioni energetiche degli edifici.

Come minimo la selezione dovrà basarsi su un Curriculum adeguato e, in relazione al contesto di mercato del software finale, dovrà garantire l'assenza di eventuali conflitti di interesse che potrebbero pregiudicare la linearità e terzietà d'azione della comunità di sviluppo.

4.3.8 Definizione delle modalità di verifica e accettazione delle modifiche ed integrazioni al codice sorgente

Una volta creato l'intero castello e popolata la comunità di sviluppo con le relative regole, è necessario individuare le modalità di verifica e accettazione di ogni modifica e/o integrazione del codice sorgente. In funzione di quanto deciso nei pertinenti punti precedenti e sulla base dell'approccio generale che le comunità di sviluppo adottano, è possibile tracciare ogni modifica ed integrazione, nonché numerare ogni nuova versione di codice, sia esso l'intero codice o una sua parte o il file eseguibile. L'aspetto più significativo però è proprio validare ogni modifica prima di renderla ufficiale, ma prima ancora, un ulteriore aspetto da non sottovalutare, è se lo sviluppo del codice debba seguire o no uno schema ben definito in fase di progettazione dal Team di Gestione oppure può proseguire secondo la disponibilità, volontà, proattività dei componenti la comunità. Si intende in particolare focalizzare l'attenzione sui gradi di libertà che si possono concedere alla comunità nello sviluppare nuove funzionalità e moduli del software. Ad esempio, è importante stabilire fin dall'inizio se sia possibile espandere un ramo particolare del software fino ai dettagli più spinti mentre altri rami rimangono a livello generale oppure è necessario sviluppare in modo organico e parallelo tutti i rami. Calando il concetto su OpenBPS, una decisione da prendere in fase di progettazione dell'intera struttura di gestione è se sia possibile spingere lo sviluppo della modellazione dell'involucro edilizio a livelli significativi di dettaglio senza che ci sia una pari crescita del modello relativamente all'impiantistica, oppure se la modellazione dell'impiantistica per la climatizzazione invernale possa andare avanti anche in assenza di un lavoro simile sulla climatizzazione estiva. Decisioni in tal senso possono essere legate, ad esempio, alla maturità dell'intero codice o alla volontà di diffondere parti di eseguibile che lavorano indipendentemente si singoli parti del sistema edificio-impianto.

Una volta deciso che tipo di nuove funzioni o parti di codice è possibile accettare, si passa alla definizione delle regole di accettazione. Generalmente le comunità di sviluppo prevedono dei meccanismi chiamati "*request for merging*" o "*merge request*" altri parlando di "*patches*". In questa sede non si approfondisce ulteriormente il tema, ma solo l'approccio alla gestione di modiche ed integrazioni. Le richieste di fatto hanno come oggetto l'inclusione di una modifica o di un nuovo modulo nel codice presente nel repository base. In funzione dei diritti assegnati agli sviluppatori è possibile che si passi o meno da una fase di review della richiesta. Questa fase prevede generalmente la individuazione di veri e propri revisori che potrebbero essere sviluppatori essi stessi o avere esclusivamente la funzione di verifica e accettazione della novità.

La review è basata quindi su una richiesta avanzata dallo sviluppatore interessato, e autorizzato, e dalla successiva analisi del revisore che una volta formulato eventuali commenti e interagito, se del caso, con lo sviluppatore concede il suo benestare affinché avvenga la fusione.

Le regole di approvazione devono essere chiare in modo da non generare fraintendimenti e rendere la loro gestione lineare e adeguata alla complessità del software nonché della nuova porzione di codice; ci possono essere ad esempio regole di approvazione semplici per modifiche marginali e più complesse per modifiche consistenti. È comunque fondamentale che le regole vengano definite e che venga definito il processo di accettazione altrimenti ogni sviluppatore potrebbe apportare modifiche senza una reale garanzia che queste possano essere utili, funzionanti e in linea con lo sviluppo che si vuole dare al software.

Utile richiamare in questa fase quanto anticipato in premessa, relativamente ad eventuali conflitti tra modifiche eseguite da più sviluppatori sulla stessa parte di codice sorgente. Le regole di accettazione devono prevedere anche la risoluzione di tali situazioni.

Nella fase di accettazione delle modifiche ricade anche la fase di test mediante utenti esterni al sistema o in qualche modo coinvolti nella comunità. Se ne è parlato poco prima accennando alla necessità di avere due livelli di componenti della comunità: gli sviluppatori veri e propri e alcuni esperti di settore che intervengono ad un livello diverso ma con competenze adeguate a giudicare tutti gli aspetti ritenuti significativi del nuovo software. Si tratta in qualche modo di simulare l'utente finale secondo uno schema definito a priori, in fase di progettazione dell'intero sistema di gestione e sviluppo, e volto, tra l'altro, a ridurre al minimo eventuali criticità applicative del prodotto finito.

Non è possibile chiudere questa parte, se non si affronta anche brevemente la modalità di validazione e accettazione del codice originale OpenBPS che costituisce il cuore su cui inizierà ad operare la comunità. Come anticipato poco sopra, nonostante la affidabilità del contesto in cui è stato sviluppato OpenBPS, è necessario procedere ad una verifica degli algoritmi di modellazione energetica contenuti nel codice iniziale al fine anche di permettere alle figure tecniche coinvolte a vario titolo nella comunità di conoscere a fondo l'intero modello.

La validazione, in questo caso, dovrebbe prendere spunto dal processo di validazione delle singole modifiche, come descritto in precedenza. È quindi necessario individuare un gruppo di esperti, selezionato dal Team di Gestione, che analizza in dettaglio il codice sorgente, eventualmente mediante l'aiuto di interfacce ad hoc o di software interprete, e ne acquisisce le modalità di calcolo e modellazione dell'edificio rilasciando opportuni commenti e richieste di chiarimenti da indirizzare agli sviluppatori originali. Tra gli aspetti da verificare in questa fase vi è eventualmente, in base alle decisioni prese su obbiettivi e marcato di riferimento, quello non banale della conformità alla normazione tecnica di settore che viene resa obbligatoria, cioè cogente, dalla legislazione in materia di prestazioni energetiche degli edifici. Il livello di conformità a tale quadro normativo dovrà essere definito dal Team di Gestione, in quanto legato ad una delle decisioni principali più volte sottolineata nel presente rapporto: il contesto di mercato entro cui si andrà a diffondere il software e i relativi utilizzi.

Per questa parte specifica del protocollo si ritiene importante l'eventuale futuro coinvolgimento del Comitato Termotecnico Italiano per il ruolo che oggi ha quadro legislativo, oltre che normativo, della prestazione energetica degli edifici. Quale ente di normazione tecnica di settore il CTI, già oggi, su mandato del legislatore, garantisce la conformità dei software di calcolo commerciali delle prestazioni degli edifici alla normazione tecnica, pertanto è auspicabile che il know- how acquisito e la conoscenza approfondita delle norme non vadano perdute a tutto vantaggio della qualità del risultato finale.

4.3.9 Definizione della modalità di compilazione dell'applicativo

Il codice sorgente rappresenta, sinteticamente, l'insieme degli algoritmi elaborati dalla comunità di sviluppatori scritti nel concordato linguaggio di programmazione. Nel caso di OpenBPS esso traduce in linguaggio informatico le indicazioni, costituite da formule, parametri, dati di input e di output, ecc. fornite dagli esperti di modellazione dell'edificio e il linguaggio di programmazione utilizzato è C#. Il codice è sostanzialmente costituito da un testo contenuto in uno o più file (sorgente) che ne definisce anche il flusso di esecuzione. Il codice sorgente, o meglio il/i file sorgente, però non è direttamente eseguibile da un computer che è invece in grado di eseguire quello che si chiama un codice linguaggio macchina. Senza entrare troppo nel dettaglio e con un certo grado di approssimazione, è utile chiarire che questo secondo linguaggio è utilizzato per i cosiddetti codici oggetto o codici eseguibili. Quindi un codice sorgente deve essere tradotto in un linguaggio direttamente comprensibile dal/dai processore/i, cioè in codice eseguibile. Tale traduzione può avvenire in due modi sostanzialmente differenti: tramite dei software chiamati **compilatori,** che producono il file oggetto (in codice macchina) che combinato successivamente con le necessarie librerie da un altro software, il **linker**, porta alla creazione del file eseguibile, contenente cioè codice eseguibile dell'interro programma; tramite altri software chiamati **interpreti**, che invece non creano alcun file eseguibile ma traducono e eseguono una linea di codice alla volta utilizzando direttamente il codice sorgente. Di conseguenza, al di là delle diverse velocità di esecuzione (molto più veloce il codice compilato di quello interpretato), mentre nel caso di un codice compilato è possibile utilizzare un programma disponendo del

solo codice eseguibile, nel caso di un codice interpretato occorre necessariamente disporre del codice sorgente.

Siccome l'eventuale scelta di un linguaggio e ambiente di sviluppo interpretato, al di là delle considerazioni prestazionali, non avrebbe consentito di mantenere i gradi di libertà che il codice scritto in un linguaggio compilabile consente (distribuzione o del solo sorgente o del solo eseguibile o di entrambi), si è optato, come già indicato per un linguaggio "*compilabile*".

Il tema della "*compilazione*" non è banale in quanto un software compilatore traducendo un codice sorgente in un codice eseguibile utilizza un "*vocabolario*" che può non essere univoco. Il processo di traduzione può essere equiparato per semplicità alla traduzione di un testo da una lingua all'altra ma con la variabile significativa delle sfumature dialettali che un computer, facendo le opportune distinzioni e generalizzazioni, non è sempre in grado di cogliere. Questo si traduce in possibili differenze nei risultati finali del software eseguibile se compilato da compilatori diversi tra loro.

Per questo motivo si è scelto di utilizzare fin dall'inizio un linguaggio di programmazione soggetto a standard internazionale (quindi con vocabolario univoco), quale C#, e come ambiente di sviluppo e compilazione il framework .NET, supportato dalla interfaccia visuale Microsoft Visual Studio, di libero uso nella versione Community, per l'ambiente Window, e dall'interfaccia visuale Mono, anch'essa d'uso gratuito, per gli ambient Linux e IOS.

In questo modo si evita una potenziale ramificazione del protocollo di cui al presente rapporto in relazione alla distribuzione di un codice univoco. Infatti, pur nell'eventualità di avere disponibile il codice sorgente OS, non vi è ambiguità nella compilazione del linguaggio standardizzato, evitando possibili difficoltà di un utilizzo univoco del software, qualora lo si ritenga adeguato a produrre documenti con valore legale (Attestato di Prestazione Energetica) e per i quali sia necessaria una certa univocità di risultati. Resta invece aperta un'altra possibile scelta da effettuare, quella relativa alla modalità di distribuzione e fruizione: se, ad esempio, distribuire l'applicativo compilato e non il codice sorgente per evitare alterazioni non dichiarate dello stesso, oppure ancora se adottare un sistema basato su concetto di Web Service in cui l'applicativo viene eseguito su una sola macchina gestita dal Team di Gestione e agli utenti vengono solamente forniti dei template per la definizione dei dati di input.

4.3.10 Definizione delle modalità di verifica della conformità degli applicativi commerciali al codice sorgente originale

Come ultima parte della sequenza di azioni che costituiscono, o dovrebbero costituire il protocollo di sviluppo di OpenBPS, si ritiene necessario approfondire il tema delle modalità di verifica e validazione della conformità dell'applicativo informatico (codice eseguibile) al sorgente originale.

Si tratta di una fase che potenzialmente può essere superflua o rappresentare un requisito essenziale, anche se in realtà già nelle note di apertura del presente rapporto emerge la necessità di un processo di validazione.

La scelta tra le due opzioni o una soluzione intermedia è conseguenza di valutazioni fatte durante la fase di progettazione dell'intero protocollo. Sia nei paragrafi introduttivi del presente capitolo che nella descrizione delle varie fasi si è accennato di volta in volta a potenziali conseguenze che l'approccio OSS potrebbe avere sulla difficoltà di garantire a fine processo la qualità del dato (output) finale prodotto dal software eseguibile. Cercando di sintetizzare e semplificare, un software OSS porta con sé un elevatissimo grado di libertà nella diffusione di applicativi per i quali non è oggettivamente possibile garantire la conformità al codice sorgente originale, a meno di ricostruire con un lavoro non banale l'intera filiera che ha portato alla singola versione compilata dallo sviluppatore "x".

A questo scenario estremo, allineato ai cosiddetti software liberi, si associa infatti il concetto che ogni sviluppatore, sulla base della licenza OSS associata al sorgente, potrà produrre una nuova versione del codice, anch'essa accompagnata da adeguata, e simile, licenza e un eseguibile potenzialmente diverso. Dall'altro lato, lo scenario opposto è quello di un codice sorgente sviluppato in un contesto completamente privato

anche se da una comunità più o meno complessa, che non viene distribuito, ma compilato in proprio e reso utilizzabile mediante server accessibile agli operatori. Una soluzione intermedia prevede la possibilità di diffondere dei file eseguibili, siano essi l'intero software o singole librerie, firmati digitalmente.

A questo punto è necessario richiamare le funzioni crittografiche hash, descritte nei capitoli introduttivi, in quanto parte integrante del discorso che si sta sviluppando.

Per procedere ulteriormente è però importante sottolineare nuovamente che un software completamente libero non dovrebbe aver bisogno di protezioni. Ognuno è libero di fare quello che ritiene, pur nel rispetto delle licenze associate al sorgente e agli applicativi. Mentre un software che deve garantire all'utente o al sistema certe prestazioni è opportuno che venga in qualche modo "*blindato*".

Le funzioni hash consentono di gestire tutto questo purché sia chiara la modalità con cui vengono create e utilizzate. Una chiave hash, ad esempio, può identificare il codice sorgente nel suo complesso, una singola parte dello stesso, una libreria statica o dinamica. Ma può anche identificare l'input e/o l'output di un software. In questo modo ora potrebbe essere più evidente come sia possibile garantire che il software utilizzato sia conforme a quello che gli sviluppatori hanno prodotto o che un dato di output sia stato elaborato proprio dall'algoritmo contenuto nel codice sorgente, partendo da determinati dati di input.

In conclusione, per garantire la conformità del software o del risultato (output) di un software è opportuno progettare un sistema di crittografia che può interessare il codice sorgente, il codice eseguibile e le librerie, i risultati dell'esecuzione del software assieme o meno ai dati di input.

Una decisione in tal senso deve essere presa a monte dell'intero processo di progettazione del sistema di gestione e sviluppo.

Gli scenari che si delineano per completare il quadro, a questo punto si ritiene possano essere:

- ***sviluppo di un codice sorgente in comunità privata con licenza tale da consentire la distribuzione di un software eseguibile senza sorgente****:*

  si tratta di una configurazione che consentirebbe un elevato grado di sviluppo e di diffusione del software per ricerca e applicazioni a differenti livelli, ma al contempo riduce molto l'affidabilità del processo di verifica e validazione. La disponibilità di codici sorgente liberi, seppure con differenti livelli di licenza, porterebbe la moltiplicazione di possibili copie del sorgente anche se in numero decisamente inferiore a quello che si avrebbe in presenza di un OSS. La certificazione mediante funzioni hash e chiavi crittografate connesse avverrebbe a questo punto piùsul codice stesso o sugli applicativi e librerie che non sui risultati del calcolo. Ciò potrebbe comunque avere un certo valore anche se inferiore rispetto alla configurazione descritta poco più avanti. Infatti, in questo caso andrebbe regolamentata opportunamente la gestione del processo di marcatura rendendola comunque centralizzata a livello di Team di Gestione onde evitare che la funzione hash relativa venga diffusa in modo incontrollato.

- ***sviluppo di un ambiente del tipo "web service****":*

  in questa situazione, il software eseguibile viene lasciato su un server protetto e gli utenti vengono provvisti di codici identificativi per l'accessoall'eseguibile, template per l'ingresso dei dati di input e format dei dati di output. Il software è compilato sulla base del codice sorgente presente nel repository principale e viene aggiornato sulla base della programmazione dei lavori di integrazione stabiliti dal Team di Gestione. Codice sorgente, eseguibile, dati di input e dati di output vengono marcati con chiave SHA-256 in modo che siano univocamente identificabili. Si tratta della configurazione più chiusa e affidabile dal punto di vista della validità del dato finale che, a questo punto, potrebbe essere utilizzato anche a fini legali (ad esempio APE o calcolo di progetto). Il mercato

non dispone dei codici sorgente e il motore di calcolo gira su una sola macchina dove è caricato solo dopo aver superato i testi di approvazione da parte della comunità di sviluppo.

4.4 Considerazioni conclusive

Vista la complessità delle varie tematiche approfondite nei paragrafi precedenti, di seguito si riporta una sintesi delle principali considerazioni emerse.

4.4.1 Flusso di azioni

Il flusso di azioni principali da intraprendere per la gestione di OpenBPS in una comunità di sviluppo è costituito da:

11. Definizione di un Team di Gestione dell'intero progetto
12. Individuazione dell'oggetto su cui lavorare e del suo mercato di riferimento
13. Individuazione della licenza o delle licenze associabili al progetto nel suo complesso e/o a parti dello stesso
14. Scelta della piattaforma di gestione del progetto
15. Individuazione delle caratteristiche di visibilità del repository (pubblico o privato)
16. Creazione del repository, individuazione di un metodo di numerazione delle versioni successive, definizione delle regole con cui deve essere gestita la documentazione che deve accompagnare ogni singola parte del software sia per gli sviluppatori che per gli utenti
17. Assegnazione dei ruoli dei singoli attori e delle regole di ingaggio
18. Definizione delle modalità di verifica e accettazione delle modifiche ed integrazioni al codice sorgente
19. Definizione della modalità di compilazione dell'applicativo
20. Definizione delle modalità di verifica della conformità degli applicativi commerciali al codice sorgente originale

Alcune di queste azioni, soprattutto le prime, richiedono decisioni puntuali che influenzano scelte o azioni successive in quanto ne determinano i gradi di libertà e conseguentemente influenzano le possibili utilizzazioni del software da parte del mercato.

La scelta principale, che di fatto instrada l'intero percorso di sviluppo del software e del protocollo per la sua gestione, è legata **all'individuazione del mercato di riferimento**. Gli elementi di scelta sono:

d. software per uso libero da parte del mercato per simulazioni in ambito di ricerca, universitario, sperimentazione,
e. software destinato a produrre output avente un ruolo legale, come ad esempio un calcolo di progettazione o un attestato di prestazione energetica,
f. una soluzione intermedia alle due.

La scelta tra a, b e c determina in cascata:

- la licenza associabile all'utilizzo finale del software
- la piattaforma di riferimento per la gestione del processo di sviluppo del software e le modalità di gestione della stessa
- la selezione degli sviluppatori e di coloro che aiuteranno a testare il software o sue parti

La soluzione scelta influenza anche la futura scelta del livello di sicurezza e affidabilità della conformità dell'eseguibile o dei calcoli derivanti dalla sua applicazione al codice sorgente base.

4.4.2 Figure coinvolte

Le principali figure e/o ruoli coinvolti, indipendentemente dal percorso scelto sono:

- Team di Gestione con ruoli manageriali sull'intero processo composto da un rappresentante ENEA e del Politecnico di Milano, possibilmente da un rappresentante del CTI e da figure tecniche quali: un progettista software, un legale, uno o più esperti settoriali di modellazione energetica del sistema edificio impianto.
- Comunità di sviluppo composta da:
    - Amministratore della comunità
    - Amministratore informatico della comunità
    - Esperti di modellazione dei singoli componenti del sistema edificio-impianto (indicativamente: involucro, climatizzazione invernale, climatizzazione estiva, automazione e controllo)
    - Sviluppatori
    - Esperti di settore per le fasi di test

4.4.3 Scenari possibili

Oltre l'approccio totalmente Open Source (sviluppo di un codice sorgente in comunità pubblica con licenza Open Source), i due scenari alternativi che si ritiene essere più sostenibili sono:

- sviluppo di un codice sorgente in comunità privata con licenza tale da consentire la distribuzione di un software eseguibile senza aver associato il sorgente. Validazione mediante chiave hash (SHA-256) sul software.
- sviluppo di un codice sorgente in comunità privata ma con licenza tale da consentirne l'utilizzo solo in un ambiente del tipo "web service". Validazione mediante chiave hash (SHA- 256) sull'input e output rilasciato all'utente remoto.

4.4.4 Azioni da intraprendere

Al termine del progetto occorrerà coordinare con le parti potenzialmente interessate allo sviluppo di OpenBPS e alla sua possibile applicazione a contesti anche regolamentati per legge, cioè gli sviluppatori da un lato (Politecnico di Milano), i finanziatori dall'altro (ENEA e MISE), eventuali associazioni tecnico scientifiche, come ad esempio AiCARR, IBPS-Italy, ecc., soggetti normatori come ad esempio il Comitato Termotecnico, una possibile linea di ulteriore sviluppo tramite l'identificazione del mercato di riferimento e quindi l'attuazione delle decisione che ne conseguono in cascata, come evidenziato nei precedenti paragrafi.

In assenza di tale identificazione e determinazione, lo sviluppo del codice OpenBPS e della relativa Community di sviluppo si muoverà nell'ottica puramente open source software per uso libero da parte del mercato per simulazioni in ambito di ricerca, universitario, sperimentazione.

5 Sito Web

5.1 Implementazione sito web

Il sito web OpenBPS è stato implementato su sistema operativo open source, Linux, tramite un motore CMS (Content Management System) open source, Drupal [27]. In realtà, Drupal funziona su diversi sistemi operativi, tra cui Windows, Mac OS X, Linux e qualsiasi piattaforma software che supporti i web server Apache (versione 1.3 o superiore) o IIS (versione 5 o superiore) e il linguaggio PHP (versione 4.3.3 o superiore) e utilizza un database per memorizzare i contenuti, come MySQL, anch'esso open source e disponibile per i diversi sistemi operativi.

Il sito, sviluppato e implementato in lingua italiana, è disponibile anche in lingua inglese, per costituire il punto di riferimento per una comunità di sviluppatori e utenti eventualmente anche su base internazionale. IL sito ha, oltre la "landing page" (la pagina di presentazione e accesso) **OpenBPS**, altre pagini principali, quali:

- **Licenza d'uso**,
  che riporta le condizioni di licenza d'uso delle librerie basata sulla European Free/Open Source Software (F/OSS) licence (EUPL v.1.1), sviluppata dall'Unione Europea per il software prodotto dalle amministrazioni pubbliche dell'Unione;
- **Caratteristiche**
  che riporta le principali caratteristiche funzionali del software;
- **Comunità**,
  che riporta lo scopo e le modalità di accesso alla comunità di sviluppatori;
- **Documentazione**,
  che riporta la documentazione, sia come pagine html, sia come link a file pdf;
- **Download**
  che riporta i link alle librerie del software;
- **Collaborazioni**
  che riporta i loghi e i link ai siti dei partner del progetto;
- **Contattaci**
  che riporta un modulo per inviare ogni tipo di richiesta al gestore del sito;
- **News,**
  che è la pagina principale di pubblicazione delle novità, organizzate per datazione;
- **FAQ**
  che è pagina principale di pubblicazione delle domande ricorrenti, organizzate per datazione.

Nelle seguenti figure sono riportati rispettivamente la parte superiore della pagina d'accesso, il banner inferiore della stessa, una schermata parziale della pagina che riporta le condizioni di licenza d'uso, una schermata parziale della pagina che riporta le caratteristiche di OpenBPS, una schermata della pagina che riporta le modalità di partecipazione alla comunità, della documentazione, del download, delle collaborazioni, dei contatti, della pubblicazione delle news e delle FAQ (domande più frequenti).

Un backup del sito e del suo data base è disponibile su DVD.

5.2 Pagine del sito web

Figura 31 – Schermata della pagina principale del sito OpenBPS

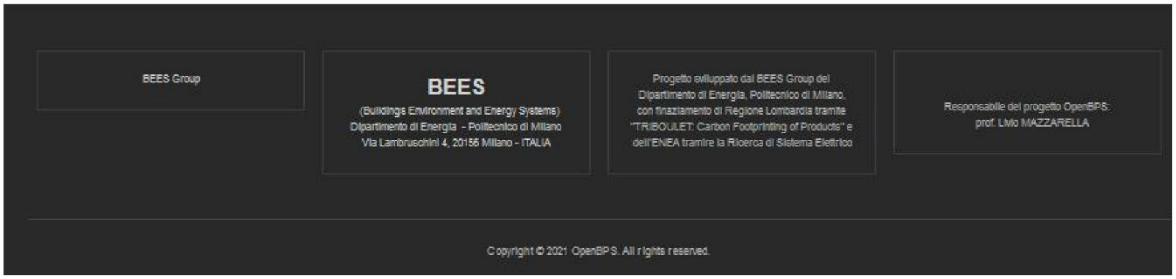


Figura 32 – Schermata banner inferiore del sito OpenBPS

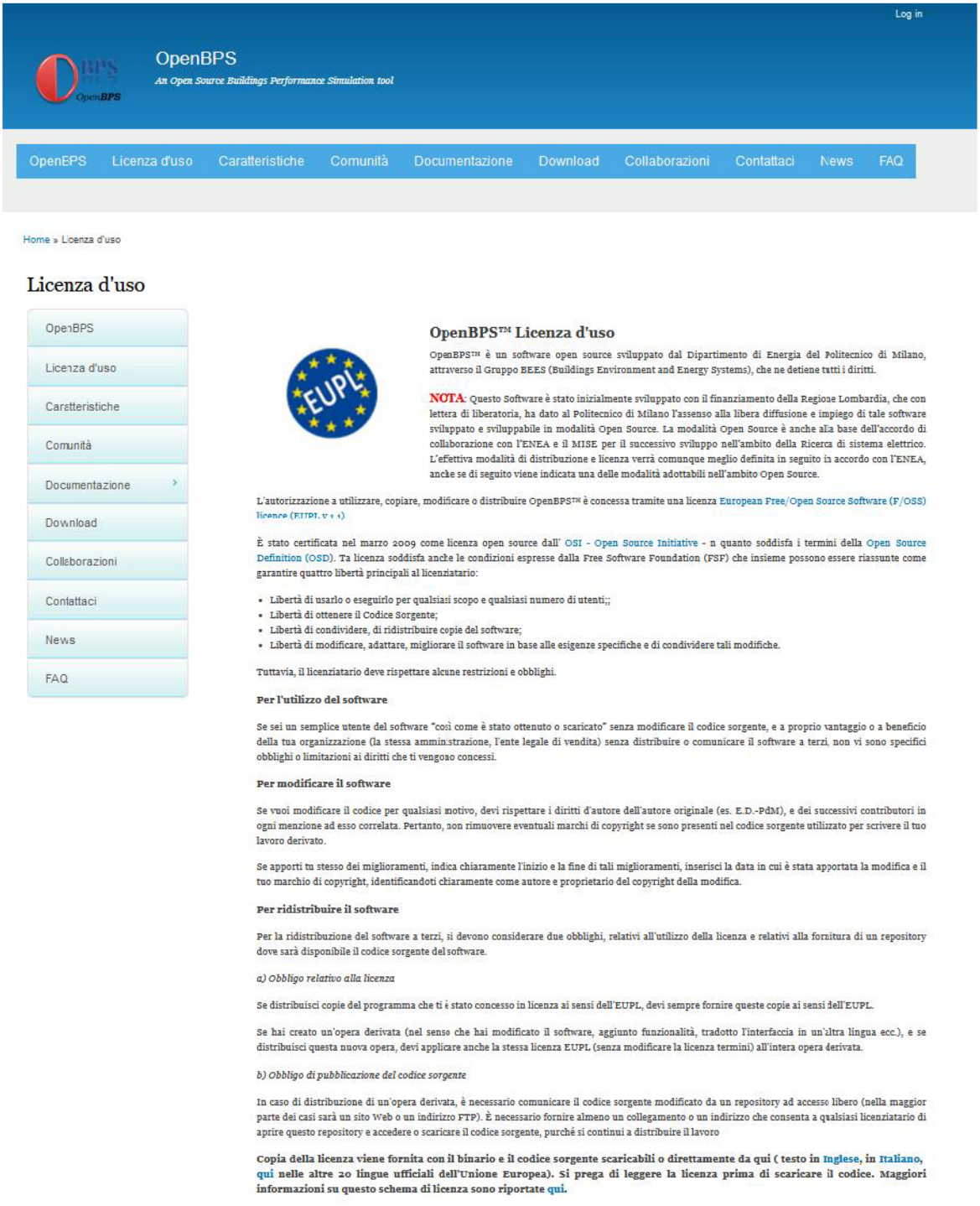
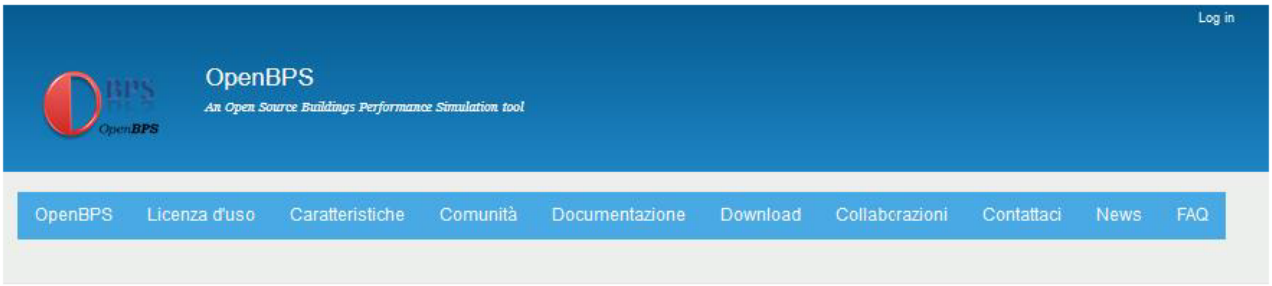


Figura 33 – Schermata parziale della pagina che riporta le condizioni di licenza d'uso



Home » Caratteristiche

Caratteristiche

- OpenBPS
- Licenza d'uso
- Caratteristiche
- Comunità
- Documentazione
- Download
- Collaborazioni
- Contattaci
- News
- FAQ

1. Introduzione

Negli ultimi anni presso il Politecnico di Milano (Politecnico di Milano), un gruppo di ricerca ha lavorato allo sviluppo di uno strumento di simulazione delle prestazioni energetiche degli edifici di "prossima generazione". Nel 2009 un progetto di ricerca preliminare, svolto attraverso un dottorato di ricerca (Pasini, 2009), ha tracciato il percorso verso la concettualizzazione e lo sviluppo di un modello orientato agli oggetti per la simulazione del sistema edilizio (Mazzarella et al., 2009). L'idea principale era combinare la potenza dei linguaggi di programmazione orientati agli oggetti oggi disponibili con la natura orientata agli oggetti d: un edificio. Nello sviluppo dello strumento è stata posta grande cura nella sua progettazione per garantire modularità e manutenibilità, attraverso un approccio di sviluppo open source (OS). La metodologia di sviluppo è stata essa stessa parte del lavoro di sviluppo, volto a creare un framework comune per una comunità di sviluppatori in grado di gestire l'intero ciclo di vita dello sviluppo del software (Mazzarella et al., 2015a). La validazione del codice è stata quindi un secondo punto chiave nello sviluppo dello strumento: sono stati utilizzati test sia analitici che comparativi per valutare la qualità degli algoritmi implementati. I primi risultati di una validazione comparativa effettuata su tale strumento, secondo lo standard BESTEST, seno stati presentati alla Conferenza Internazionale IBPSA 2013 (Mazzarella et al., 2013). Alcuni altri test comparativi tra diverse soluzioni numeriche dell'equazione differenziale di conduzione del calore sono stati presentati alla 6th International Building Physics Conference, IBPC 2015 (Mazzarella et al., 2015b). Tale lavoro di sviluppo è stato in parte finanziato da Regione Lombardia ed è stato documentato in relazioni tecniche, che saranno rese disponibili con lo strumento in futuro. Lo sviluppo del codice di calcolo, con l'implementazione una parte dell'impiantistica, è ripreso nel 2019 tramite un finanziamento dell'ENEA nell'ambito della Ricerca di Sistema terminata nel dicembre 2021 e la decisione finale su come distribuirlo e sostenerlo è ancora in sospeso. Il nome dello strumento è stato invece scelto come OpenBPS (Open Building Performance Simulator). Questo documento descrive la struttura, le caratteristiche e le capacità di OpenBPS.

