UNIVERSITÀ DI PISA

Dipartimento di Informatica
Corso di Laurea in Informatica

EasyRegatta

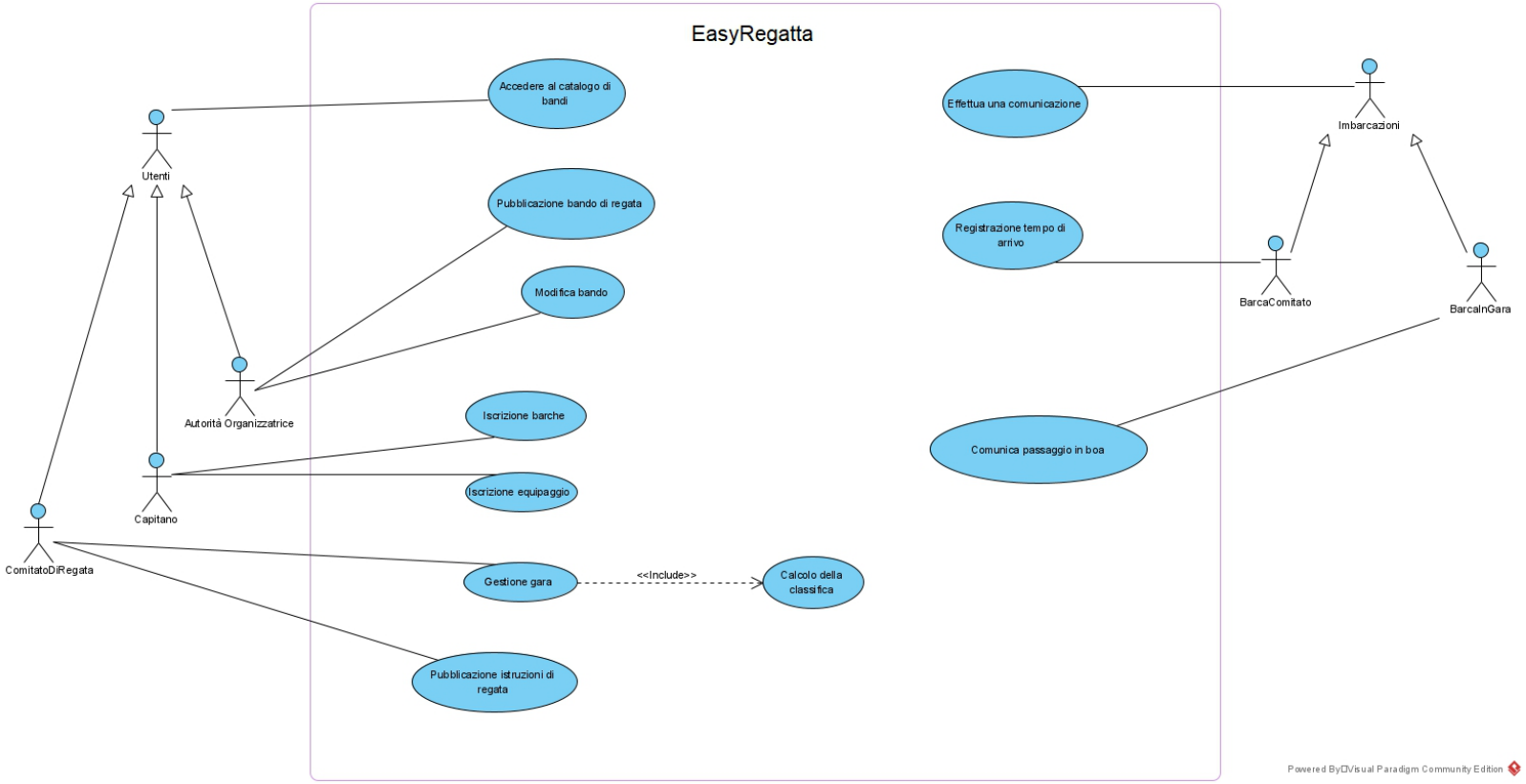
Ingegneria del Software a.a. 2020/2021

Prof.ssa Roberta Gori

F D L -
A F -
Giacomo Trapani -

Esercizio 1

Il diagramma dei casi d'uso descrive le funzionalità che il sistema mette a disposizione all'esterno. Degni di nota sono il caso d'uso relativo alle comunicazioni - ("**Effettua una comunicazione**") - che prevede che il sistema faccia da tramite per le comunicazioni tra imbarcazioni, alle informazioni sui tempi di gara - (i.e. "**Registrazione tempo di arrivo**", "**Comunica passaggio in boa**") - che implicano che il sistema venga utilizzato per segnare in tempo reale i dati riguardo le posizioni in classifica, alla gestione di una singola gara - ("**Gestione gara**") - che astrae le features necessarie a descrivere, condurre e terminare una gara.



