JConfigurator

Alessandro D'Amico, Raluca Buzamurga

21 Maggio 2020

1 Introduzione

JConfigurator è un tool pensato affinchè l'utilizzatore possa comporre una configurazione personalizzata del PC a seconda della compatibilità hardware dei componenti.
L' assemblaggio avviene scegliendo numericamente i componenti da una lista e alla fine della procedura guidata si ottiene un resoconto con il consumo complessivo della configurazione ed il prezzo d'acquisto.

2 Target commerciale

Il tool è pensato per agevolare la preventivazione di **PC custom** assemblati dagli shop informatici: tramite JConfigurator il cliente può relizzare autonomamente la fattibilità (anche economica) del prodotto desiderato, senza dover telefonare o inviare email. JConfigurator è dunque uno strumento utile ad accrescere la **QoE**[1] della clientela. Potenzialmente il software è inseribile in un contesto dotato di un database per memorizzare le configurazioni ottenute nello step finale come ordini online effettivi.

3 UML, Struttura e funzionamento del tool

Il progetto si articola su 5 package che vengono descritti nel seguito. Come macro-architettura il tool è molto simile ad un progetto JEE, con la differenza che il managament dei dati degli oggetti non avviene su un RDBMS ma avviene invece in locale mediante JAXB (Figure 1).

I package configuratorEngine + guiDemo + sequentialAssembler realizzano il Model View Controller (la FullConfig in configuratorEngine contiene rappresenta il Model, la view è la classe UserGui e i Controller utilizzati sono le classi all'interno del package sequentialAssembler vi sono le varie classi che fungono da Controller). Allo stesso modo i package configuratorEngine + ShopDemoGui + shopDataManagement realizzano un Model View Controller (anche in esso il model è sempre la FullConfig, mentre la View è la classe ShopDemoGui ed i Controller sono le classi all'interno del package ComputerShopGui).

3.1 configuratorEngine

il primo package, "ConfiguratorEngine" (Figure 2) riguarda il core concettuale dell'oggetto da comporre: la classe FullConfig rappresenta la configurazione completa ed è stata realizzata come composizione dei suoi componenti fisici.