2. Cosa è OpenBPS

OpenBPS è un nuovo programma di simulazione delle prestazioni degli edifici progettato principalmente come libreria software open source multipiattaforma (Windows, Mac e Linux). È un motore di simulazione e non esiste un'interfaccia utente formale. A scopo di sviluppo e test, viene fornita una semplice GUI, che è in grado di importare la geometria dell'edificio fornita dal plug-in OpenStudio per SketchUp o di importare direttamente progetti definiti tramite file di input EnergyPlus (.idf), anche se non ancora completamente implementati. È codificato da zero con un linguaggio di programmazione orientato agli oggetti multipiattaforma, C#, che è un linguaggio open source per .NET Framework basato sugli standard ECMA. Questo linguaggio ha consentito facilmente alla parallelizzazione del codice nativo di sfruttare i processori multi-thread/multi-core oggi disponibili.

2.1 Codice orientato agli oggetti

Uno degli obiettivi principali di OpenBPS è creare una struttura modulare avanzata che faciliti l'aggiunta di nuove funzionalità e consenta l'utilizzo della libreria da parte di qualsiasi programma di hosting. Un linguaggio di programmazione orientato agli oggetti, come C#, è stato selezionato per raggiungere questo obiettivo perché:

- è una ricca implementazione del paradigma orientato agli oggetti, che include l'incapsulamento, l'ereditarietà, il polimorfismo e il metodo overriding;
- è un linguaggio orientato agli oggetti facile ed efficiente: gli sviluppatori possono tradurre le loro idee/algoritmi per risolvere problemi complessi più facilmente rispetto al C++;
- è uno dei principali linguaggi che funziona in multipiattaforma utilizzando il framework .NET: è disponibile nei sistemi operativi Windows, Linux e MacOS;
- è più sicuro dai tipi rispetto a C++; le uniche conversioni implicite di default sono quelle considerate sicure, come l'allargamento di interi; la memoria gestita non può essere liberata in modo esplicito; invece, viene automaticamente raccolta i rifiuti;
- può produrre applicazioni che funzionano alla stessa velocità delle applicazioni C++, utilizzando il compilatore Just In Time (JIT), che può ottimizzare l'ottimizzazione del codice sull'hardware della macchina in esecuzione;
- è oggi un linguaggio di programmazione standard e open source (ECMA-334 e ISO/IEC 23270:2006).

Figura 34 – Schermata parziale della pagina che riporta le caratteristiche di OpenBPS

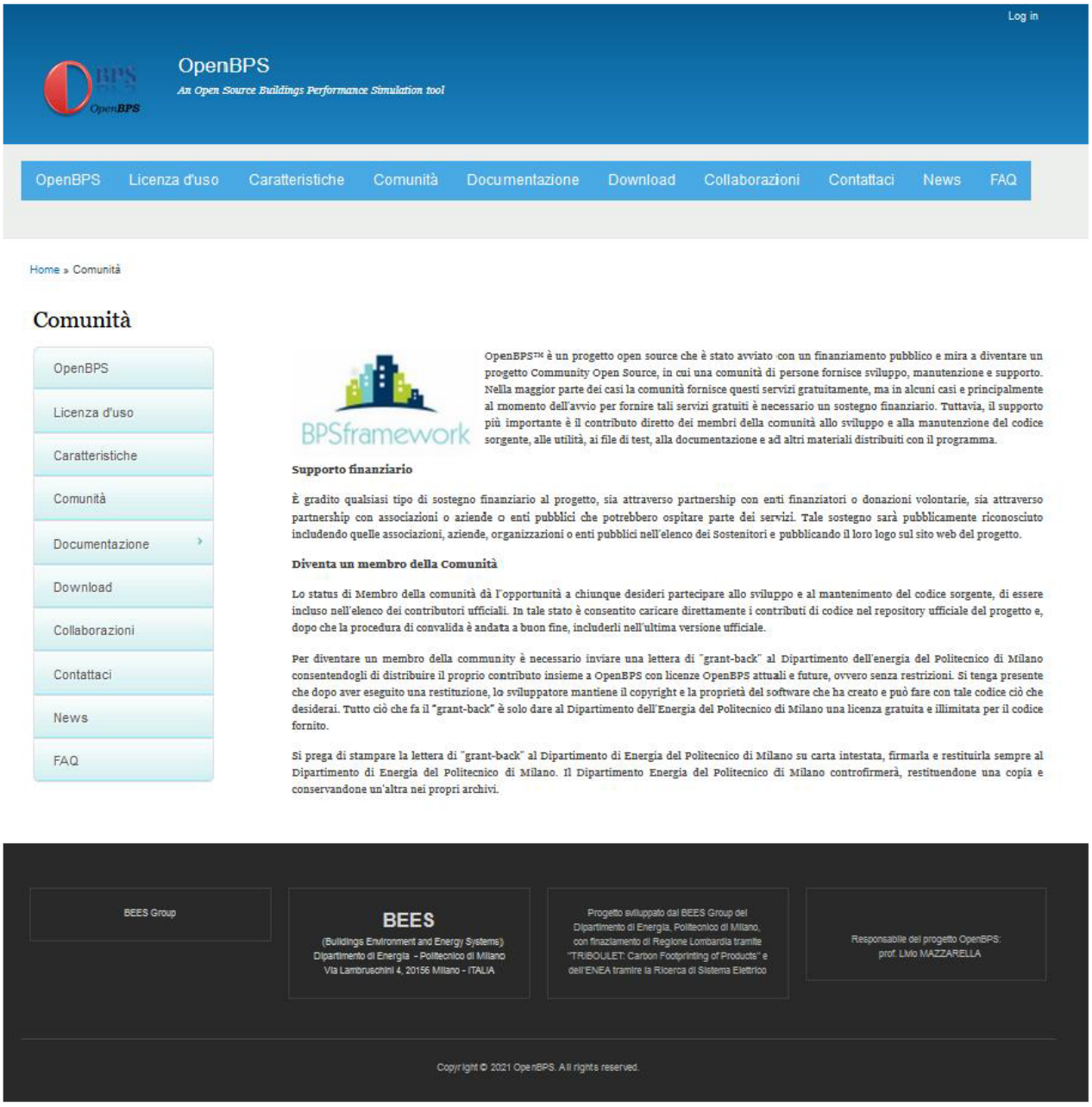


Figura 35 – Schermata della pagina che riporta le modalità di partecipazione alla comunità

