PROGETTAZIONE

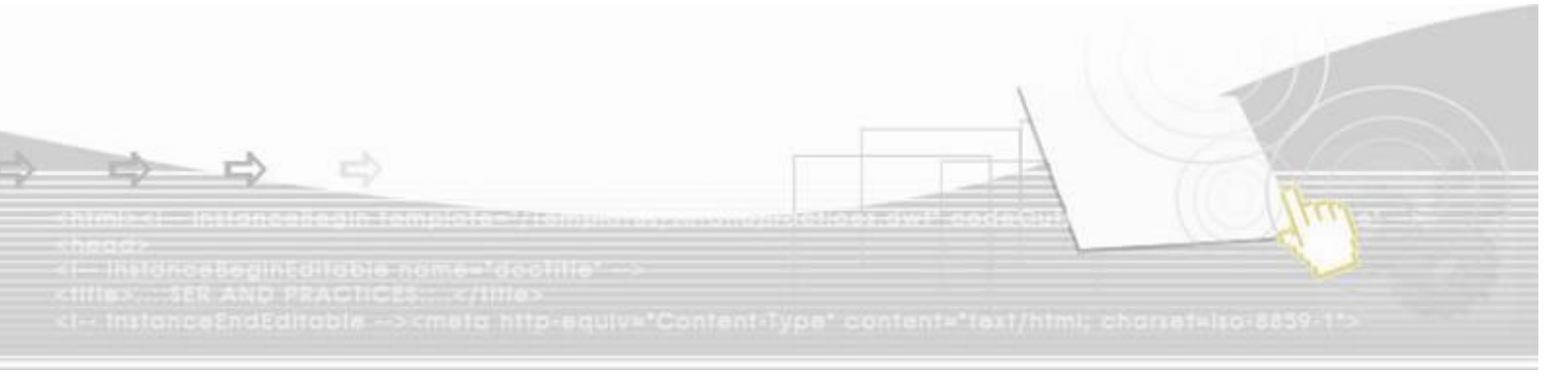
Nome del sistema: ADISysServer

Versione n°: 3.0

Data consegna: 09/03/2014

INGEGNERIA DEL SOFTWARE A.A. 2012-2013

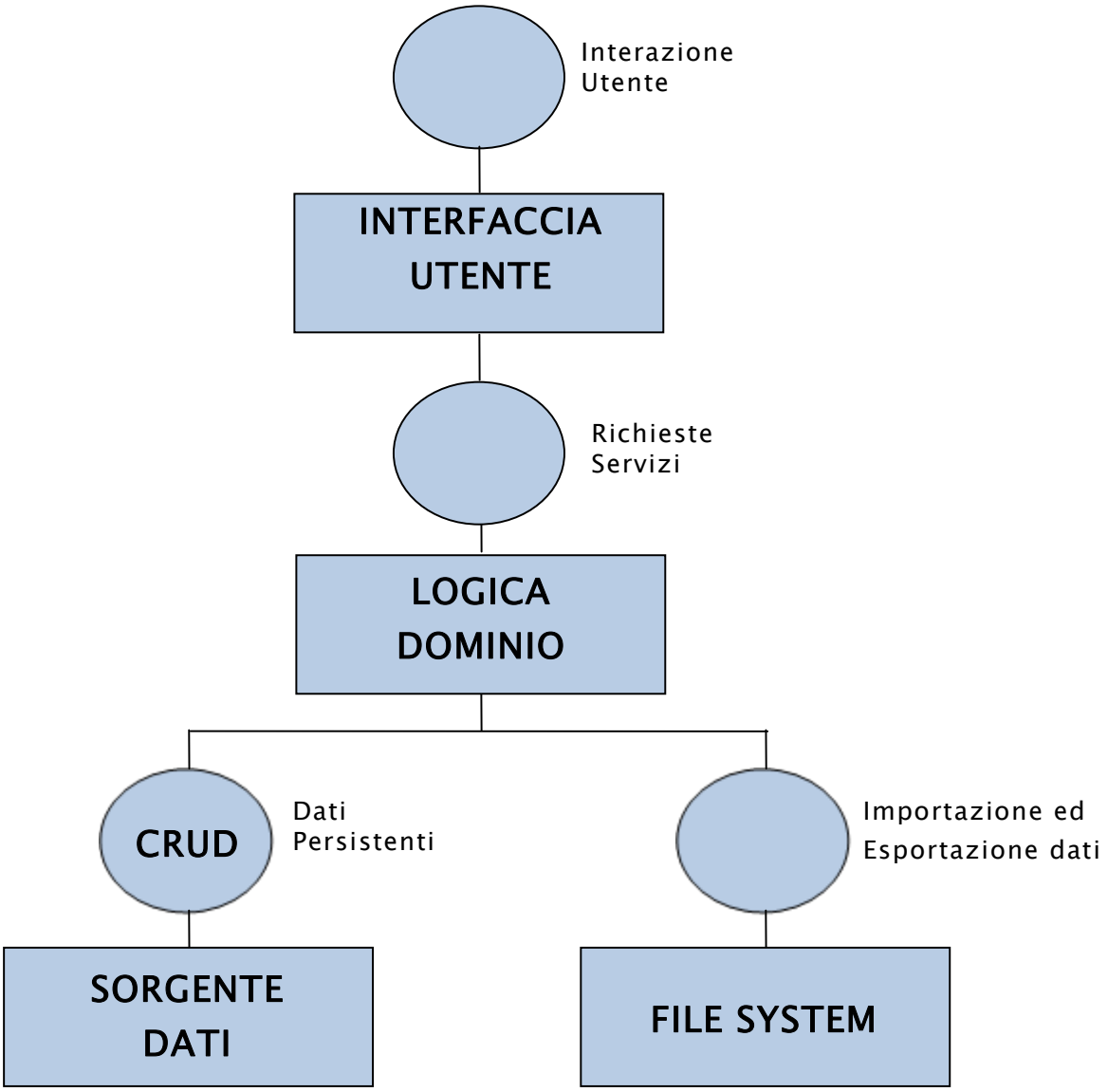
COGNOME	NOME	MATRICOLA	CORSO DI LAUREA	E-MAIL
DE TUGLIE	GIANLUCA	520343	ICD TA	GIANLUCADETUGLIE@MAIL.COM
RANDISI	MARCO	504385	ICD TA	MRANDISI@GMAIL.COM



