UNIVERSITÀ DEGLI STUDI DI CAGLIARI

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI

Corso di Laurea Specialistica in
Tecnologie Informatiche

Anno Accademico 2005-2006

Applicazioni Distribuite mediante Web Service e sistemi di Workflow

Tesi di Laurea di
Antonio Pintus

Relatore
Prof. Andrea Bosin

Questo lavoro è dedicato in particolare modo a Claudia,

per tantissimi e importanti motivi ma soprattutto per il sostegno e l'incoraggiamento che negli ultimi due anni non è mai mancato.

Ringrazio la mia famiglia: papà Franco, mamma Anna e mia sorella Maria Delfina, per l'appoggio costante e per essere come sono.

Ringrazio il Prof. Andrea Bosin per l'aiuto, la pazienza e la competenza.

Un caloroso ringraziamento va agli amici Stefano Sanna, Andrea Piras e Luca Atzori, per esserci, sempre.

Indice generale

1. Introduzione 11

2. Service Oriented Architecture 13

2.1.Tecnologia dei Web Service 14

2.1.1.Web Services Description Language (WSDL) 17

2.1.2.Universal Description Discovery and Integration (UDDI) 21

2.1.3.Simple Object Access Protocol (SOAP) 23

2.2.Web Services Orchestration 29

2.2.1.BPEL, Business Process Execution Language (for Web Services) 30

2.3.Ambienti e tool di sviluppo per i Web Service in Java 37

2.3.1.Apache Tomcat 38

2.3.2.Apache Axis 41

3. Sistemi di gestione di Workflow 43

3.1.Process 50

3.2.Parallel Routing 50

3.3.Sequential Routing 50

3.4.AND-Split 51

3.5.AND-Join 51

3.6.OR-Split 52

3.7.OR-Join 53

3.8.Iteration..... 53

3.9.Pre-Condition..... 54

3.10.Post-Condition..... 54

3.11.Transition..... 54

3.12.Transition Condition..... 54

4. Analisi di due Workflow Management System per l'e-Science..... 55

4.1.Taverna 55

4.2.Triana..... 63

4.2.1.Distributed Computing con Triana..... 64

4.3.Funzionalità di Taverna e Triana a confronto..... 74

4.3.1.Sequential Routing..... 78

4.3.2.Parallel Routing..... 79

4.3.3.Conditional Processing..... 81

4.3.4.Iteration..... 83

5. Bioinformatica e servizi per la bioinformatica..... 87

5.1.Un esempio di applicazione dei Web Service e sistemi di Workflow alla bioinformatica 89

5.2.Un possibile scenario applicativo..... 100

6. Conclusioni..... 103

7. Bibliografia..... 105

8. APPENDICE A..... 111

Indice delle figure

Figura 1: I tre attori principali nell'architettura dei Web Service..... 16
Figura 2: Il descrittore WSDL per un semplice Web Service.....19
Figura 3: Il documento XML Schema con la dichiarazione dei tipi di dato utilizzati nel WSDL della figura precedente.....20
Figura 4: UDDI Registry, attori e suo utilizzo.....22
Figura 5: La struttura di un generico messaggio SOAP..... 24
Figura 6: Messaggio SOAP inviato dall'applicazione client al Web Service per ottenere la somma degli interi 2 e 3.....25
Figura 7: Messaggio SOAP, inviato dal Web Service al client, per comunicare la somma dei due interi specificati nella richiesta, ossia il risultato dell'operazione remota 26
Figura 8: Esempio di documento di descrizione di un workflow espresso in BPEL.....34
Figura 9: Rappresentazione grafica di un processo BPEL (creato con Netbeans, un popolare ambiente di sviluppo per Java).....35
Figura 10: Apache Tomcat visto come una "blackbox".....38
Figura 11: L'architettura interna di Apache Tomcat.....39
Figura 12: Il Workflow Reference Model secondo la Workflow Management Coalition (WfMC).....44
Figura 13: Architettura generica della definizione di un processo di workflow secondo la WfMC.....45
Figura 14: Il meta-modello di base per la definizione di un workflow così come definito dalla WfMC.....47
Figura 15: Un processo come sequenza di attività.....50
Figura 16: AND-Split.....51

Figura 17: AND-Join..... 52
Figura 18: OR-Split..... 52
Figura 19: OR-Join..... 53
Figura 20: Iteration..... 53
Figura 21: Il workbench di Taverna con un workflow in fase di design..... 56
Figura 22: Esecuzione di un workflow in Taverna, la finestra dell'Enactor mostra il flusso d'esecuzione e gli eventuali problemi, come in questo caso..... 57
Figura 23: Documento in formato XSCUFL per la descrizione di un workflow in Taverna..... 60
Figura 24: L'ambiente di Triana con la composizione e la personalizzazione di un workflow..... 64
Figura 25: Architettura generale di Triana..... 65
Figura 26: Formato XML nativo di Triana per la descrizione dei workflow..... 73
Figura 27: Workflow realizzato con Taverna per incrementare di uno la somma di due numeri casuali..... 79
Figura 28: Workflow realizzato con Triana per incrementare di uno la somma di due numeri casuali..... 79
Figura 29: Parallel Routing in Taverna..... 80
Figura 30: Parallel Routing in Triana..... 80
Figura 31: Lo script in Java, per il confronto di due numeri in virgola mobile, che viene eseguito da BeanShell all'interno di Taverna..... 82
Figura 32: Workflow con salto condizionato in Taverna..... 82
Figura 33: Implicit Iteration in Taverna; se ad un processore che accetta un parametro a in input viene passata una lista di elementi, Taverna applica automaticamente un iterazione invocando il processore per ciascun elemento in lista..... 84
Figura 34: Le due diverse configurabili strategie di iterazione in Taverna nel caso

