Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

XR Bridge: un wrapper OpenXR-OpenGL per applicazioni VR

Studente/i

Lorenzo Adam Piazza

Relatore

Peternier Achille

Correlatore

-

Committente

Peternier Achille

Corso di laurea

Ingegneria informatica (Informatica TP)

Modulo

C10826

Anno

2023 / 2024

Data

19 agosto 2024

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Lo scopo del progetto è sviluppare un wrapper (d'ora in poi chiamato XrBridge) attorno a OpenXR per permettere di sviluppare applicazioni che fanno uso di realtà virtuale e OpenGL in modo più semplice. XrBridge andrà a sostituire OvVR, un'implementazione simile già esistente sviluppata dal docente responsabile che fa uso di OpenVR. Dal momento che XrBridge verrà utilizzato nel corso di realtà virtuale (successore del corso di grafica), esso dovrà essere il più simile a OvVR possibile.
Aver personalmente seguito sia il corso obligatorio di grafica e il corso opzionale di realtà virtuale mi ha permesso di meglio capire i requisiti del progetto, sia dal punto di vista del docente che dovrà lavorare con XrBridge durante il suo corso, sia dal punto di vista dello studente che dovrà sviluppare un progetto che verrà poi valutato facendo uso di XrBridge.
L'aspetto più importante di XrBridge (oltre al fatto che deve funzionare) è la semplicità. Una soluzione troppo complessa sarebbe un problema sia per il docente, sia per lo studente. Una soluzione troppo complessa forzerebbe il docente a dedicare più tempo a spiegare agli studenti come funziona e come si utilizza XrBridge; il corso, infatti, è dedicato alla realtà virtuale, non a XrBridge. Tale soluzione causerebbe troppe difficoltà per gli studenti, poiché aggiungerebbe ancora più materiale da studiare e comprendere. Per sviluppare XrBridge è dunque molto importante sempre considerare il contesto in cui esso verrà utilizzato.
Lo scopo del corso di realtà virtuale è sviluppare un'applicazione in C++ che fa uso di realtà virtuale senza utilizzare game engines interagendo direttamente con API di basso livello come OpenGL e OpenVR. Il corso opzionale di realtà virtuale si basa sul corso obligatorio di grafica, dove lo scopo è sviluppare un'applicazione grafica 3D (ad esempio un piccolo gioco) senza fare affidamento a game engines esistenti o strumenti simili. Gli studenti imparano ad utilizzare OpenGL e a sviluppare personalmente un game engine che dovranno poi utilizzare per sviluppare l'applicazione grafica. Il corso di realtà virtuale consiste, in poche parole, ad estendere l'applicazione grafica sviluppata nel corso di grafica per aggiungere la funzionalità di realtà virtuale.

Capitolo 2

Stato dell'arte

Grazie all'evoluzione della tecnologia hardware e software nel corso degli ultimi decenni, è diventato sempre più facile sviluppare applicazioni di realtà virtuale di qualità sempre maggiore e sempre più immersive. È facile riconoscere questo progresso; basta confrontare le prime esperienze di realtà virtuale, come il Nintendo Virtual Boy, e confrontarle con i videogiochi VR di ultima generazione. Nel corso del tempo sono nati una moltitudine di strumenti per facilitare lo sviluppo di applicazioni VR, da hardware facilmente accessibile a utenti casalinghi a programmi che permettono di creare scenari immersivi trascinando con il mouse oggetti in una scena virtuale.

Ci sono diversi metodi e strumenti per sviluppare applicazioni VR; di seguito ne elencherò alcuni e descriverò in che modo XrBridge si differenzierà dalle soluzioni già esistenti. Alcuni di questi strumenti sono già utilizzati nel corso di realtà virtuale.