[1] QoE: Quality Of Experience

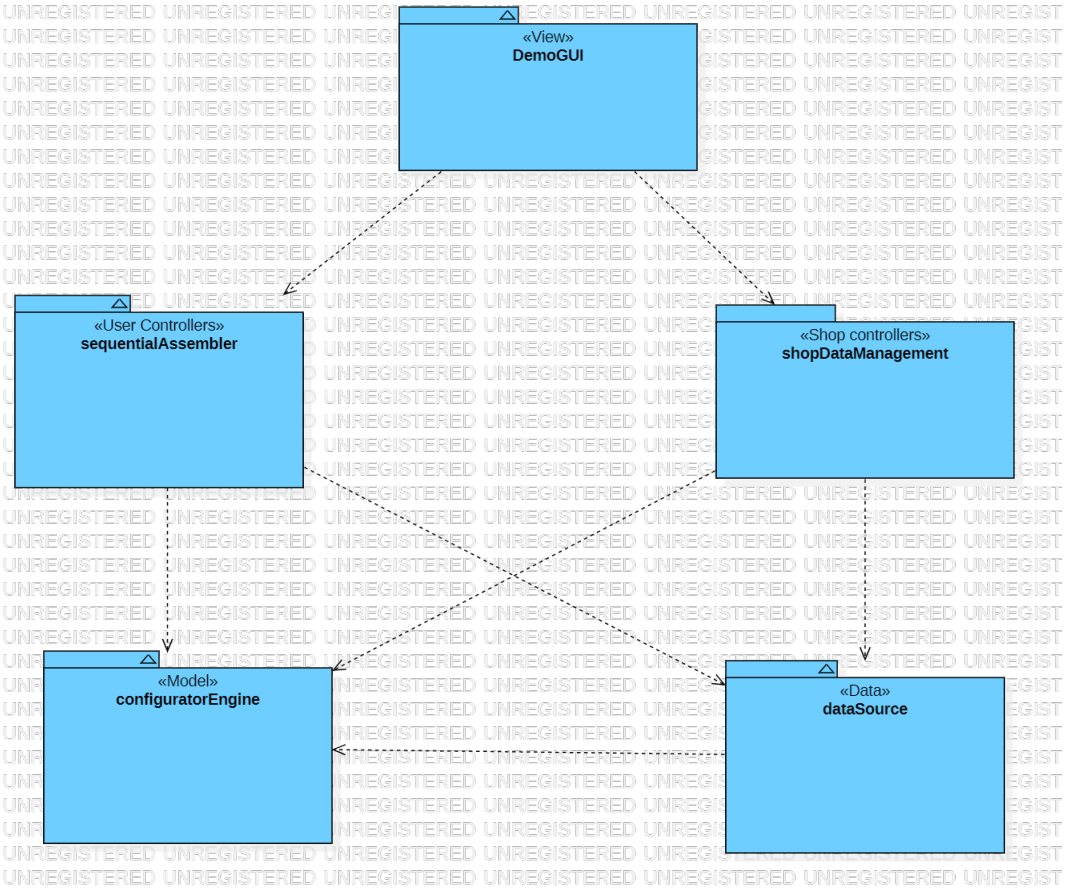

Figure 1: Package Diagram di JConfigurator

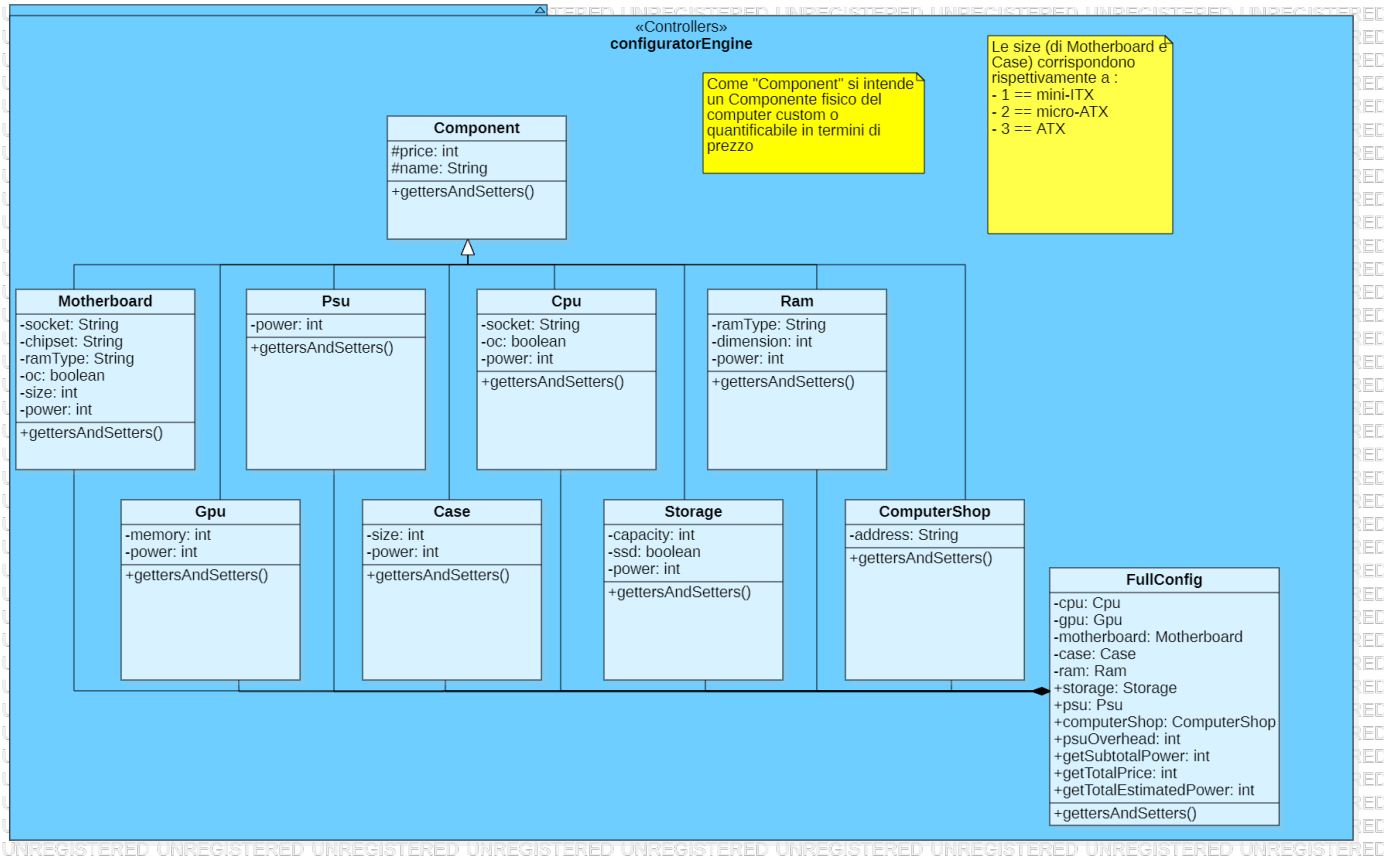

Figure 2: configuratorEngine package

3.2 dataSource

Il secondo package, "dataSource" (Figure 3) è dedicato alla gestione delle operazioni di CRUD (degli oggetti corrispondenti allo schema precedentemente citato) nei file Xml in cui sono stati precedentemente scritti: Il codice è condiviso su ComponentDao e le classi figlie, specificando il componente per cui devono effettuare l'operazione devono specificare il modo con cui operano le varie operazioni (attulmente sfruttano il codice già presente sulla classe padre, ma sono possibili altre implementazioni). Si è reso necessario creare una classe per componente in quanto ogni componente ha chiaramente attributi diversi e oltre a ciò è necessario un contenitore per poter riuscire a "incapsulare" le ArrayList degli oggetti dei vari componenti: un'alternativa potrebbe essere stata inserire le liste e le operazioni di CRUD all'interno del primo schema, ma ciò avrebbe causato una forte dipendenza con la modalità di acquisizione degli oggetti, oltre a "sporcare" tali classi con ruoli che non vi appartengono. i file Xml da cui attingono i DAOs sono 3 per ciascun Component: 1 file Xml contenente una lista vuota, 1 file con i componenti di default ed 1 file con gli oggetti attualmente accessibili al cliente.