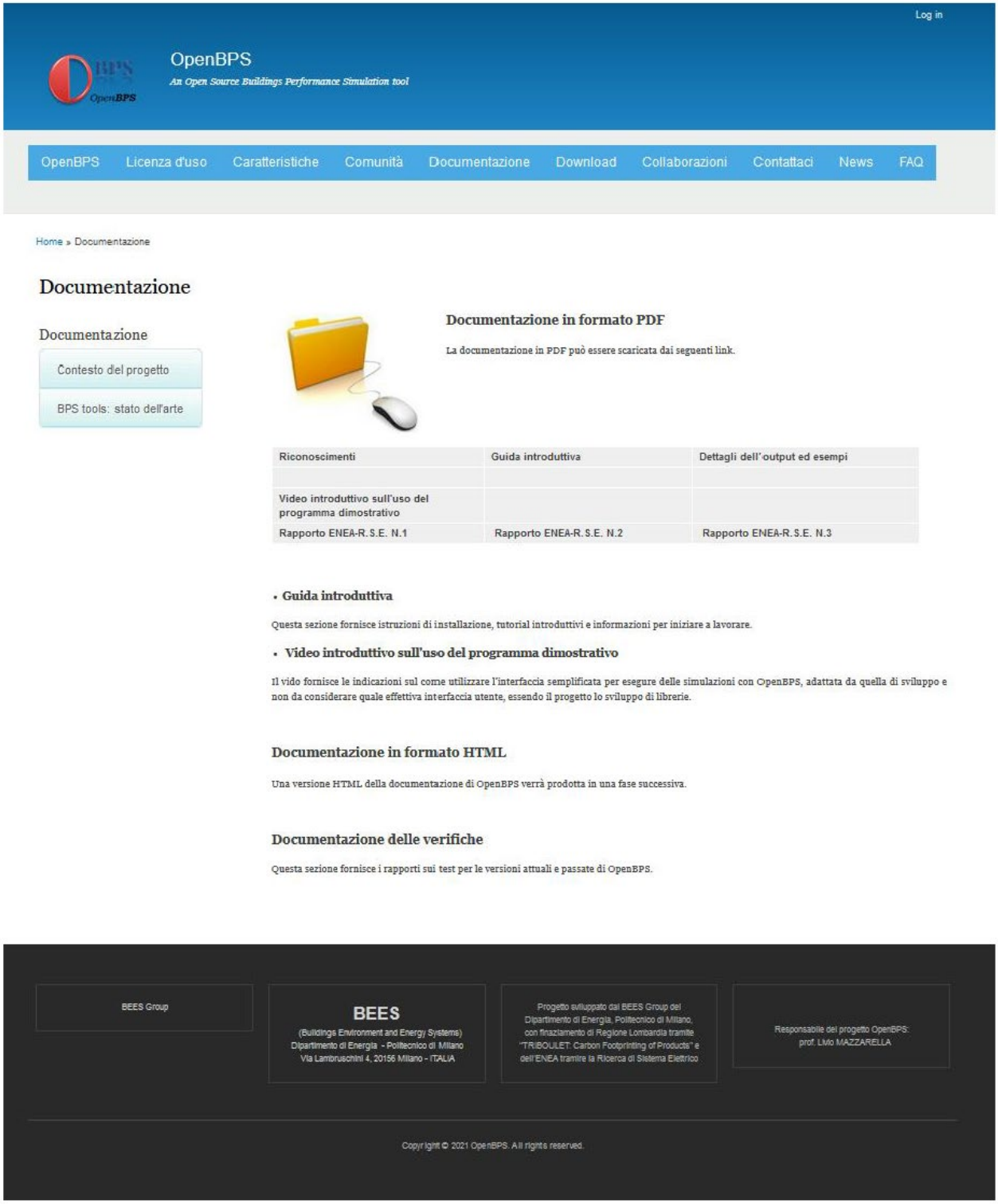


Figura 36 – Schermata della pagina che riporta la documentazione

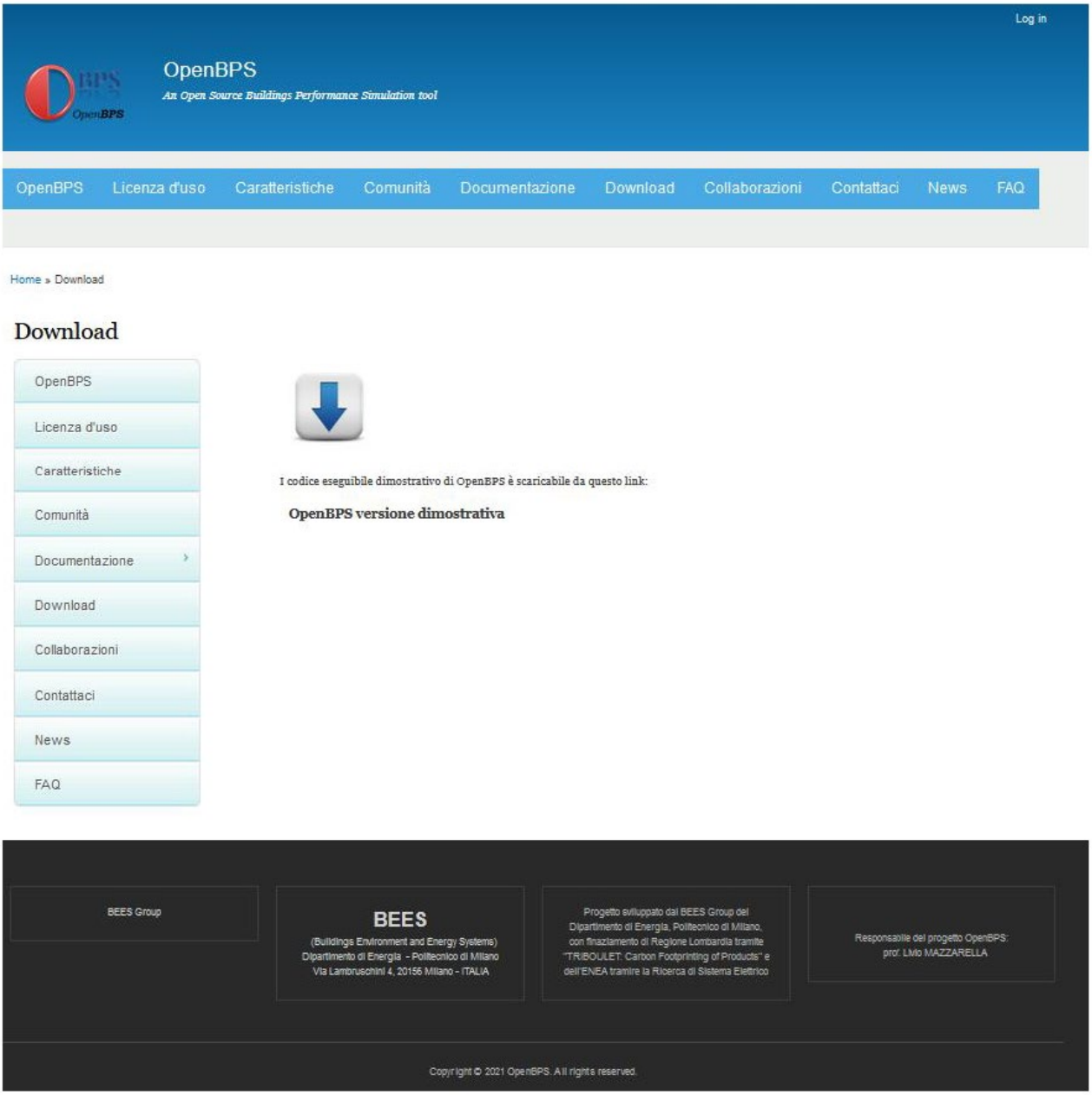


Figura 37 – Schermata della pagina del download del programma dimostrativo

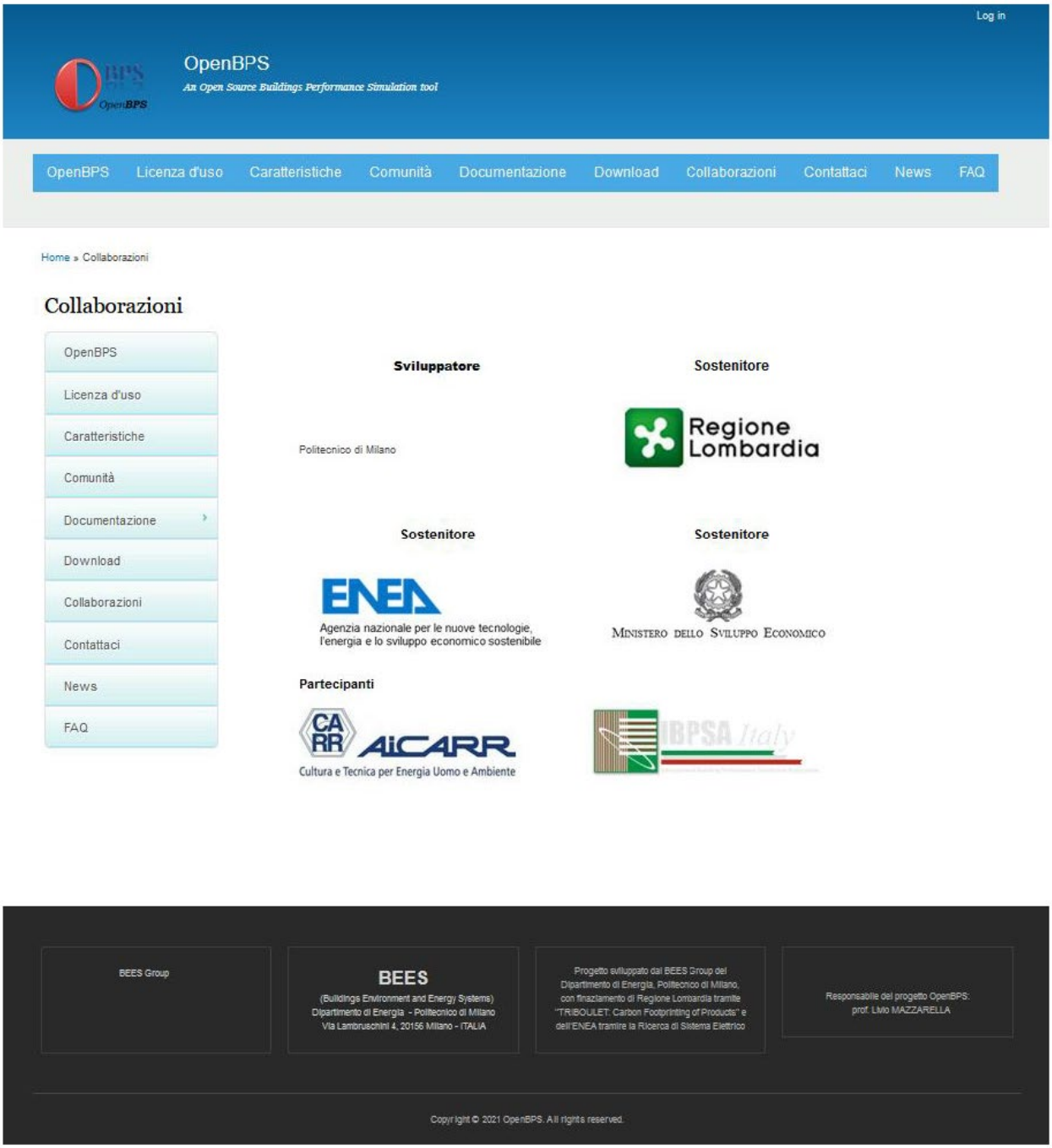


Figura 38 – Schermata della pagina dei finanziatori e collaboratori

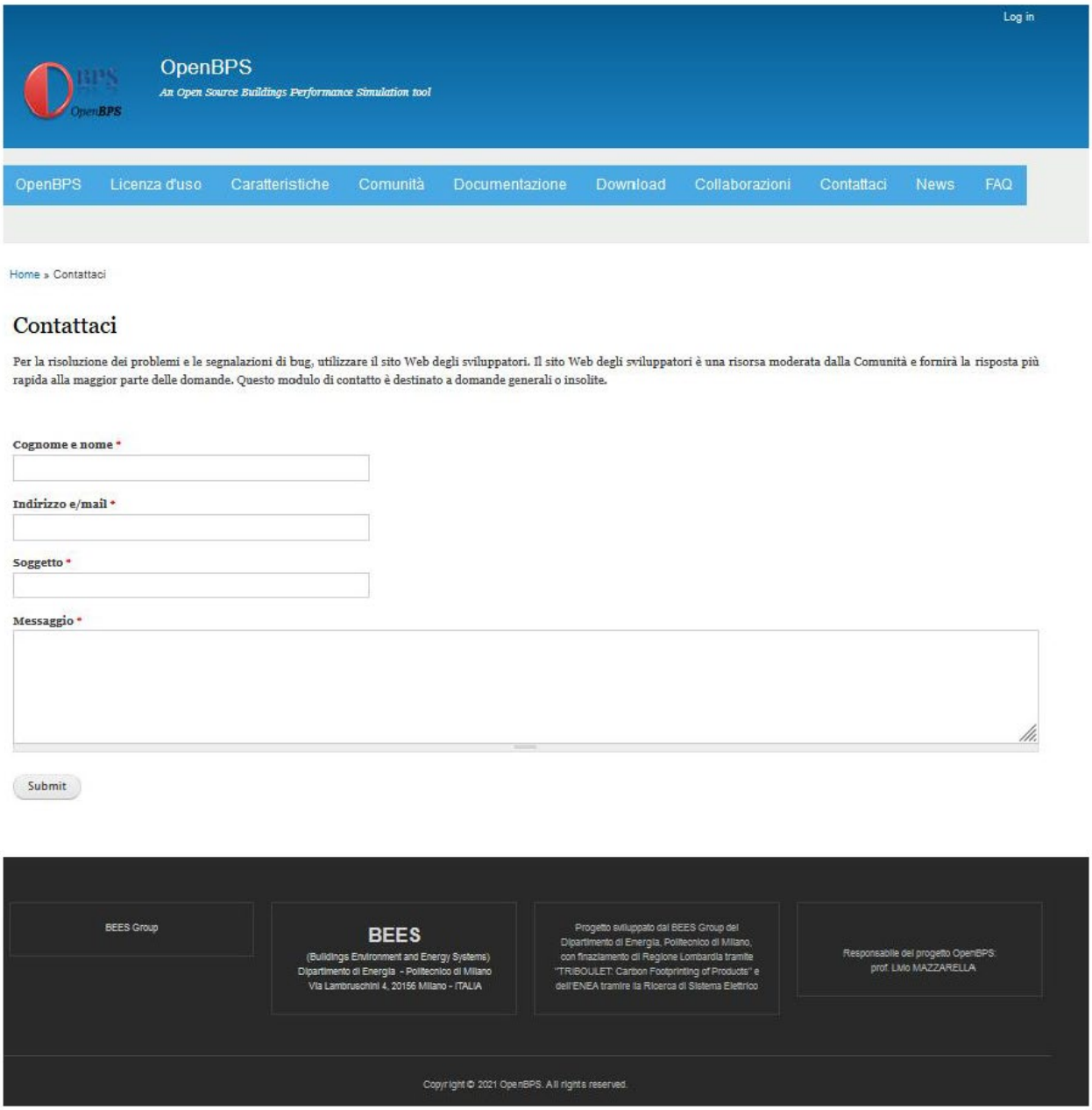


Figura 39 – Schermata della pagina per i contatti con gli sviluppatori

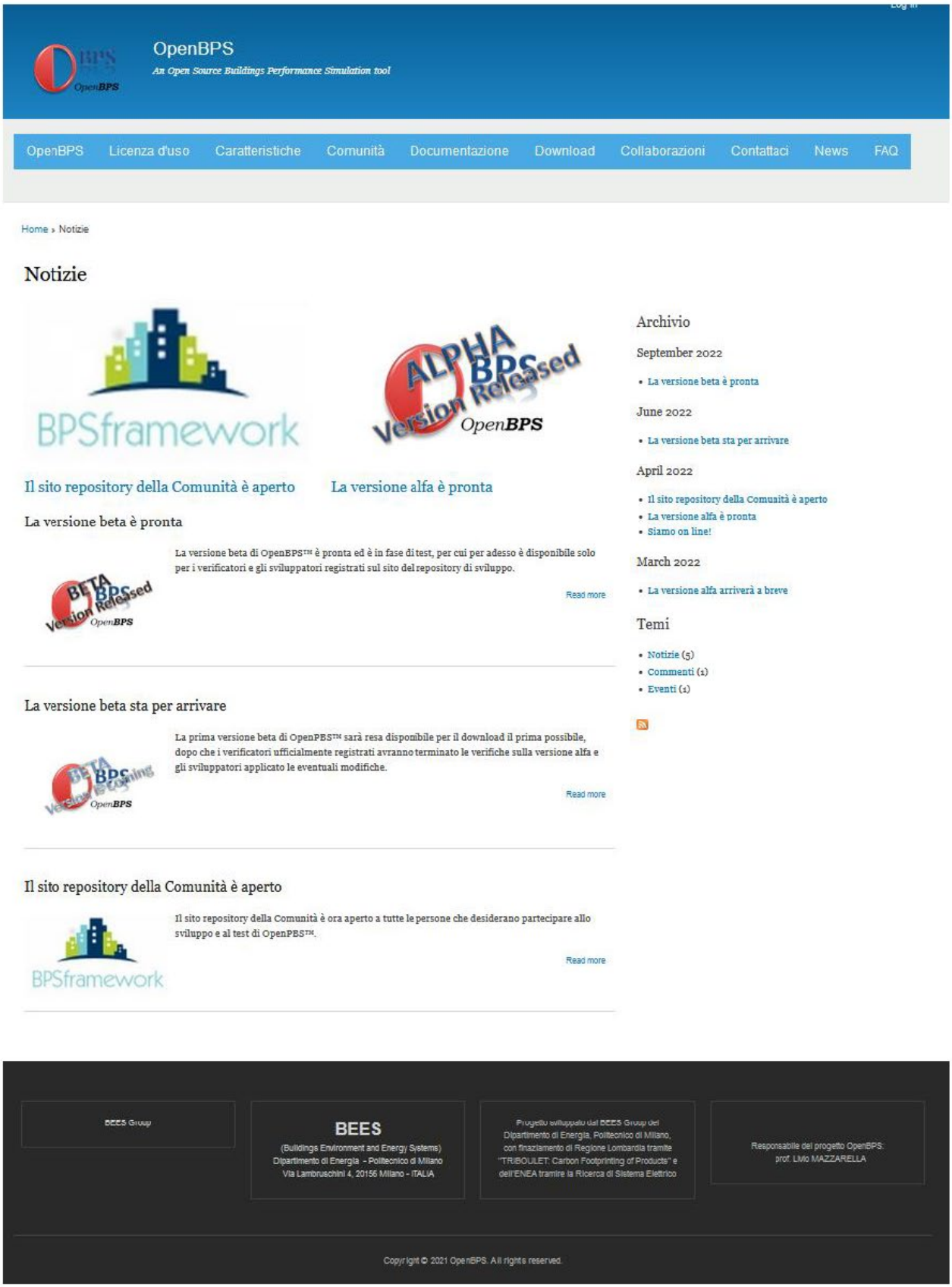


Figura 40 – Schermata della pagina principale delle notizie

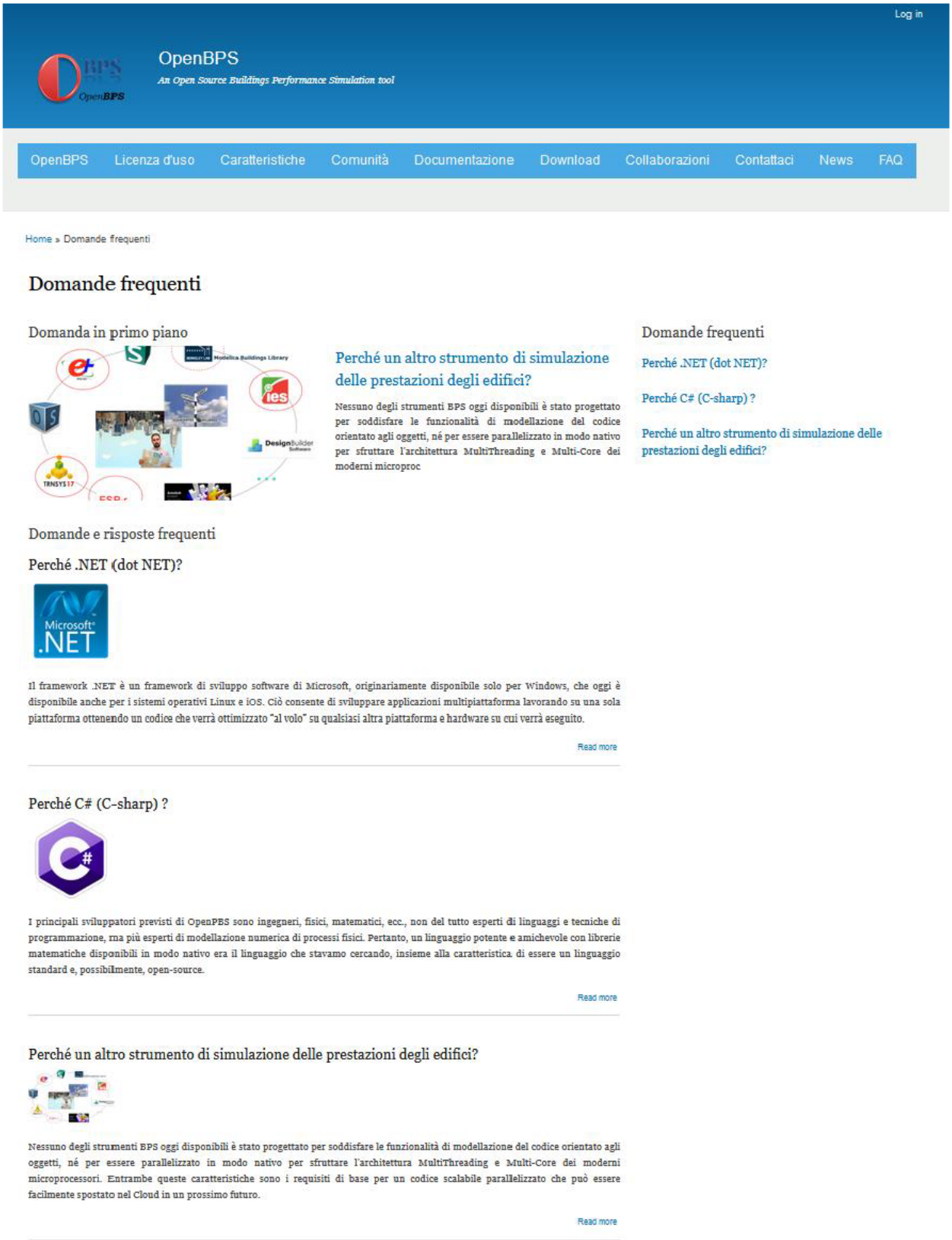


Figura 41 – Schermata della pagina principale delle FAQ

Inoltre il sito è popolato da pagine secondarie che possono crescere in numero in funzione delle esigenze di pubblicazione. In particolare la pagina principale della documentazione dispone di un menù secondario (vedasi Figura 36) che richiama pagine di documentazione del progetto, parzialmente di seguito riportate, quali la pagina che riporta il contesto in cui si è sviluppato il progetto (Figura 42) e la pagina che riporta lo stato dell'arte sugli strumenti di simulazione delle prestazioni degli edifici (BPSt) (Figura 43).

Infine in Figura 44 è riportata la pagina che espone l'archivio storico delle notizie, ordinato per data decrescente (dalla più recente al più vecchia) e in Figura 45 una pagina d'esempio di risposta a domande frequenti.

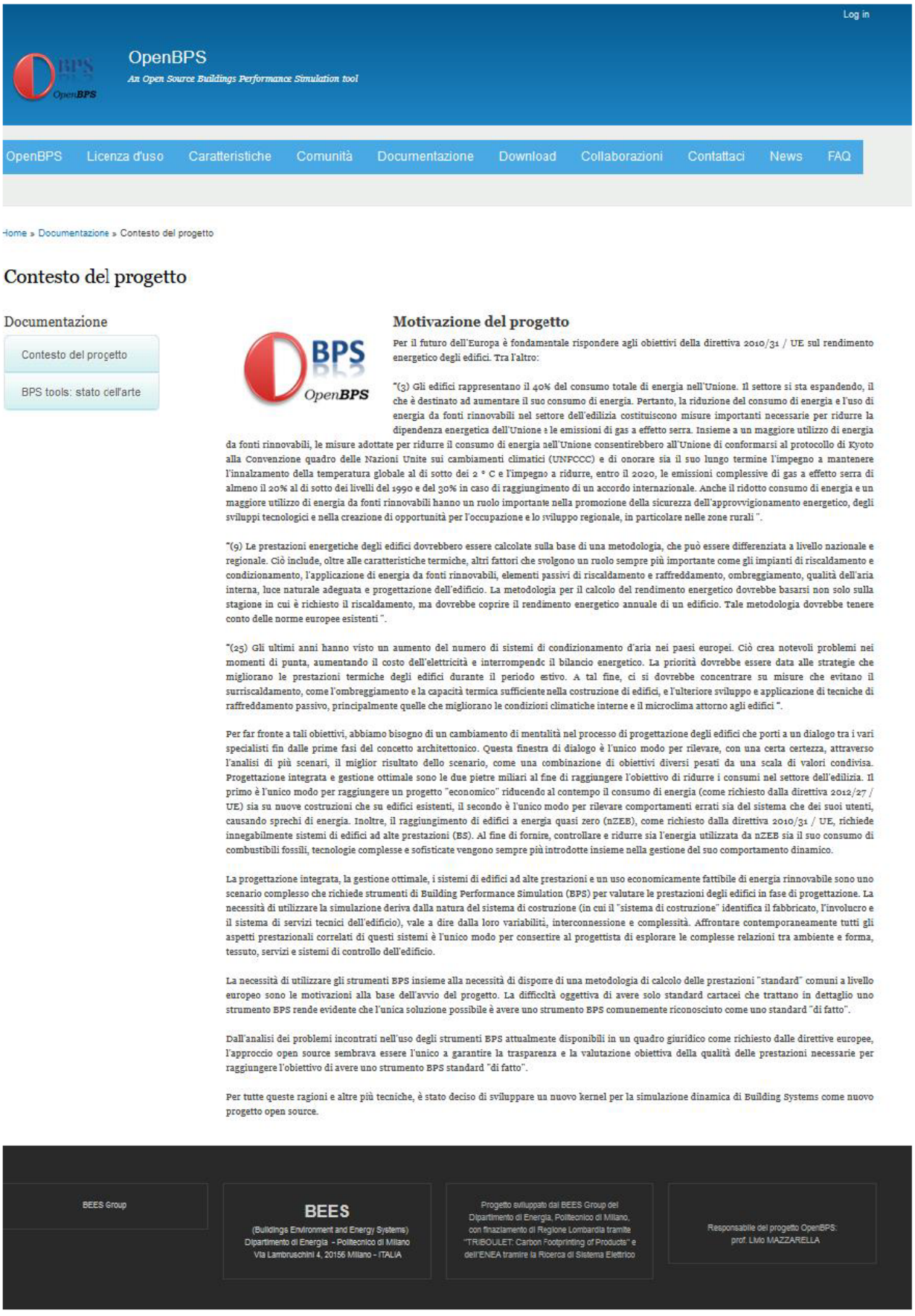


Figura 42 – Schermata della pagina che riporta il contesto in cui si è sviluppato il progetto

Log in

OpenBPS

An Open Source Buildings Performance Simulation tool

OpenBPS | Licenza d'uso | Caratteristiche | Comunità | Documentazione | Download | Collaborazioni | Contattaci | News | FAQ

Home » Documentazione » Strumenti BPS: stato dell'arte

Strumenti BPS: stato dell'arte

Documentazione

Contesto del progetto

BPS tools: stato dell'arte

Contenuti

SOMMARIO

Negli ultimi 30 anni sono stati sviluppati numerosi Building Simulation Codes (BSC). Tuttavia nessuno di essi è ancora diventato uno "standard", né è stato adottato un linguaggio standard per la descrizione dei servizi o dei componenti di tali strumenti. Focalizzando l'attenzione sull'utilizzo della Programmazione Orientata agli Oggetti avanzata, viene qui effettuata una rassegna dei BSC più utilizzati. Un primo obbiettivo di questo scritto è quello di esaminare principalmente le differenze di base tra questi strumenti e di identificare la ragione di questa mancanza di uniformità, come ad esempio, diversi obiettivi affrontati, problemi nell'implementazione pratica o carenza di basi teoriche. Infatti, le differenze rilevate vanno dai modelli fisici implementati e dai metodi utilizzati per il loro accoppiamento e risoluzione, alla struttura interna del software e alle interfacce grafiche utente (GUI). Questa disuniformità è stata favorita anche dall'avvento dei Linguaggi di Modellazione Object-Oriented, dalla diffusione di strumenti come Dymola e Simulink e dagli sviluppi raggiunti nel campo del Data Base, dei Web Services, del soft computing e dell'analisi numerica. Nel frattempo, il lavoro in corso nel mondo del Computer Aided Design (CAD), che mira a standardizzare lo scambio di dati e la formalizzazione delle conoscenze, è un ulteriore elemento da prendere in considerazione per trarre alcune riflessioni sulle evoluzioni dei BSC. Queste riflessioni mirano a capire fino a che punto simili formalizzazioni, se introdotte nell'ambito sella simulazione, possano consentire: un calcolo distribuito più semplice, responsabilità distribuite nello sviluppo di parti di codice, un'evoluzione esponenziale delle routine algebriche - grazie a una più facile estensibilità e manutenzione del codice - e una maggiore produttività e uso diffuso di questi strumenti, grazie alla progettazione e all'automazione dei processi.

Introduzione (top)

Poiché la progettazione di edifici ad alta efficienza energetica è oggi obbligatoria, è imperativo condurre, durante la fase di progettazione, una stima accurata del fabbisogno energetico dell'edificio. Tuttavia, il processo di progettazione e il suo risultato devono affrontare problematiche spesso tra loro antitetiche, come: salute e comfort; energia incorporata e operativa; costi di costruzione e progettazione, ecc..

Per essere in grado di gestire in qualche modo tali obiettivi discordanti, è importante sviluppare strumenti, e più precisamente strumenti di simulazione, che abbiano lo scopo di "informare" il processo di progettazione. La necessità di utilizzare la simulazione deriva dalla natura del sistema edificio (dove l'edificio identifica l'insieme del fabbricato e dei suoi sistemi tecnici), ovvero dalla loro variabilità, interconnessione e complessità. Affrontare contemporaneamente tutti gli aspetti prestazionali correlati di questi sisterni è l'unico modo per consentire al progettista di esplorare le complesse relazioni tra l'ambiente e la forma, il fabbricato, i servizi e i sistemi di gestione e controllo dell'edificio. Catturare questa complessità in un programma di simulazione richiede un insieme di modelli matematici coerenti e integrati adeguati per descrivere compitamente il problema.

Simulazione integrata (top)

Anche se la simulazione integrata non è un concetto nuovo (Clarke et al., 1998; Clarke, 2001b; Citherlet,2001; Kelly e Strachan, 2001), oggi si fa avanti una nuova consapevolezza dell'importanza della simulazione integrata (Trcka et al., 2007) rispetto all'impiego di strumenti totalmente separati e non interagenti per trattare i diversi aspetti della prestazione degli edifici (termica, acustica, illuminotecnica, ecc.).

Clarke (Clarke & Tang, 2004) riporta alcuni esempi di prestazioni dell'edificio in cui diversi domini interagiscono richiedendo l'uso di modelli multi-dominio: i processi di scambio termico nell'edificio e la distribuzione dell'illuminamento naturale (relativi al compito visivo, al guadagno invernale e al surriscaldamento estivo); i processi di scambio termico tra edificio, impianto e flusso d'aria interzona o movimento dell'aria all'interno della zona; flussi di termici nelle strutture e umidità di costruzione, ecc.

Sfortunatamente, gli strumenti di simulazione delle prestazioni degli edifici (BPS) disponibili non sono ugualmente adatti per emulare tutti gli aspetti rilevanti del comportamento dei sistemi edilizi. Alcuni strumenti sono più addati per la simulazione dell'involucro edilizio (es. EnergyPlus [1], ESP-r [2]), altri per quella dei sistemi tecnici (es. OpenModelica[3], TRNSYS[4]), ecc.

Nei prossimi paragrafi, faremo qualche considerazione sulla diffusione degli strumenti BPS e sul conseguente tasso di adozione (par. Tasso di adozione), esamineremo le differenze tra alcuni degli strumenti disponibili e cercheremo di capire il motivo di questa disuniformità (par. Confronto degli strumenti), analizzeremo il paradigma di interoperabilità (par. Il paradigma dell'interoperabilità), prenderemo in considerazione il processo di sviluppo del software (par. Processo di sviluppo del software) e la modularità del software (par. Modularità del software), e cercheremo di comprendere alcune delle caratteristiche chiave che gli strumenti BPS dovrebbero implementare per progredire nella curva del tasso di adozione (par. Caratteristiche chiave).

Tasso di adozione (top)

Gli utenti del settore di solito adottano una nuova tecnologia solo se è disponibile una suite completamente sviluppata di applicazioni software di alta qualità. La dispersione delle forze nello sviluppo di strumenti BPS, anche se si è mirato a traguardi importanti, non ha portato ancora al raggiungimento di una loro grande diffusione. Inoltre si ha anche un basso tasso di sviluppo in quanto gli sviluppatori di software non investiranno grandi risorse fino a quando non avranno una certa fiducia che il mercato necessita effettivamente di tali strumenti. Questo crea il classico problema dell'uovo e della gallina (Froese, 2002).

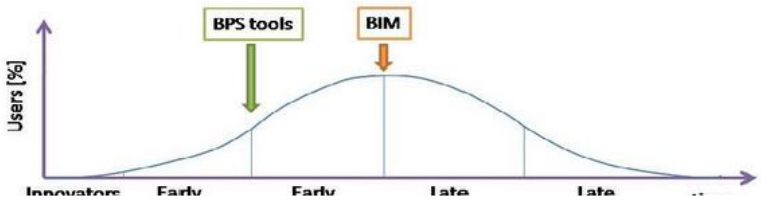


Figura 43 – Schermata della pagina che riporta il contesto in cui si è sviluppato il progetto

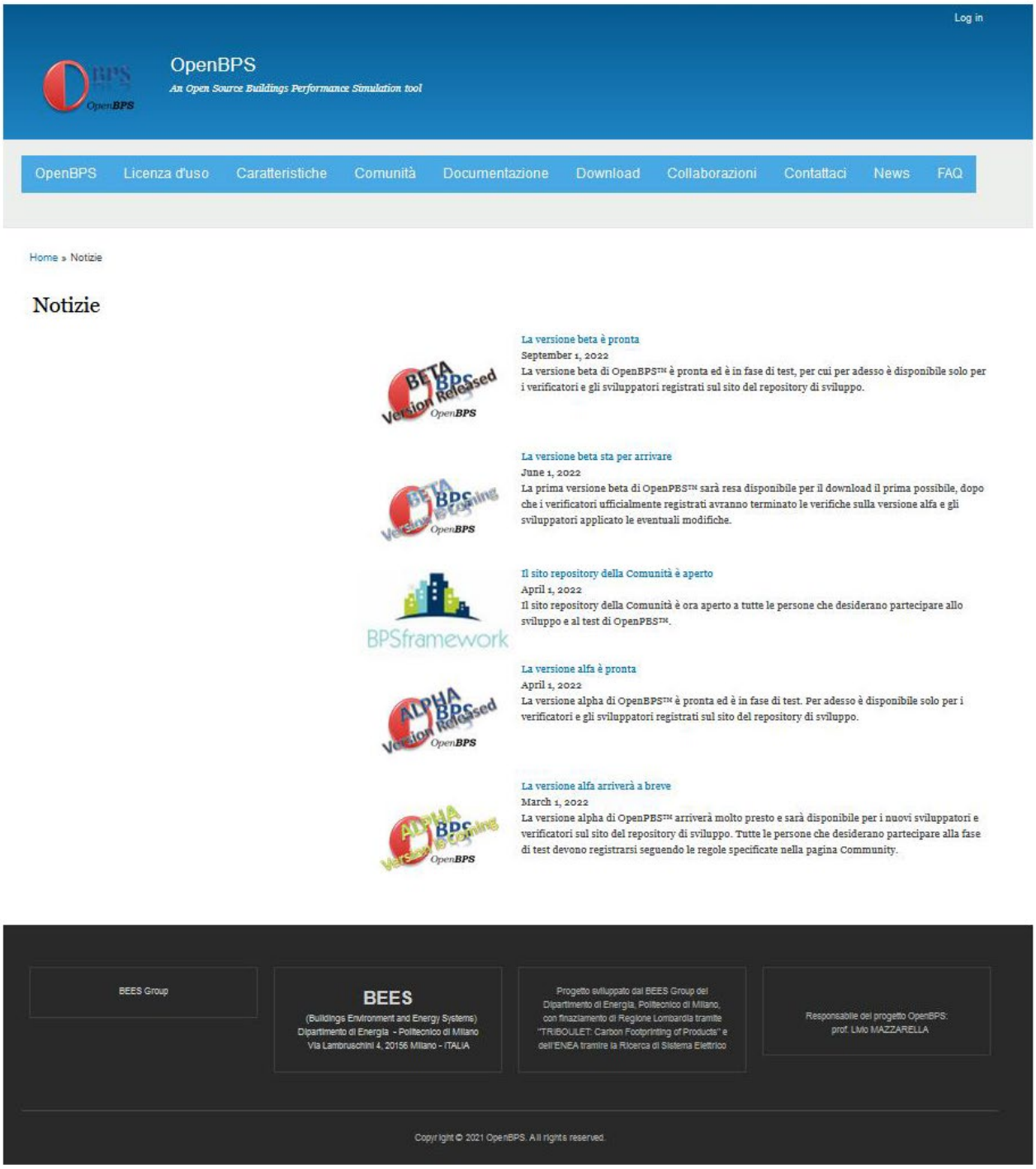


Figura 44 – Schermata della pagina che riporta l'archivio storico delle notizie ordinato per data decrescente

Figura 45 – Schermata della pagina che riporta la risposta alla domanda sulla necessità di sviluppare un altro strumento di simulazione delle prestazioni degli edifici (BPSt)

6 Repository

6.1 Scelta del repository

A prescindere da quelli che saranno gli sviluppi futuri dei risultati del progetto di ricerca, che, per quanto riguarda la creazione di una Community e delle modalità di distribuzione e partecipazione allo sviluppo del software OpenBPS, che, come detto nel capitolo precedente, dipendono pesantemente dalla scelta del marcato di riferimento, non obbiettivo di questa ricerca, si è comunque proceduto alla scelta ed implementazione di un sistema di condivisione e controllo di versione del codice a partire dagli obbiettivi principali riassumibili come segue:

Obiettivo del progetto: Sviluppo di una nuova libreria open-source per la simulazione dinamica sistema edificio, aperta al contributo di una comunità distribuita.
Il software open source può essere definito come qualsiasi software per computer, generalmente sviluppato come una collaborazione pubblica, il cui codice sorgente è reso liberamente disponibile, al fine di beneficiare della collaborazione pubblica, non legata a una singola società di sviluppo, e della verificabilità.

Attività volte a consentire il contributo di una comunità distribuita: Sviluppo di una Piattaforma Web per il Project Management (PM).
PM si concentra sul controllo dell'introduzione dei cambiamenti desiderati, comprendendo le esigenze degli utenti finali, suggerendo cosa deve essere fatto, quando, da chi e a quali standard, costruendo e motivando l'ulteriore sviluppo, coordinando il lavoro di diverse persone, monitorando il lavoro svolto, gestendo eventuali modifiche al piano e fornendo risultati di successo.

Requisiti della piattaforma Web Software Forge:
tale software dovrebbe:

- ✓ ospitare il repository di codice;
- ✓ implementare il controllo della versione del codice sorgente, il fork e l'unione;
- ✓ essere accessibile via web;
- ✓ essere ospitato da un server di proprietà del Politecnico di Milano;
- ✓ essere implementato seguendo il paradigma open-source o utilizzando un progetto open source;
- ✓ facilitare la navigazione all'interno della libreria sviluppata;
- ✓ facilitare la pianificazione, l'organizzazione e la gestione dello sviluppo della libreria;
- ✓ facilitare la comunicazione tra i membri della comunità;
- ✓ consentire il download della libreria;
- ✓ consentire la navigazione e il download della documentazione della libreria;
- ✓ consentire un'elevata correlazione tra la documentazione e il codice implementato.

6.1.1 Analisi degli strumenti esistenti e motivazione delle scelte effettuate

L'analisi è iniziata con un'indagine volta a determinare quali soluzioni sono state seguite da comunità simili, in particolare le comunità Esp-r ed Energy+.

La community esp-r si affida alla piattaforma CloudForge, mentre Energy+ ha due diverse community, una ufficiale che per il Change and Configuration Management utilizza Borland StarTeam e una non ufficiale che utilizza l'unica soluzione open source, ovvero SourceForge. [2][3][4]
Poiché uno dei requisiti era l'hosting della piattaforma web da parte di un server di proprietà del Politecnico, abbiamo focalizzato la seguente ricerca su un software di project management open source (PMS). È possibile accedere a un software di gestione dei progetti basato sul Web tramite Internet o WAN / LAN utilizzando un browser Web. Non è necessario installare altri software sul sistema. Il software può essere utilizzato facilmente dalle funzioni di controllo degli accessi (multiutente).
Tra i diversi software di gestione dei progetti open source, Redmine[5] e Allura[6] sono più vicini alle nostre esigenze.
Apache Allura è progetto della Apache Software Foundation, sponsorizzato dall'Apache Incubator PMC. Allura è un'implementazione open source del software "forge", un sito web che gestisce repository di codice sorgente, segnalazioni di bug, discussioni, mailing list, pagine wiki, blog e altro ancora per qualsiasi numero di singoli progetti. È scritto in Python e sfrutta un gran numero di pacchetti Python esistenti.
Le funzionalità principali di Allura sono le seguenti:

- monitoraggio dei problemi
- forum di discussione in thread / Mailing List
- Wiki
- repository di codice
- strutture di documentazione
- formattazione markdown ovunque
- indicizzazione Solr per la ricerca completa su tutti gli strumenti
- reticolazione tra elementi in qualsiasi strumento
- integrazione e-mail (ogni applicazione dello strumento ottiene il proprio indirizzo e-mail)
- Git
- SVN ·
- Mecurial

Redmine è un'applicazione web flessibile per la gestione dei progetti. Scritto utilizzando il framework Ruby on Rails, è multipiattaforma e cross-database e ha le seguenti caratteristiche:

- monitoraggio flessibile dei problemi
- controllo degli accessi flessibile basato sui ruoli
- Gant, grafico e calendario
- gestione di notizie, documenti e file
- Wiki per progetto
- forum per progetto
- monitoraggio del tempo
- campi personalizzati per problemi, voci temporali, progetto e utenti
- supporto per l'autenticazione LDAP multipla[7]

[2] http://espr.trac.cvsdude.com/esp-r/wiki
[3] http://www.borland.com/products/starteam/
[4] http://sourceforge.net/projects/epx/
[5] http://www.redmine.org/
[6] https://forge-allura.apache.org/p/allura/wiki/Allura%20Wiki/
[7] http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol

- [8]integrazione SCM (SVN, CVS, Git, Mercurial, Bazaar e Darcs)[9][10][11]

Il cuore del progetto open source è il codice sorgente disponibile nel repository. In Allura la navigazione del codice è possibile e si possono collegare i *commit* ai *ticket* e ad altri artefatti. Il forking e l'unione (*merge*) dlele richieste con git e hg sono le altre caratteristiche del suo repository. Inoltre la cronologia dei *commit* può essere visualizzata in forma grafica. [12]

In Redmine un progetto viene collegato a un repository solo dopo aver creato e abilitato, mentre in Allura si crea prima un progetto e poi si aggiungere un repository dedicato.

In Allura ci sono diversi modi per creare la documentazione del progetto, come creare pagine wiki con markdown, allegati, discussioni in thread e collegamenti di artefatti. Inoltre si può installare e utilizzare qualsivoglia strumento nel spazio web del progetto.
L'unica caratteristica mancante in Allura è lo strumento agile, che nel nostro caso non è critico, e potrebbero essere utilizzati altri strumenti agili esistenti.

Entrambe le applicazioni hanno mostrato buone caratteristiche.

Tuttavia, si deciso di utilizzare il software open source di gestione del progetto Apache Allura, poiché è più recente, risponde ai requisiti ed è uni sviluppo della Apache Software Foundation, sponsorizzata dall'Apache Incubator PMC.

6.2 Implementazione del repository

Il sito per gli sviluppatori è quindi basato su un altro ambiente open source sviluppato dalla Apache Foundation che è Allura [28]. Apache Allura è un software fucina open-source per la gestione di repository di codice sorgente, segnalazioni di bug, discussioni, pagine wiki, blog e altro per un qualsiasi numero di singoli progetti. Il sito "sourceforge" per lo sviluppo e la diffusione di progetti open source è basato su Allura. In particolare, si è deciso che, tra tutte le possibilità disponibili (git, SVN e Mercurial), di implementare il controllo della versione corrente del software tramite Git [29]. Allura inoltre, come già detto, include:

- la navigazione tra i file e gli invii degli aggiornamenti (commit) da parte degli sviluppatori basata su browser;
- visualizzazione delle differenze tra versioni diverse di codice o con codice-colore unificato o con visualizzazione side-by-side,
- l'evidenziazione della sintassi
- la gestione delle richieste di biforcazione e unione di parti di codice;
- la visualizzazione grafica della stria degli invii da parte degli sviluppatori.