2.1 Game engines

Nella maggior parte dei casi, appoggiarsi su di un game engine già esistente è la scelta migliore, semplice e diretta per sviluppare un'applicazione di realtà virtuale. Un game engine permette di concentrarsi interamente sul contenuto dell'applicazione senza dover pensare a tutti i detagli necessari per avere un'applicazione grafica funzionante. Alcuni dei game engines più conosciuti che supportano lo sviluppo di applicazioni VR sono Unity, Unreal Engine e Godot.

Ci sono però alcuni scenari dove potrebbe essere necessario sviluppare un engine da zero; è il caso di un'applicazione con bisogni molto specifici non coperti da un engine già esistenti oppure, come in questo caso, se l'obiettivo è imparare a sviluppare un'applicazione partendo dalle basi. In questi casi, è necessario imparare ad utilizzari strumenti a livelli più bassi; le prossime sezioni sono dedicate ad alcuni di questi strumenti.

2.2 OpenVR

OpenVR si tratta di un SDK e API sviluppati da Valve per facilitare lo sviluppo do applicazioni VR. OpenVR è progettato per essere semplice da usare e si concentra principalmente su applicazioni per Head Mounted Displays (classici visori a occhiali). Per questo motivo, OpenVR è più limitato a confronto con OpenXR.

2.3 OpenXR

OpenXR si tratta di uno standard aperto creato da Khronos (lo stesso gruppo che ha sviluppato OpenGL e Vulkan) con lo scopo di sviluppare applicazioni che fanno uso di realtà virutale e realtà aumentata. La prima versione completa (versione 1.0) è stata rilasciata nel 2019 con lo scopo di risolvere la frammentazione che esiste attualmente nel mondo della realtà virtuale. A differenza di OpenVR, OpenXR è solamente uno standard che descrive una API e non offre nessun software già pronto ed è compito di produttori di hardware o piattaforme di sviluppare i software che implementano la API. Questi software vengono chiamati *runtime*. Il vantaggio di avere una API standard è quello di permettere di sviluppare applicazioni che possono essere eseguite su una moltitudine di dispositivi. OpenXR è progettato per supportare moltissimi possibili scenari, dalla realtà aumentata, alla realtà virtuale utilizzando un headset ad un sistema cave. Questa flessibilità però viene al prezzo di una maggiore complessità rispetto a OpenVR.

2.4 SteamVR

SteamVR si tratta di un software sviluppato da Valve e distribuito attraverso la piattaforma Steam. SteamVR supporta Windows e Linux. SteamVR funge sia da implementazione di OpenVR, sia come runtime di OpenXR. Tutti i videogiochi che fanno uso di realtà virtuale distribuiti attraverso Steam fanno uso di SteamVR.

2.5 OvVR

OvVR è una libreria che funge da wrapper attorno a OpenVR sviluppata dal docente responsabile ed ha lo scopo di semplificare lo sviluppo di applicazioni VR per il corso di realtà virtuale. XrBridge andrà a sostituire questa libreria. OvVR è scritto in C++ ed è interamente contenuto in un singolo file header.

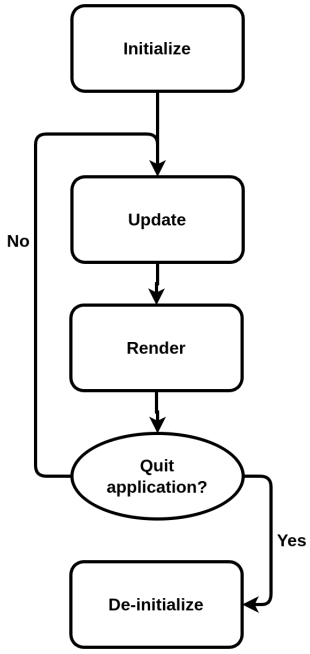
Capitolo 3

Design e implementazione

3.1 API

Di seguito è riportato il flusso di esecuzione di XrBridge:

Figura 3.1: Flusso di esecuzione di XrBridge



Lo schema riporta le operazioni fondamentali necessarie per sviluppare un'applicazione grafica.