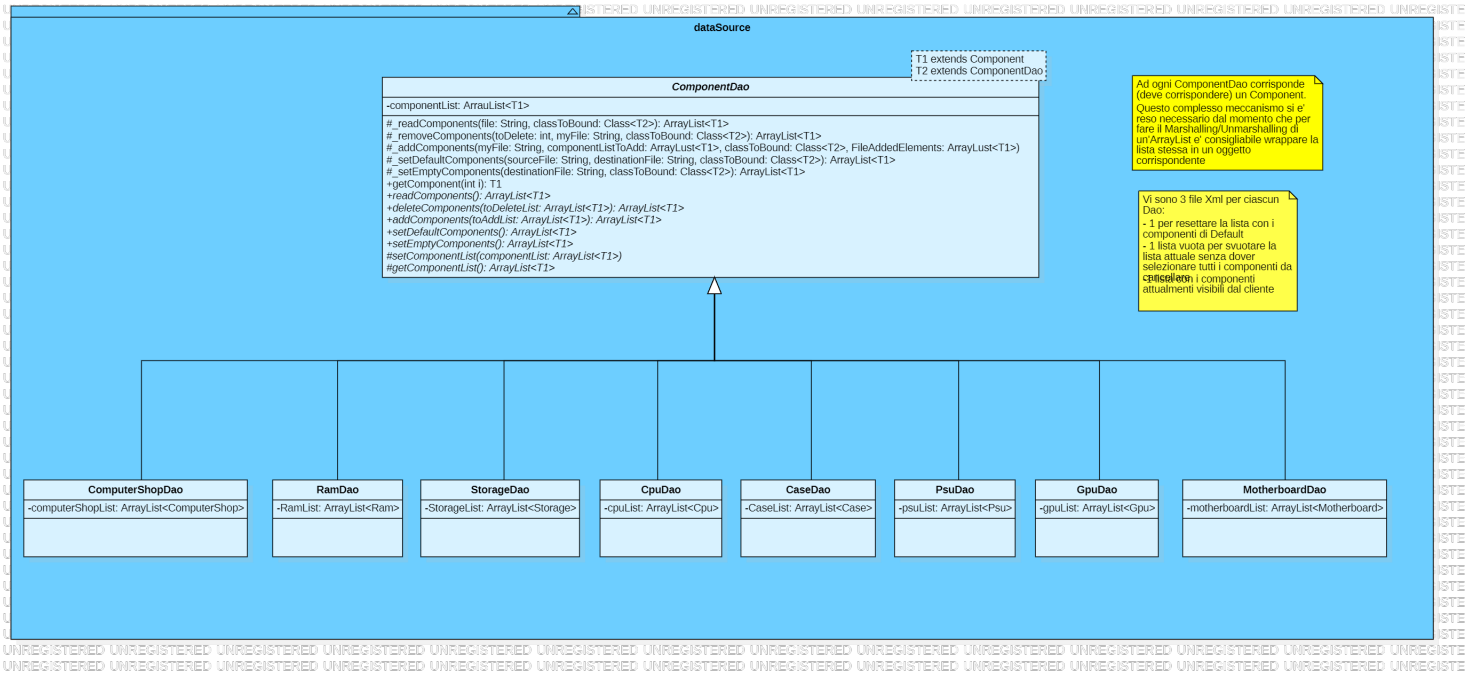

Figure 3: dataSource package

Nella funzione _readComponents (Figure 4) viene creato un contesto JAXB affinchè un apposito Unmarshaller possa estrarre dal file Xml il contenuto della classe specificata (il parametro class2Bound). Viene dunque restituita l'ArrayList dei componenti memorizzati all'interno del file.
Nella funzione _removeComponents (Figure 5) viene effettuata la medesima operazione sopra citata e successivamente vengono rimosse le componenti dall'ArrayList ottenuto. Infine viene effettuato il re-Marshalling e quindi sovrascritta la lista nell'Xml originario.
Nella funzione _setDefaultComponents (Figure 6) viene effettuato l'unmarshalling dei componenti del file il cui percorso è indicato come sourceFile come in _readComponents per poi effettuare il Marshalling della classe ottenuta sul file il cui percorso è indicato su destinationFile.

```
final protected ArrayList<T1> _readComponents(String myFile, Class<T2> class2Bound) throws JAXBException {

    JAXBContext ctx = JAXBContext.newInstance(class2Bound);
    Unmarshaller unm = ctx.createUnmarshaller();
    File fout = new File(myFile);
    T2 newComponent = class2Bound.cast(unm.unmarshal(fout));
    ArrayList<T1> co = newComponent.getComponentList();
    return co;

}
```

Figure 4: _readComponents() function

```
final protected ArrayList<T1> _removeComponents(ArrayList<T1> toDeleteList, String myFile, Class<T2> class2Bound)
        throws JAXBException {

    JAXBContext ctx = JAXBContext.newInstance(class2Bound);
    Unmarshaller unm = ctx.createUnmarshaller();
    File fout = new File(myFile);
    T2 componentDao = class2Bound.cast(unm.unmarshal(fout));

    ArrayList<T1> componentDaoList = componentDao.getComponentList();
    componentDaoList.removeAll(toDeleteList);
    componentDao.setComponentList(componentDaoList);

    Marshaller m = ctx.createMarshaller();
    m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
    m.marshal(componentDao, fout);

    return componentDao.getComponentList();
}
```