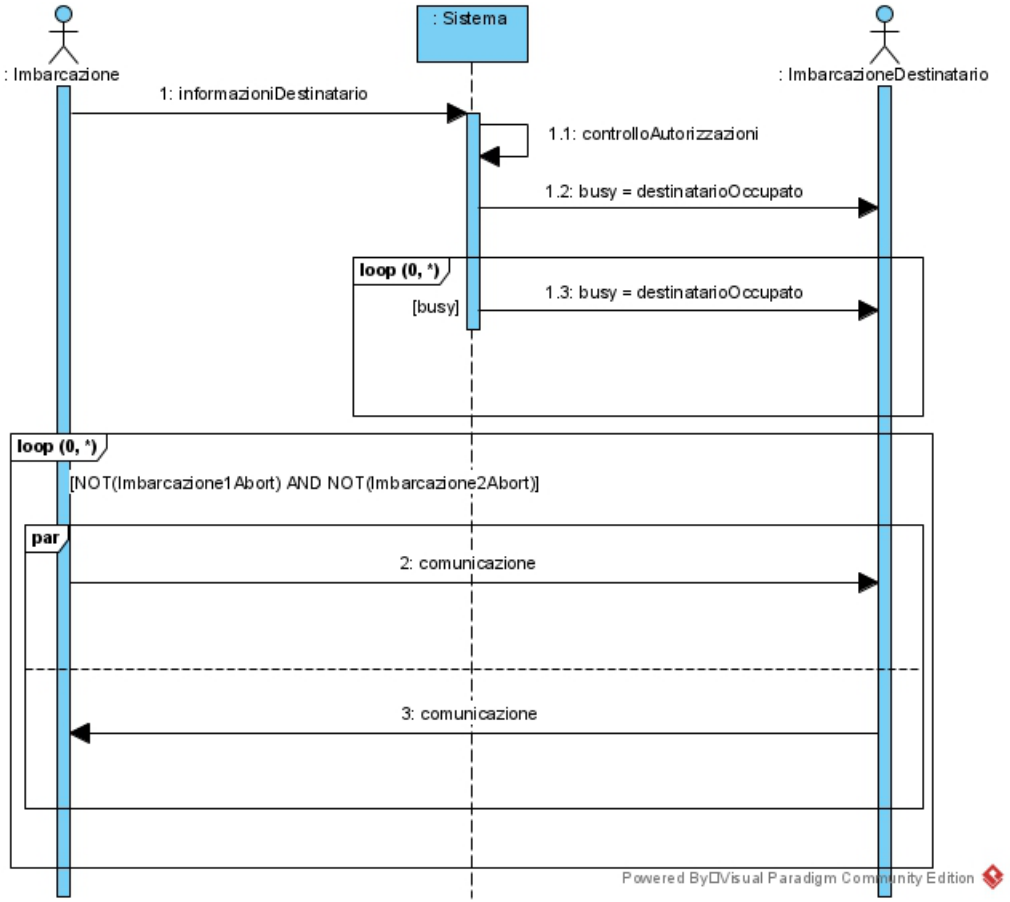
Narrativa.

Si fornisce la narrativa del caso d'uso "**Modifica bando**".

Nome	Modifica bando.
Breve Descrizione	Effettua la modifica del bando aggiornando il catalogo, invia una notifica a tutte le barche che si erano iscritte prima della modifica.
Attori Primari	Autorità Organizzatrice.
Attori Secondari	Nessuno.
Precondizioni	Il bando esiste all'interno del Sistema.
Sequenza degli eventi principale	Accerta che i nuovi dati siano corretti. Modifica il bando. Aggiorna il catalogo. Notifica la modifica del bando a tutti i suoi iscritti.
Postcondizioni	Il bando è stato modificato, tutte le barche che si sono iscritte prima della modifica hanno ricevuto una notifica.
Sequenze alternative degli eventi	Dati non corretti, modifica annullata.