di parametri multipli: strategie di cross e dot product..... 85
Figura 35: Implementazione di un loop in Triana: utilizzando il componente Loop ed impostando la condizione d'uscita tra le sue proprietà possiamo creare un ciclo condizionato. In questo esempio viene invocato un Web Service che genera un numero casuale il quale viene incrementato (sempre mediante l'invocazione di un servizio remoto) per 10 volte..... 86
Figura 36: Esempio di un microarray con circa 40000 DNA spot.....90
Figura 37: Esperimento di classificazione di attributi genomici mediante workflow composto con Triana: i componenti in celeste sono i componenti locali all'ambiente, in rosso i componenti distribuiti, ovvero i Web Service dislocati su macchine remote.....93
Figura 38: Esperimento di classificazione di attributi genomici mediante workflow composto con Taverna: i componenti in violetto sono i componenti locali all'ambiente, in verde i componenti distribuiti, ovvero i Web Service dislocati su macchine remote; in figura sono messi in evidenza anche le operazioni invocate..... 94
Figura 39: I risultati dell'esperimento effettuato vengono sia scritti su file in locale, sia mostrato a video come richiesto dai due workflow, creati ed eseguiti in Triana e Taverna, per un attributeNumber uguale a 10..... 97
Figura 40: Risultati per l'esperimento ripetuto facendo uso dell'implicit loop di Taverna, in input la lista di valori 10, 50, 80, 100, 120, 150, 180, 200, 220, 250, 280, 300, 320, 350, 380, 400 come attributeNumber, ovvero il numero di feature da selezionare di volta in volta.....98
Figura 41: Il nuovo workflow, realizzato con Triana, per ripetere ciclicamente l'esperimento con una lista di attributeNumber passata in input mediante il parametro inputList.....99
Figura 42: Scenario collaborativo per esperimenti scientifici, reso possibile

dall'utilizzo congiunto di SOA, Web Service e di Workflow Management System 101

Indice delle tabelle

Tabella 1: Le possibili combinazioni di Binding Style, Encoding Style in un messaggio SOAP..... 27
Tabella 2: Operazioni esposte dal SimpleMathWebService, Web Service di test 76
Tabella 3: Le operazione esposte dal UtilitiesWebService, Web Service di test... 77
Tabella 4: Operazioni remote messe a disposizione dai Web Service implementati per esperimenti di bioinformatica 92

1. Introduzione

A partire dagli anni '70, sono state date molteplici definizioni di *sistema distribuito,* naturalmente ognuna risentiva delle tecnologie e della visione dell'hardware e del software all'epoca prevalenti. Nel 1995, ad esempio, *Andrew S. Tanembaum* dava la seguente definizione: *"un sistema distribuito consiste di un insieme di calcolatori indipendenti che appaiono all'utente del sistema come un singolo calcolatore"* [22], ma anche questa definizione era decisamente orientata verso una definizione di un *sistema operativo distribuito* piuttosto che verso una visione fatta di *applicazioni* distribuite.
In questo lavoro, per sistemi distribuiti, intendiamo piuttosto *applicazioni indipendenti, distribuite in rete, che cooperano per il raggiungimento di un risultato comune e la comunicazione tra di esse avviene mediante lo scambio di messaggi.*
Lo studio e la realizzazione di tecnologie e strumenti per l'implementazione di una reale cooperazione tra sistemi, anche decisamente eterogenei tra loro, è stato il motore che ha fatto si che fossero creati sistemi come *CORBA, Java RMI, DCOM, XML-RPC* ed altri. Negli ultimi anni, con l'introduzione della tecnologia dei *Web Service* e la connessa definizione di protocolli per lo scambio dei messaggi tra applicazioni, come *SOAP*, si è assistito ad una ulteriore evoluzione nel campo delle applicazioni distribuite. Infatti, con il grande successo che questa tecnologia ha avuto e sta avendo, e quindi grazie ad una sua ampia adozione nei più eterogenei domini applicativi, per esempio applicazioni di *e-Business*, *e-Learning, e-Science* e *bioinformatica*, tanto per citarne qualcuna, si sta vivendo ora in una nuova era dei sistemi distribuiti, quella delle *architetture orientate ai servizi (Service Oriented Architectures)* nelle quali le applicazioni distribuite in rete sono viste come *servizi,* perché in grado di "fare" o "fornire" qualcosa di utile a qualunque altra applicazione ne avesse bisogno per il completamento delle proprie funzionalità o