Figure 5: _removeComponents() function

```
final protected ArrayList<T1> _setDefaultComponents(String sourceFile, String destinationFile,
        Class<T2> class2Bound) throws JAXBException {

    JAXBContext context = JAXBContext.newInstance(class2Bound);
    Unmarshaller ums = context.createUnmarshaller();
    T2 componentDao = class2Bound.cast(ums.unmarshal(new File(sourceFile)));

    Marshaller mrs = context.createMarshaller();
    mrs.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
    mrs.marshal(componentDao, new File(destinationFile));

    return componentDao.getComponentList();
}
```

Figure 6: _setDefaultComponents() function

3.2.1 dataSource tests

I test sul package dataSource sono stati effettuati simulando le operazioni CRUD sugli oggetti in locale e dunque comparando il risultato di esse con le operazioni eseguite sugli Xml mediante JAXB. Ad esempio, l'inserimento di CpuDao (Figure 7) è stato testato aggiungendo ad una lista di Cpu vuota la lista letta mediante la funzione readComponents (anch'essa testata) per poi aggiungere sia alla lista gestita dal CpuDao che alla lista gestita in locale due Cpu (cpu1 e cpu2) e dunque comparando i due risultati ottenuti. Le altre funzioni del package sono state testate in modo similare.

```
CpuDao cpuDao;
ArrayList<Cpu> cpuList;
Cpu cpu1;
Cpu cpu2;

@BeforeEach
void init() {
    cpuDao = new CpuDao();
    cpuList = new ArrayList<Cpu>();
    cpu1 = new Cpu("Cpu1", 10, 10, "socket", true);
    cpu2 = new Cpu("Cpu2", 10, 10, "socket", true);
    cpuList.add(cpu1);
    cpuList.add(cpu2);
}

@Test
void testAddComponents() throws JAXBException {
    ArrayList<Cpu> localCpuList = cpuDao.readComponents();

    cpuDao.addComponents(cpuList);
    localCpuList.addAll(cpuList);

    assertEquals(localCpuList, cpuDao.readComponents(), "Add a cpu list");

}
```

Figure 7: tests on dataSource

3.3 sequentialAssembler

il terzo package, "sequentialAssembler", contiene i controller con cui viene assemblato il PC custom: a ogni componente corrispondono delle operazioni di check che vengono svolte in sequenza per poter esporre all'utente solo i componenti compatibili con i componenti inseriti nei passi precedenti. I check si basano sulle comparazioni tra specifici attributi, come ad esempio il socket per la compatibilità tra Cpu e Motherboard, oppure la dimensione (intesa come form-factor) tra Motherboard e Case (Figure 2).