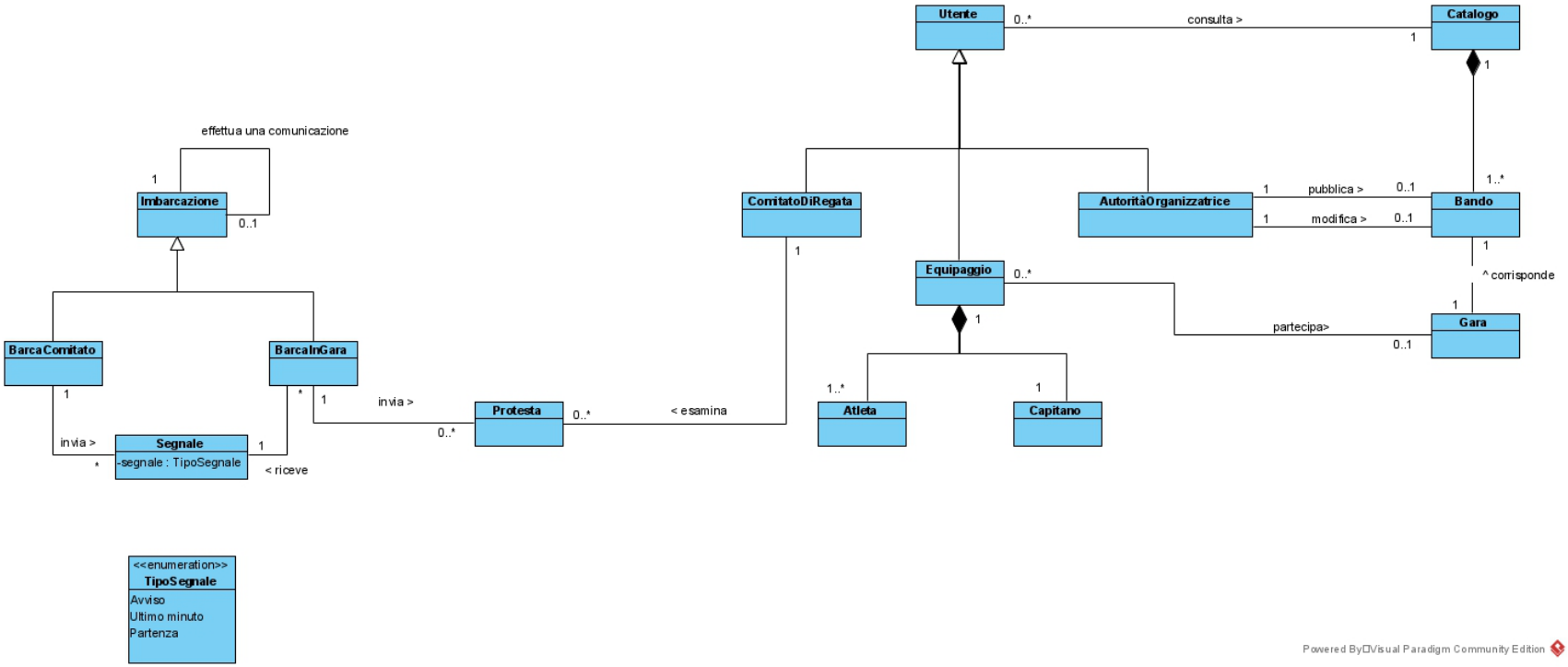
Esercizio 2

Viene fornito il diagramma di sequenza per "**Effettua una comunicazione**". Si sceglie di ripetere il tentativo di avviare una comunicazione con l'imbarcazione destinataria fino a che il mittente non riesce a connettersi rendendo il meccanismo quanto più possibile trasparente all'utente.