INDICE

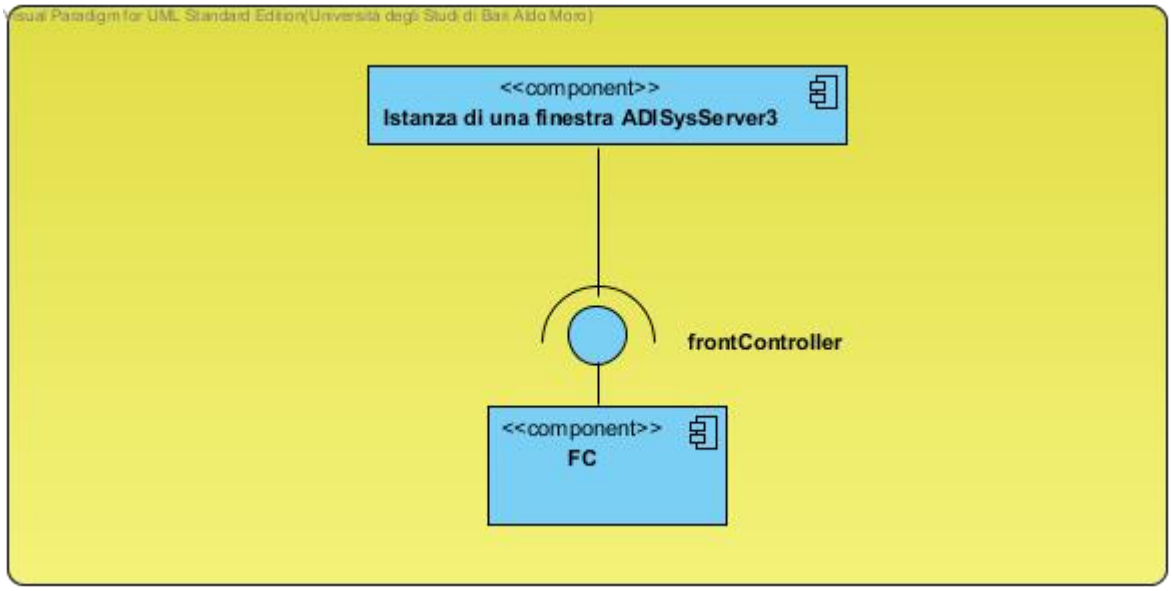
1. ARCHITETTURA

1.1 Livelli Architetturali

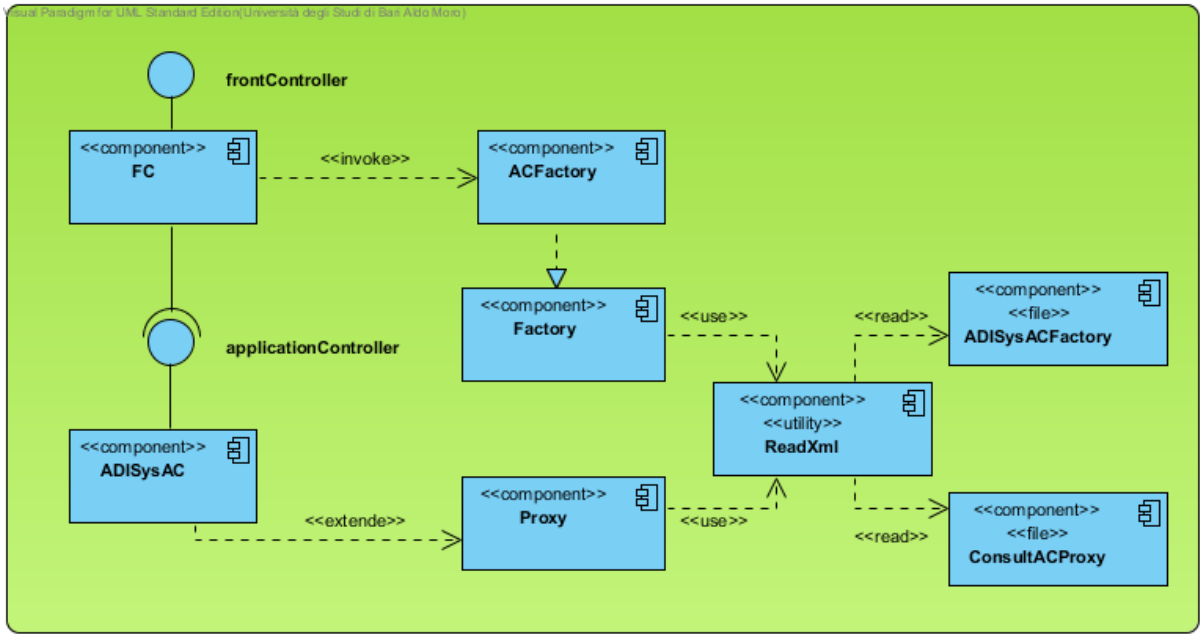


1.2 Diagramma delle Componenti

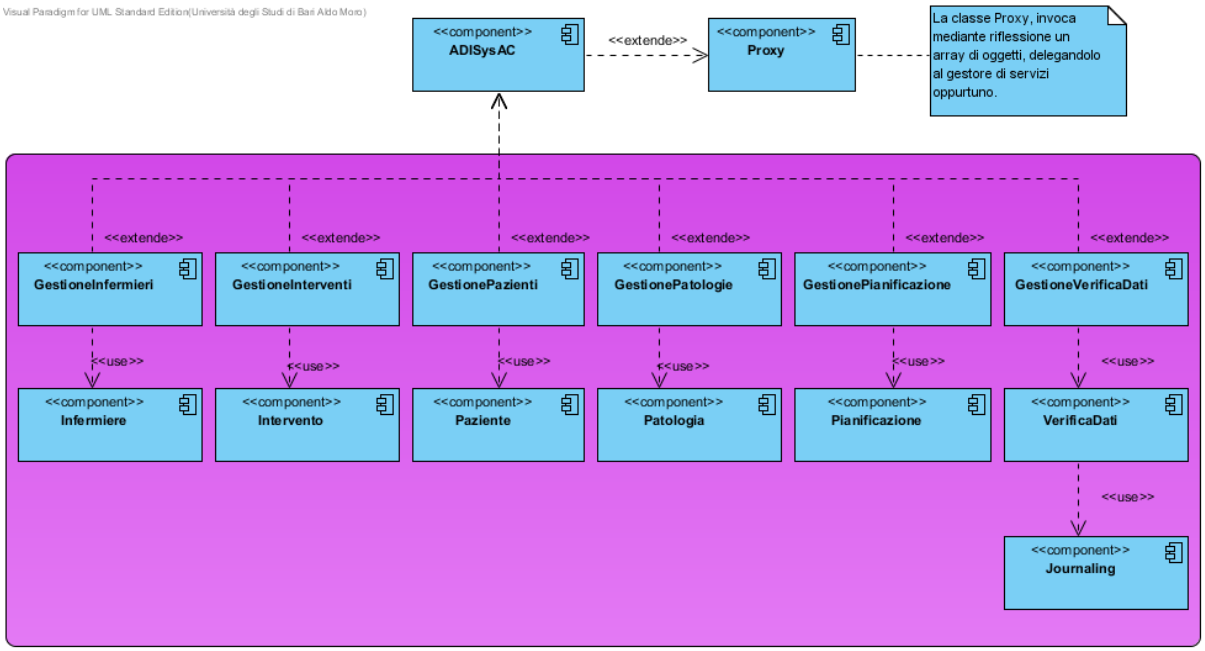
Livello di Presentazione



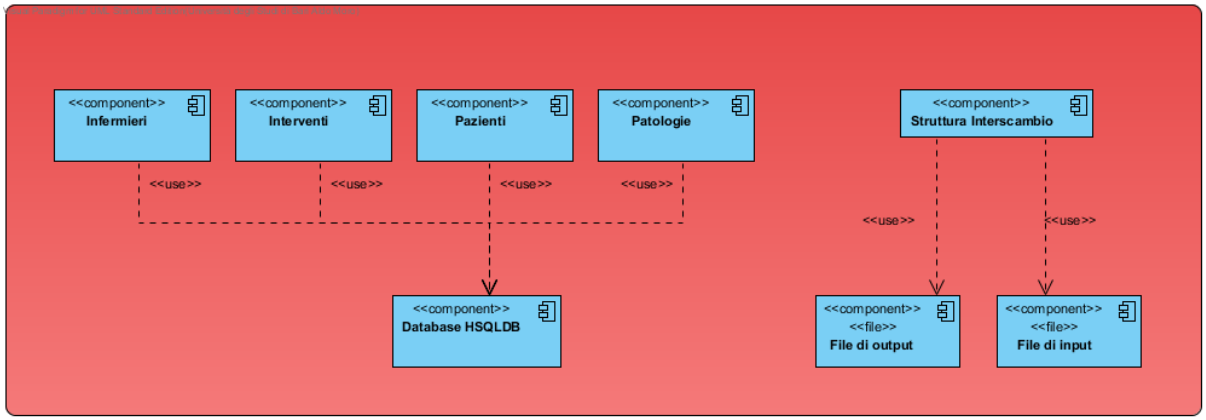
Front-end tra Presentation e Business



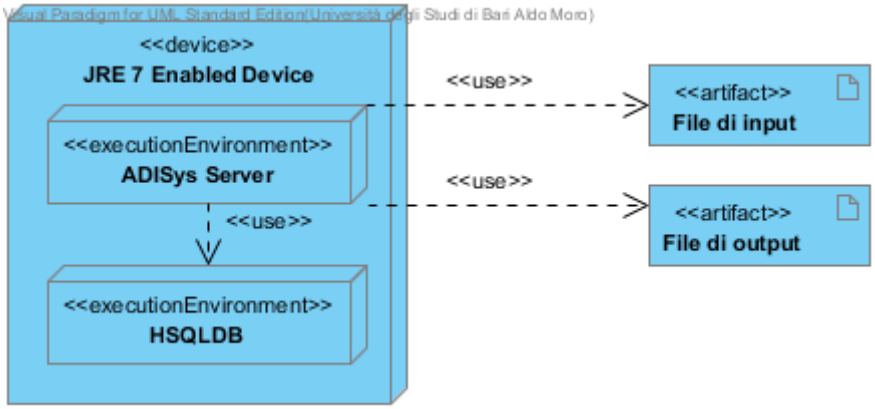
Livello di Business



Livello di Integrazione



1.3 Diagramma di Configurazione



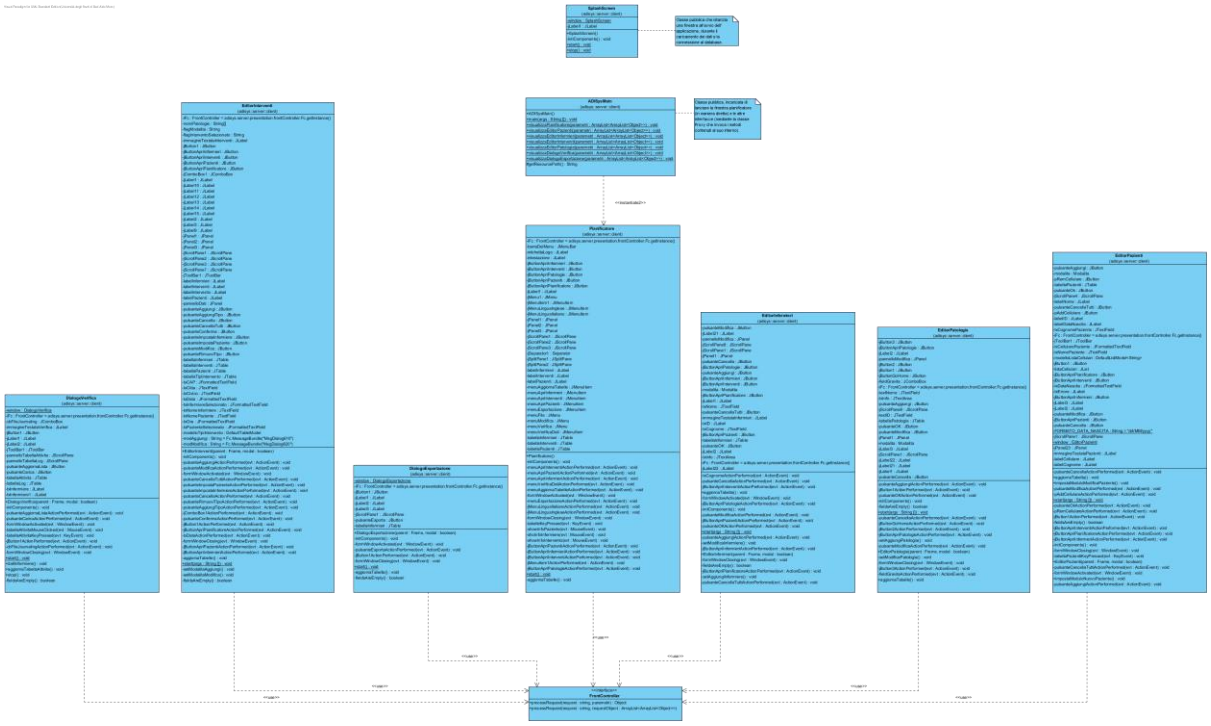
PROGETTO DI DETTAGLIO

Modifiche sulla versione precedente

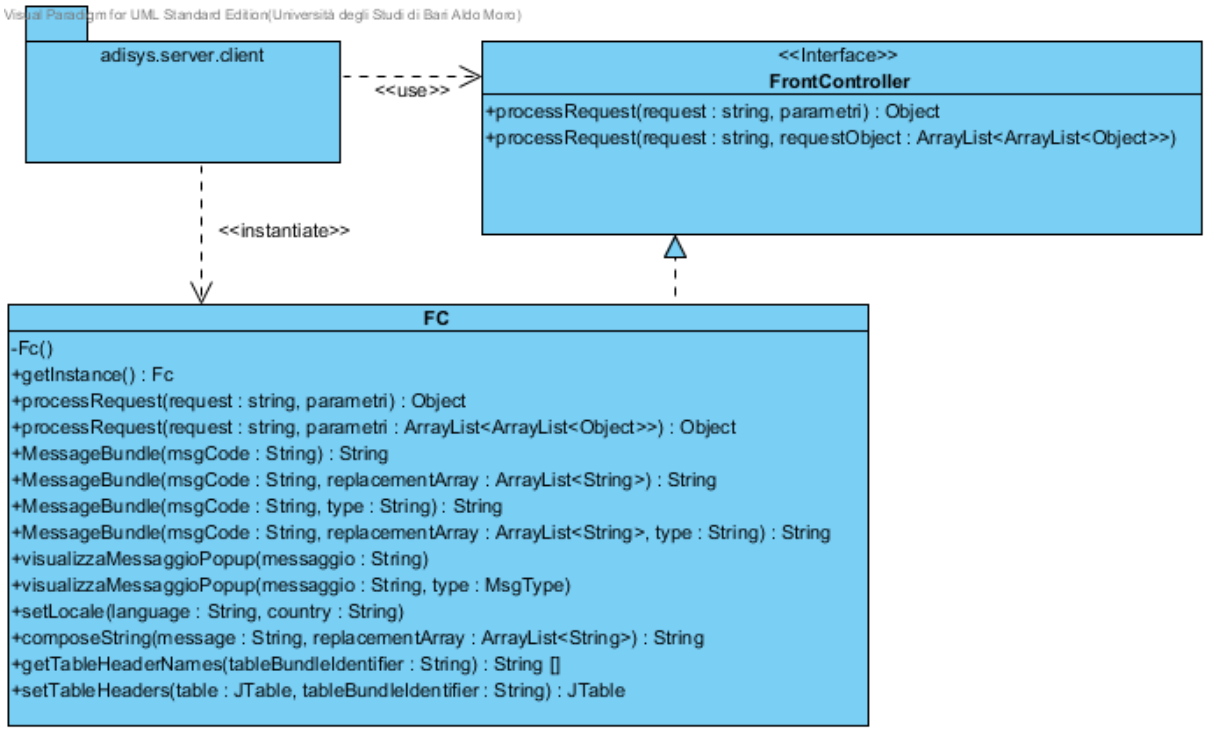
- La maggior parte delle modifiche sono state orientate alla conversione da un'organizzazione BCE a un paradigma per Tier di Java Enterprise. La modifica ha riguardato anzitutto lo scorporamento della classe ADISysController e divisione delle sue funzionalità in classi adibite a compiti specifici, tra i quali elaborazione Core (Business Object e verifica dati).
- E' stata implementata una ulteriore classe a livello client di nome ADISysMain, che ha il compito di lanciare l'interfaccia iniziale e tutte le altre interfacce GUI. In questa classe quindi, sono implementati tutti i metodi che servono a tale scopo.
- La classe ADISysControllerVerifica (come era chiamata nella versione precedente), è stata rinominata in BOGestioneADISysVerifica per questioni di coerenza e di logica, inerente a ciò che contiene.
- Si è scelto di convertire le classi Entity Infermiere, Paziente, Patologia, Intervento in Transfer Object rinominandole in TOPaziente, TOPatologia, TOIntervento, TOInfermiere mantenendo la relazione con le entità del database e trasferendo i metodi di elaborazione nelle classi specifiche.
- Sempre a livello di business, si è scelto di implementare una ulteriore classe per ogni oggetto BO (eccetto BOPianificazione e BOGestioneVerifica), che verifichi la coerenza dei dati. La scomposizione quindi, separa i metodi di verifica (contenuti precedentemente nelle varie classi di business Infermiere, Paziente, Intervento, Patologia, da noi convertite in "TO") dagli altri.
- La classe Journaling è stata spostata nella sezione di business, poiché si occupa della gestione delle informazioni di journaling demandando alla sezione di integrazione la memorizzazione dei dati.
- Per centralizzare anche la gestione dei messaggi, sono stati inseriti dei metodi appositi, nella classe FC che implementa l'interfaccia FrontController.