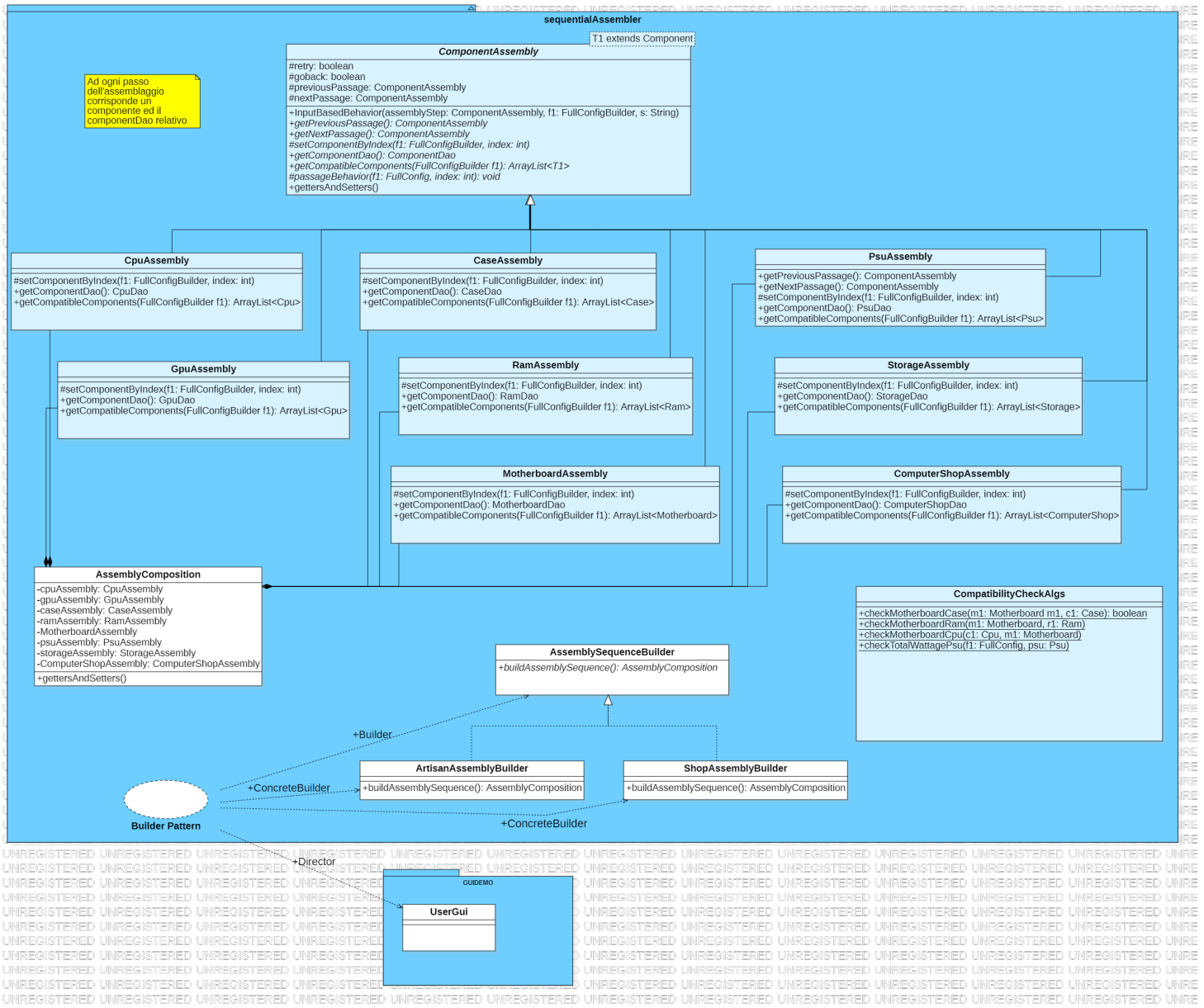

Figure 8: sequentialAssembler package

I metodi condivisi da più componenti sono nella classe CompatibilityCheckAlgs: se così non fosse si potrebbe introdurre una eventuale asimmetria oppure una dipendenza non desiderata tra un componente ed un altro (un esempio di implementazione dei due metodi è in figura 14).
All'interno del package è realizzato il pattern "Builder": AssemblySequenceBuilder è l'asbtract Builder, mentre ArtisanAssemblyBuilder e ShopAssemblyBuilder sono le sue implementazioni concrete (Figure 15, 16, 17). Il Builder è stato inserito per facilitare la gestione di assemblaggi il cui ordine avviene in modo differente oppure non vengono eseguiti tutti i passi dell'assemblaggio: sfruttando la seconda proprietà abbiamo introdotto un assemblaggio che omette la scelta del ComputerShop (così che il cliente possa scegliere se considerarlo come spesa nel resoconto finale o meno). Ciò che costruisce il builder è dunque la sequenza di assemblaggio stessa, creando una AssemblyComposition e installando le

dipendenze necessarie tra predecessore e successore. Il Director è invece il Client, che nel nostro caso corrisponde con la classe UserGui nel package guiDemo.

3.3.1 sequentialAssembler tests

I test sono stati effettuati sul check singolo del componente, verificando il risultato atteso con selezioni compatibili o incompatibili dei componenti di cui bisogna verificare la compatibilità (in Figure 10 ad esempio si comparano Motherboard e Cpu con socket diverso e con lo stesso socket). In Figure 9 viene mostrato il test del metodo che sfrutta la medesima funzione per ottenere una lista di componenti compatibili partendo da una lista generica del medesimo tipo di componenti.

```
30    @BeforeEach
31    @AfterEach
32    void init() throws JAXBException {
33        cpuAssembly = new CpuAssembly();
34        cpuDao = new CpuDao();
35        cpuDao.setEmptyComponents();
36        f1 = new FullConfig();
37    }
38
39    @Test
40    void getCompatibleComponentsTest() throws JAXBException {
41
42        Cpu cpu0 = new Cpu("Prova0", 0, 0, "AM4", true);
43        Cpu cpu1 = new Cpu("Prova1", 0, 0, "LGA1150", true);
44        Cpu cpu2 = new Cpu("Prova2", 0, 0, "LGA1151", true);
45        Cpu cpu3 = new Cpu("Prova3", 0, 0, "AM4", true);
46
47        ArrayList<Cpu> cpuList = new ArrayList<>();
48        cpuList.add(cpu0);
49        cpuList.add(cpu1);
50        cpuList.add(cpu2);
51        cpuList.add(cpu3);
52
53        cpuDao.addComponents(cpuList);
54        Motherboard motherboard = new Motherboard("MotherboardProva", 43, 3, "LGA1150", "Z77", "DDR3", false, 2);
55
56        f1.setMotherboard(motherboard);
57
58        ArrayList<Cpu> compatibleComponents = cpuAssembly.getCompatibleComponents(f1);
59
60        assertFalse(compatibleComponents.contains(cpu0),
61                "Comparing incompatible Motherboard and Cpu: different sockets");
62        assertTrue(compatibleComponents.contains(cpu1), "Comparing compatible Motherboard and Cpu: same sockets");
63        assertFalse(compatibleComponents.contains(cpu2),
64                "Comparing incompatible Motherboard and Cpu: different sockets");
65        assertFalse(compatibleComponents.contains(cpu3),
66                "Comparing incompatible Motherboard and Cpu: different sockets");
67
68    }
```