per il raggiungimento di uno scopo prefissato; di conseguenza sono stati coniati termini come *Web Service Orchestration* e *Web Service Coreography* che rendono l'idea di un nuovo modo di concepire e vedere le applicazioni distribuite ed il suo utilizzo, finalmente consentendo la comunicazione tra sistemi, piattaforme (implementative e di supporto) ed applicazioni spesso realmente eterogenee tra loro nonché dislocate fisicamente su macchine remote connesse ad Internet. Con questa nuova prospettiva per i sistemi distribuiti, ritornano *in auge*, concetti come i *workflow* e i relativi *Workflow Management System*, che trovano anch'essi nuova vita e reale terreno fertile per una loro applicazione e diffusione. In questo lavoro verranno analizzati tutti questi argomenti, sia dal punto di vista teorico e dello stato dell'arte, sia dal punto di vista applicativo, dando preferenza ai Web Service realizzati mediante *SOAP* (anziché quelli, emergenti, realizzati attraverso il modello *REST, Representational State Transfer)* e con la presentazione di due sistemi di workflow, entrambi disponibili secondo il modello *open-source*, entrambi orientati alle applicazioni di carattere scientifico. Si parlerà anche dello standard *BPEL (Business Process Execution Language)* ma quest'ultimo verrà relegato solamente al discorso sullo stato dell'arte perché uno degli scopi di questo lavoro è capire la reale applicabilità dei sistemi di workflow orientati all'*e-Science* e alla *bioinformatica* in particolare modo; a tal proposito, nell'ultima parte di questo documento, verrà presentato un reale esperimento di applicazione dei workflow e della tecnologia dei Web Service alla bioinformatica e al data-mining di dati genomici.

2. Service Oriented Architecture

La *Service Oriented Architecture (SOA)* nasce principalmente per rispondere all'esigenza di integrare tra loro sistemi, generalmente eterogenei, orientati al *Business to Business (B2B).* Quindi l'introduzione delle tecnologie riunite attorno al concetto di SOA, agiscono con lo scopo primario di realizzare un passaggio da un mondo di architetture eterogenee alla creazione di una infrastruttura di integrazione che ne permetta un loro collegamento.

In modo più astratto: la SOA può essere considerata come un'architettura il cui scopo è il raggiungimento di un *disaccoppiamento* tra agenti software interagenti tra loro. Un *servizio* è inteso, quindi, come una unità di lavoro eseguita da un particolare *fornitore di servizi* in grado di soddisfare una particolare esigenza per un *consumatore* di quel servizio che ne fa esplicita richiesta. Entrambi i ruoli di *fornitore* e *consumatore* del servizio sono ricoperti da agenti software sotto richiesta dei loro implementatori [1]. Per completare il nostro punto di vista, la SOA può essere considerata sia come una architettura avente le caratteristiche principali sopra riportate, sia un *modello di programmazione*, ovvero una filosofia per la costruzione di software; infatti, la SOA, permette ora di progettare sistemi software con l'idea (di più alto livello) che essi forniscano dei servizi rivolti ad altre applicazioni, esponendo una interfaccia ben definita che può essere pubblicata e ricercata ed i servizi offerti possano essere invocati attraverso una rete.

I "pilastri" su cui si basa la SOA sono costituiti da una serie di standard e tecnologie che vengono illustrate nelle sezioni seguenti; parte fondamentale per una reale implementazione della SOA, la ricopre la tecnologia dei *Web Service:* implementando la SOA utilizzando questa tecnologia, significa implementare applicazioni software con un modello di programmazione potente e più flessibile, potendo prescindere dalle piattaforme e dai linguaggi utilizzati.

Il futuro della SOA, come ogni altro modello o tecnologia, è rappresentato dalla sua reale applicazione e già due concetti "emergenti" stanno cominciando a diventare significativi: *GRID computing* e *on-demand computing* [23]. Rappresentando ogni applicazione, risorsa o funzionalità business come un servizio dotato di una interfaccia standardizzata, si raggiunge l'opportunità di combinare applicazioni esistenti e nuove per la costruzione di soluzioni di successo nei più svariati campi d'applicazione.

2.1. Tecnologia dei Web Service

La tecnologia dei Web Service, negli ultimi anni, ha cambiato e sta cambiando Internet [3] introducendo la possibilità di usare la rete per costituire un *Web transazionale*; con l'introduzione dei Web Service, si è permessa l'evoluzione del Web da quello più prettamente *program-to-user* o *business-to-Consumer (B2C)* al Web transazionale con l'aggiunta di applicazioni del tipo *program-to-program,* ovvero le cosiddette applicazioni *Business-to-Business (B2C).*

Questa evoluzione è alimentata dalla tecnologia dei Web Service, la quale si basa su alcuni standard, ora ben definiti, alcuni dei quali esistenti da tempo; essi sono: *HyperText Transfer Protocol, (HTTP)*, *Extensible Markup Language (XML)*, *Simple Object Access Protocol (SOAP)*, *WebServices Description Language (WSDL)*, e *Universal Description, Discovery, and Integration (UDDI).*

Prima di esaminare brevemente cosa queste tecnologie permettono di fare e come sono usate, possiamo affermare che la tecnologia Web Service ha permesso di definire un sistema, indipendente dai linguaggi di programmazione, indipendente dalle piattaforma utilizzata e che permette di ridurre i tempi di

integrazione nelle applicazioni di classe *enterprise*. Questa integrazione può ora avvenire mediante un basso livello di accoppiamento tra i sistemi di business.