1.4 Diagramma delle Classi

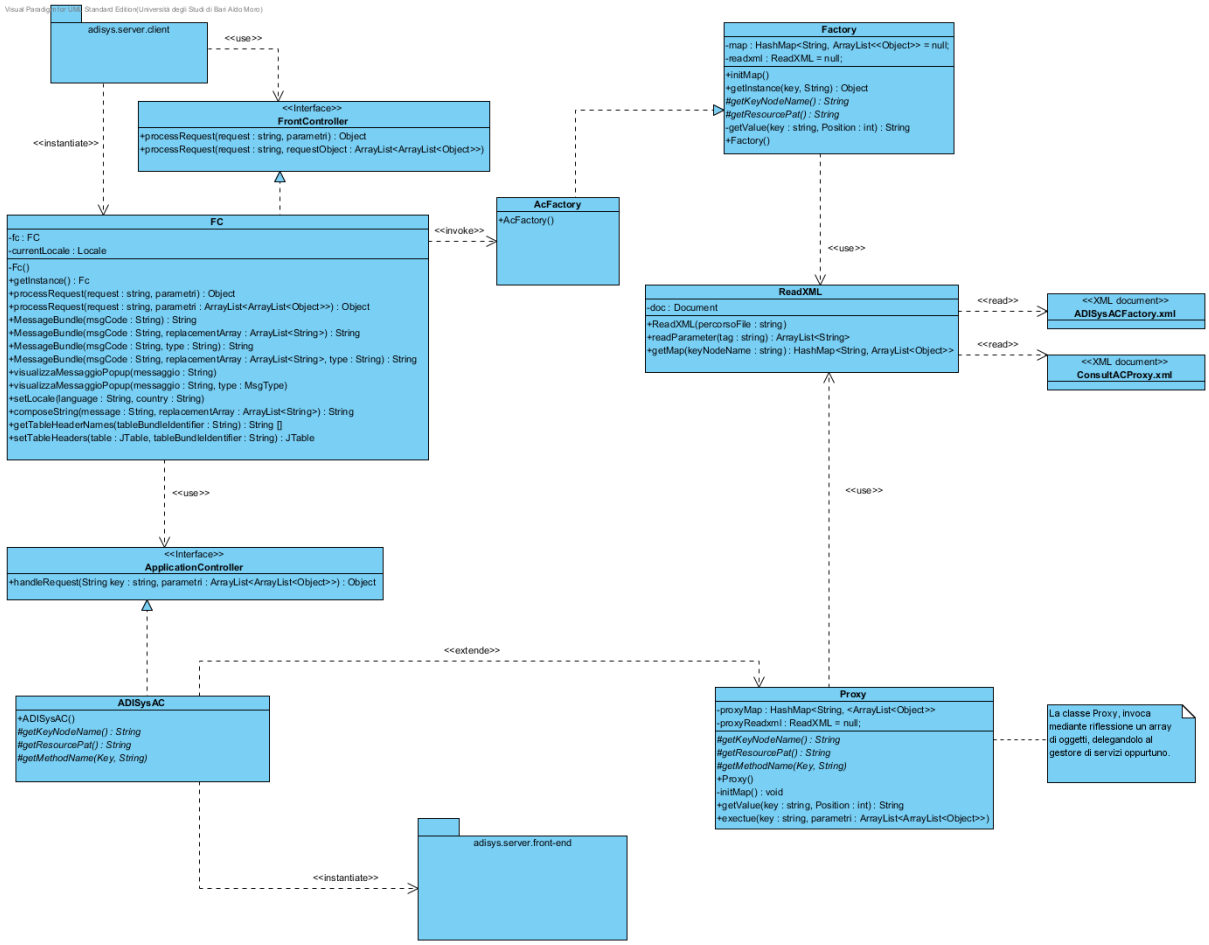
Client



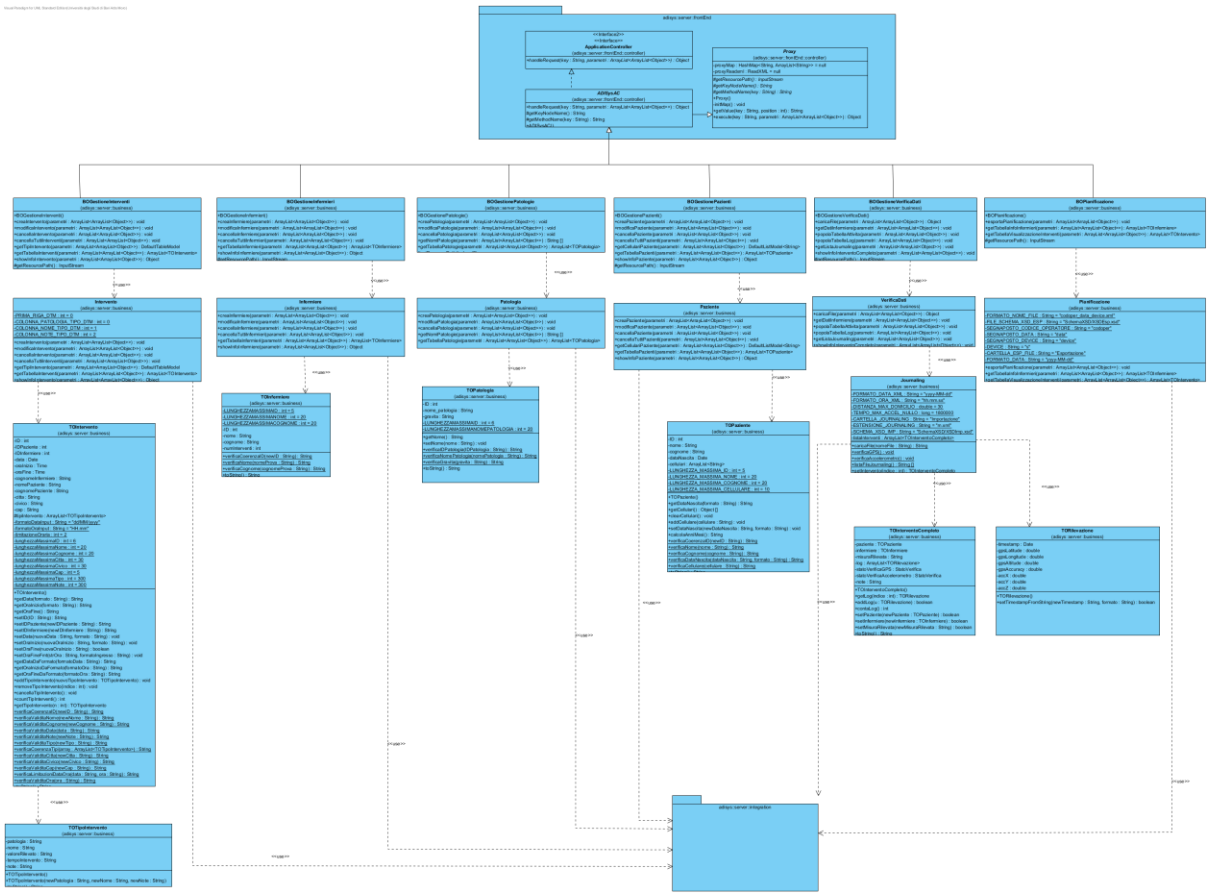
Livello di Presentazione



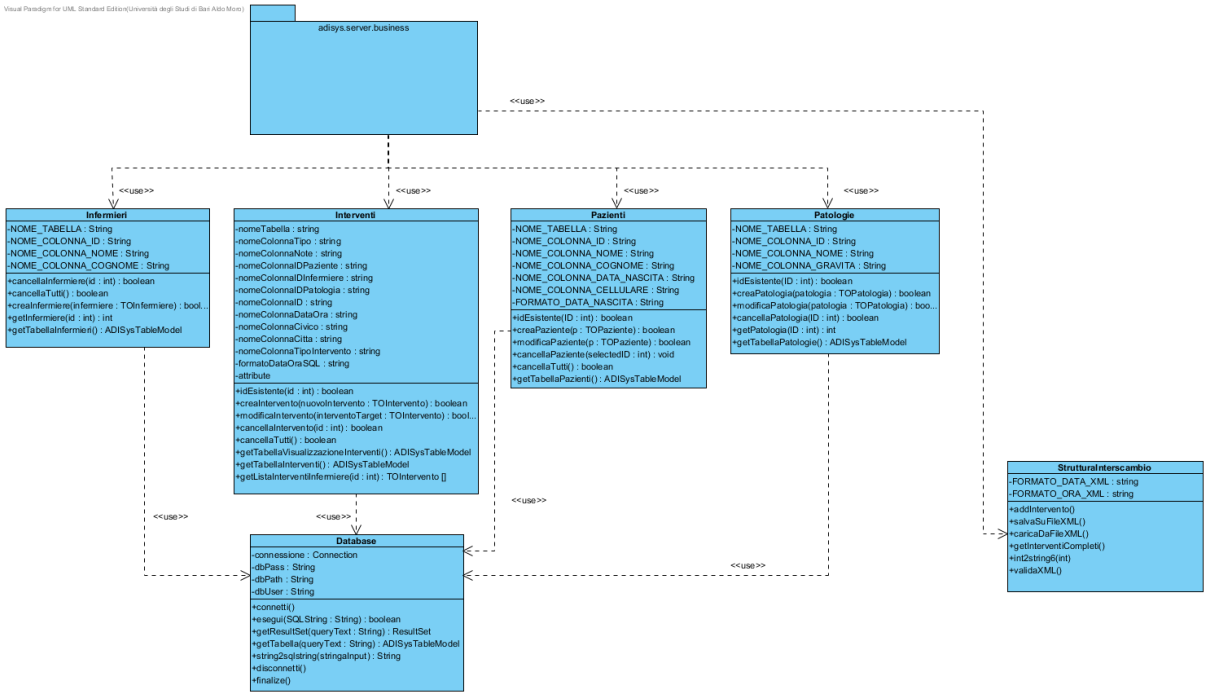
Front-end tra Presentazione e Business



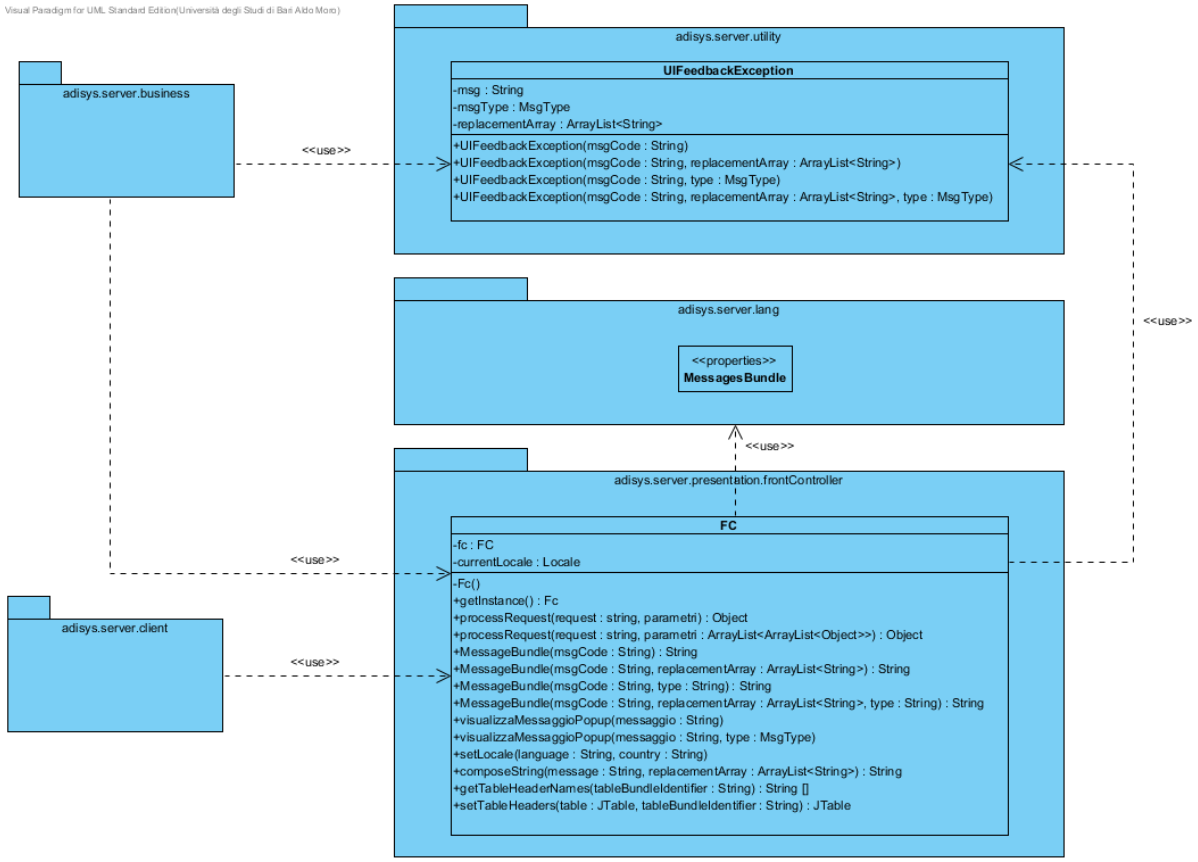
Livello di Business



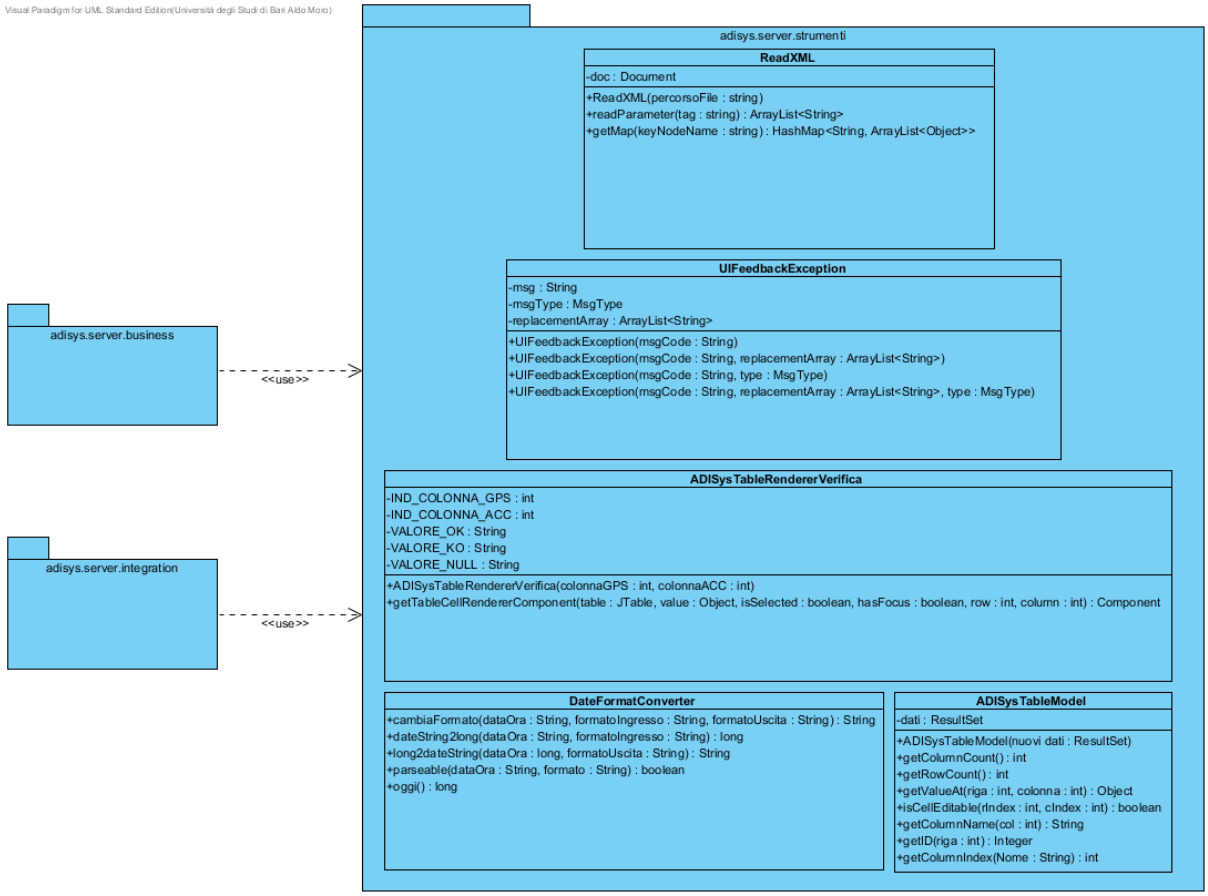
Livello di Integrazione



Gestione messaggistica



Strumenti di utility



1.5 Specifiche delle classi

1.5.1 Livello Client

SplashScreen

Classe pubblica che istanzia una finestra all'avvio dell'applicazione, durante il caricamento dei dati e la connessione al database.

ADISysMain

Classe pubblica, incaricata di lanciare la finestra pianificatore e le altre interfacce.

Metodi:

- visualizzaPianificatore(ArrayList<ArrayList<Object>>)
- visualizzaEditorInfermieri(ArrayList<ArrayList<Object>>)
- visualizzaEditorInterventi(ArrayList<ArrayList<Object>>)
- visualizzaEditorPazienti(ArrayList<ArrayList<Object>>)
- visualizzaEditorPatologia(ArrayList<ArrayList<Object>>)
- visualizzaDialogoEsportazione(ArrayList<ArrayList<Object>>)
- visualizzaDialogoVerifica(ArrayList<ArrayList<Object>>)

Pianificatore

Istanzia la finestra principale contenente la visualizzazione globale dei dati di pazienti, infermieri ed interventi. Dispone di menu che richiamano tramite il Front Controller (metodo getBoundary) le finestre di dialogo destinate all'aggiornamento dei dati (Editor), all'esportazione del file di scambio (DialogoEsportazione) e alla sua importazione per la verifica (DialogoVerifica).
Pianificatore dispone anche della possibilità di cambiare lingua.

Editor Infermieri

Finestra che permette di inserire, modificare o cancellare i dati relativi a uno o a tutti gli infermieri. E' possibile interfacciarsi con la finestra EditorInterventi.

Editor Pazienti

Finestra che permette di inserire, modificare o cancellare i dati relativi a uno o a tutti i pazienti. E' possibile interfacciarsi con la finestra EditorInterventi.

Editor Interventi

Finestra che permette di inserire, modificare o cancellare i dati relativi a uno o a tutti gli interventi. Al suo interno è possibile interfacciarsi (per poter inserire, modificare o cancellare i dati) con le finestre EditorPazienti, EditorInfermieri, EditorPatologie.

Editor Patologie

Finestra che permette di inserire, modificare o cancellare i dati relativi a una patologia. Non è possibile cancellare tutte le patologie. La finestra Editor Patologie, è visualizzabile solo mediante Editor Interventi. E' possibile interfacciarsi con la finestra EditorInterventi.

Dialogo Esportazione

Finestra che permette l'esportazione dei dati di una lista di interventi, che ogni infermiere deve compiere.

Dialogo Verifica

Finestra contenente la visualizzazione dei contenuti di un file di interscambio prodotto da ADISys Mobile con i risultati grafici e numerici dell'analisi dei dati dell'accelerometro e GPS.

NB. Tutte le finestre sono dotate di un pulsante "home" e di un pulsante "exit". Il pulsante home, serve per tornare al menù Pianificazione. Il pulsante exit, serve per uscire dall'applicazione in qualsiasi punto ci si trovi.

1.5.2 Livello di Presentazione

FrontController

Interfaccia utilizzata dal Front Controller per il livello di presentazione dell'applicazione.

Metodi: (vedere FC)

FC

Classe derivata dal Design Pattern che gestisce le richieste provenienti dall'interfaccia grafica e si occupa della comunicazione con i componenti di business tramite AdisysAC.

Metodi:

- processRequest:(String)
  utilizzato per gestire le richieste di business; in input riceve un stringa che identifica il metodo che si vuole utilizzare. Questo metodo, fa riferimento alla processRequest contenente un ArrayList bidimensionale di Oggetti, e viene utilizzato solo nel caso in cui nell'ArrayList non ci sono parametri da passare, ma solo una stringa.
- processRequest:(String,ArrayList<ArrayList<Object>>)
  utilizzato per gestire le richieste di business; in input riceve un stringa che identifica il metodo che si vuole utilizzare e un ArrayList bidimensionale di Oggetti, dove vengono passati i parametri che il metodo utilizza.
- messagesBundle(String)
  metodo utilizzato per gestire la messaggistica; in input riceva una stringa contenente il codice identificativo di un messaggio, che verrà poi visualizzato mediante VisualizzaMessaggioPopup.
- messagesBundle(String,ArrayList<String>)
  metodo utilizzato per gestire la messaggistica; in input riceva una stringa contenente il codice identificativo di un messaggio e un arrayList contenente dei parametri, che comporranno la stringa finale che verrà poi visualizzato mediante VisualizzaMessaggioPopup.
- messagesBundle(String,String)
  metodo utilizzato per gestire la messaggistica; è invocato quando,

oltre al semplice messaggio, viene inviata anche una stringa in cui è specificato il tipo di visualizzazione del messaggio da JOptionPane (ERROR,WARNING,NOTIFY).

- messagesBundle(String,ArrayList<String>,String)
  metodo utilizzato per gestire la messaggistica; è invocato quando, oltre al semplice messaggio e all'ArrayList di stringhe, viene inviata anche una stringa in cui è specificato il tipo di visualizzazione del messaggio da JOptionPane (ERROR,WARNING,NOTIFY).
- setLocale(String,String)
  metodo statico utilizzato per settare la Lingua. Utilizza due stringhe, contenenti Lingua e Paese. In base alla scelta della lingua, nella finestra Pianificatore, seleziona il file .properties opportuno.
- visualizzaMessaggioPopup(String)
  metodo utilizzato per stampare a video, mediante una JOptionPane, il messaggio di feedback, che arriva sottoforma di stringa.
- visualizzaMessaggioPopup(String,MsgType)
  metodo utilizzato per stampare a video, mediante una JOptionPane, il messaggio di feedback, che arriva sottoforma di stringa. Oltre alla stringa, questo metodo ha anche una MsgType, che imposta il tipo di visualizzazione del pannello, per mezzo di una if, che implementa i vari casi (NOTIFY,WARNING,ERROR).
- ComposeString(String,ArrayList<String>)
  metodo che permette di implementare una stringa composta. La stringa, verrà poi visualizzata mediante VisualizzaMessaggioPupop.
- getTableHeaderName(String)
- setTableHeaders(JTable,String)

ACFactory

Classe che estende le caratteristiche del Factory; viene utilizzato dal Front Controller per capire a quale Application Controller si deve rifare per il metodo richiesto.

Metodi

- getResourcePath():metodo che restituisce il percorso del file XML contenente le informazioni da caricare nella mappa map.
- getKeyNodeName():metodo che restituisce il nome del nodo da cui estarre le informazioni.

1.5.3 frontEnd

Factory

Classe astratta che viene estesa dall'ACFactory e viene utilizzato per gestire le richieste di business dell'applicazione.

Metodi:

- initMap():metodo privato che si occupa dell'inizializzazione di map attraverso il caricamento dei dati dal file XML.
- getInstance(String): Si occupa di recuperare l'istanza della classe la cui chiave è in map key, scopo proprio del pattern Factory Method.
  Nel dettaglio il metodo recupera il nome della classe di cui si desidera l'istanza, e mediante riflessione, la crea e la istanzia dando come risultato un generico Object.
- getValue(String, int): metodo che si occupa del recupero del valore contenuto in posizione position che ha come chiave key.
- getResourcePath():metodo che restituisce il percorso del file XML contenente le informazioni da caricare nella mappa map.
- getKeyNodeName():metodo che restituisce il nome del nodo da cui estarre le informazioni.

ApplicationController

Interfaccia utilizzata dall'ADISysAC per il livello di business dell'applicazione.

ADISysAC

Classe che estende Proxy ed Implementa l'Interfaccia applicationController; viene utilizzato per capire, tramite l'ausilio del Proxy, la classe da istanziare per usufruire del metodo richiesto. Crea una sottoclasse della classe Proxy.

Metodi:

- handleRequest(String, ArrayList<ArrayList<Object>>): metodo che invoca, attraverso l'ausilio di un file XML, la classe che genera il metodo richiesto.

- getResourcePath():metodo che restituisce il percorso del file XML contenente le informazioni da caricare nella mappa map.
- getKeyNodeName():metodo che restituisce il nome del nodo da cui estarre le informazioni.

Proxy

Classe astratta delle classi di servizio del sistema. Usa ReadXml per leggere dal file Xml il metodo. E' estesa dalla classe ADISysAC, che la utilizza per interfacciarsi con le classi di business mediante il metodo execute.

Metodi:

- getResourcePath(): metodo che restituisce il percorso del file XML contenente le informazioni da caricare nella mappa proxyMap.
- getKeyNodeName(): metodo che restituisce il nome del nodo da cui estrarre le informazioni.
- getMethodName(String): metodo che in base alla chiave "key", restituisce il nome del metodo da invocare.
- initMap(): metodo privato che si occupa dell'inizializzazione di proxyMap attraverso il caricamento dei dati dal file XML.
- Proxy(): Costruttore della classe Proxy , che si occupa dell'avvaloramento dei due attributi.
- getValue(String,int): Metodo pubblico che si occupa del recupero da proxyMap del valore contenuto in posizione " position" che ha chiave " key".
- execute(String, ArrayList<ArrayList<Object>>): Metodo pubblico che si occupa di invocare mediante riflessione il metodo la cui chiave in proxyMap è "key" e i cui parametri sono contenuti in "parametri" e che quindi racchiude in sé la logica del pattern. Nel dettaglio il metodo :

  1. Recupera il nome della classe, se il nome è "this" l'invocazione viene fatta direttamente su "this" altrimenti la classe viene creata ed istanziata e su questa avviene l'invocazione;
  2. Effettua il controllo sul numero dei parametri, in base al quale avremo invocazioni differenti;

3. Effettua il controllo sul campo static, se il metodo è statico l'invocazione viene fatta sulla classe, altrimenti sull'oggetto. Il metodo gestisce in entrata ed in uscita un array di Object, parametri, questo ci permette di poter lavorare su oggetti di qualsiasi tipo rendendo il metodo riusabile in qualsiasi contesto.

1.5.4 Livello di Business

GestioneInfermieri

Classe che riceve dalla Proxy i metodi inerenti alla gestione degli infermieri, inviandoli alla classe Infermiere.

Metodi:

- creaInfermiere(ArrayList<ArrayList<Object>>)
- modificaInfermiere(ArrayList<ArrayList<Object>>)
- cancellaInfermiere(ArrayList<ArrayList<Object>>)
- cancellaTuttiInfermieri()
- getTabellaInfermieri(ArrayList<ArrayList<Object>>)

Infermiere

Classe di business incaricata di gestire i metodi inerenti agli infermieri.

Metodi:

- creaInfermiere(ArrayList<ArrayList<Object>>)
- modificaInfermiere(ArrayList<ArrayList<Object>>)
- cancellaInfermiere(ArrayList<ArrayList<Object>>)
- cancellaTuttiInfermieri()
- getTabellaInfermieri(ArrayList<ArrayList<Object>>)

GestioneInterventi

Classe che riceve dalla Proxy i metodi inerenti alla gestione degli interventi, inviandoli alla classe Infermiere.

Metodi:

- creaIntervento(ArrayList<ArrayList<Object>>)
- modificaIntervento(ArrayList<ArrayList<Object>>)
- cancellaIntervento(ArrayList<ArrayList<Object>>)
- cancellaTuttiInterventi()

- getTipiIntervento(ArrayList<ArrayList<Object>>)
- getTabellaInterventi(ArrayList<ArrayList<Object>>)

Intervento

Classe di business incaricata di gestire i metodi inerenti agli interventi.

Metodi:

- creaIntervento(ArrayList<ArrayList<Object>>)
- modificaIntervento(ArrayList<ArrayList<Object>>)
- cancellaIntervento(ArrayList<ArrayList<Object>>)
- cancellaTuttiInterventi()
- getTipiIntervento(ArrayList<ArrayList<Object>>)
- getTabellaInterventi(ArrayList<ArrayList<Object>>)

GestionePazienti

Classe che riceve dalla Proxy i metodi inerenti alla gestione dei pazienti, inviandoli alla classe Paziente.

Metodi:

- creaPaziente(ArrayList<ArrayList<Object>>)
- modificaPaziente(ArrayList<ArrayList<Object>>)
- cancellaPaziente(ArrayList<ArrayList<Object>>)
- cancellaTuttiPazienti()
- getCellulariPaziente(ArrayList<ArrayList<Object>>)
- getTabellaPazienti(ArrayList<ArrayList<Object>>)
- ShowInfoPaziente(ArrayList<ArrayList<Object>>)

Paziente

Classe di business incaricata di gestire i metodi inerenti ai pazienti.