Uno degli aspetti più importanti della API è la semplicità. È importante che l'utente che farà uso di XrBridge si possa concentrare il più possibile sullo sviluppare la propria applicazione invece di dover pensare ai dettagli di implementazione della libreria. Questo significa che la API deve esporre solamente metodi e parametri che sono assolutamente necessari per

l'utente e null'altro. Per questo motivo, è stato scelto di avere un metodo per ogni operazione fondamentale (TODO: vedi schema): inizializzazione, aggiornamento dello stato, render e de-inizializzazione. In realtà, aggiornamento e render potrebbero essere raggruppati in un'unica operazione, ma è stato deciso di lasciarli separati (se l'applicazione è in pausa e perciò non deve mostrare nulla, ma deve comunque rimanere in ascolto di eventi?). I dettagli della API e dei metodi si trovano nella documentazione apposita.
XrBridge è stato implementato sotto forma di una singola classe, dove ogni metodo pubblico rappresenta una delle operazioni fondamentali.

3.1.1 Gestione errori

3.1.2 Inizializzazione

3.1.3 Update

3.1.4 Render

3.1.5 De-inizializzazione

3.2 Strumenti e linguaggi di programmazione

Dal momento che XrBridge andrà a sostituire una libreria già in uso scritta in C++ e che i corsi di grafica e realtà virtuale fanno uso unicamente di C++, anche XrBridge verrà scritto in C++; nessun altro linguaggio di programmazione è necessario per la libreria.
Inoltre, nessuno strumento specifico è necessario per sviluppare la libreria. Ogni strumento aggiuntivo serve unicamente per sviluppare l'applicazione

3.3 OpenXR

3.3.1 Documentazione

OpenXR offre una documentazione molto estesa che descrive nei dettagli come la API deve essere utilizzata e come una runtime deve essere implementata. Ci sono due principali tipi di documentazione: un manuale mirato principalmente a chi desidera utilizzare OpenXR che spiega come fare uso della API OpenXR, e una specifica mirata a chi desidera implementare una runtime di OpenXR.
Il manuale della API è accessibile tramite il seguente link: https://registry.khronos.org/OpenXR/specs/< VERSIONE >/man/html/, dove VERSIONE è la versione di OpenXR che si sta utilizzando (Per esempio 1.0). Si può accedere velocemente al manuale di una funzione o struct specifica con un link del seguente formato: https://registry.khronos.org/OpenXR/specs/< VERSIONE >/man/html/< FUNZIONE O STRUCT >.html. Il manuale mostra informazioni come

le definizione delle funzioni e i loro parametri, possibili errori e una descrizione dettagliata della funzionalità.
La specifica di OpenXR è invece accessibile tramite il seguente link: https://registry.khronos.org/OpenXR/specs/1.0/l
Come già detto, questo è utile principalmente per chi desidera implementare una runtime e non chi a chi semplicemente desidera utilizzare la API per sviluppare un'applicazione VR. Per questo progetto, ho usato molto raramente il documento di specifica.
OpenXR offre inoltre un tutorial ufficiale che spiega come sviluppare una semplice applicazione VR facendo uso di OpenXR. Il tutorial è accessibile al seguente link: https://openxr-tutorial.com/ e permette di scegliere la combinazione di piattaforma e API grafica che si desidera utilizzare. Per questo progetto ho utilizzato Windows / OpenGL, dal momento che l'implementazione per Linux non differiva da quella di Windows. Ho seguito attentamente questo tutorial per comprendere il funzionamento di OpenXR e per sviluppare una semplice applicazione dalla quale ho poi estratto il codice necessario per sviluppare XrBridge.

3.3.2 Versioni

Al momento dello svolgimento di questo progetto, ci sono due versioni principali di OpenXR: 1.0 e 1.1. Le differenze principali tra queste due versioni sembrano minime e non importanti per questo progetto; ho scelto perciò di utilizzare la versione 1.0 per avere la massima compatiblità con le runtime.
Le principali differenze tra le due versioni sono due: sono stati apportati miglioramenti alla specifica di OpenXR e alcune estensioni sono state incluse in OpenXR core.