Essenzialmente, un Web Service, è un'interfaccia la quale descrive una collezione di operazioni che possono essere invocate da remoto attraverso un sistema di scambio di messaggi basati su un formato XML standardizzato. Un Web Service mette a disposizione, quindi, una serie di operazioni, le quali possono corrispondere ai più svariati domini di applicazione; la descrizione del particolare Web Service e di queste sue "pubbliche" funzionalità sono esplicitate mediante un altro standard, basato su XML, che viene chiamato il *descrittore* del servizio. Il descrittore non solo elenca queste funzionalità, ma fornisce tutti i dettagli necessari per l'interazione con il servizio: il formato dei messaggi, i tipi di dato utilizzati, l'URL al quale è possibile contattare il servizio, il protocollo di trasporto da utilizzare. Il descrittore è espresso mediante il formato *WSDL (WebServices Description Language*).

Per analizzare l'architettura e gli attori coinvolti nella tecnologia Web Service, individuiamo tre ruoli principali: il *Service Provider*, il *Service Client*, il *Service Registry*.

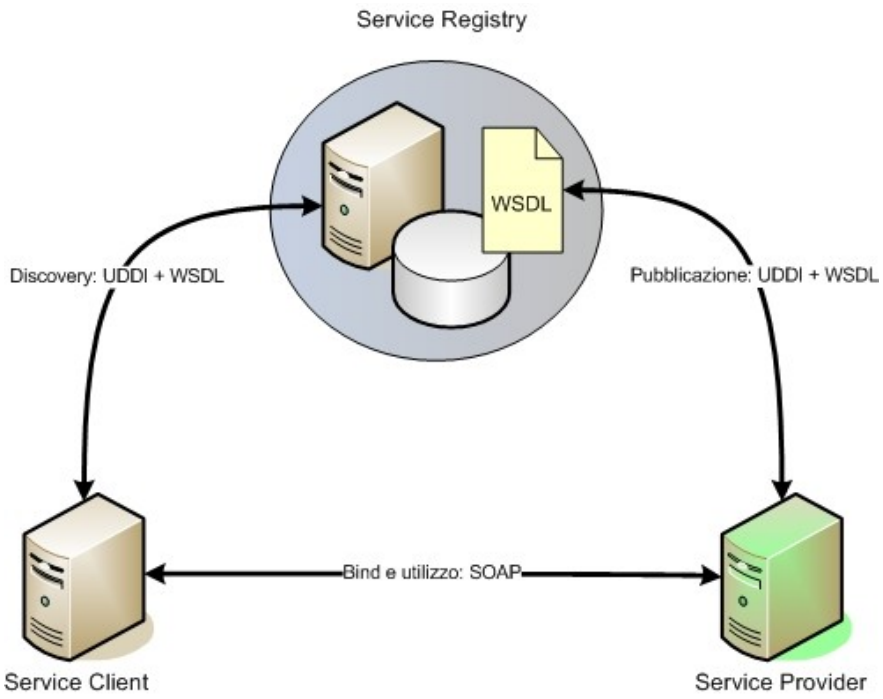

Figura 1: I tre attori principali nell'architettura dei Web Service

Come illustrato nella figura precedente, un *Service Provider* crea un Web Service e il suo file *descrittore* (WSDL) e pubblica la descrizione su un *Service Registry* basato su *UDDI (Universal Description, Discovery and Integration)*. Una volta pubblicato, un potenziale *Service Client* può trovare (*fase di discovery*) il servizio interrogando a sua volta il Service Registry ed utilizzando l'interfaccia UDDI.

Il Service Registry, se in grado, soddisfa la richiesta del Service Client fornendogli il descrittore espresso in formato WSDL e un URL al quale contattare il Web Service vero e proprio. Il Service Client, a questo punto, è in grado di effettuare il *bind* del servizio e iniziare ad invocare le operazioni remote messe a disposizione da quest'ultimo utilizzando le giuste convenzioni (nome metodo e parametri) come descritte nel suo descrittore WSDL. A questo punto le operazioni di richiesta-risposta tra Service Client e Web Service avvengono mediante lo scambio di messaggi espressi secondo le specifiche definite da

SOAP (Simple Object Access Protocol).

2.1.1. Web Services Description Language (WSDL)

WSDL è un formato XML, creato per descrivere dei web service nella rete come un insieme di *endpoint* i quali sono capaci di operare su una comunicazione basata su *messaggi,* contenenti sia informazioni orientate ai documenti, sia informazioni orientate alle procedure[11]. Le operazioni e i messaggi vengono descritti in modo astratto e successivamente vengono associate ad un protocollo di rete ben noto e ad un formato di messaggio, a scelta tra quelli consentiti, allo scopo di definire un endpoint.
In linea generale, un documento WSDL definisce i servizi come collezioni di *endpoint* o *port* e, come detto in precedenza, la definizione astratta di questi ultimi e dei messaggi è separata dalla loro concreta definizione in termini di formato di dati o protocolli di rete; questa separazione permette un riutilizzo delle definizioni astratte: messaggi, che altro non sono che le descrizioni dei dati scambiati, e tipi di porte, le quali sono collezioni astratte di operazioni. Una porta viene definita associando un indirizzo di rete ad una definizione riutilizzabile ed un insieme di porte definiscono un servizio.
Concretamente, un documento WSDL utilizza i seguenti elementi nella definizione dei servizi:

Types: contenitore di definizioni di tipi di dati facente uso di qualche sistema di definizione di tipi (per esempio XML Schema);

Message: definizione astratta e tipizzata dei dati che vengono comunicati;

Operation: descrizione astratta di un'azione messa a disposizione dal servizio;

Port Type: insieme astratto di operazioni messe a disposizione da uno o più endpoint;

Binding: protocollo concreto e definizione di un formato di dati per una particolare port type;

Port: singolo endpoint definito dalla coppia indirizzo di rete – binding;

Service: un insieme di endpoint;

Una importante caratteristica di WSDL è che esso non introduce un nuovo linguaggio di definizione dei tipi di dato; WSDL riconosce la necessità di avere tipi di dato sufficienti per descrivere dati strutturati nel formato dei messaggi, perciò, a tal uopo, supporta le specifiche del XML Schema (XSD) come il proprio sistema standard di descrizione dei tipi. Questo non limita, comunque, l'estensibilità del formato WSDL, il quale rimane aperto verso altri linguaggi di definizione dei tipi di dato.

```
<definitions targetNamespace="http://pintus.it/" name="NewWebServiceService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://pintus.it/" schemaLocation="http://localhost:8084/
WebService-exp/NewWebService?xsd=1"/>
        </xsd:schema>
    </types>
    <message name="somma">
        <part element="tns:somma" name="parameters"/>
    </message>
    <message name="sommaResponse">
        <part element="tns:sommaResponse" name="parameters"/>
    </message>
    <message name="incrementa">
        <part element="tns:incrementa" name="parameters"/>
    </message>
    <message name="incrementaResponse">
        <part element="tns:incrementaResponse" name="parameters"/>
    </message>
    <portType name="NewWebService">
        <operation name="somma">
            <input message="tns:somma"/>
            <output message="tns:sommaResponse"/>
        </operation>
        <operation name="incrementa">
            <input message="tns:incrementa"/>
            <output message="tns:incrementaResponse"/>
        </operation>
    </portType>
    <binding type="tns:NewWebService" name="NewWebServicePortBinding">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="somma">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="incrementa">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="NewWebServiceService">
        <port binding="tns:NewWebServicePortBinding" name="NewWebServicePort">
            <soap:address location="http://localhost:8084/WebService-exp/
NewWebService"/>
        </port>
    </service>
</definitions>
```

Figura 2: Il descrittore WSDL per un semplice Web Service

La seguente figura mostra un documento WSDL che descrive un semplice Web Service il quale espone due *operation,* ovvero i metodi invocabili da remoto, chiamate *somma* e *incrementa:* la prima delle due effettua la somma tra due numeri e restituisce il risultato e la seconda incrementa un valore numero passato come parametro restituendo anch'essa il risultato.

I tipi di dato utilizzati sia nei parametri, sia nei valori restituiti, sono specificati mediante un XML Schema, riportato nella seguente figura:

```
<xs:schema targetNamespace="http://pintus.it/" version="1.0">
        <xs:element type="ns1:somma" name="somma"/>
        <xs:complexType name="somma">
                <xs:sequence>
                        <xs:element type="xs:double" name="a"/>
                        <xs:element type="xs:double" name="b"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element type="ns2:sommaResponse" name="sommaResponse"/>
        <xs:complexType name="sommaResponse">
                <xs:sequence>
                        <xs:element type="xs:double" name="return"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element type="ns3:incrementa" name="incrementa"/>
        <xs:complexType name="incrementa">
                <xs:sequence>
                        <xs:element type="xs:double" name="b"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element type="ns4:incrementaResponse" name="incrementaResponse"/>
        <xs:complexType name="incrementaResponse">
                <xs:sequence>
                        <xs:element type="xs:double" name="return"/>
                </xs:sequence>
        </xs:complexType>
</xs:schema>
```

Figura 3: Il documento XML Schema con la dichiarazione dei tipi di dato utilizzati nel WSDL della figura precedente

2.1.2. Universal Description Discovery and Integration (UDDI)

Il protocollo UDDI, sviluppato dalla *Organization for the Advancement of Structured Information Standards (OASIS)*, costituisce uno dei protocolli chiave facenti parte dell'insieme di quelli definiti per l'implementazione dei fondamenti della tecnologia dei Web Service. Esso definisce una metodologia standard per la pubblicazione e il *discovery* in una *Service-Oriented Architecture (SOA)*[12].
Lo standard UDDI definisce, come illustrato in figura 1, i protocolli per poter accedere ad un *registry* di Web Service, fornisce i metodi per controllare l'accesso a questo registro e un meccanismo per distribuire o delegare i dati, presenti in esso, verso altri registri. In pratica, un registro UDDI fornisce un approccio standardizzato per consentire la localizzazione di un *servizio software*, per invocare le operazioni che tale servizio mette a disposizione e per gestirne i metadati associati. UDDI è giunto oramai alla versione 3.0, introdotta a quattro anni di distanza dalla versione 1.0, rilasciata nel 2000.

La seguente figura illustra uno scenario con gli utilizzi tipici di un registro UDDI.