Metodi:

- creaPaziente(ArrayList<ArrayList<Object>>)
- modificaPaziente(ArrayList<ArrayList<Object>>)
- cancellaPaziente(ArrayList<ArrayList<Object>>)
- cancellaTuttiPazienti()

- getCellulariPaziente(ArrayList<ArrayList<Object>>)
- getTabellaPazienti(ArrayList<ArrayList<Object>>)
- ShowInfoPaziente(ArrayList<ArrayList<Object>>)

GestionePatologie

Classe che riceve dalla Proxy i metodi inerenti alla gestione delle patologie, inviandoli alla classe Patologia.

Metodi:

- creaPatologia(ArrayList<ArrayList<Object>>)
- modificaPatologia(ArrayList<ArrayList<Object>>)
- cancellaPatologia(ArrayList<ArrayList<Object>>)
- getTabellaPatologie(ArrayList<ArrayList<Object>>)
- getNomiPatologie(ArrayList<ArrayList<Object>>)

Patologia

Classe di business incaricata di gestire i metodi inerenti alle patologie.

Metodi:

- creaPatologia(ArrayList<ArrayList<Object>>)
- modificaPatologia(ArrayList<ArrayList<Object>>)
- cancellaPatologia(ArrayList<ArrayList<Object>>)
- getTabellaPatologie(ArrayList<ArrayList<Object>>)
- getNomiPatologie(ArrayList<ArrayList<Object>>)

GestionePianificazione

Classe che riceve dalla Proxy i metodi inerenti alla gestione dell'esportazione del file di journaling, inviandoli alla classe Pianificazione.

Metodi:

- esportaPianificazione(ArrayList<ArrayList<Object>>)
- getTabellaInfoInfermieri(ArrayList<ArrayList<Object>>)
- getTabellaVisualizzazioneInterventi(ArrayList<ArrayList<Object>>)

Pianificazione

Classe di business incaricata di gestire i metodi inerenti all'esportazione del file di journaling.

Metodi:

- esportaPianificazione(ArrayList<ArrayList<Object>>)
- getTabellaInfoInfermieri(ArrayList<ArrayList<Object>>)
- getTabellaVisualizzazioneInterventi(ArrayList<ArrayList<Object>>)

GestioneVerificaDati

Classe che riceve dalla Proxy i metodi inerenti alla verifica del file di journaling, inviandoli alla classe VerificaDati.

Metodi:

- caricaFile(ArrayList<ArrayList<Object>>)
- getDatiInfermieri(ArrayList<ArrayList<Object>>)
- PopolaTabellaAttivita(ArrayList<ArrayList<Object>>)
- PopolaTabellaLog(ArrayList<ArrayList<Object>>)
- getListaJournaling(ArrayList<ArrayList<Object>>)
- showInfoInterventoCompleto(ArrayList<ArrayList<Object>>)

VerificaDati

Classe di business incaricata di gestire i metodi inerenti alla verifica dei dati del file di journaling.

Metodi:

- caricaFile(ArrayList<ArrayList<Object>>)
- getDatiInfermieri(ArrayList<ArrayList<Object>>)
- PopolaTabellaAttivita(ArrayList<ArrayList<Object>>)
- PopolaTabellaLog(ArrayList<ArrayList<Object>>)
- getListaJournaling(ArrayList<ArrayList<Object>>)
- showInfoInterventoCompleto(ArrayList<ArrayList<Object>>)

Journaling

Classe incaricata dell'importazione del file di journaling.

Metodi:

- caricaFile (String): Metodo principale che legge il file dal percorso specificato in input e prepara i dati per la verifica.
- verificaGPS(): rileva gli alert causati dai dati del GPS se condo le specifiche.
- verificaAccelerometro(): rileva gli alert causati dai dati dell'accelerometro secondo le specifiche
- listaFileJournaling(): Restituisce la lista dei file di journaling presenti nella cartella dedicata.
- getIntervento(int): restituisce l'intervento.

1.5.5 Livello entità

TOInfermiere

Classe entità utilizzata per creare oggetti contenenti i dati di un infermiere.

Metodi get:

- getID()
- getNome()
- getCognome()
- getNumInterventi()

Metodi set:

- setID(int)
- setNome(String)
- setCognome(String)
- setNumInterventi(int)

Altri metodi:

- toString()

Metodi di verifica (controllano l'esistenza degli attributi obbligatori e la lunghezza delle stringhe secondo le specifiche):

- verificaCoerenzaID(String): controlla che l'ID non sia vuoto e che non superi la lunghezza massima stabilita dalle specifiche;
- verificaNome(String): controlla che il nome non sia vuoto e che non superi la lunghezza massima stabilita dalle specifiche;
- verificaCognome(String): controlla che il cognome non sia vuoto e che non superi la lunghezza massima stabilita dalle specifiche;

TOIntervento

Classe entità utilizzata per creare oggetti contenenti i dati di un intervento.

Metodi get:

- getID()
- getIDPaziente()

- getNomePaziente()
- getCognomePaziente()
- getIDInfermiere()
- getCognomeInfermiere()
- getData(String)
- getOraInizio(String)
- getOraFine()
- getCitta()
- getCivico()
- getCap()
- getDataDaFormato(String)
- getOraInizioDaFormato(String)
- getOraFineDaFormato(String)

Metodi set:

- setID(int)
- setID(String)
- setIDPaziente(int)
- setIDPaziente(String)
- setNomePaziente(String)
- setCognomePaziente(String)
- setIDInfermiere(int)
- setIDInfermiere(String)
- setCognomeInfermiere(String)
- setData(String,String)
- setOraInizio(String,String)
- setOraFine(String)
- setOraFineFmt(String,String)
- setCitta(String)
- setCivico(String)
- setCap(String)

Altri metodi:

- TOIntervento()
- addTipoIntervento(TOTipoIntervento)
- removeTipoIntervento(int)
- cancellaTipiIntervento()
- countTipInterventi()
- getTipoIntervento(int)

- toString()

Metodi di verifica (controllano l'esistenza degli attributi obbligatori e la lunghezza delle stringhe secondo le specifiche):

- verificaCoerenzaID(String)
- verificaValiditaNome(String)
- verificaValiditaCognome(String)
- verificaValiditaData(String)
- verificaValiditaNote(String)
- verificaLimitazioniDataOra(String,String)
- verificaValiditaTipo(String)
- verificaCoerenzaTipi(ArrayList<TOTipoIntervento>)
- verificaValiditaCitta(String)
- verificaValiditaCivico(String)
- verificaValiditaCap(String)

TOTipoIntervento

Classe entità utilizzata per creare oggetti contenenti i dati di un tipo Intervento.

Metodi get:

- getPatologia()
- getNome()
- getValoreRilevato()
- getTempoIntervento()
- getNote()

Metodi set:

- setPatologia(String)
- setNome(String)
- setValoreRilevato(String)
- setTempoIntervento(String)
- setNote(String)

Altri metodi:

- TOTipoIntervento()
- TOTipoIntervento(String,String,String)
- toString()

TOPaziente

Classe entità utilizzata per creare oggetti contenenti i dati di un paziente.

Metodi get:

- getID()
- getNome()
- getCognome()
- getDataNascita(String)
- getCellulari()

Metodi set:

- setID(String)
- setNome(String)
- setCognome(String)
- setDataNascita(String)
- setNote(String)

Altri metodi:

- TOPaziente()
- addCellulare(String)
- clearCellulari()
- calcolaAnniMesi():metodo utilizzato per calcolare anni e mesi del paziente come da specifica del file di interscambio.
- toString()

Metodi di verifica (controllano l'esistenza degli attributi obbligatori e la lunghezza delle stringhe secondo le specifiche):

- verificaCoerenzaID(String): controlla che l'ID non sia vuoto e che non superi la lunghezza massima stabilita dalle specifiche;
- verificaNome(String): controlla che il nome non sia vuoto e che non superi la lunghezza massima stabilita dalle specifiche;
- verificaCognome(String):controlla che il cognome non sia vuoto e che non superi la lunghezza massima stabilita dalle specifiche;
- verificaDataNascita(String,String) :verifica esistenza e lunghezza dell'attributo.

TOPatologia

Classe entità utilizzata per creare oggetti contenenti i dati di una patologia.

Metodi get:

- getID()
- getNome()
- getGravita()

Metodi set:

- setID(int)
- setNome(String)
- setGravita(String)

Altri metodi:

- toString()

Metodi di verifica (controllano l'esistenza degli attributi obbligatori e la lunghezza delle stringhe secondo le specifiche):

- verificaCoerenzaID(String):controlla che l'ID non sia vuoto e che non superi la lunghezza massima stabilita dalle specifiche;
- verificaIDPatologia(String): controlla che l'ID non sia vuoto.
- verificaNomiPatologie(String): controlla che il nome della patologia non sia vuoto e che non superi la lunghezza massima stabilita dalle specifiche;
- verificaGravita(String):controlla che la gravità non sia vuota e che non superi la lunghezza massima stabilita dalle specifiche;

TORilevazione

Classe entità utilizzata per creare oggetti contenenti i dati di una rilevazione (Timestamp + GPS + Accelerometro).

Metodi get:

- getTimestamp()
- getGpsLatitude()
- getGpsLongitude()
- getGpsAltitude()

- getGpsAccuracy()
- getAccX()
- getAccY()
- getAccZ()

Metodi set:

- setTimestamp(Data)
- setGpsLatitude(double)
- setGpsLongitude(double)
- setGpsAltitude(double)
- setGpsAccuracy(double)
- setAccX(double)
- setAccY(double)
- setAccZ(double)
- setTimestampFromString(String,String)

Altri metodi:

- TORilevazione()
- toString()

TOInterventoCompleto

Classe entità che eredita attributi e metodi della classe Intervento e viene utilizzata per creare oggetti contenenti i dati di u n intervento completo (caricato dal file di journaling) comprendente le rilevazioni effettuate in automatico da ADISys Mobile.

Metodi get:

- getPaziente ()
- getInfermiere ()
- getLog (int)
- getMisuraRilevata()
- getStatoVerificaGPS()
- getStatoVerificaAccelerometro()
- getNote()

Metodi set:

- setPaziente(TOPaziente)

- setInfermiere(TOInfermiere)
- setMisuraRilevata(String)
- setStatoVerificaGPS(StatoVerifica)
- setStatoVerificaAccelerometro(StatoVerifica)
- setNote(String)

Altri metodi:

- addLog(TORilevazione) Aggiunge una rilevazione all'intervento;
- contaLog()Restituisce il numero di rilevazioni disponibili (utile in fasedi popolamento delle tabelle);
- toString()

1.5.6 Livello di integrazione

Infermieri

Classe che compone i predicati per il database per la gestione degli infermieri e crea gli oggetti per i livelli superiori.

Metodi:

- idEsistente(int): verifica l'esistenza o meno di un infermiere con l'id in input;
- creaInfermiere(TOInfermiere): crea un nuovo elemento nella tabella Infermieri a partire dall'oggetto in input;
- modificaInfermiere(TOInfermiere): se esiste l'infermeire con l'id indicato ne aggiorna i dati.
- cancellaInfermiere(int): cancella dal database l'infermiere con l'id indicato.
- cancellaTutti(): cancella dal database tutte le informazioni relative agli infermieri.
- getTabellaInfoInfermieriConInterventi(): restituisce il un ADISysTableModel con i dati degli infermieri ed il conto degli interventi ad essi associati.
- getTabellaInfermieri(): restituisce il un ADISysTableModel con i dati degli infermieri.
- getInfermiere():restituisce l'oggetto con i dati dell'infermiere, il cui id è fornito in input.

Pazienti

Classe che compone i predicati per il database per la gestione dei pazienti e crea gli oggetti per i livelli superiori.

Metodi:

- idEsistente(int): verifica l'esistenza o meno di un paziente con l'id in input;
- creaPaziente(TOPaziente): crea un nuovo elemento nella tabella Pazienti a partire dall'oggetto in input;
- modificaPaziente(TOPaziente): se esiste il paziente con l'id indicato ne aggiorna i dati.
- cancellaPaziente(int): cancella dal database il paziente con l'id indicato.
- cancellaTutti(): cancella dal database tutte le informazioni relative ai pazienti.

- getTabellaPazienti(): restituisce il un ADISysTableModel con i dati dei pazienti.
- getPaziente(int): restituisce l'oggetto con i dati del paziente, il cui id è fornito in input.
- recuperaID(TOPaziente): recupera l'ID del paziente voluto.

Interventi

Classe che compone i predicati per il database per la gestione degli interventi e crea gli oggetti per i livelli superiori.

Metodi:

- idEsistente(int): verifica l'esistenza o meno di un intervento con l'id in input;
- creaIntervento(TOIntervento): crea un nuovo elemento nella tabella Interventi a partire dall'oggetto in input;
- modificaIntervento(TOIntervento): se esiste l'intervento con l'id indicato ne aggiorna i dati.
- cancellaIntervento(int): cancella dal database l'intervento con l'id indicato.
- cancellaTutti(): cancella dal database tutte le informazioni relative agli interventi.
- getTabellaVisualizzazioneInterventi(): restituisce un ADISysTableModel con i dati degli interventi adattati alla visualizzazione (una view con nome di paziente ed infermiere assegnato).
- getTabellaInterventi(): restituisce il un ADISysTableModel con i dati degli interventi.
- getListaInterventiInfermiere(int): restituisce la lista degli interventi di un infermiere, avendo in input l'id dell'infermiere stesso.
- inserisciTipoIntervento(int, String,String, String): restituisce un booleano.
- cancellaTipiIntervento(int): restituisce un booleano.
- leggiTipiIntervento(int): restituisce i TipiIntervento di un determinato Intervento.
- getIntervento(int): restituisce l'intervento.

Patologie

Classe che compone i predicati per il database per la gestione delle patologie e crea gli oggetti per i livelli superiori.

Metodi:

- idEsistente(int): verifica l'esistenza o meno di un infermiere con l'id in input;
- creaPatologia(TOPatologia): crea un nuovo elemento nella tabella patologie a partire dall'oggetto in input;
- modificaPatologia(TOPatologia): se esiste la patologia con l'id indicato, ne aggiorna i dati.
- cancellaPatologia(int): cancella dal database la patologia con l'id indicato.
- getNomiPatologie(): restituisce una lista con i nomi delle patologie.
- getTabellaPatologie(): restituisce un ADISysTableModel con i dati inerenti le patologie.
- getPatologia(): restituisce l'oggetto con i dati della patologia, il cui id è fornito in input.

Database

Classe che effettua la connessione al database, effettua le query sullo stesso e restituisce i dati ottenuti.

Metodi:

- connetti(): effettua la connessione al database.
- esegui(String): esegue una operazione (non query) sul database;
- getTabella(String): esegue una query e restituisce un ADISysTableModel;
- getResultSet(String): esegue una query e restituisce un ResultSet;
- string2sqlstring(String): adatta le stringhe provenienti dall'interfaccia al linguaggio sql (risolve il problema dei doppi apici);
- finalize(): alla chiusura del programma chiama il metodo disconnetti();
- disconnetti(): chiude la connessione al database (necessario per il salvataggio delle modifiche).

StrutturaInterscambio

Classe incaricata della creazione, validazione e lettura dei file di interscambio XML.

Metodi:

- addIntervento(TOIntervento,TOpaziente): aggiunge i dati di un intervento e del paziente relativo alla struttura XML.

- salvaSuFileXML(String, String, String): scrive il file XML nel percorso specificato ed effettua la validazione.
- caricaDaFileXML(String, String): carica il file XML dal percorso specificato.
- getInterventiCompleti(): legge la struttura XML interna e restituisce un vettore di interventi pronti per verifica dei dati e visualizzazione.
- int2string6(int): metodo ausiliario per il parsing degli ID
- validaXML (File, File): metodo ausiliario per la validazione dei file XML con file XSD.

1.5.7 Gestione Messaggistica

UIFeedbackException

Classe pubblica che, attraverso i suoi metodi, gestisce le eccezioni a basso livello. Può essere lanciata da qualsiasi livello. Questa classe, invia una stringa di testo, che ritornerà al metodo usato e che verrà elaborata in seguito nella classe FC, in cui sono gestiti i metodi "messagesBundle" e "VisualizzaMessaggioPupop".

Metodi:

- UIFeedbackException(String)
- UIFeedbackException(String,ArrayList<String>)
- UIFeedbackException(String,MsgType)
- UIFeedbackException(String,ArrayList<String>,MsgType)
- getMsg()
- getType()
- getReplacementArray()

1.5.8 Strumenti di utility

ADISysTableModel

Modello di tabella che si autogenera a partire da un ResultSet. Serve a facilitare l'operazione di visualizzazione dei dati provenie nti da una query sul database.

ADISysTableRenderVerifica

Modello di visualizzazione della tabella di verifica (visualizza i simboli grafici relativi allo stato di validità dei dati).

DateFormatConverter

Classe con membri statici utilizzata per il parsing delle date da stringhe e la composizione di stringhe da oggetti GregorianCaledar, Date e DateTime(SQL) che hanno specifiche diverse.