Figure 9: tests on sequentialAssembler

```
@Test
void testCheckMotherboardCpu() {

    Motherboard motherboard1 = new Motherboard("MSI Basem", 66, 10, "LGA1151", "H110", "DDR3", false, 1);
    Motherboard motherboard2 = new Motherboard("Gigabyte ga-ab359m-g3 (AM4 DDR4)", 96, 10, "AM4", "ab359m", "DDR4",
            true, 3);
    Cpu cpu1 = new Cpu("Intel i5 7600K (lga 1151)", 222, 10, "LGA1151", true);
    Cpu cpu2 = new Cpu("AMD Ryzen 5 1400 (AM4)", 176, 10, "AM4", true);

    assertTrue(CompatibilityCheckAlgs.checkMotherboardCpu(cpu1, motherboard1),
            "Checking compatible Motherboard and Cpu: same socket");
    assertTrue(CompatibilityCheckAlgs.checkMotherboardCpu(cpu2, motherboard2),
            "Checking compatible Motherboard and Cpu: same socket");
    assertFalse(CompatibilityCheckAlgs.checkMotherboardCpu(cpu2, motherboard1),
            "Checking incompatible Motherboard and Cpu: different socket");
    assertFalse(CompatibilityCheckAlgs.checkMotherboardCpu(cpu1, motherboard2),
            "Checking incompatible Motherboard and Cpu: different socket");

}
```

Figure 10: tests on CompatibilityCheckAlgs

3.4 shopDataManagement

il package ShopDataManagement è un controller che rende possibili le operazioni che deve effettuare il negozio, quali l'inserimento di nuovi componenti, la rimozione di quelli già presenti indicando l'indice corrispondente alla lista e il reset degli Xml attualmente utilizzati. Per realizzare la funzione attraverso cui vengono scelte le liste di componenti su cui lavorare è stato realizzato un Factory Method Pattern (Figure 3), in cui il Creator è la classe astratta ComponentDataManagement e ogni classe da essa derivata rappresenta un ConcreteCreator. Dal package dataSource abbiamo invece il ComponentDao che il ruolo di Abstract Product e ogni sua classe figlia è un Concrete Product (Figure 3).