3.3.3 Runtime

OpenXR non si tratta di un software specifico, bensì di un'interfaccia standard di API che permette di sviluppare applicazioni di realtà aumentata per svariati dispositivi. Una runtime è semplicemente un software che implementa lo standard e offre alle applicazioni un'interfaccia con un dispositivo di realtà virtuale. È compito degli sviluppatori di dispositivi per realtà virtuale svilupare una runtime per il proprio dispositivo. I requisiti di questo progetto richiedono che XrBridge debba funzionare almeno con SteamVR (una runtime di OpenVR e OpenXR sviluppata da Valve); dal momento che OpenXR è uno standard, XrBridge dovrebbe funzionare con tutte le altre runtime che implementano lo standard OpenXR correttamente, salvo per piccoli aggiustamenti.
Per permettere alle applicazioni di trovare la runtime corretta installata sul computer dell'utente, ogni runtime fa uso di un file manifest, ovvero un file di formato JSON che contiene alcune informazioni basi come il nome della runtime e il suo percorso nel filesystem. Questi file manifest sono generalmente installati in percorsi standard predefiniti (definiti dallo standard OpenXR) che dipendono dalla piattaforme; in alternativa è possibile specificare un percorso non-standard attraverso una variabile d'ambiente.

3.3.4 Binding grafico

OpenXR fa uso di "graphic bindings", ovvero strutture che legano assieme una API grafica e una piattaforma. All'interno del codice, questi binding sono implementati sotto forma di struct. Esiste uno struct per ogni combinazione di API grafica (OpenGL, Vulkan, DirectX, ...) e piattaforma (Win32, X11, Wayland, ...) supportate e ogni struct richiede dei parametri legati alla piattaforma e alla API grafica scelta. Questi struct sono definiti nel file `openxr_platform.h` della libreria OpenXR. Ecco alcuni esempi:

API grafica	Piattaforma	Nome struct
OpenGL	Windows	`XrGraphicsBindingOpenGLWin32KHR`
OpenGL	Linux (Xlib)	`XrGraphicsBindingOpenGLXlibKHR`
OpenGL	Linux (Wayland)	`XrGraphicsBindingOpenGLWaylandKHR`

Durante l'inizializzazione di OpenXR, è compito dello sviluppatore istanziare e popolare uno di questi stuct per la piattaforma che so vuole utilizzare. I requisiri richiedono solamente il supporto per OpenGL su Windows e Linux. Di seguito sono descritti gli approcci all'implementazione dei binding richiesti.

3.3.4.1 Binding OpenGL + Windows

Il binding OpenGL + Windows è rappresentato dallo struct (`XrGraphicsBindingOpenGLWin32KHR`. Di seguito è riportata la definizione dello struct (i parametri `type` e `next` non sono importanti per questo capitolo):

```
typedef struct XrGraphicsBindingOpenGLWin32KHR {
  XrStructureType             type;
  const void* XR_MAY_ALIAS    next;
  HDC                         hDC;
  HGLRC                       hGLRC;
} XrGraphicsBindingOpenGLWin32KHR;
```

I due parametri importanti sono `hDC` e `hGLRC` e fanno parte della API Win32. `hDC` si tratta del *device context*, mentre *hGLRC* è il *OpenGL Rendering Context*; non è importante cosa rappresentato questi parametri, ma è necessario averli.
Qui si incontra un ostacolo: il contesto di OpenGL viene generato automaticamente da FreeGLUT. Per questioni di portabilità, FreeGLUT non offre un modo di recuperare questi parametri; a noi servono gli stessi parametri che FreeGLUT ha usato per creare il contesto OpenGL, il che significa che non possiamo generare un nuovo contesto/generare nuovi parametri. Fortunatamente, la API Win32 di Windows offre un modo per recuperare entrambi i parametri grazie alle funzioni `wglGetCurrentDC` e `wglGetCurrentContext` - queste due funzioni non accettano parametri. Grazie a queste funzioni, è possibile popolare tutta la struct e, di conseguenza, configurare OpenXR per questa piattaforma.