Include inoltre il servizio di "biglietto / tracciamento" degli errori (Ticket / Bug Tracking), che consente:

- tracciamenti multipli per progetto;
- file allegati;

[8] http://www.redmine.org/projects/redmine/wiki/RedmineRepositories
[9] http://en.wikipedia.org/wiki/GNU_Bazaar
[10] http://en.wikipedia.org/wiki/Darcs
[11] http://biz30.timedoctor.com/git-mecurial-and-cvs-comparison-of-svn-software/
[12] Un fork è una copia remota lato server di un repository, distinta dall'originale. Chiunque può biforcare un progetto aperto e l'autore originale non incorre in sanzioni o oneri a causa di ciò, l'operazione è trasparente. Un ramo è qualcosa che si trova all'interno di un repository, Rappresenta un filo di sviluppo. L'unione è il processo di combinazione delle modifiche al codice da rami diversi o da versioni diverse dello stesso ramo.

- pietre miliari, etichette e campi personalizzati;
- le ricerche salvate per un uso frequente;
- la modifica collettiva dei biglietti.

È inoltre possibile attivare forum di discussione paralleli dotati di:

- moderazione,
- risposta tramite posta elettronica;
- prevenzione dello spam.

È inclusa anche una sezione “wiki”, che consente:

- di includere allegati;
- l’evidenziazione della sintassi per frammenti di codice;
- la navigazione tra le pagine per nome o tag;
    - macro personalizzate per cose come gli elenchi di progetto, elenchi di post del blog, e l'aggiunta di un pulsante di Gittip.

Infine è disponibile la sezione “Blog”, che consente di:

- pre-pubblicare le bozze dei report;
- l’integrazione con “feed” esterni, cioè con flussi di informazione provenienti da altri siti;
- commenti alle discussioni, con la prevenzione dello spam.

Il sito di sviluppo è correntemente installato su un server Linux del Dipartimento di Energia sulla intranet dipartimentale. Quindi il suo accesso è attualmente limitato solo agli sviluppatori del Dipartimento. È comunque disponibile il file di una macchina virtuale per il software open source Virtual Box [30] (software per l'esecuzione di macchine virtuali per architettura x86 e 64bit che supporta Windows, GNU/Linux e Mac OS X come sistemi operativi host, ed è in grado di eseguire Windows, GNU/Linux, e altri, come sistemi operativi guest), che contiene tale repository. Non si trova sul DVD allegato perché ha una dimensione di circa 50 GByte e necessita di un computer con Virtual Box installato per poter essere mandato in esecuzione. Tale file è comunque sempre richiedibile, anche se per la sua stessa natura il repository di sviluppo è qualcosa che muta in continuazione e quindi ha poco senso disporne di una versione datata.
In Figura 49 è riportato uno screen-shoot del repository, con riferimento al tracciamento delle attività solte dagli sviluppatori, mentre in **Error! Reference source not found.** è riportata una schermata che mostra la struttura gerarchica del repository, sia per il codice che per la documentazione, che consente di navigarvi attraverso e di esaminarne, tramite il browser, il contenuto in modo diretto.
In Figura 54Figura 49 è riportata una schermata che mostra la struttura delle pagine “wiki”, mentre in Figura 55 quella che riguarda il Forum e in Figura 56 quella che riguarda i “ticket” di richiesta per debug di errori.

Tale sito verrà esposto al pubblico, cioè connesso ad internet, solo dopo la costituzione della comunità di sviluppatori, secondo le regole che verranno stabilite insieme con l’ENEA, dopo la conclusione del progetto.

6.3 Organizzazione del repository

Apache Allura (AA) è un sistema di controllo delle versioni e controllo delle versioni del software distribuito come software libero sotto le licenze Apache basato su Git. Gli sviluppatori utilizzano Git per mantenere le versioni correnti e storiche di file come il codice sorgente, le pagine Web e la documentazione. L’organizzazione classica di un sistema di controllo delle versioni consiste nell'avere tre cartelle definite a livello di repository (archivio del codice sorgente): trunk, branches e tag (tronco, rami e etichette).

- **Trunk o Master:** costituisce la principale area di sviluppo. Contiene tutto il codice sorgente e la documentazione per l'intero prodotto. Il nuovo codice viene archiviato nel *trunk*. Il *trunk* è il luogo in cui avviene lo sviluppo attivo di nuove funzionalità. Il *trunk* potrebbe essere descritto come "*fondamentalmente instabile*" giacché la stabilità oscilla nel corso dei mesi durante il ciclo di sviluppo. Quindi, durante le parti iniziali e centrali di un ciclo di sviluppo, il *trunk* non è molto stabile.

Avvicinandosi alla versione alfa, alla beta e alla versione finale, le cose si sistemano e il *trunk* diventa sempre più stabile. Il *Trunk* contiene i seguenti file:
  - codice sorgente;
  - documenti;
  - test.
- **Branches:** area che contiene una cartella per ogni sviluppatore. Il nuovo sviluppo e/o la modifica da parte degli sviluppatori verranno aggiunti al codice principale se i risultati del test della loro implementazione saranno accettati dal coordinato del progetto. Il *Branch* consente al team di sviluppo di lavorare contemporaneamente sulle diverse copie di un progetto.
- I **tag** contengono l'ultima versione stabile del codice, ovvero ogni versione rilasciata con documenti correlati, librerie, ecc.

6.4 Pagine del repository

Quando si accede al repository via web, la pagina principale mostra sotto l'etichetta "*All Neighborhoods*": due pulsanti, uno relativo ai progetti pubblico presenti nel repository, l'altro agli utenti accreditati (vedi Figura 46).
Facendo click sul bottone Project si accede alla pagina che riporta la lista dei progetti pubblici con una descrizione sintetica (possono esistere ance dei progetti privati che sono visibili solo a specifici soggetti all'uopo accreditati) (vedi Figura 47).
Facendo click sul nome del progetto, ad esempio OpenBPS, si accede alla pagina principale dello stesso, che mostra di default il cronologico dell'attività svolta di recente su quello specifico progetto (vedi Figura 48).
Nel menù orizzontale sono riportate le etichette per l'accesso alle altre pagine del progetto, quelle dei Tikets, del codice, del Wiki, delle discussioni e del blog (vedi Figura 48).
Andando alla pagina del codice, Figura 50, si accede per default al "trunk" del codice sorgente, indicato nella pagina e nel menù laterale come ***master,*** cioè come *master branch* ossia *trunk*.

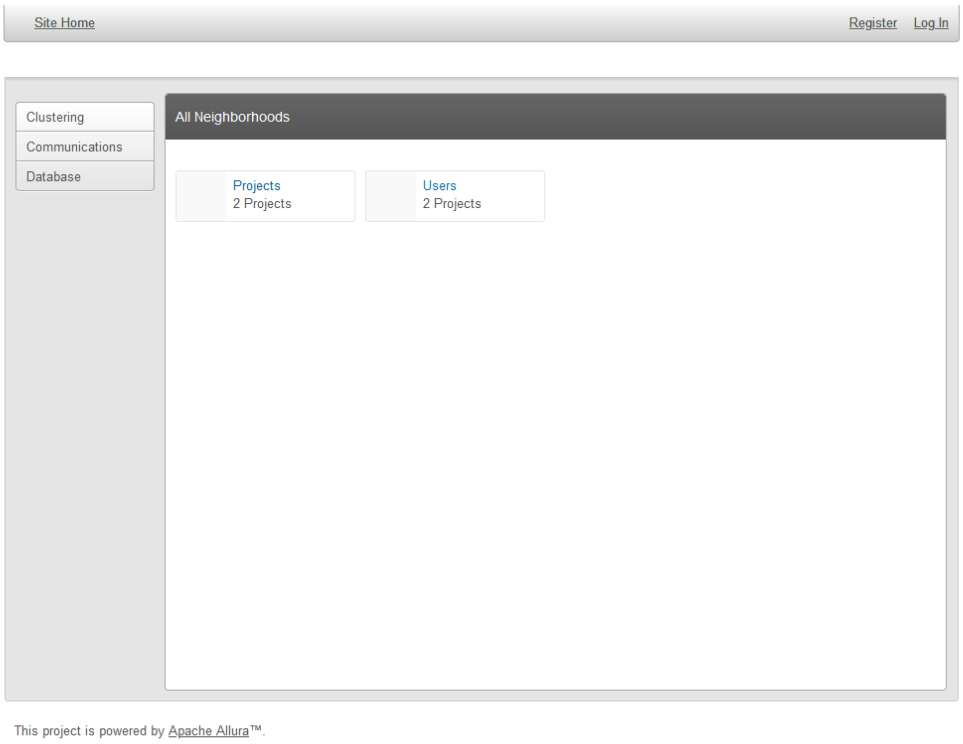


Figura 46 – Schermata di accesso al repository via web

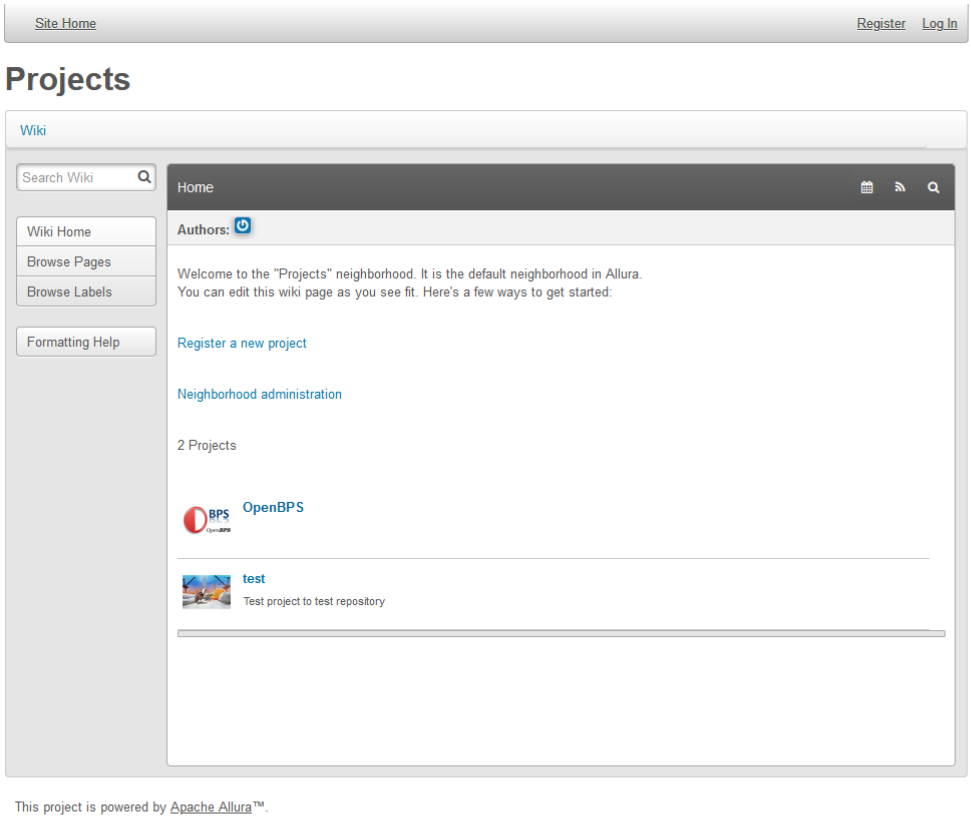


Figura 47 – Schermata di accesso ai progetti pubblici

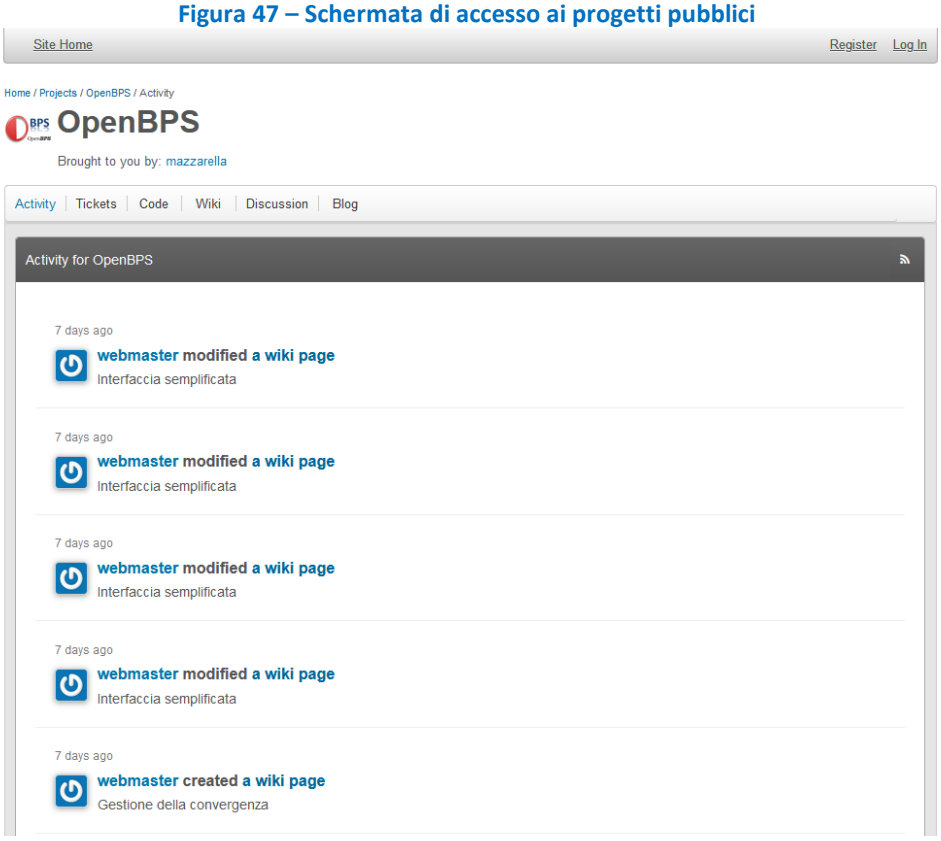


Figura 48 – Schermata principale del progetto con cronologico delle attività svolte

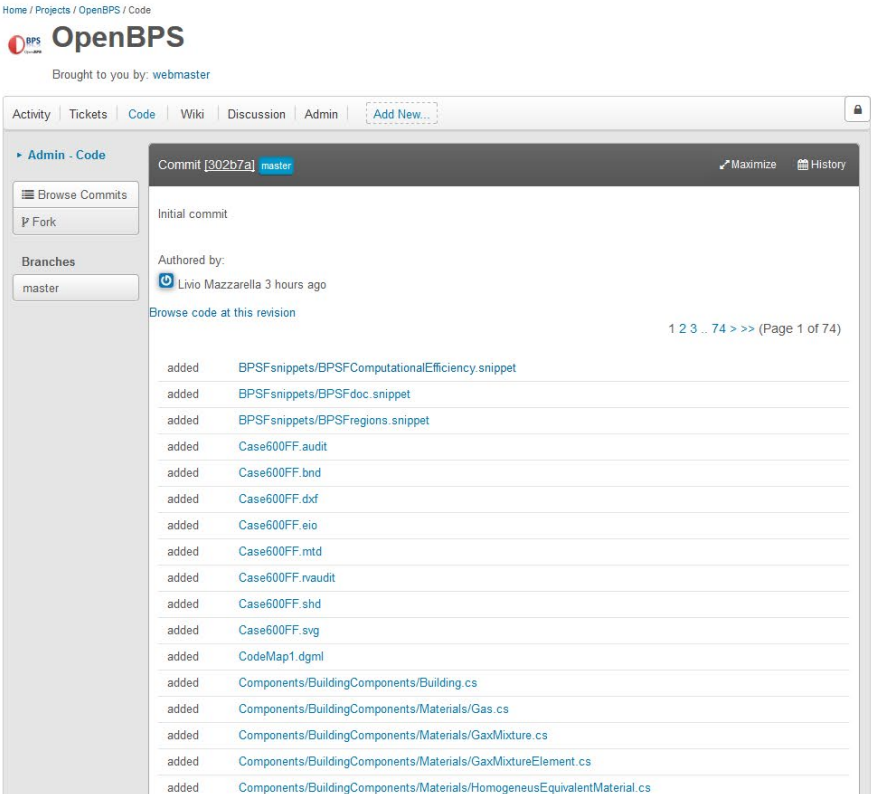


Figura 49 - Schermata che mostra il tracciamento delle attività (in questo caso il commit iniziale del progetto su repository)

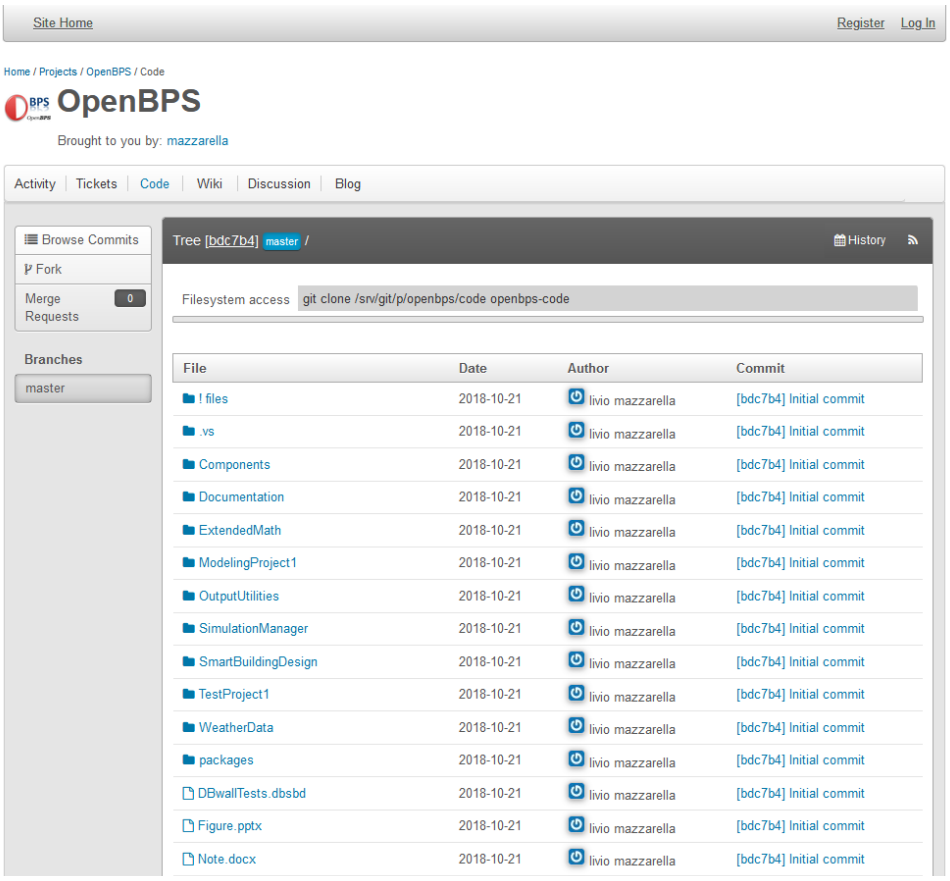


Figura 50 – Schermata che ripota il "trunk" del codice

Nella barra orizzontale della pagina del codice è riportata la modalità di accesso al codice tramite Git. Nello specifico di Figura 50, il repository non è su un sito esterno, ma locale, e quindi la funzione Git di copia del codice da repository a computer dello sviluppatore, git clone, riporta solo il cammino di accesso sul file system senza l'indirizzo IP del server su cui il repository viene installato; cioè:

```
git clone /srv/git/p/openbps/code openbps-code
```

Nel caso più generale sarà del tipo http, come:

```
git clone http://10.48.81.246:8088/git/p/openbps/code openbps-code
```

Tramite tale funzione Git si ricopia localmente tutto il codice presente nel "trunk" di OpenBPS.

Se vi sono degli sviluppi paralleli e quindi dei rami rispetto al tronco, dal menù laterale, sotto l'etichetta "*Branches*", compariranno i progetti collaterali identificati da uno specifico nome (diverso da master). Tali biforcazioni del progetto principale si generano tramite una richiesta di "fork" (bottone in alto sinistra in Figura 50).

Allo stesso modo, quando uno sviluppatore pensa di aver terminato la propria attività su una specifica parte del progetto e desidera che questa venga inclusa nel progetto definitivo o principale, attiva una "*richiesta di merge*" (bottone in alto a sinistra in Figura 50), che può essere accettata come anche rifiutata dal gestore del progetto (vedi Figura 51).

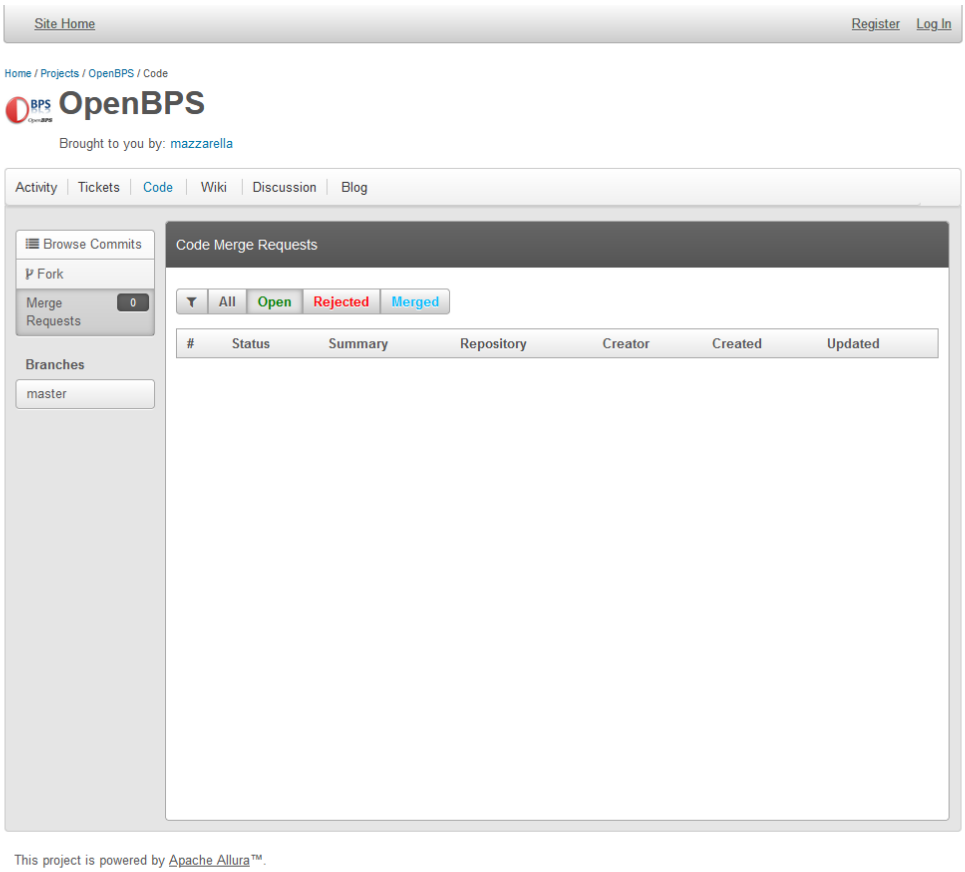


Figura 51 – Schermata che ripota le richieste merge del codice

Ovviamente lo sviluppatore può esaminate tutta la struttura gerarchica del progetto, entrando nelle varie cartelle e sottocartelle, fino al file contenete il codice di suo interesse e visualizzarne il contento, cosi come mostrato in Figura 52 relativa all' Individuazione dell'AssemblyInfo.cs del SimulationManager di OpemBPS (Tree [376594] master / SimulationManager / Properties /) e alla sua visualizzazione (Figura 53).

La visualizzazione del codice consente ovviamente anche di copiare lo stesso tramite le funzioni di Copia/Incolla del browser.

Oltre le pagine che documentano il codice sorgente e le relative attività di sviluppo, il repository di OpenBPS ospita le pagine dedicate alla “rubrica” Wiki (Figura 54), cioè un sito web che consenta agli utenti del repository di aggiungere nuovi contenuti o di modificare quelli già esistenti riguardanti lo sviluppo, l’impiego e la documentazione di OpenBPS (le pagine Wiki devono ancora essere popolate di contenuti, che però sono di fatto un contributo della Comunità e quindi seguono la sua creazione).

Inoltre il repository ospita le pagine del Forum, il luogo di incontro e discussione degli sviluppatori della Comunità su problemi e prospettive di sviluppo del codice (vedi Figura 55).

Infine, a beneficio degli utenti di OpenBPS, il sito ospita anche le pagine dei “tikets”, cioè delle segnalazioni da parte degli utenti di qualche baco nel programma, reale o presunto, dove l’utente documenta il problema riscontrato, problema che verrà assegnato ad uno sviluppatore per la sua disamina e l’identificazione della soluzione (vedi Figura 56).