ReadXML

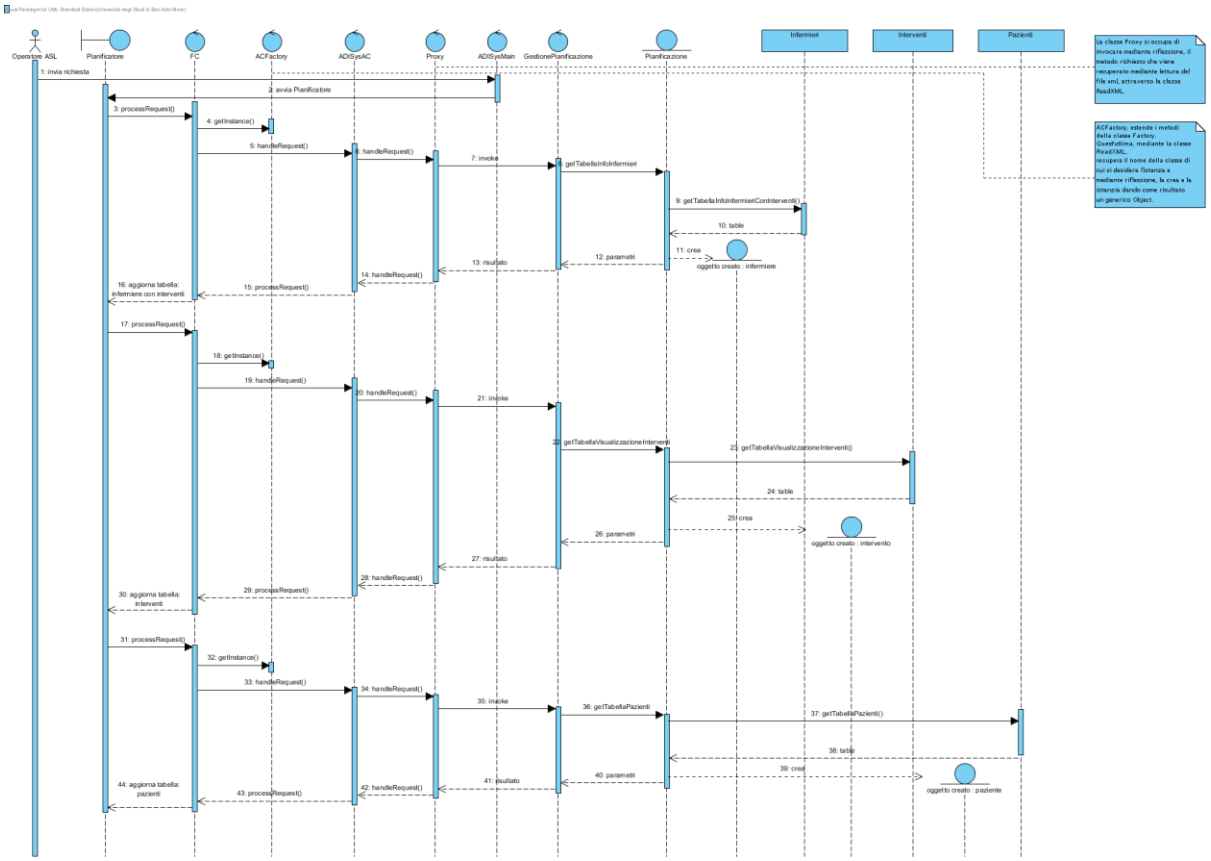
Classe di utility per la lettura del file XML.

Metodi:

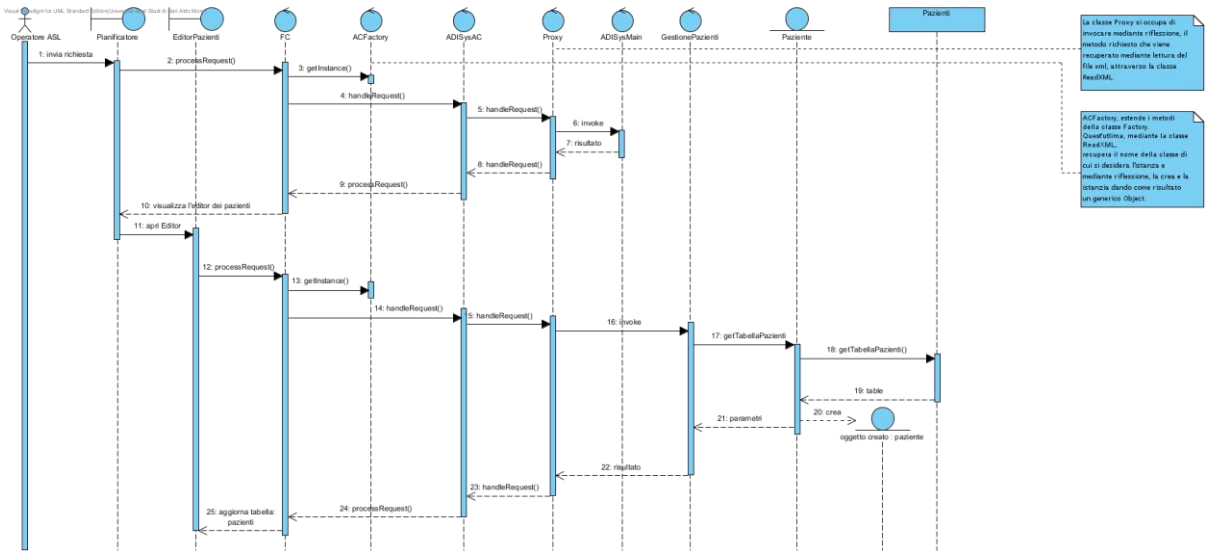
- readValue(String)
- getNodeByFirstAttrValue(String, String)
- getFirstChildValue(String, String)
- getChildrenValue(String, String)
- getNodeIstanceNumber(String)
- getFirstAttributeValue(String)
- getMap(String)