3.3.4.2 Binding OpenGL + Linux

Questo binding è più complesso rispetto a quello di OpenGL + Windows per due motivi.
La prima complicazione è dovuta al fatto che Linux non ha una API standard per quanto riguarda gli ambienti grafici. Al momento esistono infatti due principali piattaforme grafiche su Linux: X11 e Wayland. Inoltre, per X11, esistono approcci diversi per ognuna delle due librerie X11 più diffuse: Xcb e libX. Per questo motivo, esistono 3 struct per il binding OpenGL + Linux: `XrGraphicsBindingOpenGLXlibKHR` (LibX), `XrGraphicsBindingOpenGLXcbKHR` (Xcb) e `XrGraphicsBindingOpenGLWaylandKHR` (Wayland).
Inizialmente avevo deciso di utilizzare Wayland per due motivi. Il primo motivo è che lo struct per questo binding richiede un solo parametro, probabilmente rendendo più semplice l'implementazione. Il secondo motivo è il fatto che Wayland sta andando sempre di più a sostituire X11, tanto che molte delle distribuzioni di Linux principali supportano Wayland out-of-the-box. Lo svantaggio è il fatto che non è possibile eseguire applicazioni Wayland su X11. Questo approccio non è stato possibile a causa dell'assenza di supporto per Wayland in FreeGLUT. Nonostante FreeGLUT abbia un'implementazione di Wayland, come anche suggerito da un commento di uno degli sviluppatori (TODO: link to the comment), tale implementazione non è funzionante e sembra al momento abbandonata. A causa dei problemi appena menzionati, ho deciso di utilizzare l'approccio con X11.
Fortunatamente, esiste un software dal nome di XWayland che permette di eseguire quasi perfettamente molte applicazioni sviluppate per X11 in un ambiente Wayland. Usando l'approccio X11, quindi, potrò supportare entrambe le piattaforme X11 e Wayland.
Per quanto riguarda l'approccio X11, esistono due alternative: utilizzare la libreria Xlib (la libreria originale per interagire con X11) e Xcb (libreria alternativa a Xlib). Tra i due approcci, utilizzare Xlib è il più semplice e diretto, poichè non richiede di stabilire una connessione a X11. Di seguito è riportato lo struct per il binding OpenGL + Linux (X11):

```
typedef struct XrGraphicsBindingOpenGLXlibKHR {
    XrStructureType             type;
    const void* XR_MAY_ALIAS    next;
    Display*                    xDisplay;
    uint32_t                    visualid;
    GLXFBConfig                 glxFBConfig;
    GLXDrawable                 glxDrawable;
    GLXContext                  glxContext;
} XrGraphicsBindingOpenGLXlibKHR;
```

I parametri `xDisplay`, `glxDrawable` e `glxContext` sono facilmente reperibili utilizzando le funzioni `glXGetCurrentDisplay`, `glXGetCurrentDrawable` e `glXGetCurrentContext` rispettivamente. Il problema sono i parametri `visualid` e `glxFBConfig`. Anche questi sono

parametri che sono configurati da FreeGLUT ma non sono esposti e non esiste un modo per recuperarli come è possibile per i parametri precedenti.