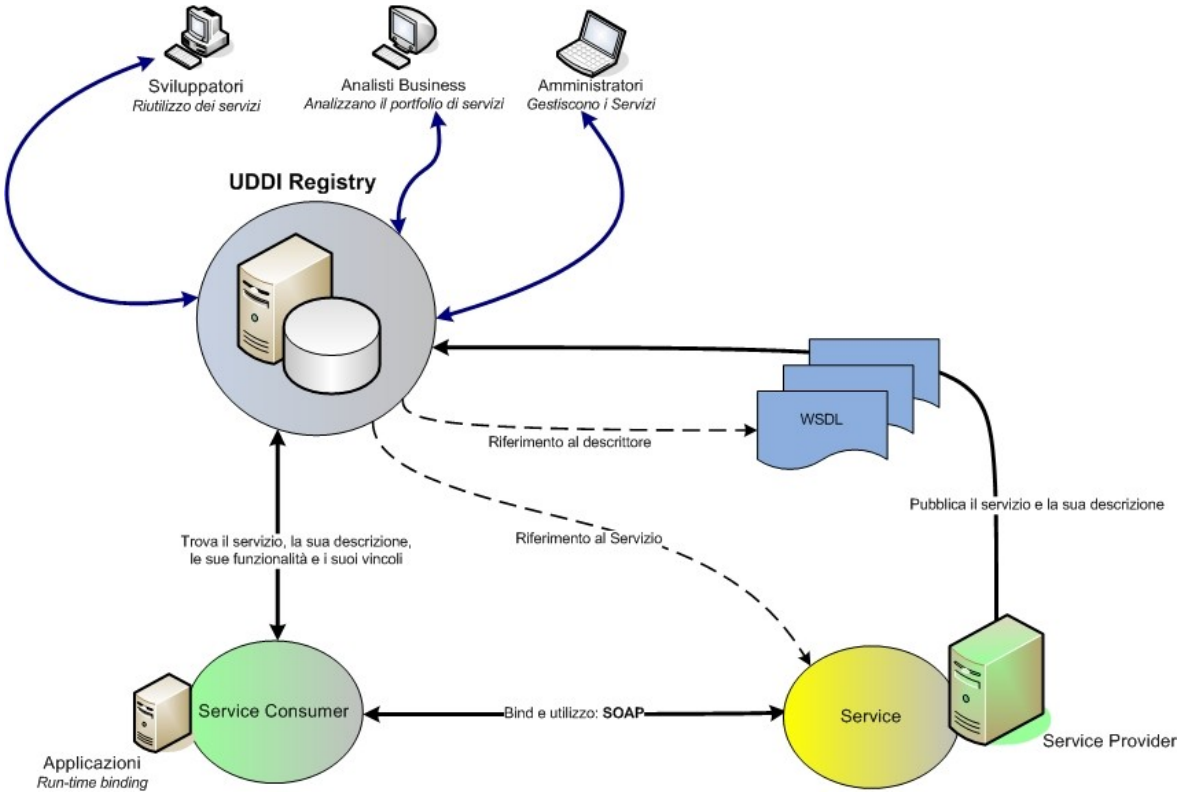

Figura 4: UDDI Registry, attori e suo utilizzo

2.1.3. Simple Object Access Protocol (SOAP)

Definito dal W3C, SOAP è un protocollo, basato su XML, per lo scambio di messaggi tra componenti software ed è divenuto il linguaggio standard con cui vengono codificati i messaggi inviati tra i Web Service e le loro applicazioni client.

Raggiunta la versione 1.2, SOAP viene definito "leggero" dal W3C, ed è stato creato, come detto precedentemente, allo scopo di realizzare lo scambio di informazioni strutturate tra applicazioni in un ambiente distribuito e decentralizzato. SOAP è basato sulle tecnologie XML allo scopo di definire un *framework* estensibile per lo scambio di messaggi fornendo un costrutto che può essere scambiato grazie ad una varietà di sottostanti protocolli di comunicazione; ancora, SOAP è stato progettato per rimanere indipendente da un particolare modello di programmazione o da altre specifiche semantiche d'implementazione [13]. Per alcuni suoi aspetti, possiamo affermare che SOAP si pone in qualche modo come un'evoluzione del più semplice protocollo di *Remote Procedure Call* noto come *XML-RPC*.

Un messaggio SOAP è un normale documento XML che contiene i seguenti elementi:

- un *Envelope,* elemento obbligatorio che identifica il documento XML come un messaggio SOAP;
- un *Header,* elemento opzionale che contiene informazioni di tipo *header*;
- un *Body*, elemento richiesto il quale contiene le informazioni sulla chiamata e la risposta;
- un *Fault*, elemento opzionale che fornisce informazioni sugli eventuali errori che potrebbero verificarsi in fase di processo dei messaggi;

La seguente figura illustra un generico messaggio SOAP.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
  ...
  ...
</soap:Header>

        <soap:Body>
                ...
                ...
                <soap:Fault>
                ...
                ...
                </soap:Fault>