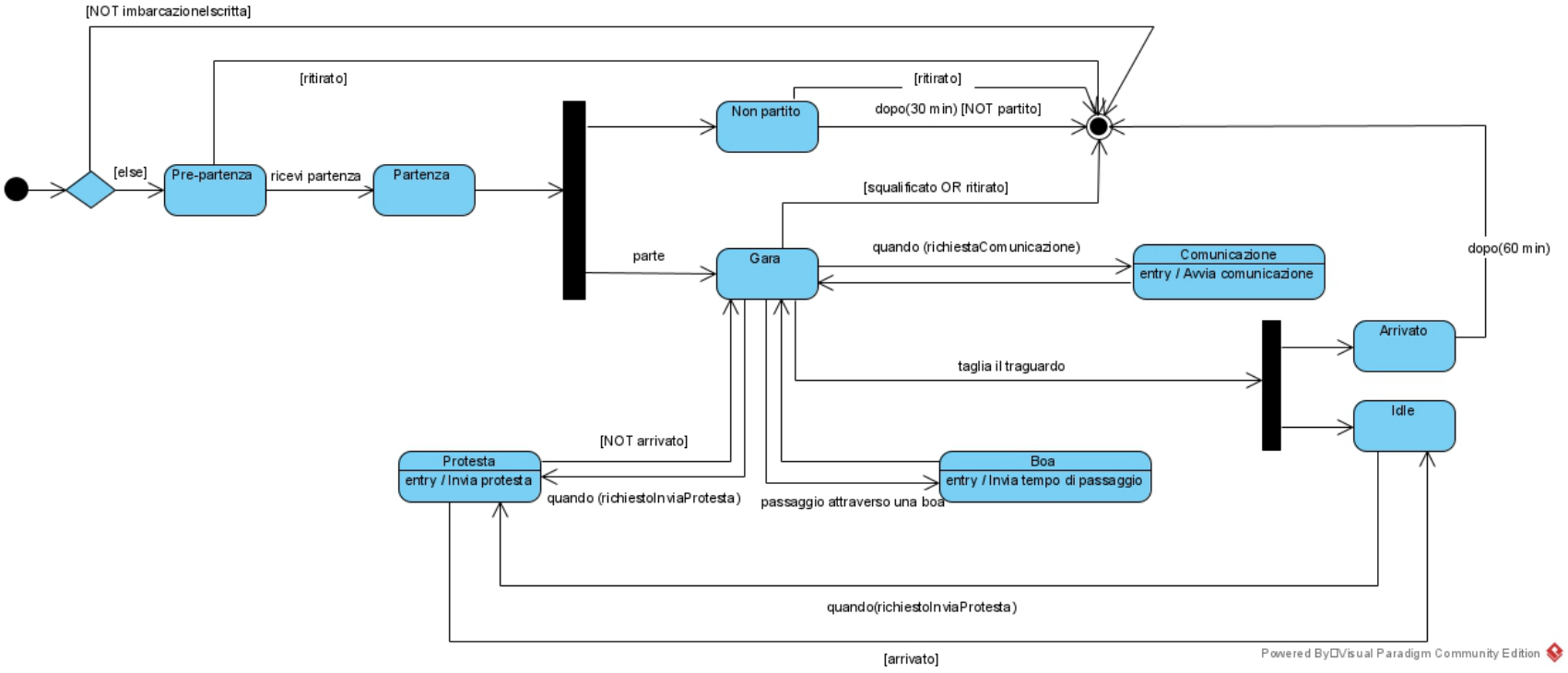
Esercizio 3

All'interno del diagramma delle classi vengono descritte le entità fondanti il dominio del progetto. Si sceglie - coerentemente con quanto descritto all'interno del diagramma dell'esercizio precedente - di modellare le comunicazioni come un qualcosa che riguarda solo le imbarcazioni coinvolte.