3.4 Limitazioni di FreeGLUT

FreeGLUT è una libreria multi-piattaforma che permette di gestire finestre, contesti OpenGL, mouse e tastiera. L'utilizzo di questa libreria è obbligatoria dal momento che viene utilizzata dal corso di grafica e il corso di realtà virtuale. Per questo progetto, è stata utilizzata la versione 3.6.0.
Come menzionato prima nel capitolo [Binding OpenGL + Linux], FreeGLUT non espone certi parametri che potrebbero essere necessari ad un'applicazione. È stato dunque necessario apportare modifiche alla libreria. L'approccio scelto è il seguente: aggiungere un nuovo file header (`freeglut_globals.h`) con all'interno delle variabili globali e salvarci dentro i parametri necessari. All'itenro di XrBridge, poi, includere questo file header e accedere alle variabili globali. Come parte della consegna del progetto, è anche presente una patch di git che descrive i cambiamenti effettuati nel dettaglio, così che possono essere facilmente applicati e analizzati.
Iniziando con il parametro più semplice: `visualid`. Questo si tratta di un semplice `uint32_t`. È bastato aggiungere una singola riga di codice che assegna una il valore della variabile `visualInfo->visualid` alla variabile globale creata. Questo viene fatto all'interno della funzione `fgPlatformOpenWindow` nel file `src/x11/fg_window_x11.c` all'itenro della repo di FreeGLUT.
Il secondo parametro è `glxFBConfig`. Questo è più complicato rispetto al precedente, pochè non si tratta di un semplice valore che può essere facilmente copiato, bensì di una struttura interna che non viene esposta all'utente se non attraverso un puntatore. Questa struttura viene generata dalla funzione `glXChooseFBConfig`, la quale accetta una serie di attributi assieme ad altri parametri e ritorna un puntatore ad una struttura di tipo `GLXFBConfig`. La soluzione è fortunatamente pittosto semplice: è sufficiente ottenere la lista di attributi usati per generare la struttura e invocare nuovamente il metodo `GLXFBConfig`. Per fare questo, è stato necessario salvare gli attributi in una delle variabili globali appena create. Questo è stato fatto nella funzione `fghChooseConfig` nel file `src/x11/fg_window_x11_glx.c`. Infine, all'interno di XrBridge, è bastato chiamare la funzione `glXChooseFBConfig` con gli attributi corretti e così si ottiene la struttura necessaria.
Una libreria alternativa a FreeGLUT è GLFW, la quale espone i parametri interni attravero dei metodi specifici. Come menzionato all'inizio del capitolo, però, non è stato posibilie utilizzare questa libreria.

Capitolo 4

Risultati

Capitolo 5

Conclusioni

5.0.1 Applicazione di test

Dal momento che il progetto consiste nello sviluppare una libreria, sarà necessario un'applicazione che farà uso di XrBridge. Questa applicazione avrà il doppio scopo di aiutare nello sviluppo e test della libreria e

5.1 Filosofia

5.2 API

Capitolo 6

Titolazione

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

6.1 Sezione

Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit. Esempio di citazione [1], [2, 3], [4]. [1] [2] Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

6.1.1 Sotto sezione

`Questo testo ha una spaziatura fissa`

Questo testo è in italico

Questo testo è in grassetto

QUESTO TESTO È IN MAIUSCOLETTO

[1] Questa è una nota a pié di pagina.
[2] Questa è un'altra nota a pié di pagina.

Questo testo è sottolineato

Citazione:

> Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

Capitolo 7

Titolazione

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

- Elemento A
- Elemento B
- Elemento C

Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

Bibliografia

[1] C. Yuen and A. Oudalov. The feasibility and profitability of ancillary services provision from multi-microgrids. In *Power Tech, 2007 IEEE Lausanne*, pages 598 –603, july 2007.

[2] H. Farhangi. The path of the smart grid. *Power and Energy Magazine, IEEE*, 8(1):18 –28, january-february 2010.

[3] L.H. Tsoukalas and R. Gao. From smart grids to an energy internet: Assumptions, architectures and requirements. In *Electric Utility Deregulation and Restructuring and Power Technologies, 2008. DRPT 2008. Third International Conference on*, pages 94 –98, april 2008.

[4] F. Blaabjerg, R. Teodorescu, M. Liserre, and A.V. Timbus. Overview of control and grid synchronization for distributed power generation systems. *Industrial Electronics, IEEE Transactions on*, 53(5):1398 –1409, oct. 2006.