        </soap:Body>

</soap:Envelope>
```

Figura 5: La struttura di un generico messaggio SOAP

Le regole principali che un messaggio SOAP deve seguire sono:

- deve essere codificato in XML valido;
- deve far uso del SOAP Envelope Namespace;
- deve far uso del SOAP Encoding Namespace;
- non deve contenere riferimenti ad un DTD come pure non deve contenere istruzioni di XML processing;

Vediamo ora un semplice esempio di comunicazione SOAP tra un Web Service e un suo consumer. Il Web Service possiede un operazione denominata *sum* che effettua la somma di due numeri interi passati come parametri e restituisce il

risultato.

Assumendo che sia ben noto l'URL del servizio (quest'ultimo potrebbe anche essere stato, ad esempio, ricavato mediante interrogazione di un UDDI Registry), l'applicazione client, il consumer, che vuole effettuare la somma degli interi *2* e *3* può invocare l'operazione remota inviando un messaggio SOAP, la figura seguente mostra tale messaggio inviato dal client al Web Service:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
        <soapenv:Body>
                <ns1:sum soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns2="http://www.w3.org/2001/XMLSchema" xmlns:ns1="http://DefaultNamespace">

                        <i1 xsi:type="ns2:int">2</i1>
                        <i2 xsi:type="ns2:int">3</i2>

                </ns1:sum>
        </soapenv:Body>
</soapenv:Envelope>
```

Figura 6: Messaggio SOAP inviato dall'applicazione client al Web Service per ottenere la somma degli interi 2 e 3

Come si può notare in Figura 6, il Body del messaggio SOAP contiene il nome dell'operazione remota invocata e i parametri, di tipo intero, dei quali si richiede la somma.

Il Web Service è capace di processare la richiesta e risponde al client con un messaggio SOAP, che contiene la risposta all'invocazione dell'operazione di somma, come illustrato nella figura seguente:

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://
www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <soapenv:Body>

                <ns1:sumResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/" xmlns:ns1="http://DefaultNamespace">
                        <sumReturn xsi:type="xsd:int">5</sumReturn>
                </ns1:sumResponse>