1.6 Diagrammi di Sequenza

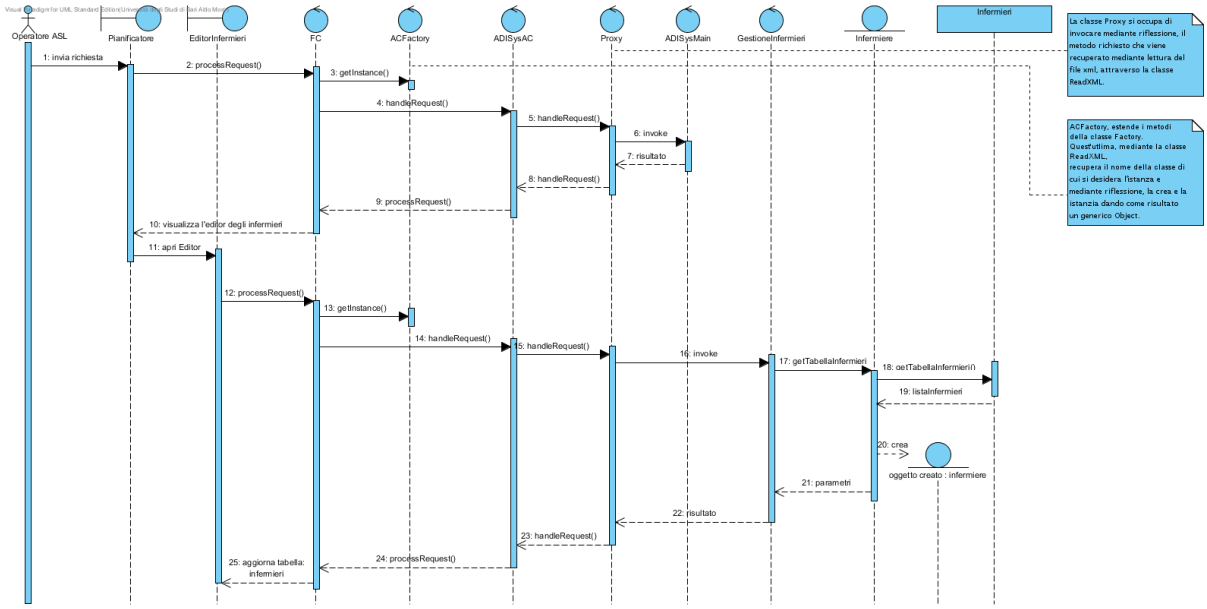
Visualizza pianificazione



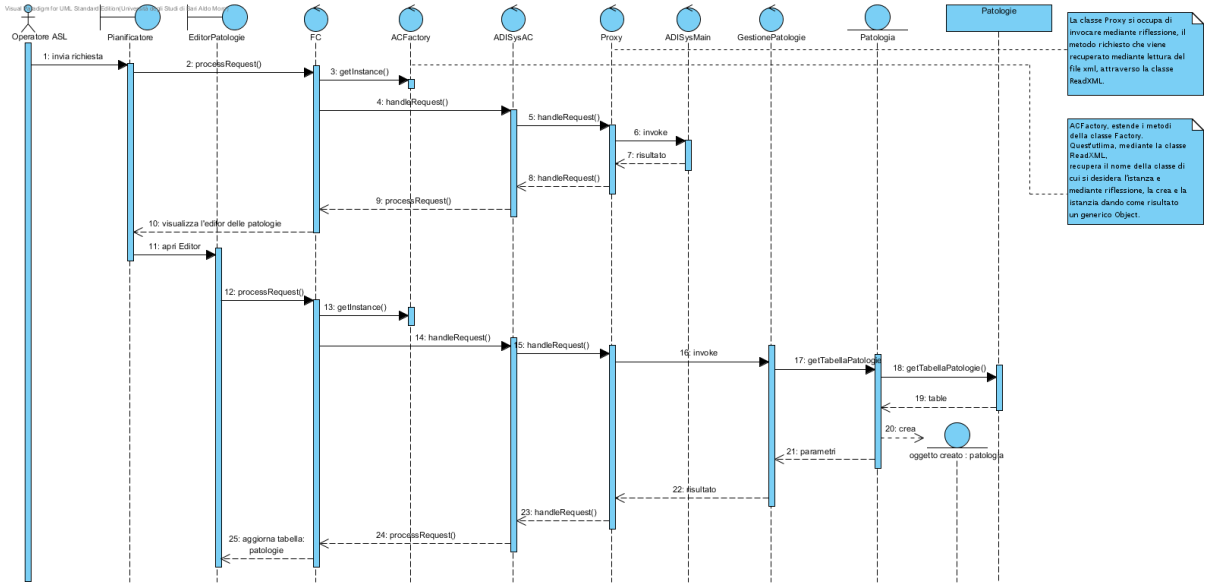
Visualizza elenco pazienti



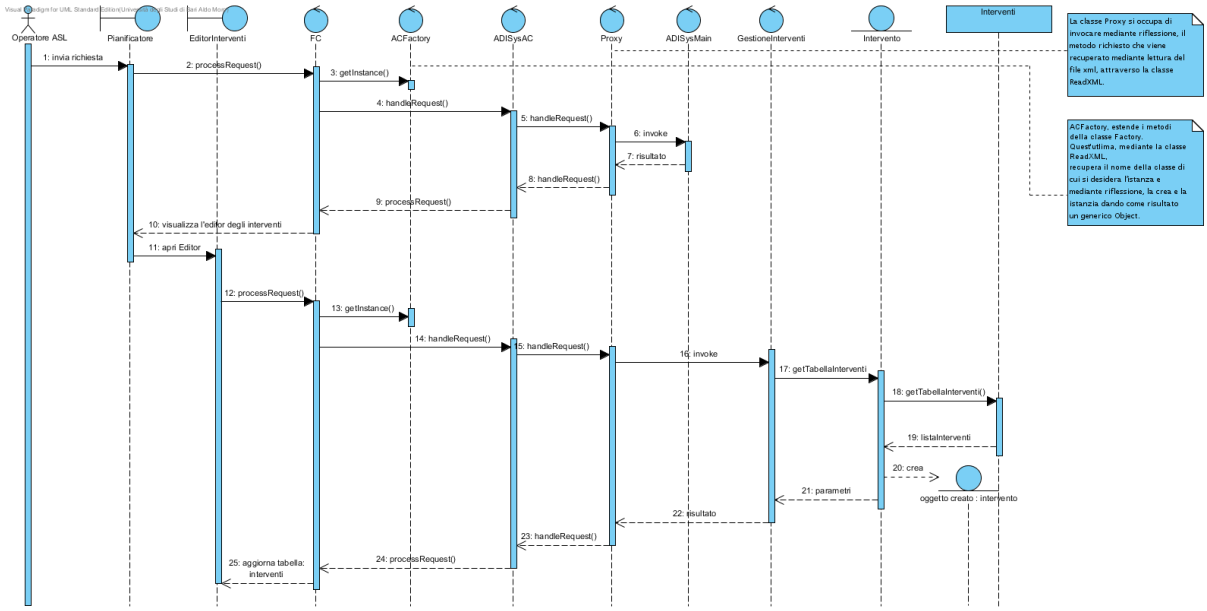
Visualizza elenco infermieri



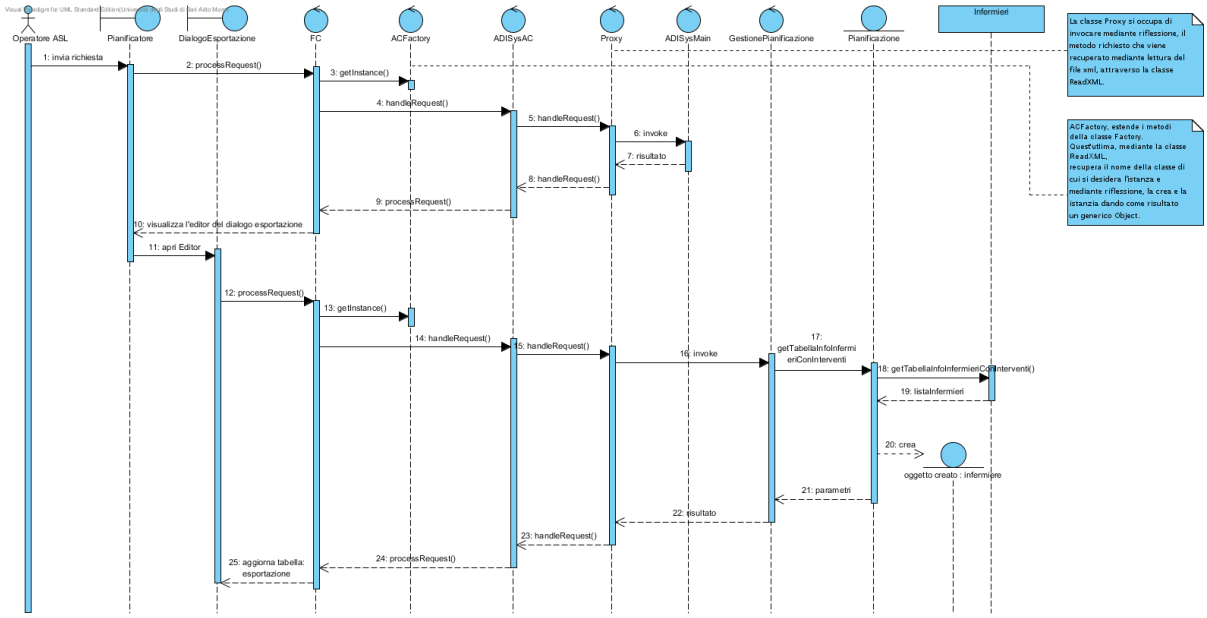
Visualizza elenco patologie



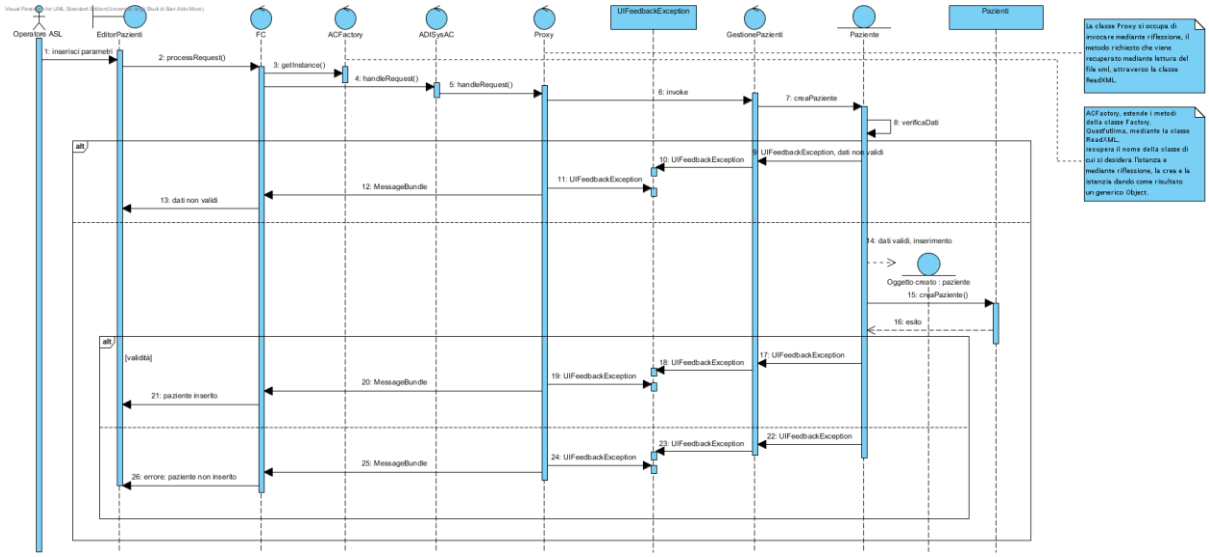
Visualizza elenco interventi



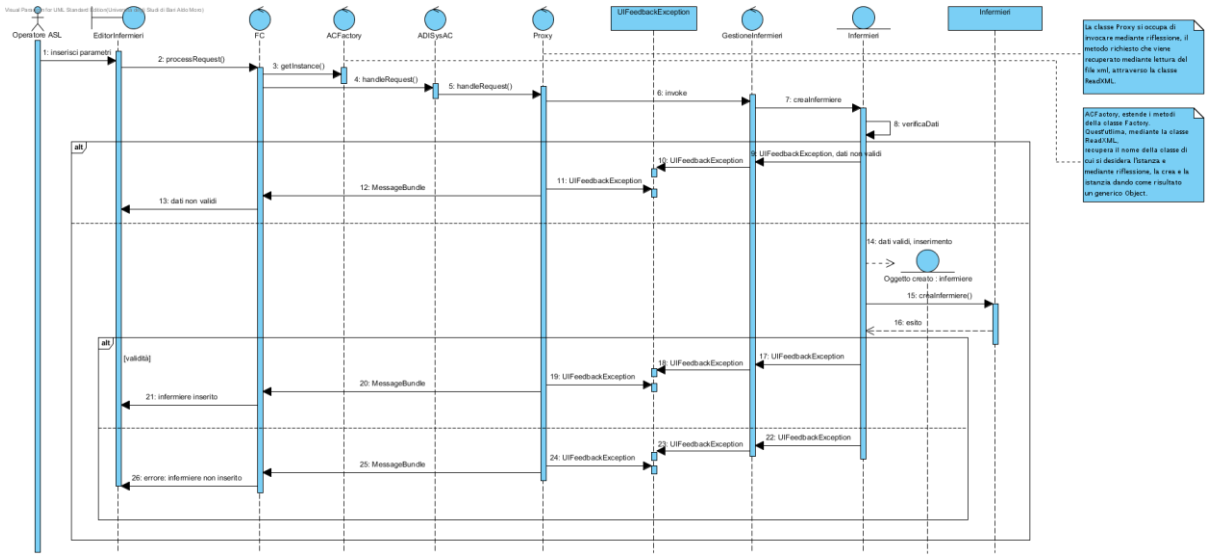
Visualizza lista esportazione



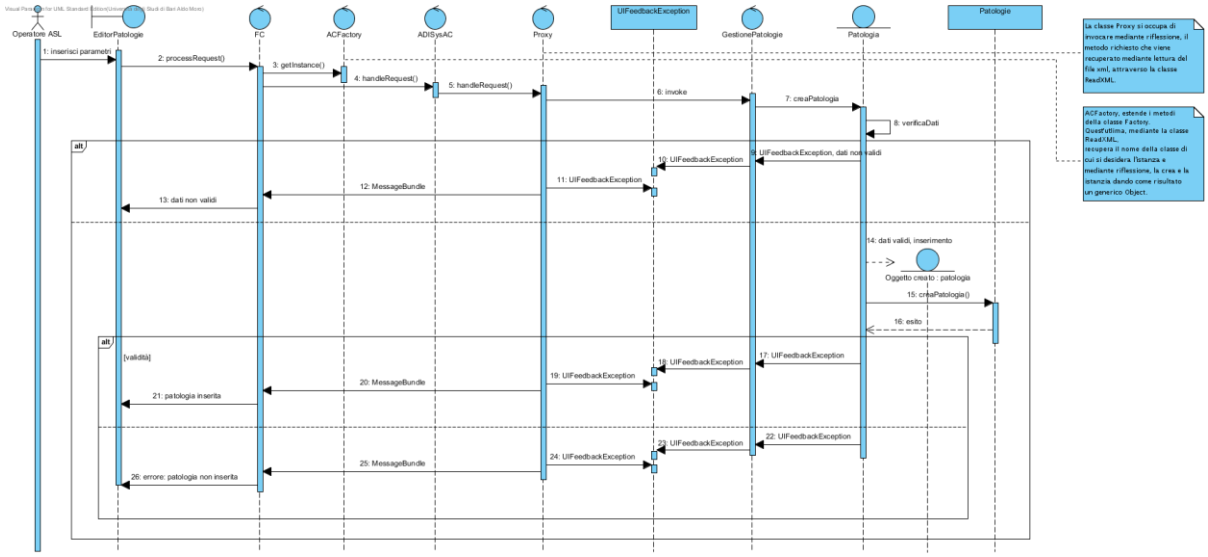
Inserisci paziente



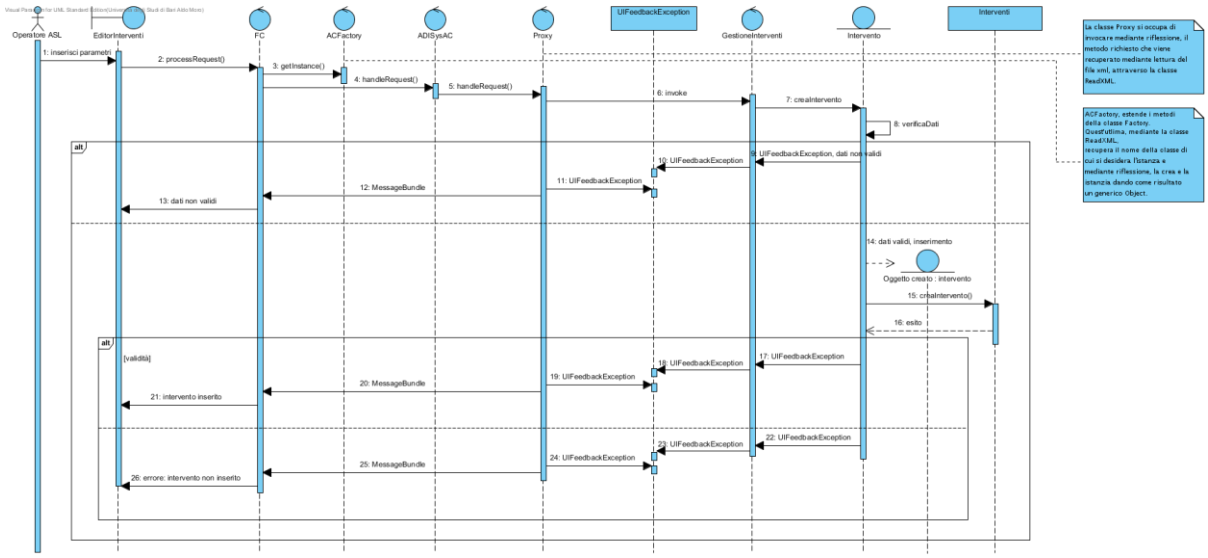
Inserisci infermiere



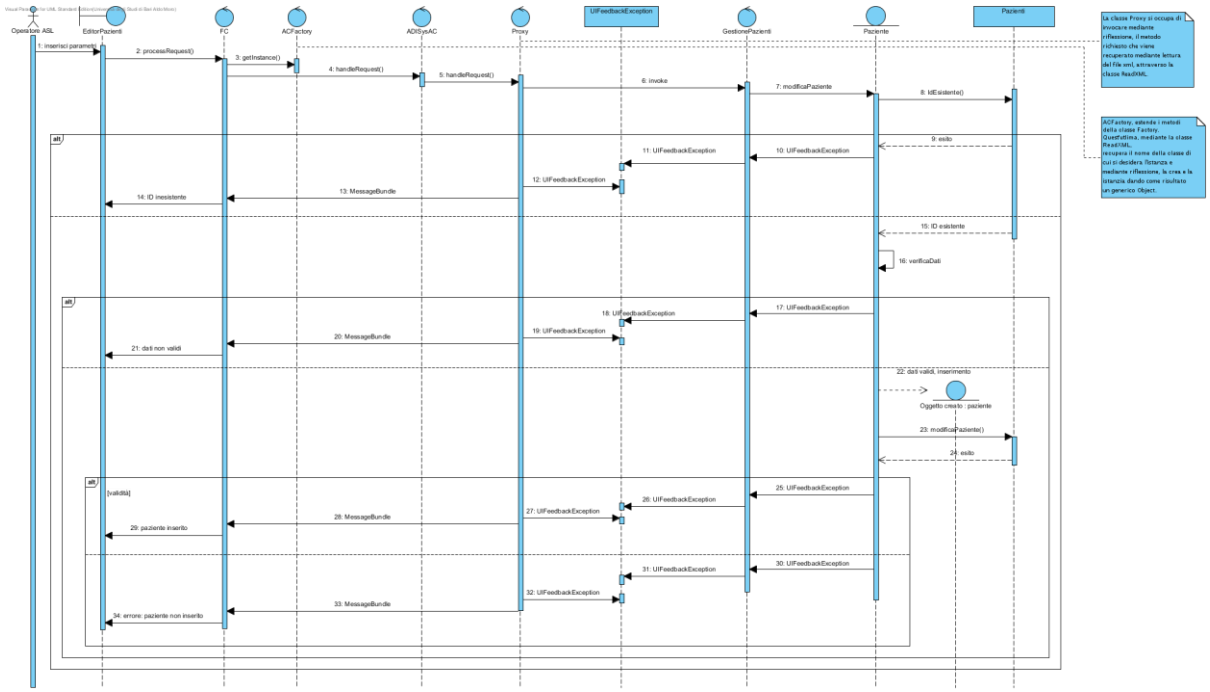
Inserisci patologia



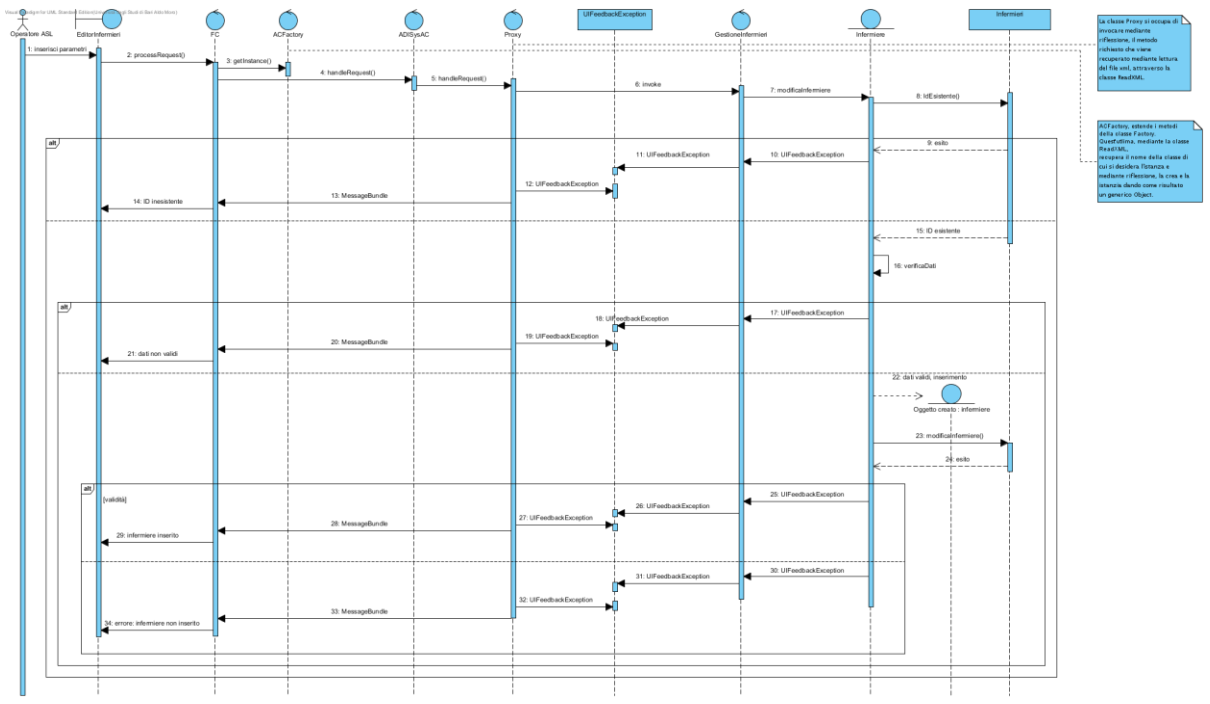
Inserisci intervento



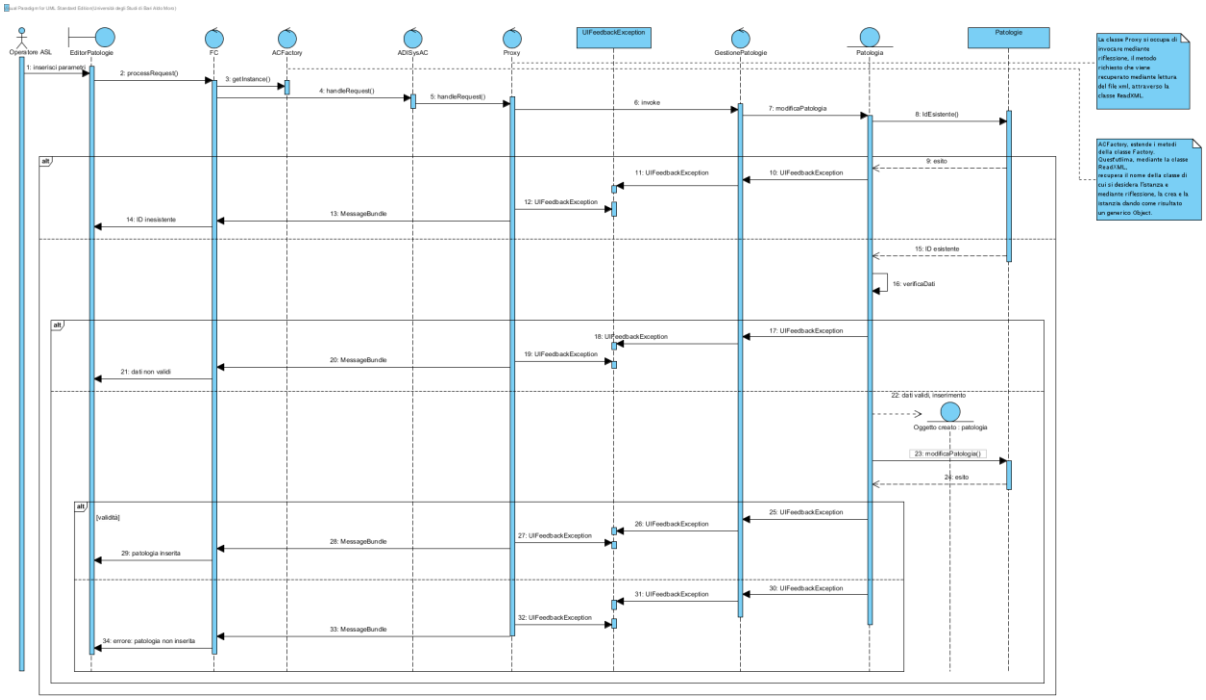
Modifica paziente



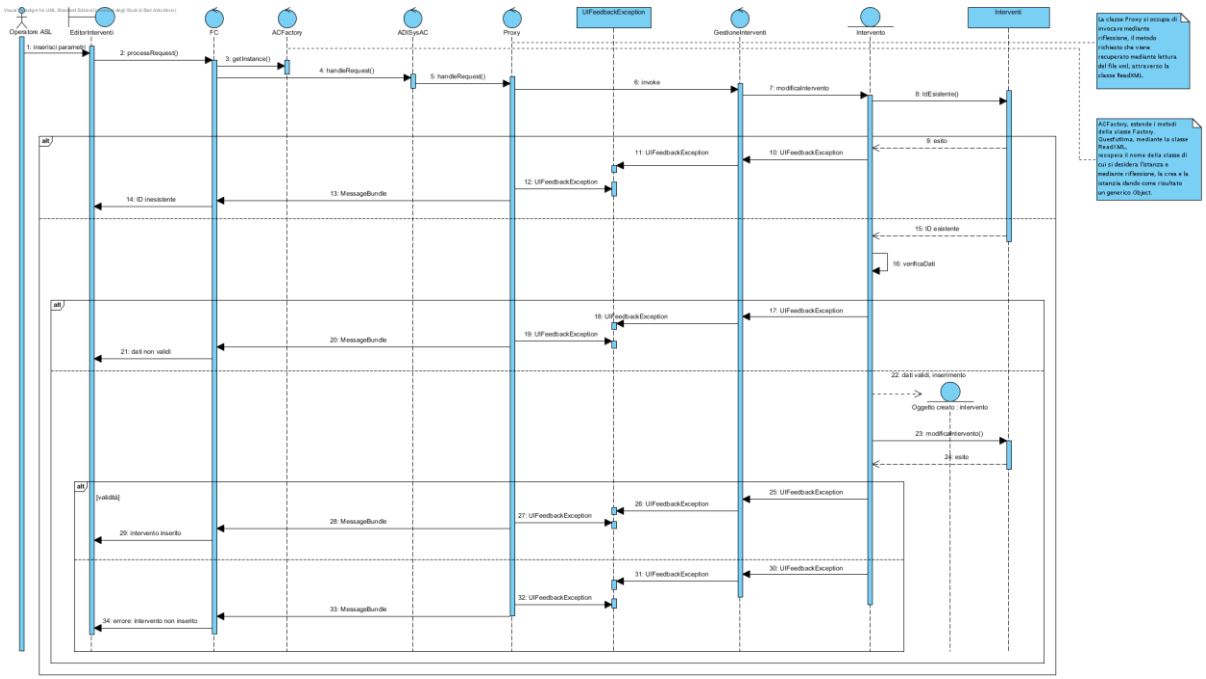
Modifica infermiere



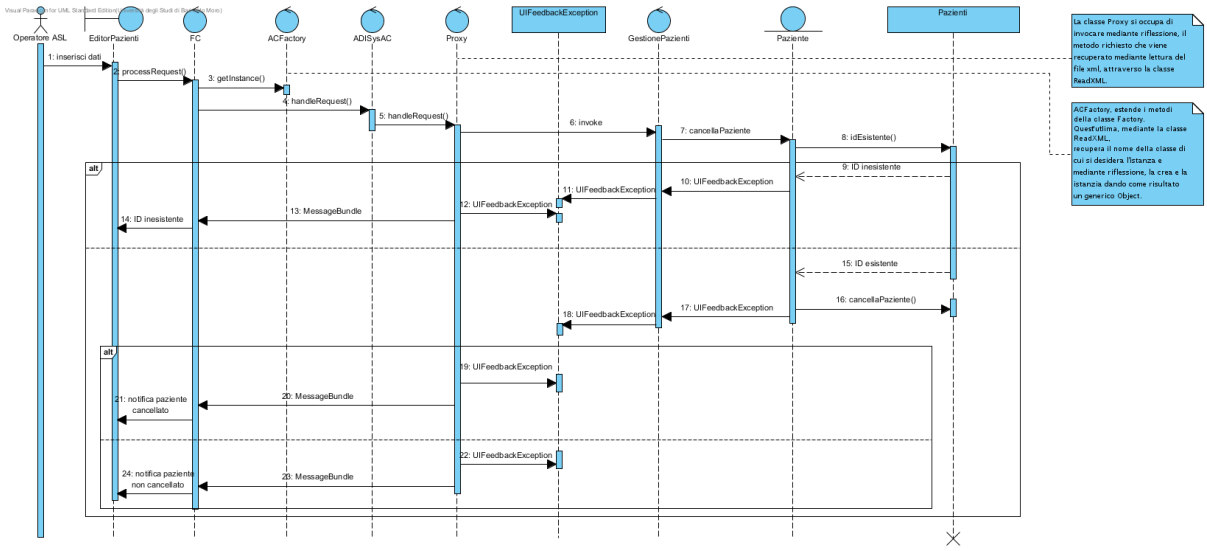
Modifica Patologia



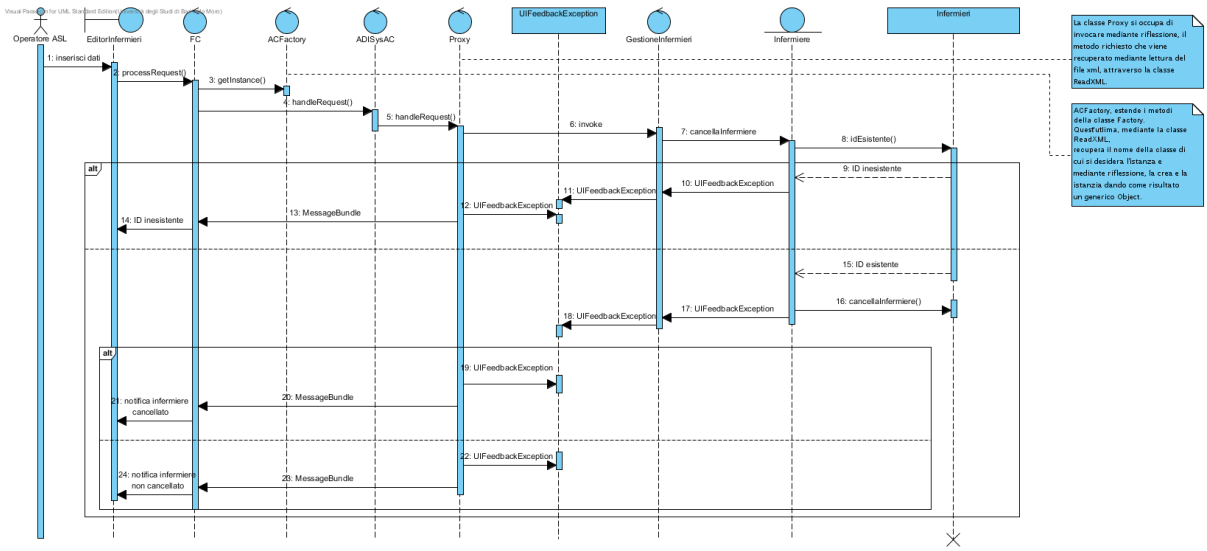
Modifica intervento



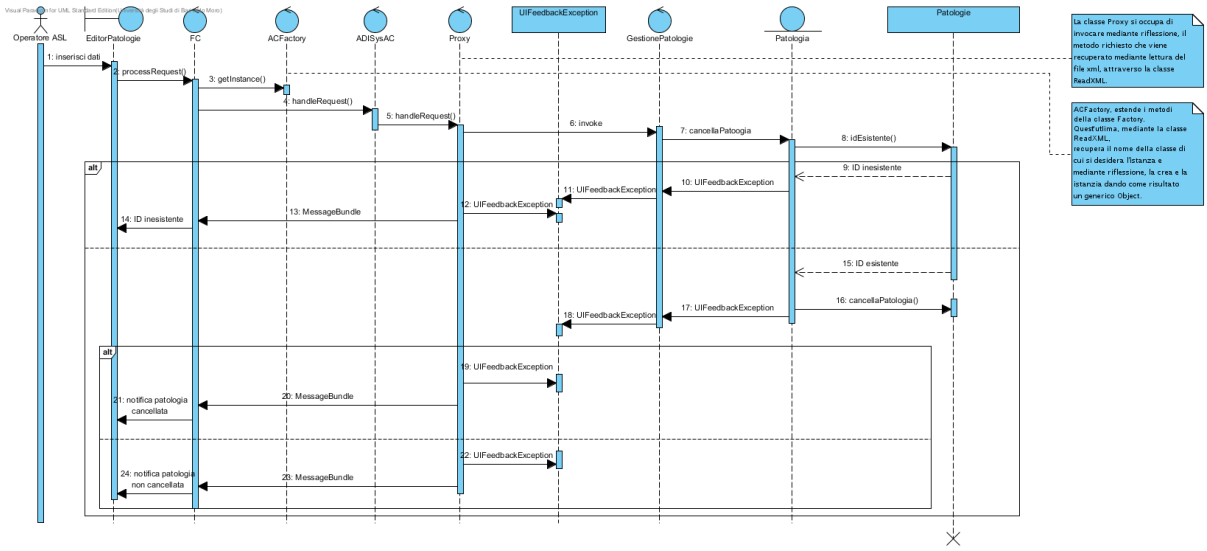
Cancella paziente



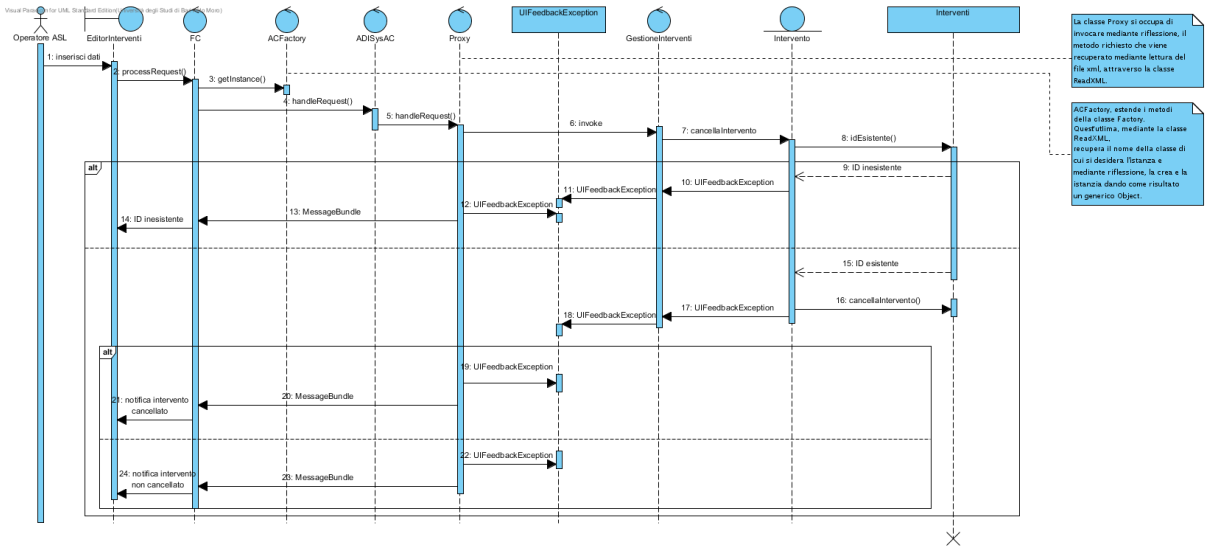
Cancella infermiere



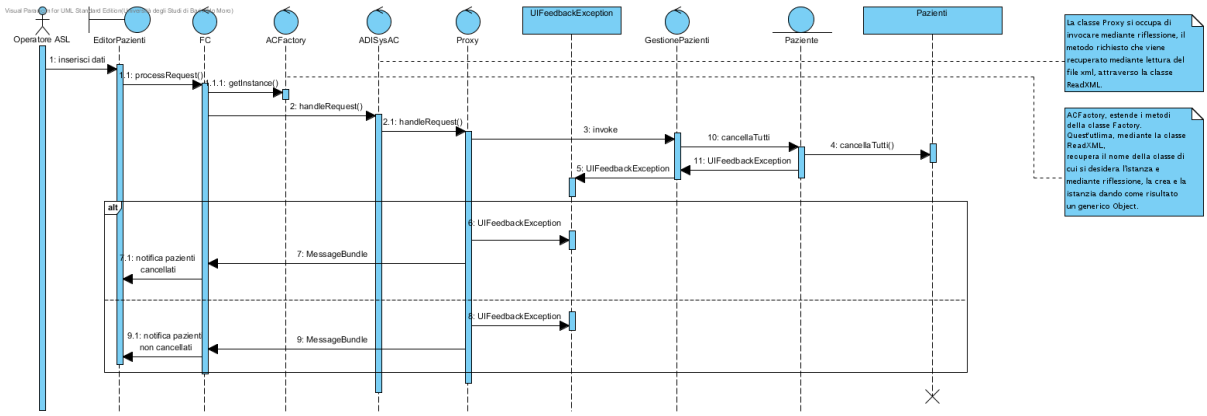
Cancella patologia



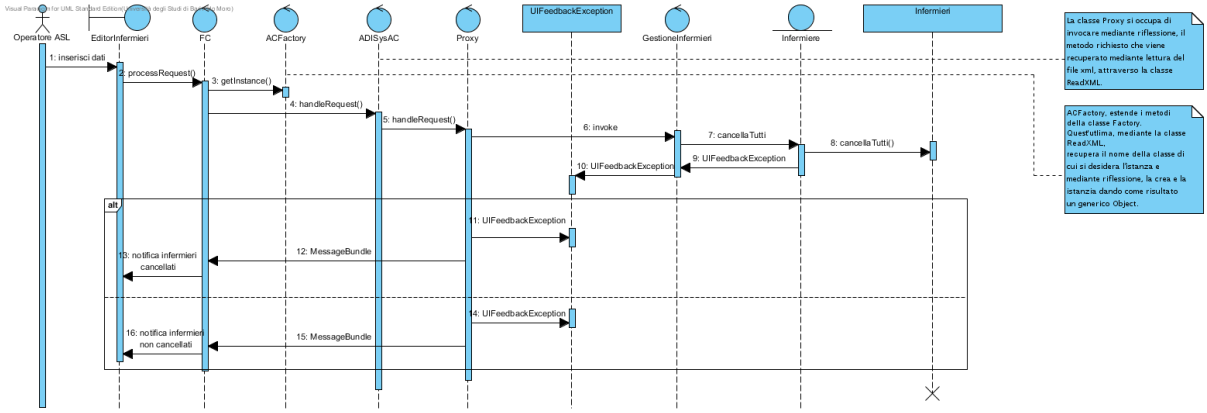
Cancella intervento



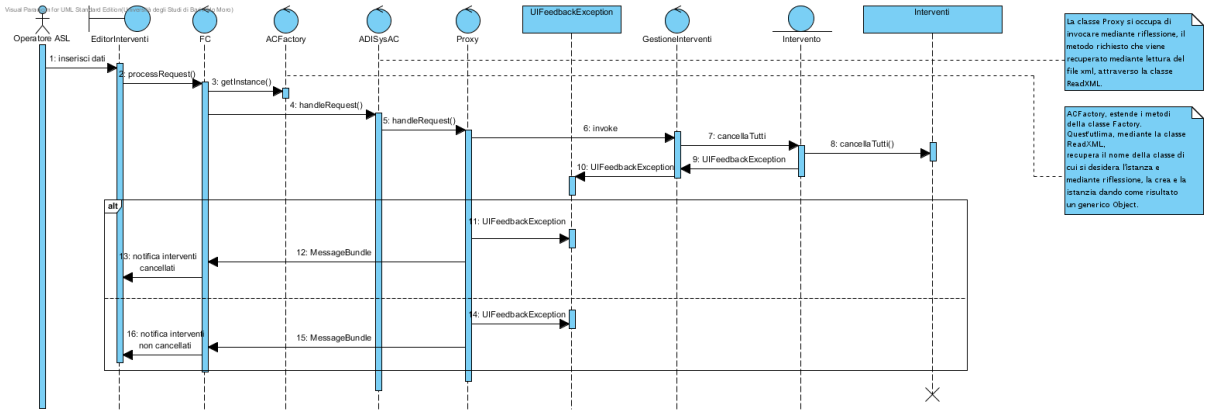
Cancella tutti i pazienti



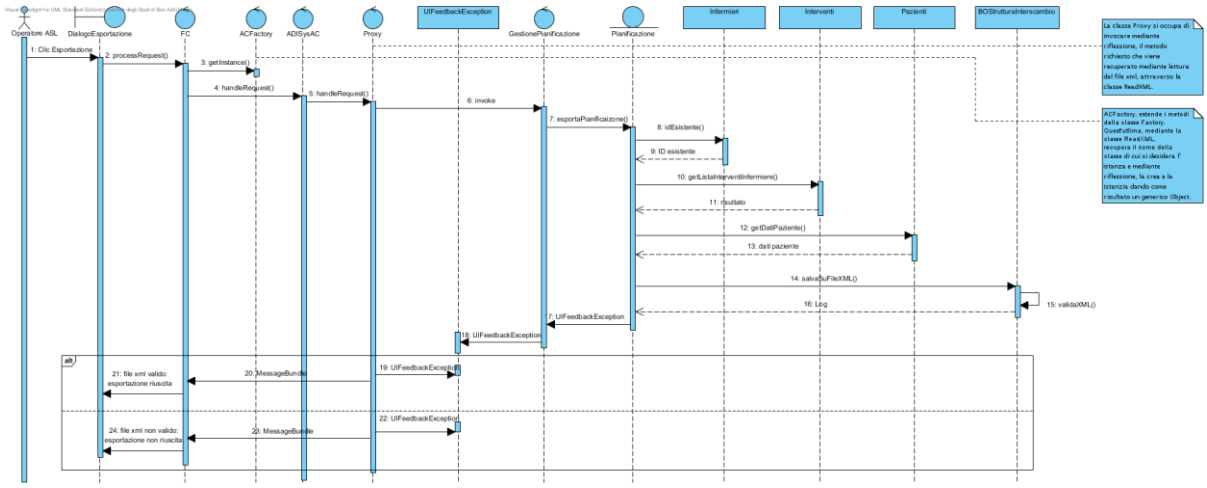
Cancella gli infermieri



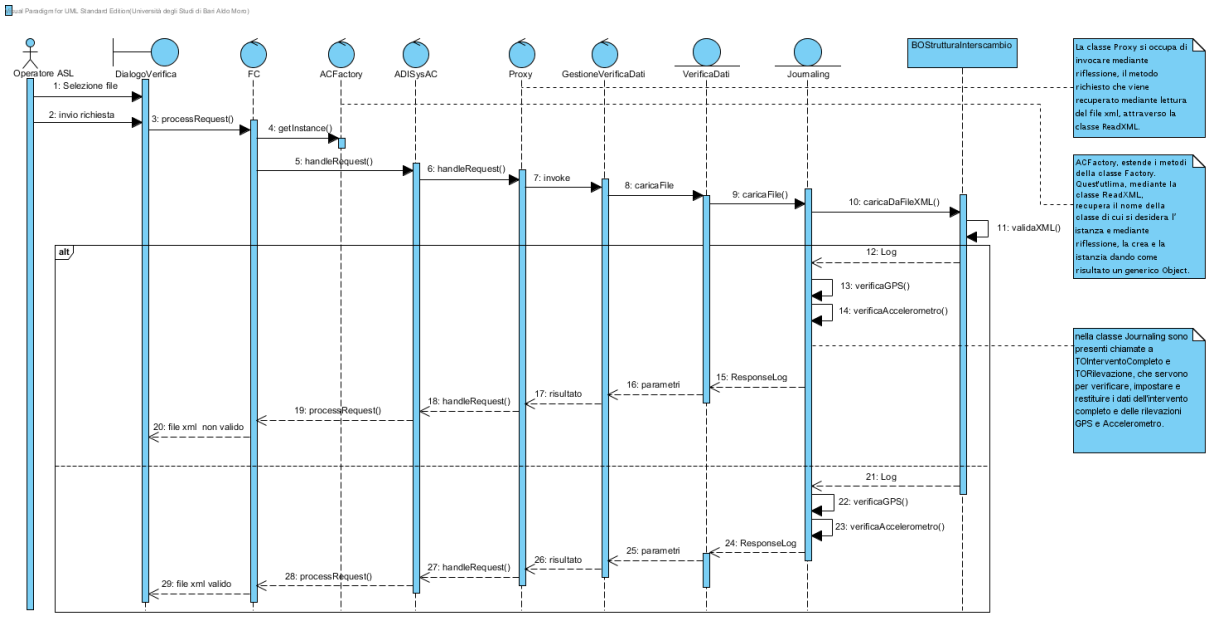
Cancella tutti gli interventi



Esporta Pianificazione



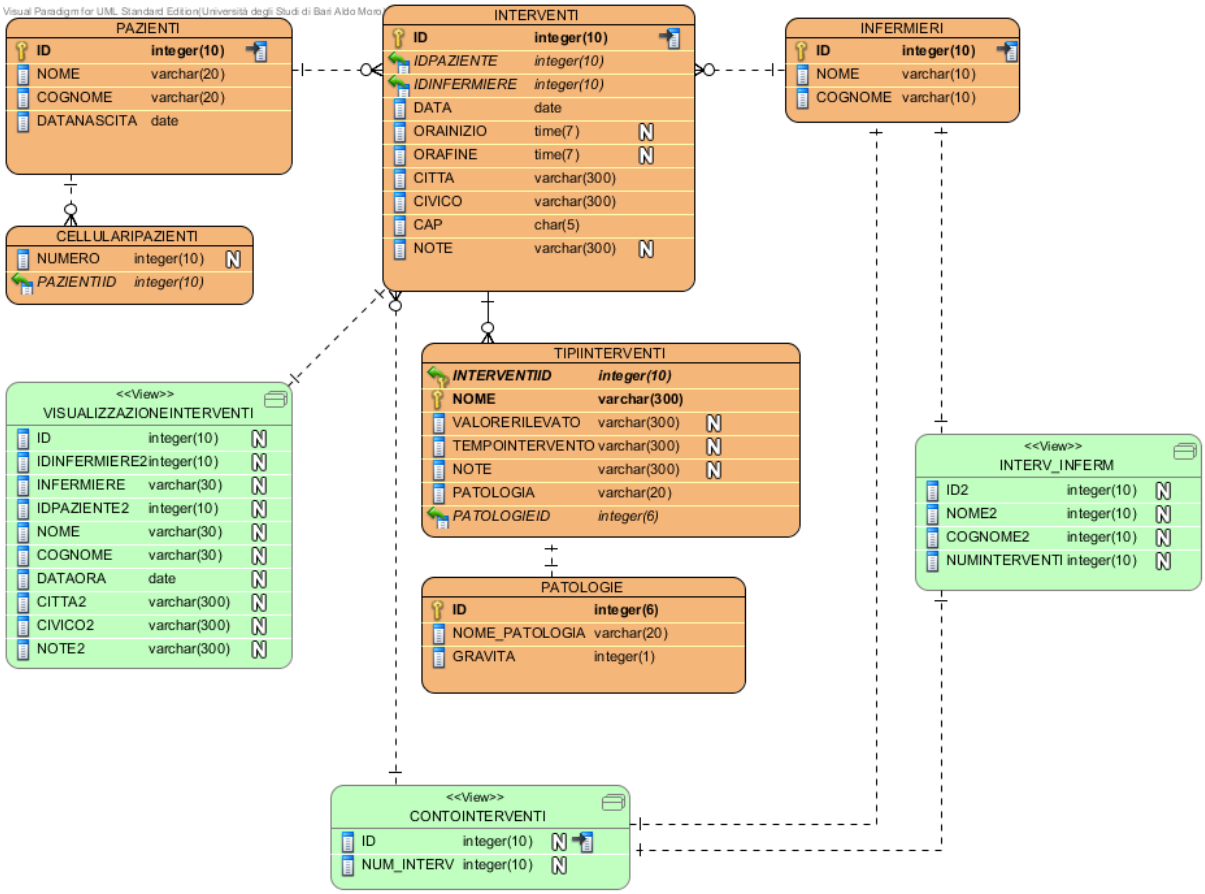
Caricamento file di journaling



2. PROGETTO DEI DATI

2.1 Database

2.1.1 Diagramma delle Dipendenze dei Dati



2.1.2 Modello del Database

Per gestire al meglio le varie entità dell'applicazione è stato scelto di utilizzare un HSQLDB (HSQL Database Engine), cioè un RDBMS (Relational Database Management System) basato sul modello relazionale, scritto in Java.

2.1.3 Dettaglio dei Dati

Tabella PAZIENTI

ID: intero di massimo 10 cifre, id entifica l'ID di un paziente. Chiave primaria della tabella.
NOME: stringa di massimo 20 caratteri, identifica il nome di un paziente.
COGNOME: stringa di massimo 20 caratteri, identifica il cognome di un paziente.
DATANASCITA: di tipo data, identifica la data di nascita di un paziente.

Tabella CELLULARIPAZIENTE

NUMERO: intero di massimo 10 cifre, identifica il numero di telefono di un paziente.
PAZIENTIID: intero di massimo 10 cifre, identifica l'id del paziente associato a un numero di telefono; vincolo di integrità referenziale con cardinalità molti ad uno con l'attributo ID della tabella PAZIENTI.

Tabella INFERMIERI

ID: intero di massimo 10 cifre, identifica l'id di un infermiere. Chiave primaria della tabella.
NOME: stringa di massimo 10 caratteri, ide ntifica il nome di un infermiere.
COGNOME: stringa di massimo 10 caratteri, identifica il cognome di un infermiere.

Tabella INTERVENTI

ID: intero di massimo 10 cifre, identifica l'id di un intervento. Chiave primaria della tabella.
IDPAZIENTE: intero di massimo 10 cifre, identifica l'id del pazienti associato all'intervento; vincolo di integrità referenziale con cardinalità molti ad uno con l'attributo ID della tabella PAZIENTI.
IDINFERMIERE: intero di massimo 10 cifre, identifica l'id dell'infermiere associato all'intervento; vincolo di integrità referenziale con cardinalità molti ad uno con l'attributo ID della tabella INFERMIERI.
DATA: data, identifica la data di un intervento.
ORAINIZIO: time di lunghezza massima 7, identifica l'ora d'inizio di un intervento.

ORAFINE: time di lunghezza massima 7, identifica l'ora della fine di un intervento.
CITTA: stringa di massimo 300 caratteri, identifica la città dove si svolgerà un intervento.
CIVICO: stringa di massimo 300 caratteri, identifica il civico dove si svolgerà un intervento.
CAP: stringa di massimo 5 caratteri, identifica il CAP dove si svolgerà un intervento.
NOTE: stringa di massimo 300 caratteri, identifica le note di un intervento.

Tabella TIPIINTERVENTI

INTERVENTIID: intero di massimo 10 cifre, identifica l'id di un tipo intervento. Chiave primaria della tabella insieme all'attributo NOME. Vincolo di integrità referenziale con cardinalità molti ad uno con l'attributo ID della tabella INTERVENTI.
NOME: stringa di massimo 300 caratteri, identifica il no me di un tipo intervento. Chiave primaria della tabella insieme all'attributo INTERVENTIID.
VALORERILEVATO: stringa di massimo 300 caratteri, identifica il valore rilevato di un tipo intervento.
TEMPOINTERVENTO: stringa di massimo 300 caratteri, identifica il tempo totale di un tipo intervento.
NOTE: stringa di massimo 300 caratteri, identifica le note di un tipo intervento.
PATOLOGIA: stringa di massimo 20 caratteri, identifica il nome di una patologia. Vincolo di integrità referenziale con cardinalità uno a molti con l'attributo NOME_PATOLOGIA della tabella PATOLOGIE.

Tabella PATOLOGIE

ID: intero di massimo 6 cifre, che identifica l'id di una patologia. Chiave primaria della tabella.
NOME_PATOLOGIA: stringa di massimo 20 caratteri, che identifica il nome di una patologia.
GRAVITA: intero di massimo 1 carattere, che identifica la gravità di una patologia.

View VISUALIZZAZIONEINTERVENTI

ID: intero di massimo 10 cifre, identifica l'id di un intervento.
IDINFERMIERE: intero di massimo 10 cifre, identifica l'id dell'infermiere associato all'intervento.
INFERMIERE: stringa di massimo 30 caratteri, identifica il cognome dell'infermiere associato all'intervento.