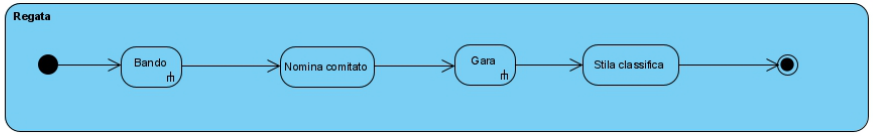
Esercizio 4

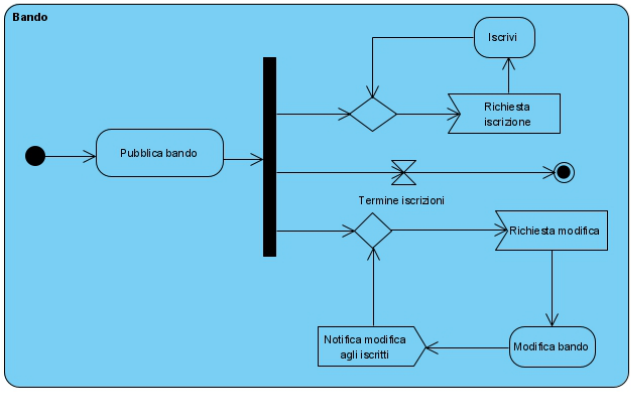
All'interno del diagramma si riconosce come stato fondamentale (in quanto quello di interesse ai fini dell'esercizio) "**Gara**", vero e proprio nodo nevralgico per tutte le interazioni che una imbarcazione può avere (i.e. effettuare comunicazioni, inviare proteste, inviare il tempo di passaggio alle boe e tagliare il traguardo). Seguendo la specifica, si sceglie di terminare la vita di una imbarcazione (relativamente a una regata) solo dopo un'ora dal momento in cui si è tagliato il traguardo poiché si deve permettere l'invio di proteste anche dopo l'arrivo (ma comunque entro un'ora da questo): per questo motivo si sceglie di adoperare uno stato "**Idle**" in modo da non uscire da "**Arrivato**" fino al timeout. La stessa logica si utilizza anche per segnare una imbarcazione come "*Non partita*" - flag che necessita di un altro timeout - a patto che siano passati 30 minuti dal momento in cui è arrivato il segnale di "*Partenza*" e questa non sia - appunto - ancora partita.

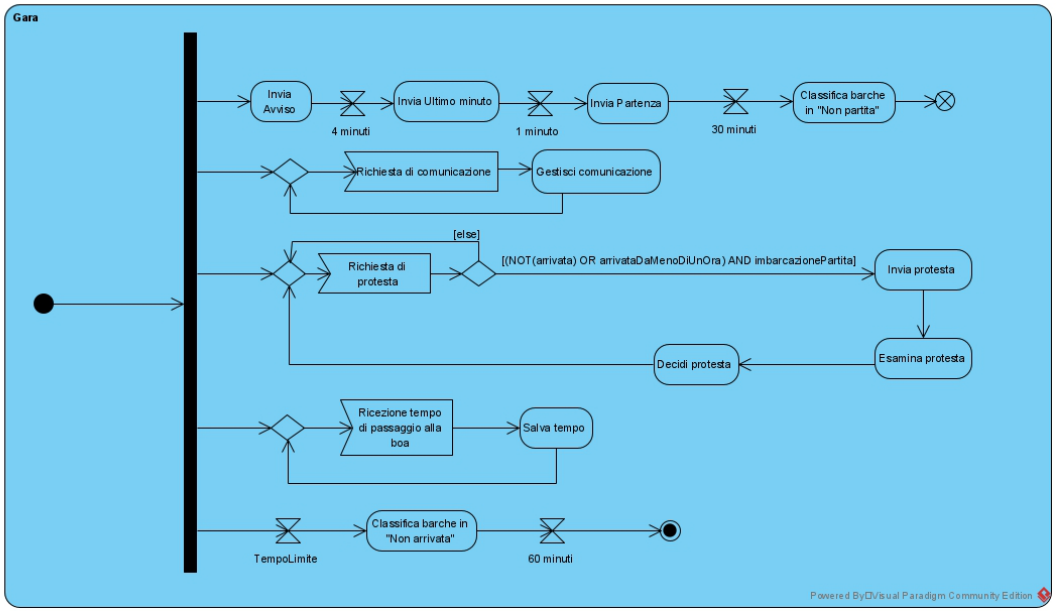


Esercizio 5

Si modellano le attività svolte dall'indizione del bando alla fine della regata. La prima sottoattività svolta - "**Bando**" - riguarda l'indizione di un bando e le operazioni che ne vengono implicate (i.e. la gestione delle iscrizioni, la possibilità di modificare un bando a patto che non si sia ancora raggiunto il termine delle iscrizioni); una volta nominato il comitato di gara (evento che si suppone avvenga al termine delle iscrizioni al bando e prima della regata) si passa a una seconda sottoattività - "**Gara**" - che modella lo svolgimento della regata (dalla gestione delle proteste a quella delle comunicazioni tra imbarcazioni, dai segnali di partenza a quelli per il passaggio alle boe); si conclude stilando la classifica delle imbarcazioni un'ora dopo il tempo limite.