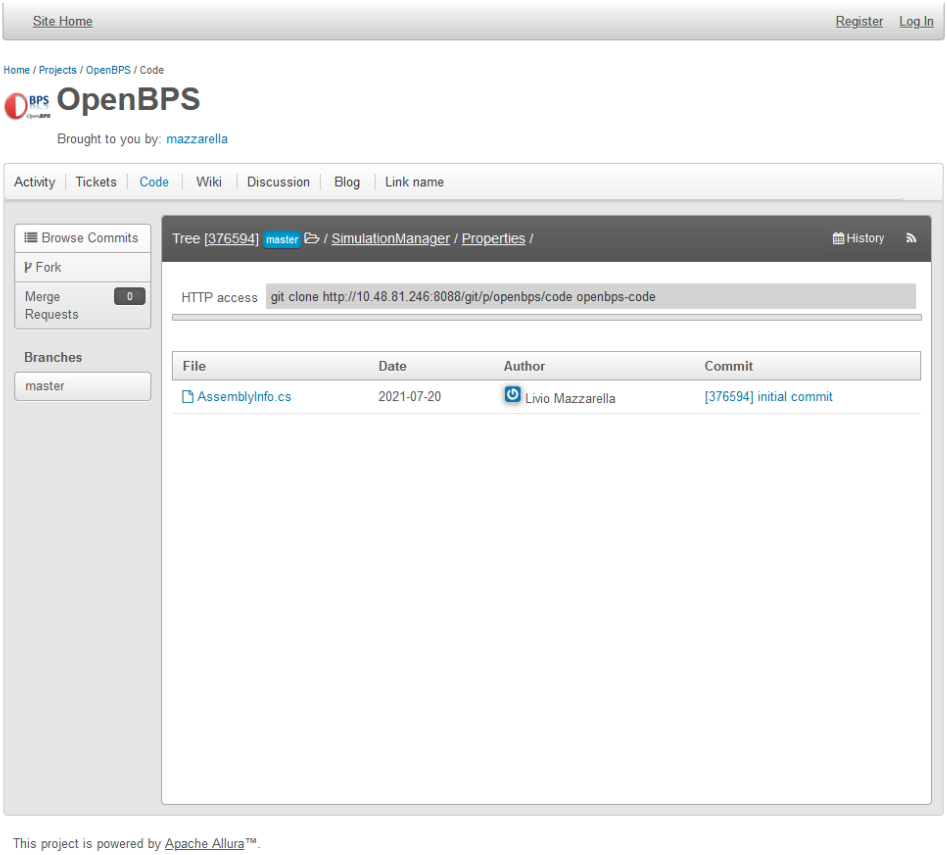


Figura 52 – Individuazione dell’AssemblyInfo.cs del SimulationManager

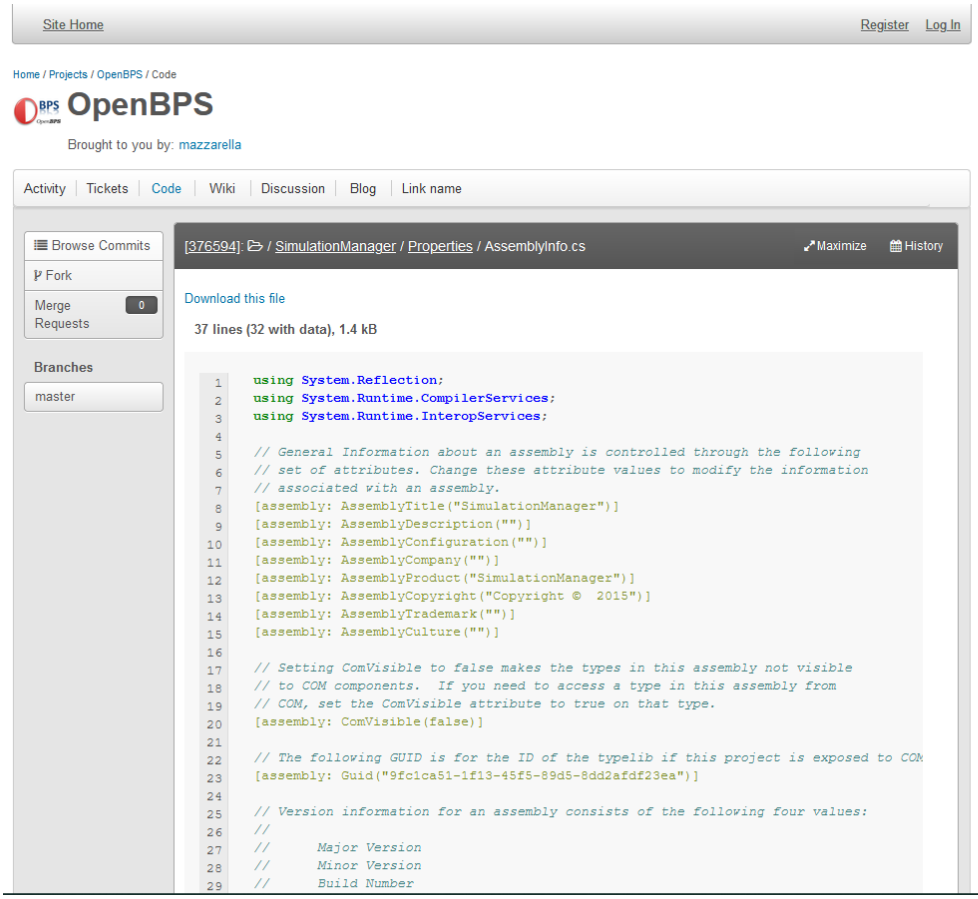


Figura 53 – Visualizzazione del codice dell'AssemblyInfo.cs del SimulationManager

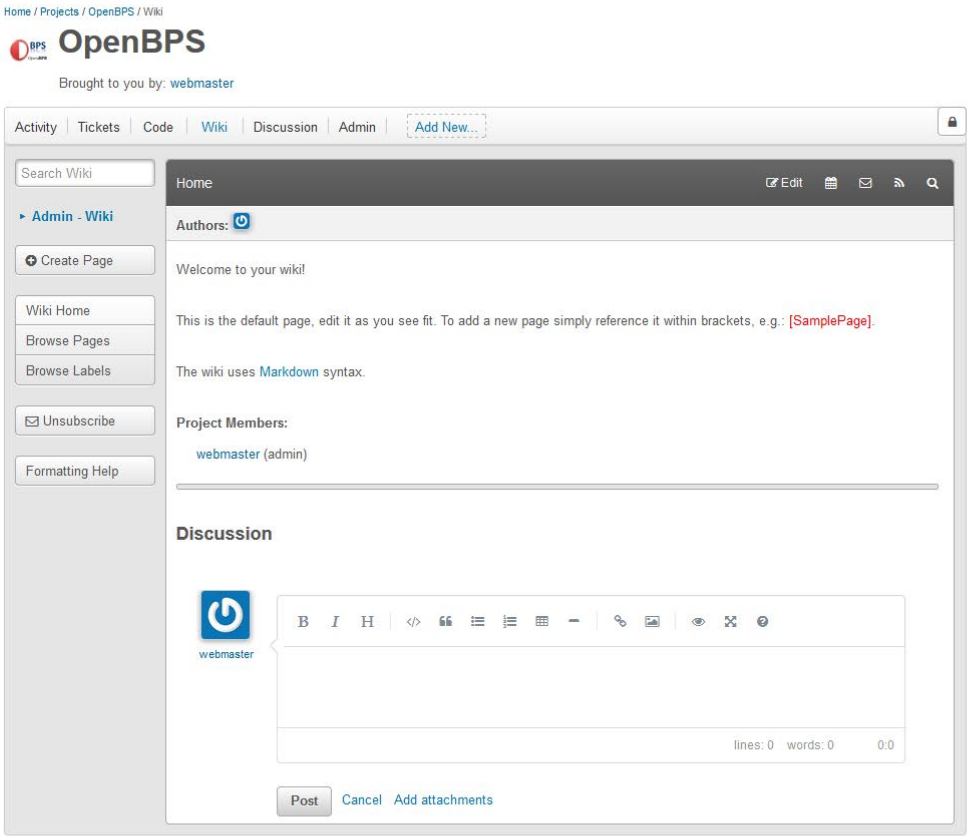


Figura 54 - Schermata che mostra la struttura delle pagine "wiki"

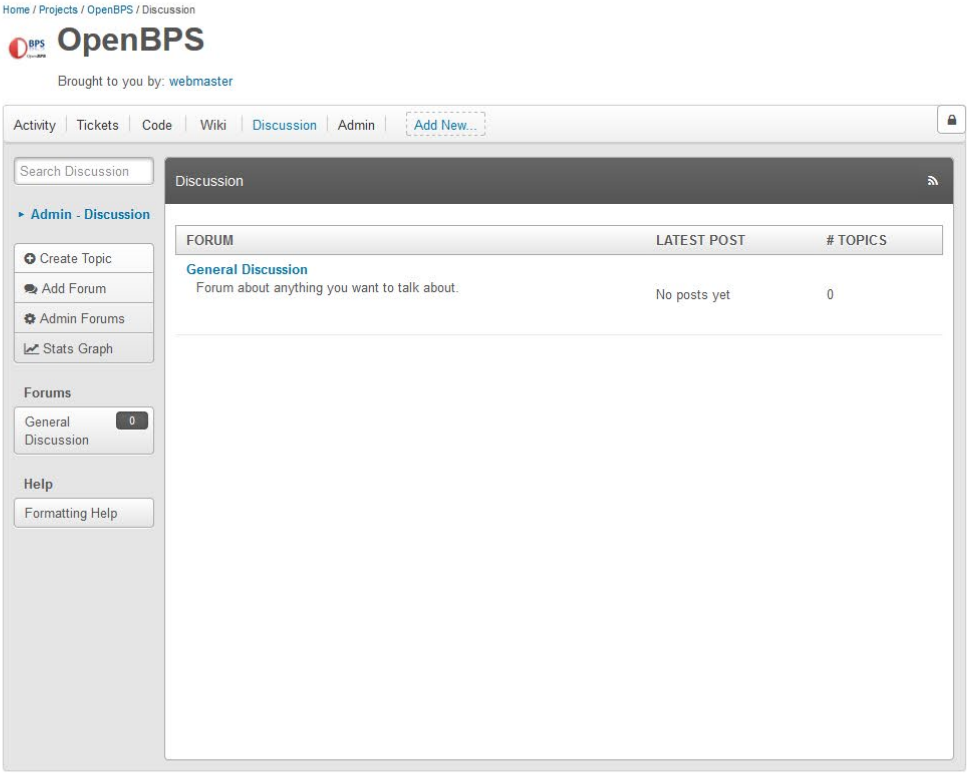


Figura 55 - Schermata che mostra la struttura delle pagine del Forum"

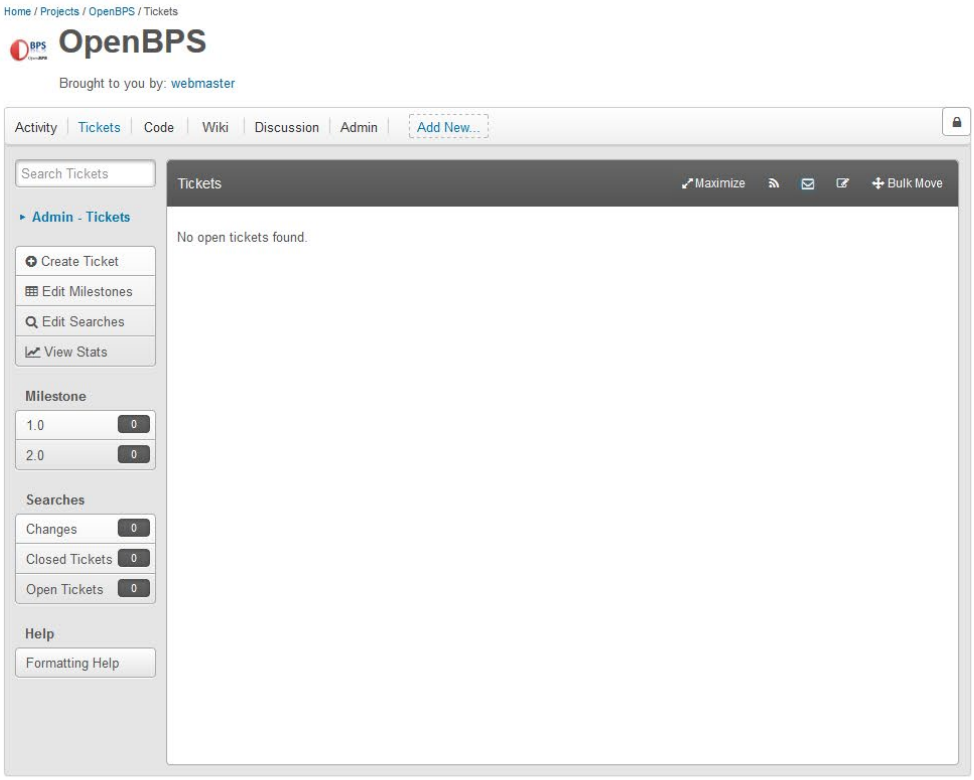


Figura 56 - Schermata che mostra la struttura delle pagine dei "tickets."

7 Conclusioni

Al termine del terzo anno di attività del progetto di ricerca mirante a realizzare un codice di calcolo per la simulazione dinamica delle prestazioni energetiche del sistema edificio che, pur consentendo un'analisi dettagliata e di qualità (base di calcolo oraria e sub-oraria), sia utilizzabile da utenti con gradi diversi di competenza e che possa evitare una grande richiesta di dati all'utente, tramite l'impiego di informazioni provenienti direttamente dai costruttori di componenti, e che sia principalmente un insieme di librerie dinamiche fruibili da interfacce sviluppate da terzi, si può concludere che:

- il codice sviluppato il primo anno, riguardante la prestazione termo-energetica dell'involucro edilizio è stato esteso questo terzo anno includendo nel "nodo" aria il bilancio del vapore d'acqua, e quindi il calcolo del carico termico latente e del fabbisogno di energia per deumidificazione e umidificazione, presente quando si vuole controllare l'umidità dell'aria interna;
- a complemento dell'attività di sviluppo del codice, sempre nell'ottica di librerie dinamiche fruibili da terzi, sono stati sviluppati una serie di metodi per il calcolo delle proprietà termofisiche delle sostanze pure, per adesso limitate a acqua e aria secca, in funzione della temperatura e a pressione costante, e per la miscela aria umida, per la quale oltre le proprietà termofisiche, parametriche rispetto alla pressione, in funzione di ogni coppia possibile di coordinate psicrometriche, sono state implementate anche le principali trasformazioni della stessa (miscelazione adiabatica, deumidificazione, umidificazione, ecc.);
- una parte importante dell'attività del terzo anno è stata rivolta alla identificazione delle problematiche relative allo sviluppo efficiente e trasparente di quando fin qui prodotto nel futuro tramite la creazione di una comunità di sviluppatori indipendenti ma cooperanti in un'ottica Open Source; dei dieci punti chiave identificati solo alcuni hanno trovato una risposta immediata che si è consolidata tramite la progettazione e la realizzazione di una repository del codice e di un sito web del progetto; altri sono stati solo parzialmente risolti con proposte interlocutrici che ruotano principalmente su un quesito fondamentale, a cui si potrà dare risposta solo successivamente alla conclusione della presente ricerca mettendo intorno ad uno stesso tavoli i potenziali futuri attori interessati: qual è il mercato di riferimento di questo codice di calcolo, un Open Sorce o un Free-to-Use?
- il sito web sviluppato, disponibile su macchina virtuale e in files di backup, è attualmente implementato su un server del Politecnico di Milano, in versione demo, cioè senza la possibilità di scaricare gli allegati, documentazione e codice eseguibile demo, per renderlo visibile all'ENEA, all'indirizzo IP 131.175.28.249, e verrà reso completamente operativo, su tale host o su altro host eventualmente messo a disposizione da ENEA, solo dopo l'approvazione finale del progetto e con il nulla osta dell'ENEA;
- il repository del codice sorgente per lo sviluppo collaborativo e il controllo di versione tramite git, dotato di pagine Wiki e un forum per le discussioni tra sviluppatori, è disponibile su macchina virtuale e in files di backup ed è anch'esso attualmente implementato su un server del Politecnico di Milano, in modalità protetta, cioè senza la possibilità di visualizzazione e accesso a terzi che non possano entrare nella Intranet di Ateno, ciò ovviamente per i motivi citati in precedenza per il sito Web.

8 Riferimenti bibliografici

1. Parlamento Europeo, “Direttiva 2002/91/EC del Parlamento europeo e del Consiglio del 16 dicembre 2002 sul rendimento energetico nell'edilizia”, Gazzetta ufficiale delle Comunità europee. L 1/65, 04.1.2003.
2. Parlamento Europeo, ”Direttiva 2010/31/UE del Parlamento europeo e del Consiglio del 19 maggio 2010 sulla prestazione energetica nell’edilizia (rifusione)”, Gazzetta ufficiale delle Comunità europee. L 153/13, 18.6.2010.
3. Parlamento Europeo, ”Direttiva 2018/844/UE del Parlamento europeo e del Consiglio del 30 maggio 2018 che modifica la direttiva 2010/31/UE sulla prestazione energetica nell’edilizia e la direttiva 2012/27/UE sull’efficienza energetica”, Gazzetta ufficiale delle Comunità europee. L 156/75, 19.6.2018.
4. Presidente della Repubblica, “Decreto Legislativo 19 agosto 2005, n. 192. Attuazione della direttiva 2002/91/CE relativa al rendimento energetico nell’edilizia”, Gazzetta Ufficiale n. 222 del 23 settembre 2005 - suppl. ord. n. 158.
5. Presidente della Repubblica, “Decreto-legge 4 giugno 2013, n. 63, Disposizioni urgenti per il recepimento della Direttiva 2010/31/UE del Parlamento europeo e del Consiglio del 19 maggio 2010, sulla prestazione energetica nell'edilizia per la definizione delle procedure d'infrazione avviate dalla Commissione europea, nonché altre disposizioni in materia di coesione sociale. (13G00107)”, Gazzetta Ufficiale n.130 del 5-6-2013
6. Presidente della Repubblica, “LEGGE 3 agosto 2013, n. 90. Conversione in legge, con modificazioni, del decreto-legge 4 giugno 2013, n. 63, recante disposizioni urgenti per il recepimento della Direttiva 2010/31/UE del Parlamento europeo e del Consiglio del 19 maggio 2010, sulla prestazione energetica nell'edilizia .. ”, Gazzetta Ufficiale Serie Generale n.181 del 03-08-2013.
7. IBPSA-USA, “BEST Directory - Building Energy Software Tools”, disponibile presso: http://www.buildingenergysoftwaretools.com/ [ultimo accesso: novembre, 2019].
8. ISO, “ISO 2533:1975 - Standard Atmosphere”, International Standard Organization, 1975, Ginevra, Svizzera
9. ISO, “ISO 2533:1975/Add 2:1977 - Extension to - 5000 m and standard atmosphere as a function of altitude in feet”, International Standard Organization, 1977, Ginevra, Svizzera
10. UNI, “UNI EN ISO 5801:2018 - Ventilatori - Verifica delle prestazioni che utilizzano vie aeree standardizzate”, UNI, 2018, Milano.
11. UNI, “UNI EN ISO 9346:2008 - Prestazione termoigrometrica degli edifici e dei materiali da costruzione - Grandezze fisiche per il trasferimento di massa – Vocabolario”, UNI, 2008, Milano
12. UNI, “UNI EN Prestazione energetica degli edifici - Ventilazione per gli edifici - Parte 5-1: Metodi di calcolo per i requisiti energetici dei sistemi di ventilazione (Moduli M5-6, M5-8, M6-5, M6-8, M7-5, M7-8) - Metodo 1: Distribuzione e generazione ”, UNI, 2018, Milano.
13. UNI, “UNI EN ISO/DIS 8502-4:2017 - Preparazione di substrati di acciaio prima dell'applicazione di pitture e prodotti similari - Prove per valutare la pulizia della superficie - Parte 4: Guida alla valutazione della probabilità di condensazione prima dell'applicazione della pittura”, UNI, 2017, Milano.
14. UNI, “UNI EN ISO 13788:2013 - Prestazione igrotermica dei componenti e degli elementi per edilizia - Temperatura superficiale interna per evitare l'umidità superficiale critica e la condensazione interstiziale - Metodi di calcolo”, UNI, 2017, Milano.
15. UNI, “UNI EN 15316-4-1:2018 - Prestazione energetica degli edifici - Metodo per il calcolo delle richieste di energia del sistema e delle efficienze del sistema - Parte 4-1: Sistemi di riscaldamento e di generazione

di acqua calda sanitaria, sistemi di combustione (caldaie, biomasse), Modulo M3-8-1, M8-8-1", UNI, 2018, Milano.
16. ASHRAE, "ASHRAE Handbook—Fundamentals 2017", American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc, Atlanta, GA, USA.
17. Wagner, W., and Pruß, A., "The IAPWS Formulation 1995 for the thermodynamic properties of ordinary water substance for general and scientific use", J. Phys. Chem. Ref. Data 31, 387-535 (2002).
18. Kretzschmar H.J. and Wagner W., "International Steam Tables - Properties of Water and Steam based on the Industrial Formulation IAPWS-IF97", 3rd Edition, 2019, Springer-Velarg GmbH, Berlin
19. Gatley D.P., Herrmann S., Kretzschemar H.-J., "A Twenty-First Century Molar Mass for Dry Air", (2008) HVAC&R RESEARCH 14(5),655-662.
20. Picard A., Davis R.S, Glaser M. and Fujii K., "Revised formula for the density of moist air (CIPM-2007)", Metrologia 45 (2008) 149–155.
21. NIST, "CODATA Internationally recommended 2018 values of the Fundamental Physical Constants", NIST Standard Reference Database 121. Last Update to Data Content: May 2019, https://physics.nist.gov/cuu/Constants/index.html
22. Lemmon E.W., Jacobsen R.T., Penoncello S.G., and Friend D.G., "Thermodynamic Properties of Air and Mixtures of Nitrogen, Argon, and Oxygen From 60 to 2000 K at Pressures to 2000 MPa", Journal of Physical and Chemical Reference Data 29, 331 (2000);
23. IAPWS, "Guideline on the Use of Fundamental Physical Constants and Basic Constants of Water, IAPWS G5-01(2020), International Association for the Properties of Water and Steam, 2020, http://www.iapws.org/index.html.
24. Mazzarella L., "The air energy balance equation paradox", Proceedings of Building Simulation Applications BSA 2013, 1st IBPSA Italy conference, Bolzano, 30th January –1st February 2013, pages 15-28, ISSN 2531-6702
25. Mazzarella L., Alongi A., Angelotti A., Filippini G., Gandelli A., Pasini M., "Metodologie di calcolo del Fabbisogno energetico degli edifici: metodi dinamici. Sviluppo e applicazione di un codice di calcolo I", Rapporto Ricerca di Sistema Elettrico, Accordo di Programma Ministero dello Sviluppo Economico – ENEA, Piano Triennale di Realizzazione 2019-2021 - I annualità, Dicembre 2019.
26. Drupal Org, "Drupal - Open Source CMS ", https://www.drupal.org/
27. The Apache Software Foundation, "Apache Allura", https://allura.apache.org/
28. Git, "GIT", https://git-scm.com/
29. Oracle, "VirtualBox", https://www.virtualbox.org/

9 Abbreviazioni e acronimi

CVCS	**C**entralised **V**ersion **C**ontrol **S**ystems
DVCS	**D**istributed **V**ersion **C**ontrol **S**ystems
EUPL	**E**uropean **P**ublic **L**icense
GPL	**G**eneral **P**ublic **L**icense
ISA	**I**nternational **S**tandard **A**tmosphere
LGPL	**L**esser **G**eneral **P**ublic **L**icense
NTP	**N**ormal **T**emperature and **P**ressure
OSS	**O**pen-**S**ource **S**oftware
PM	**P**roject **M**anagement
PMS	**P**roject **M**anagement **S**oftware
SATP	**S**tandard **A**mbient **T**emperature and **P**ressure
STP	**S**tandard **T**emperature and **P**ressure
SVN	**S**ub**v**ersio**n**
VCS	**V**ersion **C**ontrol **S**ystems

10 Autori

Prof. Livio Mazzarella

Laureato in Ingegneria Meccanica indirizzo Energetico al Politecnico di Milano nel 1982 con il massimo dei voti, consegue il titolo di Dottore di Ricerca in Energetica, primo ciclo, nel luglio 1987. Ricercatore di ruolo dal 1990, professore incaricato dal 1993 al 1997 presso l'Università di Pavia, diventa professore associato di ruolo in Fisica Tecnica Ambientale nel 1998 e successivamente Professore Straordinario sempre presso la Facoltà d'Ingegneria del Politecnico di Milano dal 1° settembre 2000. Dal 1° settembre 2003, confermato nel ruolo di Professore Ordinario di Fisica Tecnica Ambientale (ING-IND/11), è stato docente di numerosi insegnamenti nell'ambito della fisica dell'edifico. Ha svolto e svolge attualmente attività di ricerca, documentata da più di 200 pubblicazioni, sui temi dell'Energetica e della Fisica Tecnica, e più precisamente nei settori della termo fluidodinamica, modellistica numerica dei sistemi edificio-impianto, energetica negli edifici, termofisica degli edifici, conversione termica dell'energia solare, sistemi per l'accumulo di energia termica: interstagionali e non, pompe di calore, trasporto di massa ed energia nei mezzi porosi, acustica applicata. Partecipa da più di vent'anni all'attività di normativa nel settore dell'energia negli edifici e degli impianti tecnici tramite il Comitato Termotecnico Italiano, ove ha ricoperto la carica di vice-presidente del SC1 del CTI e ricopre attualmente la carica di presidente del SC5. È inoltre REHVA fellow, coordinatore del COP group di REHVA, membro della giunta direttiva di AICARR e di IBPSA-Italy.

Ing. Andrea Alongi

Laureato in Ingegneria Edile con lode nell'aprile 2007 presso il Po9litecnico di Milano, ha conseguito il Dottorato di Ricerca in Energetica con merito presso lo stesso Ateneo nel 2012 con uno studio sui componenti d'involucro edilizio permeabili all'aria. Dal 2007 al 2019è stato assegnista di ricerca presso il Dipartimento di Energia, dove ha svolto attività di ricerca sull' analisi numerica della prestazione energetica di edifici: lo studio, sia in termini di fabbisogno energetico in regime dinamico sia valutandone il comportamento in condizioni standardizzate da procedura per certificazione energetica, di edifici reali esistenti o in fase di progettazione; l'analisi parametrica tramite metodologia cost optimality di varie categorie di edifici virtuali (analisi per archetipi di residenze nuove ed esistenti, uffici nuovi ed esistenti, scuole}. Dall'ottobre 2010 è stato assunto dallo stesso Ateneo nel ruolo tecnico.

Prof. Adriana Angelotti

Adriana Angellotti laureata in Fisica nel 2000, ha conseguito il Dottorato in Ingegneria Energetica presso il Politecnico di Milano nel 2004, dove nel 2005 ha preso servizio come ricercatrice di ruolo nel settore scientifico Fisica tecnica ambientale e nel 2019 quale professore associato nello stesso SSD. Si dall'inizio ha svolto attività di ricerca nell'ambito dell'efficienza energetica del sistema edificio-impianto: prestazioni degli impianti di climatizzazione (geotermici, sistemi a bassa exergia) e delle tecnologie di involucro edilizio innovative (facciate traspiranti, facciate ventilate, inerzia termica, sistemi tessili per l'architettura); benessere termoigrometrico in ambienti interni ed esterni; ventilazione naturale alla scala urbana, con particolare riferimento alla loro modellazione ai fini della simulazione dinamica.

Ing. Martina Pasini

Laureata con lode nel 2006 con tesi “Analisi tecnica comparativa di nuove tecnologie emergenti”, riguardante principalmente la tecnologia di facciata ventilata trasparente (Double Skin Façades) e diplomatasi nello stesso anno all’Alta Scuola Politecnica, ha conseguito il titolo di dottore di ricerca, con lode, nel 2010 con la tesi “Towards an enriched modularity of building performance simulation’s programs. Reconceptualisation and development of an Object-Oriented model for the Simulation of the Building System”. Assegnista di ricerca per il Politecnico di Milano dal 2010 al 2017. Collaboratrice, come analista informatica e programmatrice, per un’industria produttrice di scambiatori di calore dal 2016 al 2019. Si occupa di simulazione dinamica dei fabbisogni del sistema Edificio-Impianto, sostenibilità ambientale, ottimizzazione di componente e di processo, algoritmi di automazione di processo, analisi dati.

11 Appendice A

11.1 Atmosfera Standard Internazionale (ISA)

L'ipotesi base dell'ISA è la distribuzione lineare della temperatura con l'altezza geopotenziale, H, cioè

$$T_{DA}(H) = T_{0;ISA} + \beta \cdot H \tag{A.1}$$

con β gradiente termico che varia nei vari strati in cui viene suddivisa l'atmosfera, come riportato in Tabella A. 1.

Tabella A. 1 – Temperature e gradienti verticali di temperatura

Altezza geopotenziale H [km]	Temperatura T [K]	Gradiente di temperatura β [K/km]
-2.00	301.15	-6.50
0.00	288.15	-6.50
11.00	216.65	0.00
20.00	216.65	+1.00
32.00	228.65	+2.80
47.00	270.56	0.00
51.00	270.65	-2.80
71.00	214.65	-20.0
80.00	196.65	

La distribuzione di pressione si ricava combinando insieme la definizione di pressione statica della colonna d'aria nel campo gravitazionale, data in forma differenziale rispetto all'altezza geopotenziale da:

$$dp = -\rho g \, dH \tag{A.2}$$

e l'equazione di stato dei gas ideali nella forma specifica all'aria secca come

$$\rho = \frac{p}{R_{DA} T} \tag{A.3}$$

cioè

$$dp = -g\,\frac{p}{R_{DA}T}dH \tag{A.4}$$

Da cui, differenziando la (A.1),

$$dT_{DA} = \beta \cdot dH \tag{A.5}$$

s sostituendo si ottiene

$$\frac{dp}{p} = -\frac{g}{\beta R_{DA}}\frac{dT}{T} \tag{A.6}$$

e quindi, integrando,

$$\ln\frac{p_f}{p_i} = -\frac{g}{\beta R_{DA}}\ln\frac{T_f}{T_i} \tag{A.7}$$

e quindi

$$p_f = p_i e^{-\frac{g}{\beta R_{DA}}\ln\frac{T_f}{T_i}} = p_i e^{\ln\left(\frac{T_f}{T_i}\right)^{-\frac{g}{\beta R_{DA}}}} = p_i\left(\frac{T_f}{T_i}\right)^{-\frac{g}{\beta R_{DA}}} \tag{A.8}$$

Sostituendo nella (A.8) la (A.1) si ottiene infine:

$$p_f = p_i \cdot \left[1 + \frac{\beta_{f-i}}{T_i}\left(H_f - H_i\right)\right]^{-\frac{g}{\beta_{f-i}R_{DA}}} \tag{A.9}$$

equazione che si applica ad ognuno degli strati identificati dalla differenza di altezza geopotenziale $H_f - H_i$.