IDPAZIENTE: intero di massimo 10 cifre, identifica l'id del paziente associato all'intervento.
NOME: stringa di massimo 30 caratteri, identifica il nome del paziente associato all'intervento.
COGNOME: stringa di massimo 30 caratteri, identifica il cognome d el paziente associato all'intervento.
DATAORA: date, identifica la data e l'ora di un intervento.
CITTA: stringa di massimo 300 caratteri, identifica la città dove si svolgerà un intervento.
CIVICO: stringa di massimo 300 caratteri, identifica il civico dove si svolgerà un intervento.
NOTE: stringa di massimo 300 caratteri, identifica le note di un intervento.

View CONTOINTERVENTI

ID: intero di massimo 10 cifre, identifica l'id di un infermiere.
NUM_INTERV: intero di massimo 10 cifre, identifica il numero degli interventi di un infermiere.

View INTER_INFERM

ID: intero di massimo 10 cifre, identifica l'id di un infermiere.
NOME: stringa di massimo 10 caratteri, identifica il nome di un infermiere.
COGNOME: stringa di massimo 10 caratteri, identifica il cognome di un infermiere.
NUMINTERVENTI: intero di massimo 10 caratteri, identifica il numero degli interventi di un infermiere.

File System

Nella directory di progetto sono presenti le cartelle utilizzate per lo scambio di dati con ADISys Mobile:

- La cartella "Importazione" contiene i file di journaling da verificare (ADISys Server può caricare tutti i file presenti nella cartella, quindi è possibile rinominare i file in modo da permettere la presenza contemporanea nella cartella);
- La cartella "Esportazione" contiene il file prodotto da ADISys Server esportando la pianificazione giornaliera di un infermiere (il file deve essere rimosso dalla cartella per la consegna all'infermiere o sarà sovrascritto dal successivo).

Nella directory principale è inoltre presente la cartella utilizzata dall'ACFactory e dalla classe Proxy per i servizi di business; in particolare:

- **ADISysACFactory.xml**: File XML utilizzato dall'ACFactory per sapere quale Application Controller utilizzare (in questa versione di ADISys Server l'unico Application Controller implementato è ADISysAC);
- **ConsultACProxy.xml**: File XML utilizzato dal Proxy per individuare la classe che contiene il metodo richiesto dal client; tale file conterrà pertanto tutti i metodi delle classi che estendono ADISysAC.

In aggiunta, è presente anche la cartella utilizzata per il multilingua; in questa cartella saranno presenti tutti i file "properties" delle lingue utilizzabili in ADISys. In particolare in questa versione di ADISys Server saranno presenti:

- **MessagesBundle**: file properties contente la lingua standard.
- **MessagesBundle_it_IT**: file properties contenente la lingua italiana di ADISys Server;
- **MessagesBundle_en_EN**: file properties contenente la lingua inglese di ADISys Server.

Per sviluppi futuri (aggiunta di lingue) sarà sufficiente aggiungere nella cartella dedicata il file properties riguardante la lingua voluta (oltre ovviamente a piccole modifiche nella selezione della lingua nella boundary del Pianificatore, tale da rendere possibile il cambio con la nuova lingua desiderata).

2.1.4 Grammatiche file XML

Si è deciso di utilizzare come formato per veicolare i dati dal Server al Mobile e viceversa un unico formato, standard, riconosciuto a livello europeo le cui specifiche sono di seguito presentate.
La richiesta di modifica del sistema impone dei nuovi requisiti informativi dovuti alla standardizzazione del formato di interscambio delle informazioni tra i sottosistemi ADISys Server ed ADISys Mobile.
Questa modifica non porta alcuna modifica ai casi d'uso del sistema, ma richiede una gestione diversa dei file secondo il formato XML con validazione da Schema XSD.

2.1.5 Formato del file di interscambio (Esportazione – Importazione)

L'albero XML è così costituito:

NODO ROOT <interventi> (contiene tutti i dati del file)

Al suo interno ogni elemento intervento è specificato così:

- attributo **id**, obbligatorio e di tipo intero a 6 cifre;
- attributi **orainizio** ed **orafine** specificati nel seguente formato "hh:mm:ss" dove hh indica le ore; mm indica i minuti; ss indica i secondi
- costituito da uno ed un solo elemento **operatoreIntervento** specificato così:
    - attributo **id**; obbligatorio e di tipo intero a 6 cifre;
- costituito da uno ed un solo elemento **data** specificato nel seguente formato "YYYY-MM-DD" dove:
    - YYYY indica l'anno
    - MM indica il mese
    - DD indica il giorno
- costituito da uno ed un solo elemento **luogo** specificato così:
    - **citta** attributo; obbligatorio di tipo stringa
    - **indirizzo** attributo; obbligatorio di tipo stringa
    - **cap** attributo; obbligatorio di tipo intero a 5 cifre
- costituito da uno ed un solo elemento **paziente** specificato così

- attributo **id**; obbligatorio e di tipo intero a 6 cifre
- uno ed un solo elemento **nome**; obbligatorio di tipo stringa
- uno ed un solo elemento **cognome**; obbligatorio di tipo stringa
- uno ed un solo elemento **dataNascita**; obbligatorio di tipo stringa
- uno ed un solo elemento **cellulari** specificato così:
  - costituito da zero, uno o enne elementi **cellulare** specificato dagli attributi seguenti
  - **numero**; obbligatorio di tipo intero a 10 cifre costituito da uno ed un solo elemento **tipiInterventi** così specificato