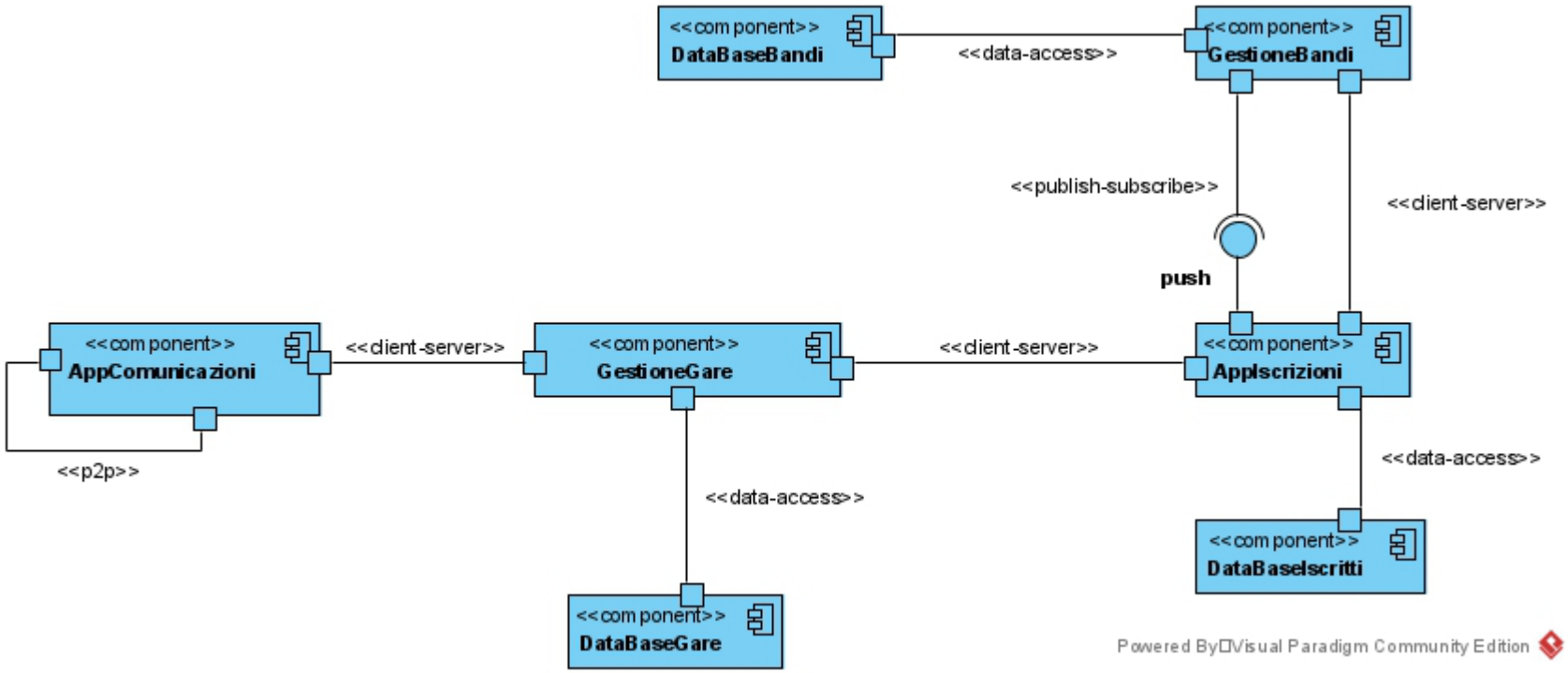




Esercizio 6

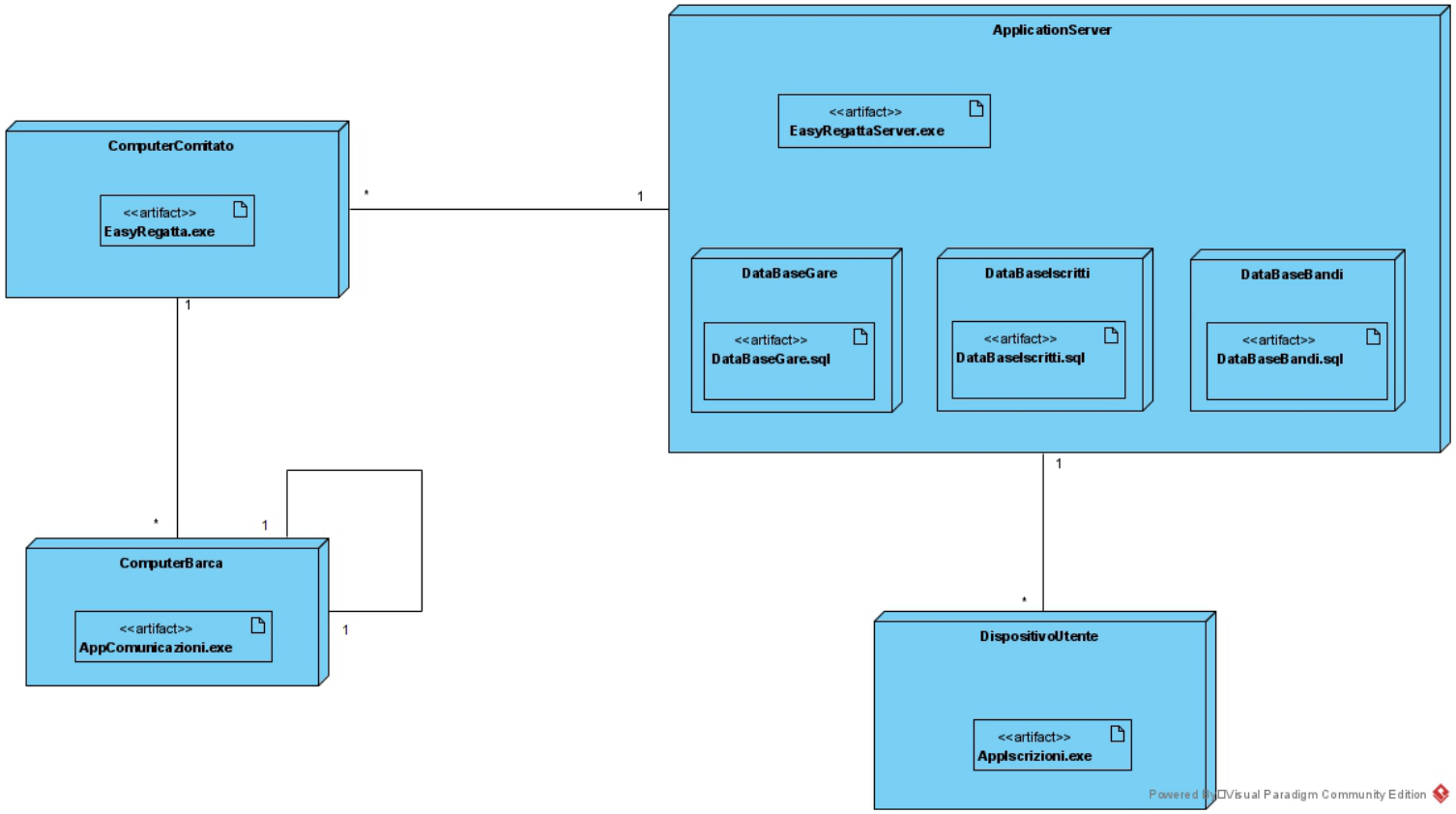
Vista componenti-connettori.

Si sceglie di dividere il sistema software *EasyRegatta* in due componenti fondamentali. La prima - "**GestioneBandi**" - si occupa della pubblicazione dei bandi (e il loro salvataggio) e permette agli utenti di consultarne comunicando con il database a questi dedicato - "**DataBaseBandi**" - abilitandone, inoltre, l'iscrizione attraverso "**AppIscrizioni**", componente con la quale si interfaccia anche per inviare notifiche agli (utenti) iscritti in seguito a modifiche; la seconda - "**GestioneGare**" - si occupa di seguire per intero una regata (i.e. stilando la classifica, permettendo di esaminare le proteste etc.), comunica con un database - "**DataBaseGare**" - nel quale salva i risultati della competizione e con "**AppComunicazioni**", utilizzata per gestire le proteste (la stessa componente viene utilizzata anche per le comunicazioni tra imbarcazioni senza che queste passino dalla componente centrale).



Vista di dislocazione.