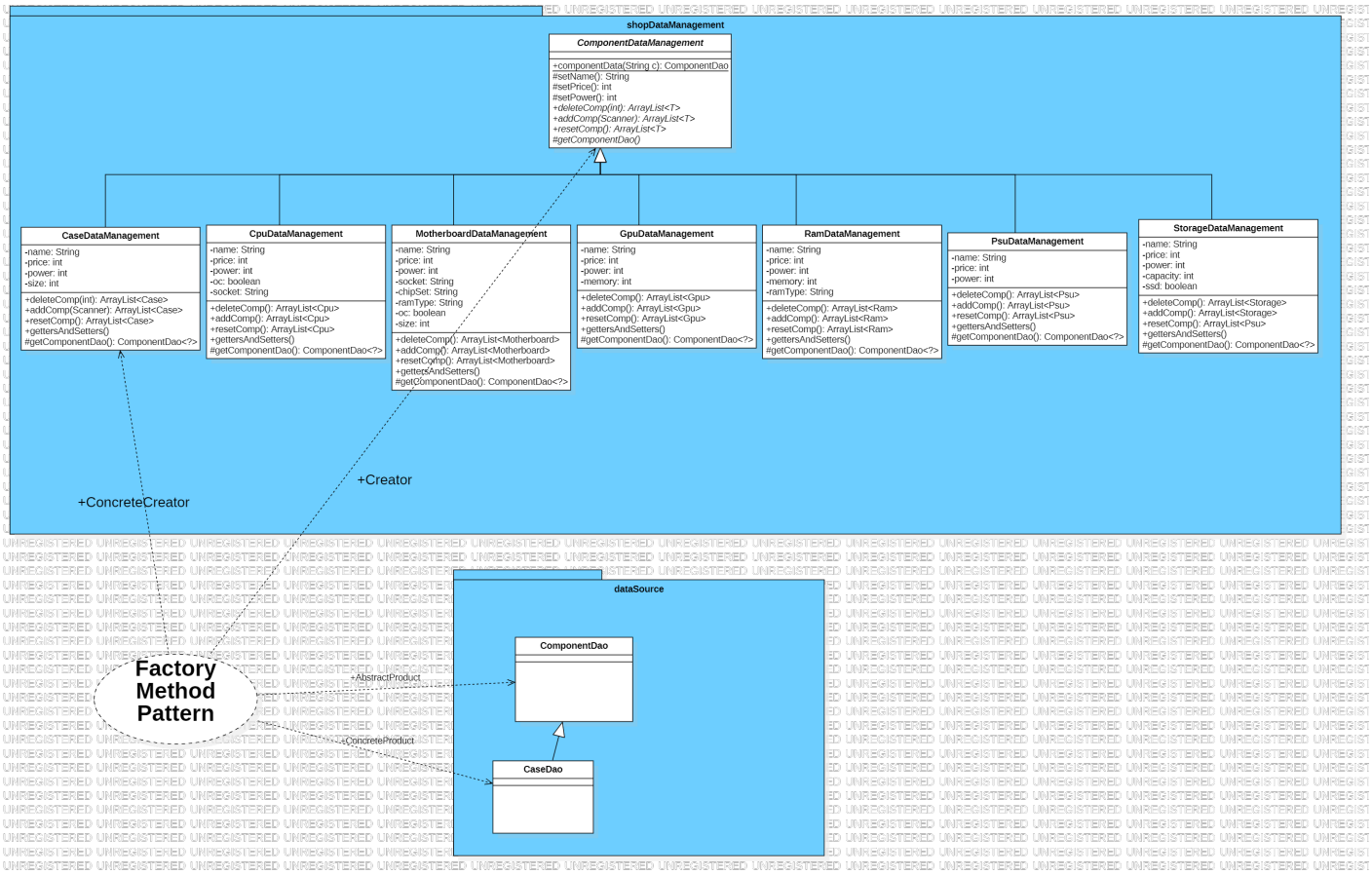

Figure 11: shopDataManagement package

3.4.1 shopDataManagement tests

I test in shopDataManagementTests, vengono eseguiti per ogni componentDataManagement: si effettuano i test dei metodi addComp, deleteComp e resetComp. Prima di eseguire i test, viene salvato in un ArrayList il contenuto del file xml corrispondente al componente di riferimento. DeleteCompTest: viene eliminata una componente di indice random dal file xml tramite il metodo deleteComp, in seguito si assicura che il contenuto del file sia diverso dal contenuto della lista salvata precedentemente. Infine viene eliminata la componente con lo stesso indice dalla lista locale e asserito che il contenuto del file sia uguale a quello della lista. AddCompTest utilizza lo stesso principio di DeleteCompTest, ma aggiungendo una componente anzicè rimuovendola. Un esempio di test è presente in Figure 12.

```java
@BeforeEach
public void init() throws JAXBException {
    cpuList = cpuDao.readComponents();
    cpuData = new CpuDataManagement();
}

@Test
public void deleteCpuTest() throws JAXBException {
    if(cpuDao.readComponents().size() > 0) {
        int index = (int)(Math.random() * cpuDao.readComponents().size());
        assertNotEquals(cpuData.deleteComp(index), cpuList, "Makes sure that components in Cpu.XML are not the same as before, after the deletion.");
        cpuList.remove(index);
        assertEquals(cpuDao.readComponents(), cpuList, "Makes sure that the componenent was removed from Cpu.XML file");
    }
}
```

Figure 12: tests on shopDataManagement

3.5 guiDemo

Questo ultimo package intende essere unicamente una implementazione semplificata mediante terminale del configuratore: UserGui contiene un main dedicato alle operazioni del Cliente (la creazione di una configurazione "artigianale" e di una configurazione "shop" che comprende nel prezzo lo shop a cui si vuole assegnare l'incarico di assemblare fisicamente il PC), mentre ShopDemoGui contiene un main dedicato allo Shop (tramite esso lo Shop può inserire, rimuovere o resettare le liste Xml a cui ha accesso il Cliente). Mediante la GUI si realizzano dunque i casi d'uso identificati (Figure 13).