- costituito da uno o enne elementi tipoIntervento specificati così
  - attributo **patologia**; obbligatorio e di tipo stringa
  - attributo **nome**; obbligatorio e di tipo stringa
  - uno ed un solo elemento **valoreRilevato** con i seguenti attributi:
  - **tempoIntervento** specificato nel seguente formato "hh:mm:ss" dove:
    - hh indica le ore
    - mm indica i minuti
    - ss indica i secondi
  - uno ed uno solo elemento **note** di tipo stringa
- costituito da uno ed un solo elemento **log** specificato così:
  - costituito da uno ed un solo elemento **rilevazioni** specificato così:
    - zero, uno o enne elementi **rilevazione** specificato così:
      - uno ed un solo elemento **timestamp** con i seguenti attributi:
        - **data** specificato nel seguente formato "YYYY-MMDD" dove:
          - YYYY indica l'anno;
          - MM indica il mese;
          - DD indica il giorno

- **ora** specificato nel seguente formato "hh:mm:ss" dove:
  - hh indica le ore;
  - mm indica i minuti;
  - ss indica i secondi;

❖ uno ed un solo elemento **gps** con i seguenti attributi:

- **latitudine**;obbligatorio di tipo decimale
- **longitudine**;obbligatorio di tipo decimale
- **altitudine**;obbligatorio di tipo decimale
- **accuratezza**;obbligatorio di tipo intero

❖ uno ed un solo elemento **accelerometro** con i seguenti attributi:

- **valorex**; obbligatorio di tipo decimale
- **valorey**; obbligatorio di tipo decimale
- **valorez**; obbligatorio di tipo decimale

Il documento di definizione della grammatica deve ess ere in formato XSD (XML Schema Definition). Ogni istanza di documento deve essere "well formed" e valida.

Il nome fisico del file file di pianificazione delle attività e del file di journaling sarà cosi definito:

codoper_data_device dove

- codoper corrisponde al valore dell'attributo id definito per l'elemento operatoreIntervento
- data corrisponde alla data di produzione del file lato server
- device indica il device che invia il file; assume i valori s ed m dove s indica il device server ed m il device mobile.

3. APPENDICE

3.1 Patterns Utilizzati

Front Controller

L'uso del Front Controller consente la centralizzazione delle richieste da parte delle interfacce. Esso intercetta le richieste provenienti dal client, controlla gli accessi ai servizi di business e restituisce le risposte. In questo modo le finestre non conterranno alcuna logica di business, potendo solamente richiedere servizi al Front Controller.
Con tale pattern si centralizza il punto di accesso alle richieste provenienti dal client che utilizza l'applicazione. Di conseguenza, esso potrà semplicemente istanziare un'altra finestra o richiedere un servizio al livello di dominio.
L'uso del Front Controller aumenta la manutenibilità del livello di dominio, che risulterà separato dalla logica di presentazione.

Factory

Tale pattern "aggira" il problema della creazione degli oggetti senza specificare l'esatta classe dell'oggetto che si vuole manipolare.
Implementata come una classe astratta, viene estesa dall'ACFactory che riprende i metodi del Factory stesso. Il processRequest del Front Controller richiama il metodo principale del pattern, il getInstance, attraverso, appunto, l'ACFactory. Il metodo, recupera il nome della classe di cui si desidera l'istanza, e mediante riflessione, la crea e la istanzia dando come risultato un generico Object. Gli altri metodi implementati nel Factory sono, oltre al costruttore, initMap, che si occupa di inizializzare la mappa rifacendosi al file XML associato, e getValue, che si occupa di recuperare il nome della classe del metodo desiderato. Sono presenti inoltre altri due metodi astratti, implementati nell'ACFactory: getResourcePath e il getKeyNodeName.

Application Controller

L'Application Controller Pattern è utilizzato al fine di centralizzare il controllo per la gestione delle richieste. In questo caso, è utilizzato per centralizzare la logica di business, avvalendosi del Proxy Pattern.

Proxy

Si tratta di un pattern strutturale basato su oggetti che viene utilizzato per accedere ad un oggetto complesso tramite un oggetto semplice.
Il Proxy espone gli stessi metodi dell'oggetto complesso che maschera e questo permette di adattare facilmente l'oggetto senza richiedere modifiche. La necessità di un Proxy è evidente qualora diversi oggetti implementino un'interfaccia comune e soprattutto quando si vuole ritardare la creazione dell'oggetto fino a quando questo non è realmente necessario all'interno del sistema.

Transfer Object

Questo pattern viene utilizzato affinché i client accedano ad altri livelli per poter raccogliere o aggiornare i dati delle entità del sistema, riducendo pertanto le richieste remote verso il Database. E' necessario quindi per poter trasferire i dati tra livelli architetturali differenti.
Il Transfer Object è un oggetto dotato di caratteristiche particolari: contiene solo proprietà e non ha all'interno alcuna logica di business. Come dice lo stesso nome del pattern è quindi un oggetto che serve per trasferire dei dati. In uno scenario di invocazione remota, fare numerose chiamate al server per ottenere dei dati causa un network overhead non indifferente; il Transfer Object aiuta a minimizzare le chiamate.
Dal punto di vista del codice quindi, viene implementato come un POJO (Plain Old Java Object , un normale oggetto Java, non un oggetto speciale).