Se invece di eliminare la massa volumica tra la (A.2) e la (A.3) si elimina la pressione ottiene la distribuzione verticale di massa volumica. Infatti, dall'equazione dei gas ideali differenziando si ha:

$$dp = \rho R_{DA}\,dT + R_{DA}T d\rho \tag{A.10}$$

Da cui sostituendo nella (A.2) si ottiene

$$\frac{d\rho}{\rho} = \frac{-1}{R_{DA}T}\left(R_{DA}\,dT + g dH\right) \tag{A.11}$$

e per la (A.1) la relazione differenziale tra ρ e T

$$\frac{d\rho}{\rho} = -\left(1 + \frac{g}{\beta R_{DA}}\right)\frac{dT}{T} \tag{A.12}$$

da cui integrando

$$\ln\frac{\rho_f}{\rho_i} = -\left(1 + \frac{g}{\beta R_{DA}}\right)\ln\frac{T_f}{T_i} \tag{A.13}$$

cioè, per la (A.1),

$$\rho_f = \rho_i \cdot \left[1 + \frac{\beta_{f-i}}{T_i}\left(H_f - H_i\right)\right]^{-\left(1 + \frac{g}{\beta_{f-i}R_{DA}}\right)} \tag{A.14}$$

12 Appendice B

12.1 Tipologie di licenze compatibili GPL e loro sintetica descrizione

Un primo esempio di licenza applicabile alla Pubblica Amministrazione è la European Union Public Licence (EUPL). Si tratta di una licenza Free/Open Source Software con effetto copyleft redatta e approvata dalla Commissione europea. Sempre da Wikipedia è possibile approfondire le caratteristiche dell'EUPL:

"*La presente licenza concede al licenziatario sull'opera originale, e per tutta la durata del diritto d'autore, una licenza di tipo mondiale, non esclusiva, gratuita e con la possibilità di essere concessa a sua volta in sub-licenza, conferendo il diritto di*:

- *utilizzare l'Opera in qualsiasi circostanza e per ogni utilizzo;*
- *riprodurre l'Opera;*
- *modificare l'Opera originaria e creare Opere derivate basate su di essa;*
- *comunicare al pubblico, anche mediante messa a disposizione o esposizione dell'Opera o di copie di essa e, a seconda dei casi, rappresentare l'Opera in forma pubblica;*
- *distribuire l'Opera o copie di essa;*
- *cedere in prestito e in locazione l'Opera o copie di essa;*
- *concedere in sub-licenza i diritti sull'opera o copie di essa.*

Tali diritti sono usufruibili mediante qualunque mezzo di comunicazione o tramite qualunque supporto attualmente esistente o di futura invenzione".

Mentre per avere un elenco di altre possibili licenze, si può consultare il sito GNU. **Quanto segue è proprio tratto da www.gnu.org/licenses/license-list.html nel novembre 2021, in una traduzione non ufficiale**. Per ogni licenza vengono definite le principali caratteristiche e spunti di possibile utilizzo. Ovviamente si tratta in buona parte di licenze per software open redatte da promotori dell'approccio OSS.

- Licenze per il software
    - Licenze di software libero compatibili con la GPL
    - Licenze di software libero incompatibili con la GPL
    - Licenze di software non libero
- Licenze per la documentazione
    - Licenze di documentazione libera
    - Licenze per documentazione non libera
- Licenze per altre opere
    - Licenze per materiale diverso da software e documentazione
    - Licenze per i tipi di carattere
    - Licenze per opere legate a un punto di vista (ad esempio, opinioni o testimonianze)
    - Licenze per la progettazione di oggetti fisici

12.1.1 Licenze per il software

12.1.1.1 Licenze di software libero compatibili con la GPL

Le seguenti licenze sono licenze per software libero, e sono compatibili con la GNU GPL.

- Licenza Pubblica Generica GNU (GNU General Public License, GNU GPL) versione 3

  Questa è l'ultima versione della GNU GPL: una licenza per software libero, una licenza con copyleft. Ne consigliamo l'uso per la maggior parte dei pacchetti software.

  Notate che la GNU GPL versione 3 (GPLv3) non è compatibile con la GNU GPL versione 2 (GPLv2). Tuttavia, la maggior parte del software rilasciato sotto GPLv2 vi permette di usare i termini di licenza anche di versioni successive della GPL; in questo caso, potete usare il codice scegliendo GPLv3 per effettuare la combinazione di codice desiderata. Per ulteriori informazioni sulla compatibilità tra licenze GNU, vedete le domande frequenti.

- Licenza Pubblica Generica GNU (GNU General Public License, GNU GPL) versione 2

  Questa è la versione precedente della GNU GPL: una licenza per software libero, una licenza con copyleft. Consigliamo l'ultima versione per la maggior parte dei pacchetti software.

  Notate che la GNU GPL versione 2 (GPLv2) non è compatibile con la GNU GPL versione 3 (GPLv3). Tuttavia, la maggior parte del software rilasciato sotto GPLv2 vi permette di usare i termini di licenza anche di versioni successive della GPL; in questo caso, potete usare il codice scegliendo la licenza GPLv3 per effettuare la combinazione di codice desiderata. Per ulteriori informazioni sulla compatibilità tra licenze GNU, vedete le domande frequenti.

- Licenza Pubblica Generica GNU Attenuata (GNU Lesser General Public License, GNU LGPL) versione 3

  Questa è l'ultima versione della LGPL: una licenza per software libero, ma non una licenza con copyleft forte, poiché permette il collegamento con moduli non liberi. È compatibile con la GPLv3. Ne consigliamo l'uso soltanto in circostanze particolari.

  Notate che la LGPLv3 non è compatibile con la GPLv2. Tuttavia, la maggior parte del software rilasciato sotto GPLv2 vi permette di usare i termini di licenza anche di versioni successive della GPL; in questo caso, potete usare il codice scegliendo la licenza GPLv3 per effettuare la combinazione di codice desiderata. Per ulteriori informazioni sulla compatibilità tra licenze GNU, vedete le domande frequenti.

- Licenza Pubblica Generica GNU Attenuata (GNU Lesser General Public License, GNU LGPL) versione 2.1

  Questa è la versione precedente della GNU GPL: una licenza per software libero, ma non una licenza con copyleft forte, poiché permette il collegamento con moduli non liberi. È compatibile con la GPLv2 e la GPLv3. In genere consigliamo l'uso dell'ultima versione della LGPL, tra l'altro solo in circostanze particolari. Per altre informazioni su come la LGPLv2.1 sia compatibile con altre licenze GNU, vedete le domande frequenti.

- Licenza Pubblica Generica GNU Affero (GNU Affero General Public License, AGPL) versione 3

  È una licenza di software libero e una licenza con copyleft. I termini d'uso sono effettivamente quelli della GPLv3 con un paragrafo aggiuntivo nella sezione 13, che permette agli utenti che interagiscono tramite una rete con il programma a cui si applica la licenza di ricevere il sorgente di quel programma. Consigliamo agli sviluppatori di considerare l'uso della GNU AGPL per qualsiasi software destinato ad essere comunemente utilizzato via rete.

Notate che la GNU AGPL non è compatibile con la GPLv2. Tecnicamente non è compatibile nemmeno con la GPLv3: non si può prendere codice rilasciato sotto GNU AGPL e distribuirlo o modificarlo alle condizioni della GPLv3 o viceversa. Tuttavia, è possibile combinare in un unico progetto moduli separati o file sorgente rilasciate sotto entrambe le licenze, il che darà a molti programmatori tutte le libertà di cui necessitano per realizzare il programma che desiderano. Si veda la sezione 13 di entrambe le licenze per ulteriori dettagli.

- Licenza GNU All-Permissive (GNU All-Permissive License)

  Questa è una licenza di software libero debole, che lascia molti permessi ed è compatibile con la GNU GPL; la consigliamo per i file README e altri piccoli file di supporto distribuiti con i programmi GNU. Tutti gli sviluppatori possono sentirsi liberi di utilizzarla in situazioni simili.

  Versioni precedenti di questa licenza non contenevano la seconda frase in cui si nega espressamente la garanzia. Questa analisi si applica ad entrambe le varianti.

- Licenza Apache, versione 2.0

  Questa è una licenza di software libero, compatibile con la versione 3 della GNU GPL.

  Notate che questa licenza non è compatibile con la versione 2 della GPL, perché ha requisiti che mancavano in quella versione della GPL: ad esempio, terminazioni dei brevetti e compensazioni. La protezione dai tranelli brevettuali è importante, e per questo motivo per programmi di una certa consistenza consigliamo la licenza Apache 2.0 anziché altre licenze permissive.

- Licenza Artistica 2.0

  Questa è una licenza di software libero, compatibile con la GPL grazie a una clausola nella sezione 4(c)(ii).

- Licenza Artistica Chiarita

  Questa è una licenza di software libero, compatibile con la GPL. Apporta le modifiche strettamente necessarie per correggere i punti della Artistic License 1.0 che non sono sufficientemente chiari.

- Licenza del Berkeley Database (nota anche come Licenza di Sleepycat Software

  Questa è una licenza di software libero, compatibile con la GNU GPL.

- Licenza Software Boost

  Questa è una licenza di software libero, debole e permissiva, senza copyleft, compatibile con la GNU GPL.

- Licenza BSD modificata

  Questa è la licenza BSD originale, modificata rimuovendo la clausola pubblicitaria. È una licenza per software libero debole e permissiva, senza copyleft, compatibile con la GNU GPL.

  Talvolta questa licenza è denominata "la licenza BSD con 3 clausole".

  La licenza BSD modificata è una buona scelta nell'ambito delle licenze deboli e permissive, anche se la licenza Apache 2.0 è preferibile. Ad ogni modo, è rischioso raccomandare l'uso della "licenza BSD", anche per casi speciali come programmi molto brevi, per una possibile confusione che può portare all'uso della problematica licenza BSD originale. Per evitare questo rischio suggerite la licenza X11. La licenza X11 e la BSD modificata sono più o meno equivalenti.

  Comunque la licenza Apache 2.0 è migliore per programmi di una certa consistenza, perché previene tranelli brevettuali.

- Licenza CC0

  CC0 è una donazione in pubblico dominio per Creative Commons. Un'opera rilasciata sotto CC0 è di pubblico dominio per l'interpretazione più ampia possibile che la legge permette di questo termine.

  Se per qualche motivo questo non è possibile, CC0 permette in subordine di ripiegare su una licenza debole e permissiva. Sono compatibili con la GNU GPL entrambe le licenze comprese in CC0.

  Se volete che il vostro lavoro sia di dominio pubblico, vi consigliamo di usare la CC0.

- Licenza CeCILL versione 2

  La CeCILL è una licenza di software libero, esplicitamente compatibile con la GNU GPL.

  Il testo della CeCILL usa alcuni termini che dovrebbero essere evitati in quanto faziosi: “proprietà intellettuale” e “protezione”; usare questi termini non è stata una buona decisione, perché la lettura della licenza tende a diffondere l'utilizzo di tali termini. Tuttavia, questo non causa alcun problema ai programmi rilasciati sotto licenza CeCILL.

  La sezione 9.4 della CeCILL obbliga gli sviluppatori del programma ad alcune forme di cooperazione con gli utenti, se qualcuno minaccia il programma con brevetti. Questo potrebbe essere un problema per lo sviluppatore; tuttavia, se siete sicuri che aiutereste comunque gli utenti come ivi stabilito, questo per voi non è un problema.

- Licenza BSD Chiara (Clear BSD)

  Questa è una licenza di software libero, compatibile sia con la GPLv2 sia con la GPLv3. E' basata sulla licenza BSD modificata, e aggiunge una condizione che chiarisce esplicitamente che non viene concesso l'utilizzo di alcun brevetto. Perciò siate cauti quando usate software sotto questa licenza; dovreste prima considerare se chi detiene il brevetto possa poi volere agire contro di voi per violazione di brevetto. Se lo sviluppatore nega responsabilità sui brevetti in modo da prepararvi una trappola, sarebbe bene evitare il programma.

- Licenza generica Cryptix

  Questa è una licenza per software libero debole e permissiva, senza copyleft, compatibile con la GNU GPL. È molto simile alla licenza X11.

- Licenza eCos, versione 2.0

  La licenza eCos, versione 2.0, è una licenza di software libero compatibile con la GPL. Consiste nella GPL con l'aggiunta di un'eccezione che permette di collegare software non coperto dalla GPL. Ha gli stessi svantaggi della LGPL.

- Licenza Educational Community 2.0

  Questa è una licenza di software libero, compatibile con la GPLv3. E' basata sulla Licenza Apache 2.0; il campo di applicazione della concessione del brevetto viene modificato stabilendo che quando il dipendente di un'organizzazione lavora su un progetto, l'organizzazione non deve concedere tutti i propri brevetti a chi riceve il programma. Questa disposizione sui brevetti e la clausola di indennizzo della sezione 9 rendono questa licenza incompatibile con la GPLv2.

- Licenza Eiffel Forum, versione 2

  Questa è una licenza di software libero, compatibile con la GNU GPL. Le versioni precedenti della licenza Eiffel non sono compatibili con la GPL.

- Licenza software EU DataGrid

Questa è una licenza di software libero, debole e permissiva, senza copyleft, compatibile con la GNU GPL.

- Licenza di Expat

  Questa è una licenza per software libero debole e permissiva, senza copyleft, compatibile con la GNU GPL. A volte viene ambiguamente chiamata con il nome di Licenza MIT.

  Per programmi di una certa consistenza è meglio usare la licenza Apache 2.0, perché previene tranelli brevettuali.

- Licenza di FreeBSD

  Questa è la licenza BSD originale, a cui sono state tolte la clausola di pubblicità e un'altra; a volte è chiamata la “licenza BSD con 2 clausole”. E' una licenza per software libero debole e permissiva, senza copyleft, compatibile con la GNU GPL.

  I nostri commenti sulla Licenza BSD modificata si applicano anche a questa licenza.

- Licenza Freetype Project

  Questa è una licenza di software libero, compatibile con la GPLv3. A causa di imposizioni sul riconoscimento degli autori è incompatibile con la GPLv2.

- Licenza Historical Permission Notice and Disclaimer

  Questa è una licenza di software libero debole e permissiva che è compatibile con la GPL. Il caso è simile a quello di Python 1.6a2 e versioni precedenti.

- Licenza della iMatix Standard Function Library

  Questa è una licenza per software libero ed è compatibile con la licenza GPL.

- Licenza di imlib2

  Questa è una licenza di software libero, compatibile con la GNU GPL. L'autore ci ha spiegato che tutte le possibilità previste dalla GPL per fornire i sorgenti sono valide per la definizione di "reso disponibile pubblicamente" usata in questa licenza.

- Licenza di Independent JPEG Group

  Questa è una licenza di software libero, compatibile con la GNU GPL. Gli autori ci hanno assicurato che gli sviluppatori che documentano le proprie modifiche in maniera conforme alla GPL rispettano anche il requisito, simile, previsto da questa licenza.

- Licenza informale

  Una “*licenza informale*” è una semplice frase come “*Fate quel che volete di questo*” o “*Siete liberi di ridistribuire e modificare questo codice*”.

  Negli Stati Uniti, queste licenze vengono interpretate sulla base di quello che sembra che l'autore intendesse, quindi in pratica il loro significato è quello che sembrano avere alla lettura. Quindi sarebbero licenze di software libero, senza copyleft, compatibili con la GNU GPL. Ma una scelta poco felice delle parole potrebbe dare significati diversi.

  Tra l'altro, molti altri paesi hanno un approccio più rigido alle licenze basate sul copyright, e non si può sapere cosa deciderebbe un tribunale in quei paesi davanti a una licenza informale. Potrebbe anche decidere che non è proprio una licenza valida.

  Se volete che il vostro codice sia libero, non causate danni inutili ai vostri utenti: scegliete ed applicate una licenza libera tra quelle più diffuse. Abbiamo una pagina con consigli per la scelta di una licenza.

- Licenza Intel Open Source

  Questa è una licenza di software libero, compatibile con la GNU GPL.

- Licenza ISC

  Questa è una licenza di software libero, debole e permissiva, compatibile con la GNU GPL.

  Questa licenza è a volte nota anche come licenza di OpenBSD, anche se c'è una piccola differenza tra le due. La licenza di OpenBSD è stata aggiornata per rimuovere l'ambiguità del termine "and/or" ("e/o") che conteneva. La licenza ISC dice "Permission to use, copy, modify, and/or distribute this software…", mentre la licenza OpenBSD dice "Permission to use, copy, modify, and distribute…".

  Quando la licenza ISC fu pubblicata, l'uso di "and/or" destava preoccupazione perché una simile clausola usata nella licenza di Pine era stata usata dall'Università di Washington per sostenere che non si potevano distribuire versioni modificate del software. Tuttavia ISC ci ha detto che non condivide l'interpretazione dell'Università di Washington, e abbiamo tutti i motivi per fidarci. Quindi non c'è ragione di evitare il software rilasciato sotto questa licenza. Comunque, per evitare che queste scelte lessicali causino problemi in futuro, preferiamo che gli sviluppatori scelgano una licenza diversa per il loro lavoro. La licenza Expat License e la licenza FreeBSD sono brevi e garantiscono permessi molto simili.

- Licenza Pubblica Mozilla (MPL), versione 2.0

  Questa è una licenza di software libero. La sezione 3.3 fornisce un'indiretta compatibilità tra questa licenza e la GNU GPL 2.0, la GNU LGPL 2.1, la GNU AGPL 3 e le loro successive versioni. Quando ricevete un'opera sotto licenza MPL 2.0, potete produrre un'"opera più ampia" che combina quell'opera con materiale sotto le licenze GNU elencate. Quando lo fate, la sezione 3.3 vi permette di distribuire il lavoro coperto da MPL ai termini e alle condizioni delle medesime licenze GNU a una condizione: dovete fare in modo che i file originariamente sotto licenza MPL siano ancora disponibili (anche) con licenza MPL. In altri termini: quando create una combinazione in questo modo, i file che erano originariamente sotto MPL avranno una doppia licenza, la MPL e la licenza (o le licenze) GNU in questione. Il risultato finale sarà che l'"opera più ampia" nel suo complesso sarà coperta dalla licenza GNU. Chi riceve da voi l'opera combinata potrà scegliere se usare i file che originariamente erano sotto MPL secondo i termini di quella licenza oppure distribuire l'"opera più ampia" in tutto o in parte secondo i termini della licenza (o licenze) GNU, senza ulteriori restrizioni.

  È importante capire che la condizione di distribuire i file sotto le condizioni della MPL vale solo per chi crea e distribuisce per primo l'"opera più ampia". Se applicata anche a chi lo riceve, diventerebbe una restrizione incompatibile con le licenze GNU. Ciò detto, quando si fanno contributi a un progetto esistente è consigliabile usare la stessa licenza anche per le modifiche, anche quando non obbligatorio. Se ricevete sotto licenza GNU un lavoro in cui alcuni file sono anche sotto MPL, dovreste togliere la MPL da tali file solo se ci sono motivazioni molto forti per farlo.

  Controllate i dettagli nelle indicazioni di licenza del software sotto MPL prima di creare un'"*opera più ampia*" in questo modo. Chi rilascia un'opera sotto licenza MPL 2.0 può scegliere di impedire la compatibilità aggiungendo un'indicazione che dica che l'opera è "Incompatibile con Licenze Secondarie". Il software che riporta questa indicazione non è compatibile con la GPL o la AGPL.

  Il software rilasciato sotto versioni precedenti della MPL può essere aggiornato alla versione 2.0, ma se non è già disponibile sotto una delle licenze GNU elencate allora deve essere marcato come "Incompatibile con Licenze Secondarie". Questo comporta che il software disponibile solo sotto versioni precedenti della MPL rimane incompatibile con la GPL e la AGPL.

- Licenza Open Source di NCSA/University of Illinois

  Questa licenza è basata sulle condizioni delle licenze Expat e BSD modificata. E' una licenza di software libero debole e permissiva, senza copyleft, compatibile con la GNU GPL.

- Licenza di Netscape Javascript

  Questa è una unione disgiunta della Licenza Pubblica Netscape e della GNU GPL. Per questo motivo è una licenza per software libero, compatibile con la GNU GPL, ma senza un forte copyleft.

- Licenza di OpenLDAP, versione 2.7

  Questa è una licenza di software libero permissiva, senza copyleft, compatibile con la GNU GPL.

- Licenza di Perl 5 e precedenti

  Questa licenza è l'unione disgiunta della Licenza Artistica 1.0 e della GNU GPL; in altre parole, è possibile scegliere tra una delle due licenze. Si qualifica come licenza per software libero, ma può non essere un effettivo copyleft. È compatibile con la GNU GPL poiché la GNU GPL è una delle possibili alternative.

  Vi raccomandiamo di utilizzare questa licenza per qualunque pacchetto Perl 4 o Perl 5 che desiderate scrivete, per promuovere la coerenza e l'uniformità nella programmazione in Perl. Al di fuori di questo, vi chiediamo di non utilizzare questa licenza; è meglio utilizzare solo la GNU GPL.

- Licenza Pubblico Dominio (Public Domain)

  Essere di dominio pubblico non è una licenza: piuttosto, significa che il materiale non è protetto da copyright e non è necessaria nessuna licenza. In pratica quindi se un'opera è di dominio pubblico, potrebbe anche avere una licenza come software libero del tutto permissiva senza copyleft. Lo stato di dominio pubblico è compatibile con la GNU GPL.

  Se volete rilasciare il vostro lavoro come "pubblico dominio", vi consigliamo di usare strumenti formali per farlo. Chiediamo a chi fa piccoli contributi a GNU di firmare un foglio di assunzione responsabilità: questa è una possibile soluzione. Se state lavorando a un progetto che non ha politiche formali sui contributi ricevuti, CC0 è un buono strumento, utilizzabile da tutti. Dona in modo formale il vostro lavoro al pubblico dominio e comprende in subordine una licenza permissiva da usare nei casi in cui il pubblico dominio è impossibile.

- Licenza di Python 2.0.1, 2.1.1 e versioni successive

  Questa è una licenza per software libero ed è compatibile con la GNU GPL. Si noti che le versioni intermedie di Python (dalla 1.6b1 alle 2.0 e 2.1) sono rilasciate sotto una diversa licenza (si veda sotto)

- Licenza di Python 1.6a2 e versioni precedenti

  Questa è una licenza per software libero ed è compatibile con la GNU GPL. Si noti che le versioni più recenti di Python sono rilasciate sotto altre licenze (si veda sopra e sotto).

- Licenza di Ruby

  Questa è una licenza di software libero, compatibile con la GNU GPL tramite un'esplicita clausola di doppia licenza.

- SGI Free Software License B, versione 2.0

  La SGI Free Software License B versione 2.0 è una licenza di software libero. E' essenzialmente uguale alla licenza X11, con una possibilità alternativa di fornire il testo della licenza.

  Le versioni precedenti della SGI Free Software License B non erano, nonostante il nome, licenze di software libero. Tuttavia, tutte includevano clausole che permettevano all'utente di passare a sua scelta a versioni più recenti della licenza. Quindi se un programma è stato rilasciato sotto qualsiasi versione della SGI Free License B lo si può utilizzare secondo i termini di questa versione libera.

- Standard ML of New Jersey Copyright License

  Questa è una licenza di software libero, debole e permissiva, senza copyleft, compatibile con la GNU GPL.

- Unicode, Inc. License Agreement for Data Files and Software

  Questa è una licenza che Unicode, Inc. ha applicato al suo Unicode Character Database: vari file di dati che gli sviluppatori possono usare per semplificare l'implementazione dello standard Unicode nei propri programmi. È una licenza debole e permissiva, compatibile con tutte le versioni della GPL.

  Se volete utilizzare file coperti da questo accordo di licenza nel vostro software, non dovrebbero esserci problemi, ma consigliamo di includere anche una copia completa dell'accordo. Alcuni dei file contengono licenze alternative che non sono libere, o nessuna informazione sulla licenza, quindi inserire una copia dell'accordo di licenza aiuterà ad evitare la confusione se altri dovessero ridistribuire

  il vostro software. Ovviamente, sarete obbligati a seguire le condizioni dell'accordo di licenza quando distribuirete i file, ma si tratta di condizioni molto semplici.

  Assicuratevi che i file che utilizzate siano coperti proprio da questo accordo di licenza. Altri file pubblicati da Unicode, Inc. sono coperti dagli "Unicode Terms of Use", una licenza diversa, non libera, che appare sulla stessa pagina ma copre file diversi. Una breve spiegazione all'inizio dell'accordo di licenza spiega a quali file esso si applica.

  Vi sconsigliamo di utilizzare questo accordo di licenza per il software che scrivete. Se volete usare una licenza debole e permissiva, considerate la licenza di Expat per programmi brevi e la licenza Apache 2.0 per programmi consistenti. Queste due licenze sono molto più comuni e hanno riconoscimento più ampio nella comunità del software libero.

- Universal Permissive License (UPL)

  Questa è una licenza debole permissiva senza copyleft, compatibile con la GNU GPL. La licenza dà la possibilità di coprire i brevetti oltre che il software, ma continuiamo a consigliare la licenza Apache 2.0 per prevenire tranelli brevettuali quando si scegli di usare uan licenza permissiva.

- Unlicense

  La Unlicense è una donazione in pubblico dominio. Un lavoro rilasciato sotto la Unlicense diventa di dominio pubblico nella più ampia accezione del termine permessa dalla legge, e ha inoltre una licenza debole e permissiva aggiuntiva che copre i casi in cui la donazione è problematica. Entrambe le licenze comprese in Unlicense sono compatibili con la GNU GPL.

  Se volete rilasciare il vostro lavoro come "pubblico dominio", vi consigliamo di usare CC0. Anche CC0 dona il vostro lavoro al pubblico dominio e comprende in subordine una licenza permissiva, ed è inoltre meglio studiato e più maturo di Unlicense.

- Licenza di Vim, versione 6.1 o successiva

  Questa è una licenza per software libero, con copyleft parziale, non completo. E' compatibile con la GPL grazie ad un'esplicita clausola di conversione.

- W3C Software Notice and License

  Questa è una licenza per software libero ed è compatibile con la licenza GPL.

- Licenza di WebM

  L'implementazione di WebM di Google è coperta dalla Licenza BSD modificata. Google fornisce inoltre una licenza di brevetti separata (che impropriamente chiama “Additional IP Rights Grant”) per

brevetti che Google possiede o controlla e che sono violati dalla loro implementazione di WebM. Il software GPL può essere distribuito rispettando i termini di questa licenza: permette ai distributori di esercitare tutti i diritti garantiti dalla GPL, e ne soddisfa tutte le condizioni. In conclusione, la licenza completa di WebM è libera e compatibile con la GPL.

- Licenza WTFPL, versione 2

  Questa è una licenza di software libero, debole e permissiva, senza copyleft, compatibile con la GNU GPL.

  Sconsigliamo questa licenza. Se volete una licenza permissiva per un piccolo programma, consigliamo la licenza X11. Per un programma più grande suggeriremmo una licenza copyleft o, se desiderate usare una licenza permissiva, consigliamo la licenza Apache 2.0 perché protegge gli utenti da tranelli brevettuali.

- Licenza WxWidgets Library

  La licenza di WxWidgets è una licenza di software libero compatibile con la GPL. È formata dalla LGPL

  2.0 o successiva, con un permesso aggiuntivo che permette che distribuzioni binarie che usano la libreria possano avere qualsiasi licenza (anche proprietaria) scelta da chi effettua la distribuzione. È una licenza con copyleft debole, ancora più della LGPL, consigliata in casi particolari.

- Licenza WxWindows Library

  Un vecchio nome della WxWidgets Library license.

- Licenza di X11

  Questa è una licenza per software libero debole e permissiva, senza copyleft, compatibile con la GNU GPL. Le vecchie versioni di XFree86 utilizzavano la stessa licenza, e così pure alcune varianti attuali di XFree86. Le versioni successive di XFree86, comunque, sono distribuite sotto la licenza XFree86 1.1.

  Questa licenza è a volte chiamata licenza MIT, ma questo termine è improprio, perché il MIT ha usato molte diverse licenze per il proprio software.

  Questa licenza va bene per un piccolo programma. Per un programma più grande suggeriremmo una licenza copyleft o, se desiderate usare una licenza permissiva, consigliamo la licenza Apache 2.0 perché protegge gli utenti da tranelli brevettuali.

- Licenza XFree86 1.1

  Questa è una licenza di software libero debole e permissiva, senza copyleft, compatibile con la versione 3 della GPL.

  Notate che questa licenza è incompatibile con la versione 2 della GPL, a causa di requisiti imposti a tutta la documentazione distribuita che contenga crediti.

  Ci sono, al momento, diverse varianti di XFree86, e solo alcune di esse usano questa licenza. Alcune continuano a usare la licenza di X11.