A livello hardware si identificano 4 differenti ambienti di esecuzione: il server centrale - "**ApplicationServer**" - utilizzato anche per le basi di dati mantenute dal sistema; il dispositivo di un utente - "**DispositivoUtente**" - (e.g. uno smartphone, un computer etc.) su cui viene eseguito l'applicativo che consente l'iscrizione a un bando; il dispositivo utilizzato dal comitato di regata - "**ComputerComitato**" - che viene utilizzato durante lo svolgimento della gara (e.g. può essere utilizzato per la gestione delle proteste); il dispositivo utilizzato all'interno delle imbarcazioni - "**ComputerBarca**" - utilizzato per la gestione delle comunicazioni (tra imbarcazioni).



Esercizio 7

Per impostare il test, si individuano delle categorie secondo le quali partizionare i valori in input:

- nome := [a-zA-Z][a-zA-Z]+ | "" | null | NOT([a-zA-Z][a-zA-Z]+). Il primo pattern corrisponde alle stringhe che contengono solo spazi, lettere maiuscole e lettere minuscole (e che non iniziano con uno spazio). Si considera valido solo se (il match col primo pattern) ha successo. La notazione utilizzata per l'ultima è per identificare tutte le stringhe che falliscono il match con la prima espressione regolare.
- rating := $\{x : x \in Z \ \wedge \ x \in [-\infty; 0]\}$ | $\{x : x \in N \ \wedge \ x \neq 0\}$. Si considera valido solo per valori maggiori di zero.
- temporeale = $\{x : x \in Z \ \wedge \ x \in [-\infty; 0]\}$ | $\{x : x \in N \ \wedge \ x \neq 0\}$. Si considera valido solo per valori maggiori di zero.
- distanza = $\{x : x \in Z \ \wedge \ x \in [-\infty; 0]\}$ | $\{x : x \in N \ \wedge \ x \neq 0\}$. Si considera valido solo per valori maggiori di zero.
- Tempo Compensato = $\{x : x \in Z \ \wedge \ x \in [-\infty; 0]\}$ | $\{x : x \in N \ \wedge \ x \neq 0\}$. Si considera valido solo per valori maggiori di zero.
- rep[i] = null | definito. Si considera valido se tutti i campi sono validi.
- rep = null | [] | array con almeno un elemento. Si considera valido se tutti gli elementi sono validi.

Si distinguono dunque 384 possibili combinazioni, tra queste se ne scelgono 11, una per ogni caso non valido (il metodo fornito non li gestisce) + una valida. Si sceglie un criterio *white box*.

Si scrive la seguente batteria di test:

Input	**output atteso**
{<null, 5>}	errore: array null.
{<[],3>}	errore: array vuoto.
{<[{"Luca", 4.5, 10}, {"Antonio, 5.5, 11}, {"Mario", 1.2, 23}], -8>}	errore: distanza non positiva.
{<[{"Fabio", 2.3, 20}, {null, 4.1, 10}], 4>}	errore: rep[1].nome null.
{<[{"Francesco",-1.3,23}], {"Alberto",3.2, 14}], 2>}	errore: rep[0].rating non positivo.
{<[{"Luca", 4.5, -3}, {"Antonio, 5.5, 11}, {"Mario", 1.2, 23}], 4>}	errore: rep[0].tempoReale non positivo.
{<[{"Aldo", 4.5, 30}, {"", 3.4, 25}], 4>}	errore: rep[1].nome vuoto.
{<[{"Marco", 2.5, 31}, {"!!?", 3.5, 21}], 2>}	errore: rep[1].nome non valido.
{<[{"Giovanni", 9.1, 35}, null], 6>}	errore: rep[1] null.
{<[{"Federico", 2.5, 52}], 10>}	errore: Tempo Compensato non positivo.
{<[{"Giacomo", 4.5, 40}, {"Alberto", 1.2, 20}, {"Francesco", 2.3, 45}], 8>}	"Giacomo".

Si implementa il seguente test driver:

```
bool testDriver()
{
    Report [] r = new Report[3];
    r[0] = new Report("Giacomo", 4.5, 40);
    r[1] = new Report("Alberto", 1.2, 20);
    r[2] = new Report("Francesco", 2.3, 45);
    String v = calcolaVincitore(r, 8);
    return v.equals("Giacomo");
}
```