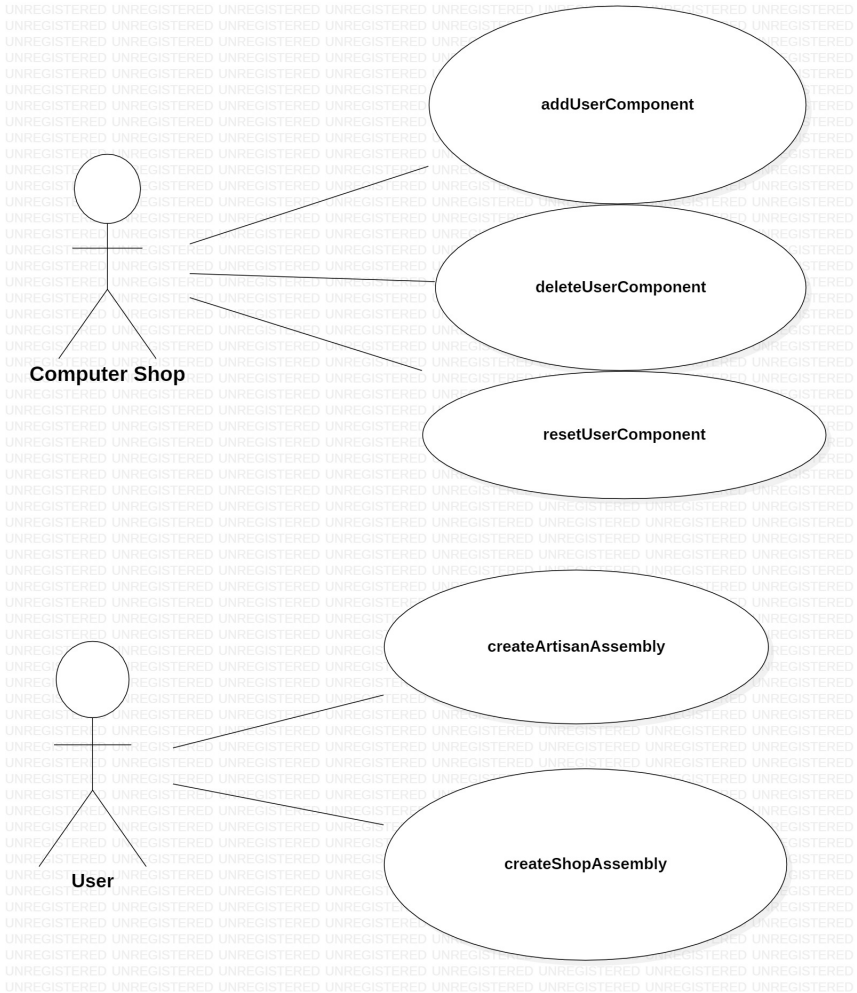

Figure 13: Use Case Diagram

```java
public class CompatibilityCheckAlgs {

    /**
     * Checks if a motherboard and a case are compatible by comparing the sizes on
     * both
     *
     * @param m1 Motherboard to be checked
     * @param c1 Case to be checked
     * @return the result of the check (true if compatible, false if not)
     */
    static public boolean checkMotherboardCase(Motherboard m1, Case c1) {
        if (c1.getSize() >= m1.getSize())
            return true;
        else
            return false;
    }

    /**
     * Checks if a motherboard and a ram are compatible by comparing the type of ram
     * socket on both
     *
     * @param m1 Motherboard to be checked
     * @param r1 Ram to be checked
     * @return the result of the check (true if compatible, false if not)
     */
    static public boolean checkMotherboardRam(Motherboard m1, Ram r1) {
        if (r1.getRamType().contentEquals(m1.getRamType()))
            return true;
        else
            return false;
    }
```

Figure 14: CompatibilityCheckAlgs sample

```java
package sequentialAssembler;

public interface AssemblySequenceBuilder {
    public AssemblyComposition buildAssemblySequence();

}
```

Figure 15: Builder

```java
package sequentialAssembler;

public class ArtisanAssemblyBuilder implements AssemblySequenceBuilder {

    @Override
    public AssemblyComposition buildAssemblySequence() {
        AssemblyComposition myComposition = new AssemblyComposition();

        // set step 1: Cpu
        myComposition.getCpuAssembly().setNextPassage1(myComposition.getGpuAssembly());
        // set step 2: Gpu
        myComposition.getGpuAssembly().setPreviousPassage1(myComposition.getCpuAssembly());
        myComposition.getGpuAssembly().setNextPassage1(myComposition.getMotherboardAssembly());
        // set step 3: Motherboard
        myComposition.getMotherboardAssembly().setPreviousPassage1(myComposition.getGpuAssembly());
        myComposition.getMotherboardAssembly().setNextPassage1(myComposition.getRamAssembly());
        // set step 4: Ram
        myComposition.getRamAssembly().setPreviousPassage1(myComposition.getMotherboardAssembly());
        myComposition.getRamAssembly().setNextPassage1(myComposition.getCaseAssembly());
        // set step 5: Case
        myComposition.getCaseAssembly().setPreviousPassage1(myComposition.getRamAssembly());
        myComposition.getCaseAssembly().setNextPassage1(myComposition.getStorageAssembly());
        // set step 6: Storage
        myComposition.getStorageAssembly().setPreviousPassage1(myComposition.getCaseAssembly());
        myComposition.getStorageAssembly().setNextPassage1(myComposition.getPsuAssembly());
        // set step 7: Psu
        myComposition.getPsuAssembly().setPreviousPassage1(myComposition.getStorageAssembly());

        return myComposition;
```

Figure 16: Concrete Builder ArtisanAssemblyBuilder

```java
package sequentialAssembler;

public class ShopAssemblyBuilder implements AssemblySequenceBuilder {

    @Override
    public AssemblyComposition buildAssemblySequence() {
        AssemblyComposition myComposition = new AssemblyComposition();

        // set step 1: Cpu
        myComposition.getCpuAssembly().setNextPassage1(myComposition.getGpuAssembly());
        // set step 2: Gpu
        myComposition.getGpuAssembly().setPreviousPassage1(myComposition.getCpuAssembly());
        myComposition.getGpuAssembly().setNextPassage1(myComposition.getMotherboardAssembly());
        // set step 3: Motherboard
        myComposition.getMotherboardAssembly().setPreviousPassage1(myComposition.getGpuAssembly());
        myComposition.getMotherboardAssembly().setNextPassage1(myComposition.getRamAssembly());
        // set step 4: Ram
        myComposition.getRamAssembly().setPreviousPassage1(myComposition.getMotherboardAssembly());
        myComposition.getRamAssembly().setNextPassage1(myComposition.getCaseAssembly());
        // set step 5: Case
        myComposition.getCaseAssembly().setPreviousPassage1(myComposition.getRamAssembly());
        myComposition.getCaseAssembly().setNextPassage1(myComposition.getStorageAssembly());
        // set step 6: Storage
        myComposition.getStorageAssembly().setPreviousPassage1(myComposition.getCaseAssembly());
        myComposition.getStorageAssembly().setNextPassage1(myComposition.getPsuAssembly());
        // set step 7: Psu
        myComposition.getPsuAssembly().setPreviousPassage1(myComposition.getStorageAssembly());
        myComposition.getPsuAssembly().setNextPassage1(myComposition.getShopAssembly());
        // set step 8: Shop
        myComposition.getShopAssembly().setPreviousPassage1(myComposition.getPsuAssembly());

        return myComposition;

    }
```

Figure 17: Concrete Builder ShopAssemblyBuilder

(Documentazione aggiuntiva ottenuta mediante JavaDocs è reperibile nella cartella doc del progetto)