- Licenza di ZLib

  Questa è una licenza per software libero ed è compatibile con la licenza GPL.

- Licenza Zope Public, versioni 2.0 e 2.1

  Questa è una licenza di software libero debole e permissiva, senza copyleft, compatibile con la GNU GPL.

12.1.1.2 Licenze di software libero incompatibili con la GPL

Le seguenti licenze sono licenze per software libero, ma non sono compatibili con la GNU GPL.

- Licenza Generica Pubblica Affero, versione 1

  La Licenza Generica Pubblica Affero è una licenza di software libero, con copyleft, incompatibile con la GNU GPL. Consiste nella versione 2 della GNU GPL con l'aggiunta di una sezione inserita da Affero dietro approvazione della FSF. La nuova sezione, 2(d), copre la distribuzione di applicazioni che operano tramite servizi web o reti di computer.

  Questa licenza è stata seguita dalla Licenza Pubblica Generica GNU Affero, versione 3; vi consigliamo di usare quest'ultima.

- Academic Free License, tutte le versioni fino alla 3.0

  La Academic Free License è una licenza di software libero, senza copyleft, incompatibile con la GNU GPL. Le ultime versioni contengono clausole contrattuali simili alla Open Software License, e dovrebbero essere evitate per le stesse ragioni.

- Licenza Apache, versione 1.1

  Questa è una licenza di software libero permissiva, senza copyleft. Ha alcuni requisiti che la rendono incompatibile con la GNU GPL, come forti restrizioni all'uso di nomi legati ad Apache.

- Licenza Apache, versione 1.0

  Questa è una licenza di software libero debole e permissiva, senza copyleft con una clausola di pubblicità. Questa clausola crea problemi pratici come quelli della licenza BSD originale, compresa l'incompatibilità con la GNU GPL.

- Licenza Apple Public Source (APSL), versione 2

  Questa è una licenza di software libero, incompatibile con la GNU GPL. Consigliamo di non usarla per nuovo software scritto da voi, ma è ammissibile utilizzare e migliorare il software rilasciato sotto questa licenza. Ulteriori dettagli.

- Licenza Open Source di BitTorrent

  Questa è una licenza di software libero, ma è incompatibile con la GPL, per gli stessi motivi per cui la è la licenza Open Source di Jabber.

- Licenza BSD originale

  Talvolta questa licenza è denominata "la licenza BSD con 4 clausole".

  Questa è una licenza per software libero debole e permissiva, senza copyleft con un problema serio: la "fastidiosa clausola pubblicitaria BSD". Il problema non è determinante: non rende il software non libero. Ma causa molti problemi pratici, inclusa l'incompatibilità con la GNU GPL.

  Vi chiediamo di non utilizzare la licenza BSD originale per il vostro software. Se desiderate una licenza per software libero debole e permissiva, senza copyleft vi consigliamo l'uso della licenza BSD modificata della licenza X11 o della licenza Expat. E per programmi significativi è meglio utilizzare la licenza Apache 2.0 che protegge da tranelli brevettuali.

  Ad ogni modo, non sussiste una valida ragione per evitare di utilizzare programmi rilasciati sotto la licenza BSD originale.

- Licenza CeCILL-B versione 1

La licenza CeCILL-B + una licenza di software libero, incompatibile con la GPL perché ha dei requisiti che non sono presenti nella GPL. I riconoscimenti richiesti nella sezione 5.3.4 vanno al di là di quanto previsto dalla GPL. Inoltre, è presente una strana richiesta che si faccia “un ragionevole tentativo” di convincere le terze parti a “rispettare gli obblighi previsti da questo articolo”.

Vi sconsigliamo di usare questa licenza per il vostro software.

- Licenza CeCILL-C versione 1

Questa è una licenza per software libero, con debole copyleft, vagamente simile alla LGPL. È incompatibile con la GNU GPL perché non contiene la clausola esplicita presente nella basic CeCILL.

Vi sconsigliamo di usare questa licenza per il vostro software.

Common Development and Distribution License (CDDL), versione 1.0 (#CDDL)

Questa è una licenza di software libero. Ha un copyleft debole, a livello di file, come la versione 1 della Mozilla Public License, che la rende incompatibile con la GNU GPL. Ciò significa che un modulo coperto dalla GPL e un modulo coperto dalla CDDL non possono essere collegati tra loro. Per questo motivo vi chiediamo di non usare la CDDL.

Per un esempio che illustra perché non si dovrebbero combinare opere rilasciate con licenza CDDL e opere rilasciate con licenza GPL si veda questa dichiarazione della FSF: Interpretare, fare valere e cambiare la GNU GPL: un esempio dalla combinazione di Linux e ZFS.

Da notare, nella CDDL, anche l'impropria scelta di utilizzare il termine “proprietà intellettuale”.

- Common Public Attribution License 1.0 (CPAL)

Questa è una licenza di software libero. È basata sulla Mozilla Public License versione 1, ed è incompatibile con la GPL per gli stessi motivi: ha vari requisiti sulle versioni modificate che nella GPL non esistono. Vi chiede, inoltre, di pubblicare i sorgenti del programma se permettete ad altri di usarlo.

- Licenza Common Public versione 1.0

Questa è una licenza di software libero. Purtroppo, il suo copyleft debole e la scelta della legge applicabile la rendono incompatibile con la GNU GPL.

- Licenza Condor Public

Le ultime versioni di Condor (dalla 6.9.5 in poi) sono distribuite sotto Apache License 2.0. Solo le vecchie versioni di Condor usano questa licenza.

La Condor Public License è una licenza di software libero. Ha un paio di requisiti che la rendono incompatibile con la GNU GPL, tra cui forti restrizioni all'utilizzo di nomi legati a Condor e impone ai redistributori di garantire che rispetteranno le leggi statunitensi sull'esportazione. (Se il rispetto delle leggi statunitensi sull'esportazione fosse una effettiva condizione della licenza, la licenza non sarebbe libera).

- Licenza pubblica Eclipse, versione 1.0

La Licenza Pubblica Eclipse è simile alla Licenza pubblica comune e i nostri commenti sulla CPL si applicano nella stessa misura in questo caso. L'unica differenza è che la EPL elimina le frasi sulle azioni legali per violazioni di brevetti nei confronti di chi contribuisce al programma rilasciato sotto EPL.

- Eclipse Public License Version 2.0

In termini di compatibilità con la GPL, la Eclipse Public License versione 2.0 è essenzialmente equivalente alla versione 1.0. L'unica modifica è che offre esplicitamente la possibilità di usare la GNU GPL versione 2 o successiva come “licenza secondaria” per una certa porzione di codice.

Se un contributore iniziale rilascia del codice e indica la GNU GPL versione 2 o successiva come licenza secondaria, ciò garantisce compatibilità esplicita con quelle versioni della GPL per quel codice. Farlo è più o meno equivalente, per gli utenti, a rilasciare quel codice con doppia licenza EPL o GPL. Comunque, la EPL2 senza questa indicazione rimane incompatibile con la GPL.

- Licenza Pubblica dell'Unione Europea (EUPL) versione 1.1

  Questa è una licenza di software libero. Di per sé, ha un copyleft paragonabile a quello della GPL. Tuttavia, permette ai licenziatari di distribuire l'opera alle condizioni previste da altre licenze, alcune delle quali (in particolare, la licenza pubblica Eclipse e la Licenza Pubblica Comune) prevedono solo un copyleft debole. Quindi gli sviluppatori non possono fare affidamento su questa licenza se desiderano un copyleft forte.

  La EUPL permette di cambiare la licenza in GPLv2, poiché questa è elencata tra le varie licenze alternative a cui gli utenti possono convertire il programma. Indirettamente, è possibile anche cambiare la licenza in GPLv3, perché è possibile cambiare la licenza alla CeCILL v2, e la CeCILL v2 dà la possibilità di cambiare ulteriormente licenza in qualsiasi versione della GNU GPL.

  Per questa variazione di licenza in due passi, dovrete prima scrivere una porzione di codice da rilasciare sotto licenza CeCILL v2, o trovare un modulo adatto già disponibile a quelle condizioni, e aggiungerlo al programma. A questo punto, secondo le condizioni della EUPL, l'aggunta di quel modulo vi dà la possibilità di convertire l'intero programma a CeCILL v2. Poi dovete scrivere un'ulteriore porzione di codice da rilasciare sotto licenza GPLv3+, o trovare un modulo adatto già disponibile a quelle condizioni, e aggiungerlo al programma. A questo punto, secondo le condizioni della CeCILL, l'aggunta di quel modulo vi dà la possibilità di convertire l'intero programma a GPLv3+.

- Licenza Pubblica dell'Unione Europea (EUPL) versione 1.2

  Questa è una licenza di software libero. Di per sé, ha un copyleft paragonabile a quello della GPL. Tuttavia, permette ai licenziatari di distribuire l'opera alle condizioni previste da altre licenze, alcune delle quali (in particolare, la licenza pubblica Eclipse) prevedono solo un copyleft debole. Quindi gli sviluppatori non possono fare affidamento su questa licenza se desiderano un copyleft forte.

  La EUPL permette di cambiare la licenza solo in GPLv2 o in GPLv3, poiché queste sono elencate tra le varie licenze alternative a cui gli utenti possono convertire il programma. Indirettamente, è possibile anche cambiare la licenza in GPLv3 o successive, perché è possibile cambiare la licenza alla CeCILL v2, e la CeCILL v2 dà la possibilità di cambiare ulteriormente licenza in qualsiasi versione della GNU GPL.

  Per questa variazione di licenza in due passi, dovrete prima scrivere una porzione di codice da rilasciare sotto licenza CeCILL v2, o trovare un modulo adatto già disponibile a quelle condizioni, e aggiungerlo al programma. A questo punto, secondo le condizioni della EUPL, l'aggunta di quel modulo vi dà la possibilità di convertire l'intero programma a CeCILL v2. Poi dovete scrivere un'ulteriore porzione di codice da rilasciare sotto licenza GPLv3+, o trovare un modulo adatto già disponibile a quelle condizioni, e aggiungerlo al programma. A questo punto, secondo le condizioni della CeCILL, l'aggunta di quel modulo vi dà la possibilità di convertire l'intero programma a GPLv3+.

- Licenza Fraunhofer FDK ACC

  Questa è una licenza software gratuita per quanto possibile. È incompatibile con qualsiasi versione della GNU GPL

  Ha un pericolo particolare sotto forma di un termine che afferma espressamente che non si concede alcuna licenza di brevetto, con un invito ad acquistarne una. Per questo motivo, e poiché l'autore della licenza è un noto "sottrattore" di brevetti, ti invitiamo a fare attenzione nell'usare o ridistribuire il software ai sensi di questa licenza: dovresti prima considerare se il licenziante possa mirare ad attirarti in una violazione di brevetto. Se si conclude che il programma è un'esca per una trappola per brevetti, sarebbe saggio evitare il programma.

È possibile che i relativi brevetti siano scaduti. A seconda che Fraunhofer abbia ancora brevetti attivi che coprono il lavoro, il software potrebbe essere una trappola ora o meno. (Naturalmente, qualsiasi programma è potenzialmente minacciato dai brevetti e l'unico modo per porre fine a ciò è modificare la legge sui brevetti per rendere il software protetto dai brevetti.)

- Licenza di Gnuplot

  Questa è una licenza di software libero, incompatibile con la GNU GPL.

- Licenza IBM Public, versione 1.0

  Questa è una licenza per software libero ma è sfortunatamente incompatibile con la GNU GPL per una clausola sulla scelta dell'ordinamento legale che si applica.

- Licenza Jabber Open Source, versione 1.0

  La licenza è per software libero, incompatibile con la GPL. Permette il re-licenziamento all'interno di una specifica classe di licenze, quelle che soddisfano tutti i requisiti della licenza di Jabber. La GPL non fa parte di questa classe di licenze, quindi la licenza di Jabber non permette il re-licenziamento sotto GPL. Perciò non è compatibile.

- Licenza pubblica del Progetto LaTeX 1.3a

  Non abbiamo ancora scritto un'analisi dettagliata specifica, ma questa è una licenza di software libero, con richieste meno stringenti sulla distribuzione di quanto previsto dalla LPPL 1.2 di seguito descritta. Rimane incompatibile con la GPL perché alcune versioni modificate devono includere una copia o un collegamento alla versione originale.

- Licenza pubblica del Progetto LaTeX 1.2

  Questa licenza è una parte dei termini di distribuzione per LaTeX. Per il suo stato attuale è una licenza per software libero, ma incompatibile con la GPL poiché ha molti requisiti che non si trovano nella GPL.

  Questa licenza contiene restrizioni complesse e fastidiose su come pubblicare una versione modificata, inclusa una richiesta che rientra appena in ciò che si può accettare: che qualunque file modificato deve avere un nuovo nome.

  Il motivo di questa richiesta è accettabile per LaTeX poiché LaTeX ha la caratteristica di poter tracciare i nomi dei file, per specificare opzioni tipo "*utilizza il file pluto quando viene richiesto il file paperino*". Con questa caratteristica, la richiesta è decisamente fastidiosa; senza, la stessa richiesta diventerebbe un ostacolo molto serio, e quindi siamo arrivati alla conclusione che questa licenza rende il programma non libero.

  Questa condizione può causare problemi con modifiche sostanziali. Per esempio, se si vuole portare software coperto da LPPL su un sistema in cui non esiste il tracciamento dei nomi dei file, ma che richiedesse sempre di richiedere il file per nome, si dovrebbe implementare una funzione di tracciamento dei nomi dei file per mantenere libero il software. Questo sarebbe un fastidio, ma il fatto che una licenza possa rendere non libero un programma se trapiantato in un contesto molto diverso non lo rende non libero nel contesto originario.

  La LPPL dice che alcuni file, in determinate versioni di LaTeX, possono avere ulteriori restrizioni, che potrebbero renderli non liberi. Per questa ragione, bisogna stare molto attenti nel produrre una versione libera di LaTeX.

  La LPPL fa l'affermazione controversa che semplicemente il fatto di avere dei file in una macchina sulla quale altre persone possono fare login e accedervi viene considerata una forma di distribuzione. Crediamo che i tribunali non sosterranno questa rivendicazione, ma è un brutto segno che si inizi a farla.

Per favore non utilizzate questa licenza per nessun altro progetto.

Nota: Questi commenti si riferiscono alla versione 1.2 (3 settembre 1999) della LPPL.

- Lucent Public License Version 1.02 (Plan 9 license)

  Questa è una licenza di software libero, ma è incompatibile con la GNU GPL per la scelta della legge di riferimento. Non la consigliamo per nuovo software scritto da voi, ma è accettabile usare e migliorare Plan 9 sotto questa licenza.

- Microsoft Public License (Ms-PL)

  Questa è una licenza di software libero; ha un copyleft che non è forte, ma è incompatibile con la GNU GPL. Per questo motivo sconsigliamo l'utilizzo della Ms-PL.

- Microsoft Reciprocal License (Ms-RL)

  Questa è una licenza di software libero. E' basata sulla Microsoft Public License con una clausola aggiuntiva che rende il copyleft un po' più forte. Rimane incompatibile con la GNU GPL. Per questo motivo sconsigliamo l'utilizzo della Ms-RL.

- Licenza Pubblica Mozilla (MPL), versione 1.1

  Questa è una licenza per software libero che non ha un forte copyleft; diversamente dalla licenza X11, ha alcune restrizioni complesse che la rendono incompatibile con la GNU GPL. Per questo, un modulo rilasciato sotto licenza GPL ed uno rilasciato sotto MPL non possono essere legalmente uniti assieme. Vi invitiamo a non utilizzare la MPL 1.1 per questo motivo.

  Ad ogni modo, la MPL versione 1.1 ha una clausola (sezione 13) che permette ad un programma (o ad una sua parte) di offrire la possibilità di scegliere anche un'altra licenza. Se parte di un programma permette la GNU GPL come scelta alternativa, o qualunque altra licenza compatibile con la GPL, quella parte di programma ha una licenza compatibile con la GPL.

  La versione 2.0 della MPL presenta numerosi miglioramenti, tra cui la compatibilità con la GPL nell'utilizzo standard. Per dettagli, vedere la sezione dedicata.

- Licenza Open Source di Netizen (NOSL), versione 1.0

  Questa è una licenza per software libero che è essenzialmente uguale alla Licenza Pubblica Mozilla versione 1.1. Come la MPL, la NOSL ha alcune restrizioni complesse che la rendono incompatibile con

  la GNU GPL. Cioè, un modulo rilasciato con licenza GPL e uno con licenza NOSL non possono essere legalmente uniti assieme. Vi invitiamo a non usare la NOSL per questo motivo.

- Licenza Pubblica Netscape (NPL), versioni 1.0 e 1.1

  Questa è una licenza per software libero, con un debole copyleft, incompatibile con la GNU GPL. Consiste nella Licenza Pubblica Mozilla versione 1.1 con l'aggiunta di una clausola che permette a Netscape di utilizzare il codice che voi aggiungete anche nelle loro versioni proprietarie del programma. Ovviamente, loro non permettono a voi di utilizzare il loro codice in modo analogo. Vi invitiamo a non utilizzare la NPL.

- Licenza Nokia Open Source

  Questa licenza è simile alla Licenza Pubblica Mozilla versione 1: una licenza per software libero incompatibile con la GNU GPL.

- Vecchia licenza di OpenLDAP, versione 2.3

Questa è una licenza per software libero permissiva, senza copyleft che include alcune richieste (nelle sezioni 4 e 5) che la rendono incompatibile con la GNU GPL. Si noti che le ultime versioni di OpenLDAP hanno una nuova licenza compatibile con la GNU GPL.

Vi invitiamo a non utilizzare la licenza OpenLDAP per il vostro software. Ad ogni modo, non sussiste una valida ragione per evitare di utilizzare programmi rilasciati sotto questa licenza.

- Open Software License, tutte le versioni fino alla 3.0

  La Open Software License è una licenza di software libero. È incompatibile con la GNU GPL in vari modi.

  Versioni recenti della Open Software License chiedono ai distributori di tentare di ottenere un assenso esplicito alla licenza. Quindi distribuire software OSL su normali siti FTP, inviare patch o memorizzare il software in un comune sistema di controllo di versione sarebbe una violazione della licenza e ne comporterebbe la terminazione. Quindi la Open Software License rende molto difficile sviluppare software con gli strumenti comunemente utilizzati per il software libero. Per questo motivo, e per l'incompatibilità con la GPL, sconsigliamo l'utilizzo della OSL per qualsiasi programma.

  Vi invitiamo a non utilizzare la Open Software License per il vostro software. Ad ogni modo, non sussiste una valida ragione per evitare di utilizzare programmi rilasciati sotto questa licenza.

- Licenza OpenSSL

  La licenza di OpenSSL unisce due licenze: la cosiddetta "licenza di OpenSSL" e la licenza di SSLeay. Dovete seguirle entrambe. La combinazione è una licenza di software libero, copyleft, incompatibile con la GNU GPL. Ha anche una clausola di pubblicità come la licenza BSD originale e la licenza Apache 1.

  Vi invitiamo a utilizzare GNUTLS anziché OpenSSL per il vostro software. Ad ogni modo, non sussiste una valida ragione per evitare di utilizzare OpenSSL e applicazioni che lavorano con OpenSSL.

- Licenza Phorum, Version 2.0

  Questa è una licenza per software libero ma è incompatibile con la GPL. La sezione 5 la rende incompatibile con la GPL.

- Licenza PHP, versione 3.01

  Questa licenza è utilizzata da gran parte di PHP4. E' una licenza per software libero senza copyleft. È incompatibile con la GNU GPL a causa di forti restrizioni sull'uso del termine "PHP" nei nomi di prodotti derivati.

  Sconsigliamo di usare questa licenza per tutto quello che non siano estensioni di PHP.

- Licenza di Python 1.6b1 fino alla 2.0 e 2.1

  Questa è una licenza per software libero ma è incompatibile con la GNU GPL. Il problema principale è che questa licenza di Python è regolata alle leggi dello Stato della Virginia, negli Stati Uniti, e la GPL non lo permette.

- Q Public License (QPL), versione 1.0

  Questa è una licenza per software libero senza copyleft che è incompatibile con la GNU GPL. Crea inoltre rilevanti inconvenienti di tipo pratico, poiché i sorgenti modificati possono essere distribuiti solo sotto forma di patch.

  Raccomandiamo di non utilizzare la QPL per il vostro software ed utilizzare pacchetti rilasciati sotto questa licenza solo se strettamente necessario. Ad ogni modo, questo avvertimento non vale per Qt, dato che Qt adesso viene rilasciato anche sotto licenza GNU GPL.

Dato che la QPL è incompatibile con la GNU GPL, non è possibile prendere un programma rilasciato sotto GPL e uno rilasciato sotto licenza QPL per unirli assieme, non importa come.

Ad ogni modo, se avete scritto un programma che utilizza una libreria rilasciata sotto licenza QPL (chiamata, per esempio, RUBI) e volete rilasciare il vostro programma con la licenza GNU GPL, potete facilmente farlo. È possibile risolvere il conflitto *per il vostro programma* aggiungendo una nota come questa:

```
Come eccezione, avete il permesso di unire questo programma alla libreria XYZ e
di distribuire eseguibili se rispettate termini della licenza GNU GPL per quanto
riguarda tutto il software nell'eseguibile eccetto la parte XYZ.
```

È possibile fare questo, legalmente, se siete i detentori del copyright del programma. Aggiungetelo nei file sorgente, dopo la nota che specifica che il programma è rilasciato sotto licenza GNU GPL.

- RealNetworks Public Source License (RPSL), versione 1.0

  La RPSL è una licenza di software libero incompatibile con la GPL per vari motivi: richiede che i lavori derivati siano distribuiti sotto i termini della RPSL e stabilisce che la sede legale per discutere controversie sia Seattle, Washington.

- Sun Industry Standards Source License 1.0

  Questa è una licenza per software libero, con debole copyleft, incompatibile con la GNU GPL a causa di piccoli dettagli piuttosto che per l'impostazione generale.

- Sun Public License

  Questa licenza è essenzialmente uguale alla Licenza Pubblica Mozilla versione 1: una licenza per software libero incompatibile con la GNU GPL. Non confondetela con la Sun Community Source License che non è una licenza per software libero.

- Licenza di xinetd

  Questa è una licenza per software libero con copyleft, incompatibile con la GPL. Lo è poiché aggiunge ulteriori restrizioni sulla ridistribuzione di versioni modificate che vanno contro i requisiti di distribuzione della GPL.

- Yahoo! Public License 1.1

  Questa è una licenza di software libero. Ha un copyleft simile a quello della Mozilla Public License. Prevede anche una scelta di legge di riferimento, nella sezione 7. Per questi motivi è incompatibile con la GPL. Questa licenza utilizza anche il termine improprio “proprietà intellettuale”.

- Zend License, versione 2.0

  Questa licenza è usata da una parte di PHP4. E' una licenza di software libero, senza copyleft, incompatibile con la GNU GPL, e ha problemi pratici come quelli della licenza BSD originale.

  Sconsigliamo l'utilizzo di questa licenza per il vostro software.

- Zimbra Public License 1.3

  Questa licenza è identica alla Yahoo! Public License 1.1, tranne per il fatto che questa licenza è fornita da VMWare anziché Yahoo!. I nostri commenti sono gli stessi: si tratta di una licenza di software libero, con copyleft parziale, incompatibile con la GPL.

- Licenza pubblica Zope versione 1

  Questa è una licenza per software libero debole, piuttosto permissiva e senza copyleft con problemi pratici come quelli riscontrati nella licenza originale BSD, inclusa l'incompatibilità con la GNU GPL.

Vi invitiamo a non utilizzare la ZPL versione 1 per il vostro software. Ad ogni modo, non esiste una valida ragione per non utilizzare programmi rilasciati sotto questa licenza, come le precedenti versioni di Zope.

La versione 2.0 della Zope Public License è compatibile con la GPL.

12.1.1.3 • Licenze non libere per il software

Le seguenti licenze non si qualificano come licenze per software libero. Una licenza non libera è automaticamente incompatibile con la GNU GPL.

Ovviamente, vi invitiamo ad evitare l'uso di licenze per software non libero, ed evitare il software non libero in generale.

Non esiste modo per elencare tutte le licenze di software non libero conosciute in questa pagina; in fondo, ogni azienda di software proprietario ha la propria. Ci concentriamo qui sulle licenze che spesso vengono confuse con licenze per software libero ma sono, nei fatti, licenze per software non libero.

Abbiamo fornito collegamenti a quei pacchetti quando potevamo farlo senza violare la nostra regola generale. Non forniamo collegamenti a siti che promuovono, incoraggiano o facilitano l'uso di pacchetti software non liberi. L'ultima cosa che vogliamo fare è quella di fornire a qualunque pacchetto non libero della pubblicità gratuita che possa incoraggiare le persone ad usarlo. Per la stessa ragione, abbiamo evitato di menzionare i programmi che utilizzano queste licenze, a meno che non pensiamo che per motivi particolari questo non sia controproducente.

- Nessuna licenza

  Se il codice sorgente non è accompagnato da una licenza che dà agli utenti le quattro libertà essenziali, e non è nemmeno stato collocato nel pubblico dominio in maniera corretta, non è software libero.

  Alcuni sviluppatori pensano che il codice che non ha alcuna licenza sia da considerarsi automaticamente di pubblico dominio, ma questo non è vero secondo le leggi attuali: anzi, tutte le opere soggette a diritto d'autore, compresi i programmi, sono soggette a copyright se non diversamente specificato: se la licenza non dà all'utente specifiche libertà, queste non sono previste. In alcuni paesi gli utenti che scaricano codice che non ha licenza possono trovarsi ad infrangere il copyright anche semplicemente compilando o eseguendo quel codice.

  Affinché un programma sia libero, i detentori dei diritti d'autore su quel programma devono esplicitamente dare agli utenti quattro libertà essenziali. Il documento con cui lo fanno è una licenza di software libero, ed è questo lo scopo delle licenze di software libero.

  In alcuni paesi è possibile per l'autore mettere il codice sotto pubblico dominio, ma questo richiede un'azione esplicita. In questo caso, la scelta migliore è la licenza CC0, che funziona anche in altri paesi perché offre comunque una licenza che è sostanzialmente equivalente al pubblico dominio. Ma in molti casi, per assicurarsi che le libertà vengano trasmesse a tutti gli utenti di quel codice, è consigliabile rilasciare il codice con licenza copyleft.

  Il codice scritto da impiegati federali statunitensi fa eccezione perché le leggi statunitensi lo collocano esplicitamente nel pubblico dominio; tuttavia, questo non si applica alle opere scritte da aziende assunte dagli Stati Uniti, e non si applica necessariamente agli altri paesi, in cui spesso è permesso che lo stato eserciti il copyright su quanto produce.

- Aladdin Free Public License

  A differenza di quanto espresso nel nome, questa non è una licenza per software libero perché non permette di fare pagare la distribuzione, e proibisce di mettere insieme software con questa licenza e software a pagamento.

- Apple Public Source License (APSL), versione 1.x

  Le versioni 1.0, 1.1 e 1.2 sono licenze non libere (vedere il collegamento per ulteriori spiegazioni). Ne sconsigliamo l'utilizzo e sconsigliamo anche l'uso di software distribuito sotto queste condizioni. La versione 2.0 della APSL è una licenza di software libero.

- Licenza Artistica 1.0

  Non possiamo dire che questa sia una licenza per software libero poiché è troppo vaga; alcuni passaggi sono fin troppo elaborati a loro vantaggio, e il loro significato non è chiaro. Vi invitiamo a non utilizzarla, eccetto nel caso in cui sia parte della licenza disgiunta di Perl.

- AT&T Public License

  La AT&T Public License è una licenza non libera. Ha vari problemi seri:

  La clausola sui brevetti è invalidata da qualsiasi modifica al codice relativo. Dovete fare richiesta scritta per distribuire sorgenti o patch.

  Richiede di notificare AT&T se distribuite una patch.

  La licenza può essere terminata senza vostre colpe, come da sezione 8/3. Mette come condizione della licenza il rispetto di leggi sull'esportazione. Alcune versioni della licenza vi obbligano a fornire supporto.

  Alcune versioni della licenza dicono che non potete vendere una copia del software a un costo superiore alle spese di distribuzione.

  Ha inoltre altre due caratteristiche particolarmente fastidiose:

  Ha un'ampia licenza inversa a favore di AT&T, che va molto oltre l'uso del vostro codice, anche modificato.

  Sostiene che serve una licenza di AT&T per creare collegamenti al loro sito. Questo non è un problema pratico immediato, perché la licenza dà la possibilità di creare un tale collegamento (ma non si dovrebbero creare collegamenti a siti che promuovono software non libero). Ma un concetto del genere non dovrebbe essere pubblicizzato o diffuso.

- Code Project Open License, versione 1.02

  La Code Project Open License non è una licenza di software libero. La sezione 5.6 pone restrizioni su come potete usare l'opera. La sezione 5.4 vieta la distribuzione commerciale del software da solo; e, a seconda di come si interpreta la sezione 3.4, sembra anche che non ci sia proprio la possibilità di distribuire il software da solo, con qualsiasi modalità.

- eCos Public License, versione 1.1

  Questa era la vecchia licenza eCos. Non è una licenza di software libero, perché richiede di inviare ogni versione modificata a un certo sviluppatore iniziale. Ci sono anche termini potenzialmente problematici nel testo della licenza.

  Oggi eCos è disponibile sotto la GNU GPL con permesso aggiuntivo di collegamento con programmi non liberi.

- CNRI Digital Object Repository License Agreement

  Questa licenza è non libera a causa dell'articolo 3, che a quanto pare include il requisito di non violare la licenza di alcun programma eseguito dall'utente, anche proprietario.

- GPL for Computer Programs of the Public Administration

  La GPL-PA (nome originale Portoghese: “Licença Pública Geral para Administração Pública”) è non

libera per vari motivi:

Permette solo l'uso in "circostanze normali".

Non permette la distribuzione di codice sorgente senza binari. Scade dopo 50 anni.

- Hacktivismo Enhanced-Source Software License Agreement (HESSLA)

  Non è una licenza di software libero perché restringe i compiti per cui si può usare il software, e restringe anche i compiti che possono essere svolti da versioni modificate del programma.

- Jahia Community Source License

  La Jahia Community Source License non è una licenza libera. L'uso del codice sorgente è limitato a scopi di ricerca.

- Licenza di JSON

  Questa è la licenza dell'implementazione originaria del formato di interscambio dati JSON. Questa licenza usa la licenza Expat come base, ma aggiunge una clausola che ordina che “Il Software deve essere usato a fin di bene, non di male.” Questa è una restrizione sull'utilizzo e quindi va in conflitto con la libertà 0. E' probabile che questa restrizione non possa essere applicata in concreto, ma non possiamo darlo per scontato. Quindi questa licenza è non libera.

- Vecchia licenza di ksh93

  ksh93 era distribuito con una licenza dedicata che non era libera. Ad esempio richiedeva di inviare tutte le modifiche allo sviluppatore.

- Licenza di Lha

  La licenza di lha è da ritenersi non libera perché è così vaga che non permette di capire quali permessi ha l'utente.

- Microsoft's Shared Source CLI, C#, and Jscript License

  Questa licenza non permette distribuzione commerciale, e permette l'uso commerciale solo a certe condizioni.

  Microsoft ha altre licenze che descrive come "Shared Source", alcune delle quali hanno restrizioni diverse.

- NASA Open Source Agreement

  Il NASA Open Source Agreement, versione 1.3, è una licenza non libera perché richiede che le modifiche siano una vostra “creazione originale”. Lo sviluppo di software libero dipende dall'inclusione di software altrui, e questa licenza lo vieta.

  Sconsigliamo l'uso di questa licenza e invitiamo i cittadini statunitensi a scrivere alla NASA e chiedere l'utilizzo di una vera licenza libera.

- Licenza di Oculus Rift SDK

  Questa non è una licenza per software libero; ha molti gravi problemi.

  Si può distribuire solo il programma libOVR nella sua interezza, e non una parte.

  Il permesso di ridistribuire può essere revocato sulla base di condizioni poco chiare. Chi produce una versione modificata ha l'obbligo di inviarla a Oculus su richiesta.

  L'uso è consentito solo con il loro prodotto.

Le nuove versioni della licenza sostituiscono completamente le vecchie, e i permessi già dati possono quindi essere ritirati.

Ci potrebbero essere altri problemi significativi, ma questi sono già sufficienti per concludere la nostra analisi.

- Open Public License

  Questa non è una licenza per software libero, poiché richiede l'invio di ciascuna versione modificata pubblicata allo specifico sviluppatore iniziale. Sono presenti inoltre in questa licenza alcuni paragrafi ambigui: non siamo sicuri che non possano creare problemi.

- Peer-Production License

  La Peer-Production License non è una licenza libera perché restringe chi e come può distribuire il programma e per quale scopo. Inoltre non dà a tutti il permesso di usare il programma.

  La PPL ha veri aspetti specifici per performance artistiche, e non ci opponiamo al suo uso in ambito artistico, ma, al contrario di quanto alcuni consigliano, non deve essere usata per software, manuali o altre opere che dovrebbero essere libere.

- Licenza di PINE

  La licenza di PINE non è libera perché impedisce in gran parte la distribuzione di versioni modificate. Restringe anche le modalità utilizzabili per vendere copie.

  Alpine, un successore di PINE, è rilasciato sotto la Apache License, versione 2.0.

- Vecchia licenza di Plan 9

  Questa non è una licenza per software libero. Non fornisce le libertà essenziali come il diritto di effettuare ed utilizzare modifiche private. Non utilizzate questa licenza, e vi invitiamo a non utilizzare alcun software rilasciato sotto di essa. È disponibile anche una dettagliata discussione su questa licenza.

  A settembre 2002 venne notato che la licenza pubblicata di Plan 9 era stata modificata con l'aggiunta di ulteriori restrizioni, anche se la data era sempre 20 settembre 2000. Tuttavia, ulteriori modifiche nel 2003 hanno reso Plan 9 software libero.

- Reciprocal Public License

  La Reciprocal Public License è non libera per tre motivi. 1. Limita il prezzo di una copia iniziale. 2. Richiede di avvisare lo sviluppatore originale se si pubblica una versione modificata. 3. Richiede la pubblicazione di qualsiasi versione modificata in uso, anche privato.

- Licenza di Scilab

  Questa non è una licenza di sofware libero perché non permette la distribuzione commerciale di una versione modificata. Fortunatamente, dalla versione 5.0.0, Scilab è software libero, rilasciato sotto licenza CeCILL versione 2.

- Licenza di Scratch 1.4

  Questa non è una licenza di sofware libero perché non permette la distribuzione commerciale. Inoltre la condizione numero 4 pone significative restrizioni sulla funzionalità delle versioni modificate.

  Le ultime versioni di Scratch sono distribuite con licenza GNU GPL, ma non sono tutte consigliabili perché alcune dipendono da Adobe Air, che è software proprietario.

- Simple Machines License

Nonostante il nome è una licenza software, non libera per vari motivi:

Serve il permesso dell'autore originale per distribuire il software. Non si possono vendere copie del programma.

La vostra licenza può essere terminata se avete ricevuto il software da qualcuno che non ha rispettato la licenza.

- Vecchia licenza di Squeak

  La licenza di Squeak originaria, applicata al software, non è una licenza libera perché richiede agli utenti, ovunque si trovino, di applicare le leggi statunitensi sull'esportazione. Applicata ai font, non ne permette nemmeno la modifica.

  In aggiunta chiede agli utenti di non dare responsabilità all'autore, il che potrebbe bastare a molti utenti per riflettere prima di usare il software.

  Le versioni recenti di Squeak (dalla 4.0) sono rilasciate sotto Licenza Expat con porzioni di codice sotto Apache License 2.0.

- Licenza Sun Community Source

  Questa non è una licenza per software libero. Non fornisce le libertà essenziali come la pubblicazione di versioni modificate. Non utilizzate questa licenza, e vi invitiamo ad evitare tutto il software rilasciato sotto di essa.

- Licenza Sun Solaris Source Code (Foundation Release), versione1.1

  Questa non è una licenza per software libero. Questa licenza proibisce la redistribuzione, proibisce l'uso commerciale del software e può essere revocata.

- Sybase Open Watcom Public License, versione 1.0

  Questa non è una licenza di software libero. Obbliga a pubblicare il codice sorgente ogni volta che si rientra nei casi che licenza definisce col termine inglese “deploy”, che secondo le definizioni usate nella licenza comprendono anche vari tipi di utilizzo privato.

- SystemC “Open Source” License, Versione 3.0

  Questa licenza richiede di aiutare chi fornisce il software a fare valere i suoi marchi registrati. E' una condizione che non si può porre agli utenti, quindi la licenza è non libera. Ha anche altri problemi pratici: alcuni requisiti sono vaghi, e usa il termine improprio “proprietà intellettuale”.

  Nonostante il nome non è chiaro se questa licenza sia davvero una licenza “open source”, ma il nostro giudizio non è basato su quello.

- Truecrypt license 3.0

  Questa licenza non è libera, per vari motivi. Ad esempio dice che non si può usare il programma se non si comprende la licenza, impone condizioni sul consentire agli altri di eseguire una copia del programma, impone condizioni su programmi distinti che "dipendono da" Truecrypt. La condizione sui marchi registrati si applica anche a "materiali associati".

  Ci sono altri punti della licenza che sono problematici, e la loro valutazione ha ritardato la pubblicazione delle nostre conclusioni. Ma è ora chiaro che non ci dispiace che Truecrypt sia stato chiuso. Ci sono programmi liberi equivalenti.

- University of Utah Public License

  La University of Utah Public License non è libera perché non permette ridistribuzione commerciale. Stando ai suoi termini, proibirebbe anche l'uso commerciale e anche la consulenza commerciale,

restrizioni che probabilmente non sono valide secondo la legge USA, ma che potrebbero essere valide in alcuni paesi; è eccessivo anche il solo menzionarle.

L'uso di questa licenza da parte dell'Università dello Utah, che le dà il nome, mostra la pericolosa tendenza delle università a restringere la conoscenza invece che renderla pubblica.

Se un'università cerca di imporre una licenza come questa su software scritto da voi, non disperate. Con fermezza e un po' di impegno è possibile prevalere sugli amministratori di università troppo interessati al guadagno.

Sollevate il problema il prima possibile.

- Licenza di YaST

  Questa non è una licenza per software libero. La licenza proibisce la distribuzione a pagamento e rende impossibile includere il software in molti dei CD-ROM contenenti software libero che vengono venduti da aziende e da organizzazioni.

  Inoltre possono sussistere ulteriori problemi con la sezione 2a, ma sembra che manchi una parola, quindi è difficile stabilire cosa la licenza voglia veramente dire.

  (Il programma YaST non usa più questa licenza; per fortuna è ora software libero sotto licenza GNU GPL).

12.1.2 Licenze per la documentazione

12.1.2.1 Licenze per la documentazione libere

Le seguenti licenze si qualificano come licenze per documentazione libera.

- GNU Free Documentation License

  Questa licenza è stata progettata per documentazione libera con copyleft. Pensiamo di adottarla per tutti i manuali GNU. Si presta anche ad altri tipi di lavori, come libri di testo e dizionari, per esempio. Non è applicabile solo a lavori testuali (“libri”).

- Licenza per la documentazione di FreeBSD

  Questa è una licenza permissiva senza copyleft per documentazione libera, compatibile con la GNU FDL.

- Apple's Common Documentation License, versione 1.0

  Questa è una licenza per documentazione libera che è incompatibile con la GNU FDL. È incompatibile poiché la sezione 2c dichiara "Non è possibile aggiungere altri termini o condizioni a quelli presenti in questa licenza", e la GNU FDL ha termini aggiuntivi che non si trovano nella licenza Common Documentation.

- Licenza Open Publication, versione 1.0

  Questa licenza può essere utilizzata come licenza per documentazione libera. È una licenza per documentazione libera con copyleft a condizione che il detentore del copyright non eserciti nessuna delle "OPZIONI DI LICENZA" elencate nella sezione VI della licenza stessa. Se una delle opzioni viene scelta, la licenza diventa non libera. In ogni caso, è incompatibile con la GNU FDL.

  Questo crea uno scompenso pratico nell'uso o nel raccomandare l'uso di questa licenza: se invitate ad usare la Open Publication Licence, versione 1.0 senza scegliere nessuna delle opzioni, sarebbe facile dimenticare la seconda parte della raccomandazione. Qualcuno potrebbe utilizzare questa licenza e le sue opzioni e rendere un manuale non libero pur credendo di seguire lo stesso il vostro consiglio.

Allo stesso modo, se utilizzate questa licenza senza nessuna delle opzioni per rendere il vostro manuale libero, qualcun altro potrebbe imitarvi e cambiare poi la sua idea riguardo alle opzioni pensando che si tratti solo di un dettaglio. Il risultato è che renderebbe il suo manuale non libero.

Quindi, mentre i manuali pubblicati sotto questa licenza si qualificano come documentazione libera se nessuna opzione viene utilizzata, è meglio utilizzare la Licenza GNU Free Documentation per evitare i rischi di portare qualcun altro all'errore.

Notate per favore che questa licenza non è la stessa che la Licenza Open Content. Queste due licenze vengono spesso confuse, dato che la Licenza Open Content viene spesso chiamata con l'acronimo di "OPL". Per chiarezza è meglio non utilizzare la forma abbreviata "OPL" per entrambe le licenze. È meglio scrivere il nome completo per essere sicuri che si comprenda quello che state dicendo.

12.1.2.2 • Licenze per la documentazione non libere

Le seguenti licenze non si qualificano come licenze per documentazione libera:

- Licenza Open Content, versione 1.0

  Questa licenza non si qualifica come libera, poiché sussistono restrizioni sul pagamento in denaro delle copie. Vi raccomandiamo di non usare questa licenza.

  Notate per favore che questa licenza non è la stessa che la Licenza Open Publication. Queste due licenze vengono spesso confuse, dato che la Licenza Open Content viene spesso chiamata con l'acronimo di "OPL". Per chiarezza è meglio non utilizzare la forma abbreviata "OPL" per entrambe le licenze. È meglio scrivere il nome completo per essere sicuri che si comprenda quello che state dicendo.

- Creative Commons Nocommercial, qualsiasi versione

  Questa licenza non si qualifica come libera, poiché sussistono restrizioni sul pagamento in denaro delle copie. Quindi vi raccomandiamo di non usare questa licenza per la documentazione.

  In aggiunta, presenta un problema di fondo che vale per tutti i tipi di lavori: quando una versione modificata ha molti autori, ottenere il permesso all'uso commerciale da parte di tutti loro diventa praticamente impossibile.

- Creative Commons Noderivatives, qualsiasi versione

  Questa licenza non si qualifica come libera, poiché sussistono restrizioni sulla distribuzione delle versioni modificate. Vi raccomandiamo di non usare questa licenza per la documentazione.

12.1.3 Licenze per altre opere

12.1.3.1 Licenze per materiale diverso da software e documentazione

- Licenza Pubblica Generica GNU (GNU General Public License, GNU GPL

  La GNU GPL può essere usata per dati diversi dal software, purché sia possibile determinare la definizione di “codice sorgente” nel contesto specifico. La DSL (vedere sotto) richiede allo stesso modo di determinare quale sia il “codice sorgente” usando criteri analoghi alla GPL.

- GNU Free Documentation License

  La GNU FDL è consigliata per libri di testo e materiali per l'insegnamento in tutte le materie. (“Documentazione” significa semplicemente libri di testo e altri materiali per l'apprendimento del software). Consigliamo la GNU FDL anche per dizionari, enciclopedie, e altri lavori che forniscono informazioni di uso pratico.

- Creative Commons Attribution 4.0 license (CC BY)

  Questa è una licenza libera senza copyleft adatta a lavori di arte e spettacolo, e lavori didattici. È compatibile con tutte le versioni della GNU GPL, ma, come le altre licenze CC, non è consigliabile per il software.

  (#which-cc) Creative Commons pubblica molte licenze, molto diverse tra loro. Quindi, dire che un lavoro è sotto "licenza Creative Commons" non spiega la questione fondamentale, cioè quale licenza vi si applica. Quando vedete un lavoro con una nota di licenza del genere, chiedete all'autore di specificare chiaramente quale licenza si applica di preciso. E se qualcuno propone di usare una "licenza Creative Commons" per un certo lavoro, chiedete immediatamente quale.

- Creative Commons Attribution-Sharealike 4.0 license (CC BY-SA)

  Questa è una licenza libera senza copyleft adatta a lavori di arte e spettacolo, e lavori educativi. È compatibile con tutte le versioni della GNU GPL, ma, come le altre licenze CC, non è consigliabile per il software.

  La licenza CC BY-SA 4.0 è compatibile in una direzione con la GNU GPL 3: è possibile prendere materiale che ha licenza CC BY-SA 4.0, modificarlo e distribuirlo sotto licenza GNU GPL 3, ma non è possibile fare l'operazione inversa.

  Dato che Creative Commons elenca solo la versione 3 della GNU GPL come licenza compatibile, non si può distribuire un'opera che era sotto CC BY-SA con licenza "*GNU GPL version 3, or (at your option) any later version*" (cioè "GNU GPL 3 o successiva"). Tuttavia, la sezione 14 della GNU GPL 3 permette a chi sceglie la licenza di delegare ad altri la scelta se successive versioni della GNU GPL possano essere usate. Quindi, se qualcuno adatta un'opera che era sotto CC BY-SA 4.0 e la incorpora in un progetto sotto GNU GPL 3, può specificare che delega Creative Commons (tramite http://creativecommons.org/compatiblelicenses) in modo che quando e se Creative Commons deciderà che una futura versione della GNU GPL è compatibile, sarà possibile usare quella futura versione per l'opera combinata.

  Siate sempre attenti a quale licenza Creative Commons state facendo riferimento.

- Design Science License (DSL)

  Questa è una licenza libera con copyleft adatta a dati generici. Non utilizzatela per software o documentazione, poiché è incompatibile con la GNU GPL e la GNU FDL; ma è accettabile usarla per altri tipi di dati.

- Free Art License

  Questa è una licenza libera con copyleft adatta a lavori artistici. Permette distribuzione commerciale, come previsto dai criteri del software libero. È con copyleft perché qualsiasi opera più grande che comprende parte dell'opera distribuita sotto questa licenza deve essere rilasciata, nella sua interezza, sotto la stessa licenza o sotto una licenza simile che rispetti alcuni requisiti elencati. Non utilizzatela per software o documentazione, poiché è incompatibile con la GNU GPL e la GNU FDL.

- Licenza Open Database

  Questa è una licenza libera con copyleft adatta a dati generici. È incompatibile con la GNU GPL. Non utilizzatela per software o documentazione, poiché è incompatibile con la GNU GPL e la GNU FDL a causa di richieste sconvenienti sul dover firmare contratti che cercano di imporre qualcosa simile al copyleft su dati che non sono soggetti a diritto d'autore; ma è accettabile usare dati rilasciati sotto questa licenza.

- Licenze per i tipi di carattere

Le licenze qui sotto si applicano alla codifica di un certo disegno tramite file per computer, non al disegno in sé. Per quanto ne sappiamo, la codifica implementata è sempre soggetta a copyright, mentre lo stato legale del disegno artistico di un tipo di carattere è complesso e varia a seconda della giurisdizione.

- Licenza Pubblica Generica GNU (GNU General Public License, GNU GPL)

  La GNU GPL può essere usata per caratteri. Tuttavia, non permette di includere il carattere in un documento a meno che il documento non sia sotto licenza GPL. Per farlo, usate la specifica eccezione per i caratteri. Vedete anche questo articolo esplicativo.

- Arphic Public License

  Questa è una licenza libera con copyleft, incompatibile con la GPL. Il suo uso normale è per i caratteri, e in quel campo l'incompatibilità non crea problemi.

- Licenza dei font ec per LaTeX

  Questa licenza si applica ai font European Computer Modern e Text Companion, comunemente usati con LaTeX. A seconda di come è usata può essere libera o no. Se il pacchetto dice che alcuni font nel pacchetto non possono essere modificati, allora il pacchetto non è libero, altrimenti lo è. I font originali in sé non contengono queste restrizioni, quindi sono liberi.

  In modo simile alla LaTeX Project Public License 1.2, questa licenza richiede che le versioni modificate dell'opera usino un nome diverso dal nome di qualsiasi versione precedente. Questo è accettabile per opere destinate ad essere usate con LaTeX, dato che TeX si basa sui nomi dei file, ma è molto fastidioso in altri contesti.

- IPA Font License

  Questa è una licenza di software libero con copyleft, incompatibile con la GPL. Ha una condizione scomoda che richiede che le opere derivate non usino o non comprendano il nome dell'opera originaria come nome di programma, tipo di carattere o file. Questo è accettabile per i tipi di carattere (che si possono sostituire usando strumenti liberi), ma può essere molto scomodo e fastidioso in altri contesti.

- SIL Open Font License 1.1

  La Open Font License (anche la versione originale 1.0) è una licenza libera con copyleft per i caratteri. Il suo unico requisito è che, quando li si vende, i caratteri siano distribuiti con qualche programma, non da soli. Dato che qualsiasi semplicissimo programma basta a soddisfare il requisito, questo non è un problema. Né noi né SIL consigliamo l'uso di questa licenza al di fuori dell'ambito dei caratteri.

- Licenze per opere legate a un punto di vista (ad esempio, opinioni o testimonianze)

  Lavori che presentano l'opinione di qualcuno (memorie, editoriali e così via) hanno uno scopo diverso dal software e dalla documentazione. Per questo ci aspettiamo che diano permessi diversi a chi riceve i lavori: semplicemente il permesso di copiare e distribuire il lavoro in forma letterale. Richard Stallman ne discute spesso nei suoi interventi pubblici.

  Dato che tantissime licenze rispettano questi criteri, non possiamo elencarle tutte. Tuttavia, se ne state cercando una, possiamo darvi due suggerimenti:

- GNU Verbatim Copying License

  Questa licenza è stata usata per molti anni nel sito di GNU. È molto semplice, e particolarmente adatta a lavori scritti.

- Creative Commons Attribution-NoDerivs 4.0 license (CC BY-ND)

Questa è la licenza usata nei siti di GNU e FSF. Dà circa gli stessi permessi della nostra licenza di copia letterale, ma è molto più dettagliata. La consigliamo per lavori di opinione audio o video. Vanno bene anche versioni meno recenti, ma consigliamo di aggiornare all'ultima versione se possibile. Siate sempre attenti a quale licenza Creative Commons state discutendo.

- Licenze per la progettazione di oggetti fisici

I circuiti sono pensati per l'uso pratico, quindi anche i loro progetti devono avere licenza libera. Consigliamo la licenza pubblica generica (GNU GPL) 3 o successiva; la versione 3 è stata adattata a questo caso.

Anche gli schemi per stampanti 3D relativi ad oggetti di uso pratico devono essere liberi. Consigliamo la GNU GPL o una delle licenze Creative Commons libere: CC-BY, CC-BY-SA o CC0.

Invece gli schemi per stampanti 3D per oggetti decorativi sono opere artistiche, per cui consigliamo una qualsiasi licenza Creative Commons.

13 Appendice C

13.1 Comparazione tra piattaforme per la creazione di comunità e la gestione di OSS

Esistono decine di piattaforme disponibili sul mercato. Alcune di esse totalmente gratuite in quanto basate su OSS altre a pagamento. Wikipedia elenca i cosiddetti FOSS (Forge OSS) definendone le principali caratteristiche e classificandoli in:

Free software

- Allura
- FusionForge
- Gitea
- Joinup (Drupal-based)
- Kallithea
- Launchpad Suite
- Phabricator
- Redmine
- sourcehut
- Trac
- Tuleap

Freemium software (Software gratuito ma con alcune funzioni a pagamento)

- GitLab
- GForge Advanced Server

Free online services

- Joinup collaboration platform
- Launchpad
- OSDN (Open Source Development Network)
- SourceForge
- GNU Savannah
- sourcehut (while in alpha)

Freemium online services

- GitLab
- GitHub
- Bitbucket

Discontinued software

- Savane (software)
- GForge Community Edition (last release April 23, 2010), not to be confused with the proprietary GForge first released October 1, 2018.

Discontinued online services

- Gna.org

- Google Code
- Open Source Assistive Technology Software (OATS)

Per quanto riguarda i costi, ad esempio, secondo il sito https://github.com/pricing#compare-features , consultato nel novembre 2021, la piattaforma GitHub presenta tre livelli di adesione. Il primo è il livello "Free" gratuito che consente limitati gradi di libertà. Il secondo, Team, ha un costo di 48 $/utente.anno, mentre il terzo, Enterprise, ha un costo di 252 $/utente.anno.

Per quanto riguarda invece un confronto tra alcune piattaforme, si riprende quanto riportato dal sito di Apache Allura https://allura.apache.org, che è una piattaforma opern source e gratuita.

Name	License	Language	Database	Git	Hg	SVN	Other SCM	Integrated Code Browser	Fork / Merge Requests	Notes
Allura	Apache 2.0	Python	Mongo	Yes	Yes[1]	Yes	No	Yes	Yes	DOAP description
Indefero	GPLv2	PHP	Any SQL	Yes	Yes	Yes	Yes	Yes	No	Last release 19/11/2012
Gitlab	MIT	Ruby	MySQL or PostgreSQL	Yes	No	No	No	Yes	Yes	
FusionForge	GPLv2	PHP	PostgreSQL	Yes	Yes	Yes	Yes	No	No	DOAP and ADMS.SW description
Gitorious	AGPL	Ruby	MySQL	Yes	No	No	No	Yes	Yes	Acquired by Gitlab
Libre Source	GPLv2	Java	PostgreSQL	No	No	Yes	No	No	No	Last release May 2008
Redmine	GPLv2	Ruby	MySQL, PostgreSQL, or SQLite	Yes	Yes	Yes	Yes	Yes	No	
Savane	GPLv2	PHP, Perl	MySQL	Yes	Yes	Yes	Yes	No	No	Merged with Fusionforge
Trac	BSD	Python	SQLite, PostgreSQL, or MySQL	Yes	Yes	Yes	Yes	Yes	No	
Tuleap	GPLv2	PHP	MySQL	Yes	No	Yes	Yes	Yes	Yes	
Apache Bloodhound	Apache 2.0	Python	SQLite, PostgreSQL, or MySQL	Yes	Yes	Yes	Yes	Yes	No	Based on Trac
Phabricator	Apache 2.0	PHP	MySQL	Yes	Yes	Yes	No	Yes	No	
RhodeCode	AGPL v3.0 or Commercial	Python	PostgreSQL, SQLite, or MySQL/MariaDB	Yes	Yes	Yes	No	Yes	Yes	
Pagure	GPLv2	Python	MySQL, PostgreSQL, or SQLite	Yes	No	No	No	Yes	Yes	
Gogs	MIT	Go	MySQL, PostgreSQL	Yes	No	No	No	Yes	Yes	

Name	Ticket Tracker	Wiki	Forums	Blog / News	Download Manager	Project Discovery	API	Plugins	Custom Themes	LDAP Auth	Oauth Auth
Allura	Yes	Yes	Yes	Yes	No	Yes[2]	Yes	Yes	Yes	Yes	Yes
Indefero	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No
Gitlab	Yes	Yes	No	No	No	Yes	Yes	No	Yes	Yes	Yes
FusionForge	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes
Gitorious	No	Yes	No	No	No	Yes	Yes	No	Yes	Yes	No
LibreSource	Yes	Yes	Yes	No	Yes	No	No	No	No	Yes	No
Redmine	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Savane	Yes	No	No	Yes	No	Yes	No	No	Yes	No	No
Trac	Yes	Yes	Yes[3]	Yes[4]	Yes[5]	No	Yes	Yes	Yes[6]	Yes	Yes
Tuleap	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Apache Bloodhound	Yes	Yes	Yes[3]	Yes[4]	Yes[5]	No	Yes	Yes	Yes[6]	Yes	No
Phabricator	Yes	Yes	No	No	No	No	Yes	No	No	Yes	Yes
RhodeCode	No	No	No	No	Yes	No	Yes	Yes	No	Yes	Yes
Pagure	Yes	No	No	No	Yes	No	Yes	No	No	No	No
Gogs	Yes	Yes	No	No	Yes	Yes	Yes	No	No	Yes	Yes