        </soapenv:Body>
</soapenv:Envelope>
```

Figura 7: Messaggio SOAP, inviato dal Web Service al client, per comunicare la somma dei due interi specificati nella richiesta, ossia il risultato dell'operazione remota

Nel Body possiamo individuare la risposta del Web Service che restituisce un intero di valore 5, ovvero la somma dei due numeri interi passati nella richiesta del client.

Naturalmente quanto presentato è solamente un piccolo esempio di messaggistica SOAP per la comunicazione tra due entità software distribuite; le applicazioni più comuni non si limitano ad usare i tipi di dato base ma anche tipi strutturati e complessi, segue che i messaggi SOAP avranno a che fare con tipi definiti in qualche modo attraverso grammatiche XML.

Un aspetto rilevante da considerare nella definizione dei Web Service e che ricade nella tipologia dei contenuti dei messaggi SOAP è rappresentato dal formato dei dati che viaggiano nel SOAP Envelope. Quest'ultimo e sostanzialmente è composto da due parametri:

- *Binding Style:* che può essere di tipo *document* o di tipo *rpc*;
- *Encoding Style:* che può essere di tipo *encoded* o di tipo *literal*;

Il *Binding Style* rappresenta il modo nel quale vengono descritte le operazioni e i

loro parametri durante l'invocazione di un Web Service: lo stile *rpc* viene usato per descrivere i Web Service come se fossero delle "tradizionali" procedure remote ed ogni parametro contenuto in una invocazione mediante messaggio SOAP viene "incapsulato" da dei *tag* che lo descrivono; lo stile *document* è sostanzialmente utilizzato per descrivere i Web Service come entità software orientate al consumo di documenti ed infatti ogni parametro in questo caso è rappresentato dal documento che lo descrive senza nessuna modifica.

L'E*ncoding style* specifica il modo in cui vengono scritti i dati facenti parte delle richieste e delle risposte dei Web Service; lo stile *encoded* prescrive che i dati debbano essere descritti come tipi definiti nello standard XML-Schema (a parte alcune eccezioni definite dalle specifiche di SOAP) e viene utilizzato nel caso in cui si abbia la necessità di serializzare-deserializzare grafi composti di oggetti da una parte e dell'altra delle due entità comunicanti, per i tipi complessi, invece si ricorre all'utilizzo dell'XML-Schema; lo stile *literal*, prescrive che i dati debbano essere descritti mediante un documento XML-Schema e non fa nessuna supposizione sul modo in cui i dati vengono serializzati-deserializzati; l'unico vincolo che si richiede è che il documento sia sempre descritto mediante XML-Schema.

Riassumendo si hanno, quindi, quattro possibili combinazioni, come illustrato nella figura seguente:

rpc/encoded	rpc/literal
document/encoded	document/literal

Tabella 1: Le possibili combinazioni di Binding Style, Encoding Style in un messaggio SOAP

Questa varietà di combinazioni ha introdotto diversi problemi di interoperabilità tra diverse implementazioni di Web Service SOAP, soprattuto in relazione ai

diversi linguaggi di programmazione e piattaforme utilizzati. Per questo motivo si è costituito il consorzio *WS-I (Web Service Interoperability)*[27] il cui scopo è proprio quello della creazione di specifiche per permettere una reale interoperabilità tra le diverse implementazioni dei Web Service. Per questo motivo l'organizzazione WS-I ha elaborato un set di specifiche chiamato *WS-I Basic Profile* che devono essere supportate dagli implementatori dei servizi affinche si abbia un minimo livello di interoperabilità per i Web Service, tra le altre cose, il WS-I Basic Profile prevede, per esempio, che venga utilizzato il formato *document/literal* come encoding.

2.2. Web Services Orchestration

Per Web Service Orchestration (WSO) si intende il tentativo di una integrazione e composizione di singoli Web Service in processi di business standardizzati. Ultimamente questo modello tecnologico sta diventando un componente chiave per la realizzazione di una reale e standardizzata Service Oriented Architecture, nonché un componente fondamentale nello stack di layer basati sulla tecnologia Web Service; inoltre, la sua natura di meccanismo di disaccoppiamento dei servizi costituisce un promettente avvio verso le architetture distribuite basate su servizi eterogenei ma uniti sotto una ben definita interfaccia di comunicazione e discovery. Una formulazione precisa di standard per l'esecuzione di processi business basati su Web Service porterà presto al raggiungimento di un duplice scopo: riduzione dei costi e sostanziale miglioramento dell'interoperabilità tra servizi, senza contare i benefit di un loro reale riutilizzo.

2.2.1. BPEL, Business Process Execution Language (for Web Services)

Con l'introduzione e utilizzo delle tecnologie basate su Web Service, si è assistito ad un reale cambiamento-ampliamento di Internet e del suo utilizzo quale strumento per applicazioni B2B. Attuale capitolo di questa evoluzione è quello di una formalizzazione di alcuni standard e sistemi di gestione di flussi di lavoro ed esecuzione per sistemi distribuiti basati su Web Service. Le classiche applicazioni software per il supporto di flussi di lavoro con queste caratteristiche sono i sistemi di gestione di workflow (WfMS, WorkFlow Management Systems), che consentono la gestione contemporanea di un numero, anche elevato, di istanze di processi seguendo schemi di processo predefiniti[4][5].
In questo ambito, il processo viene definito come una *rete di attività* e di relazioni esistenti tra di esse, criteri per indicare l'inizio e la fine di un processo, e informazioni riguardo alle singole attività, quali: i partecipanti, le applicazioni, i dati utilizzati e scambiati, e così via. La rappresentazione di un processo in una forma che consente l'esecuzione automatica delle attività automatizzatili e la gestione automatica del passaggio tra un'attività e quelle che seguono nel flusso di lavoro è basata su alcuni costrutti fondamentali quali:

sequenza: le attività vengono svolte l'una di seguito all'altra: una attività viene attivata solo al termine di un'attività precedente;

parallelo (AND split): dopo la terminazione di una attività vengono attivate più attività in parallelo;

alternativa (OR split): dopo la terminazione di una attività vengono attivate più attività in alternativa; vengono specificate le condizioni di attivazione delle

attività oppure può essere effettuata una scelta non deterministica;
join (AND oppure OR): per proseguire nel flusso di lavoro devono essere terminate tutte le attività precedenti (caso AND), o almeno una delle attività precedenti.

Sulla base di questi costrutti è possibile rappresentare reti di attività di tipo generale.
Nelle precedenti sezioni si è visto come lo scopo della tecnologia dei Web Services sia quello di raggiungere una interoperabilità universale tra applicazioni utilizzando il Web standard [6]. Questa tecnologia utilizza un modello di disaccoppiamento tra le parti, in modo da permettere una flessibile integrazione tra sistemi eterogenei in una molteplice varietà di domini. Tutto questo indipendentemente dalla piattaforma e dai linguaggi di programmazione utilizzati. La completa potenzialità della tecnologia dei Web Service come piattaforma di integrazione sarà raggiunta solamente quando le applicazioni e i processi di business saranno capaci di integrare le loro complesse interazioni facendo uso di un modello standardizzato di integrazione. Il modello di interazione, che è automaticamente supportato dal formato WSDL, è essenzialmente un modello *stateless sincrono* o *asincrono* mentre i modelli per le interazioni *business* assumono tipicamente la forma di sequenze di scambi di messaggi di tipo *peer-to-peer*, *(P2P)* sia sincrone che asincrone, con una gestione *stateful* di interazioni a lungo termine tra due o più parti.
Per definire questo tipo di interazioni business è necessaria una descrizione formale del protocollo di scambio dei messaggi in questo contesto aggiuntivo.
La definizione di questa tipologia di protocolli implica una precisa specifica del mutuo scambio di messaggi tra le parti coinvolte nel protocollo, senza una esposizione pubblica della loro implementazione interna. Ci sono essenzialmente

due buone ragioni per la separazione degli aspetti pubblici di un processo business dagli aspetti privati o interni: la prima, ovviamente, è data dal fatto che gli attori coinvolti nel business non vogliono rivelare le loro strategie o i loro *data management* ai corrispettivi partner d'affari; la seconda è che questa separazione introduce la libertà di cambiare gli aspetti privati dell'implementazione dei processi senza avere nessun effetto sul protocollo pubblico di business.

I protocolli di business devono essere necessariamente e chiaramente descritti in modo indipendente dalla piattaforma e dai linguaggi utilizzati.
I concetti chiave per descrivere un protocollo di business sono:

- presenza di costrutti condizionali e di time-out, poichè il protocollo include comportamenti dipendenti dai flussi di dati;

- capacità di reazione ad eventi *eccezionali*, come sequenze di *recovery*;

- ricca capacità descrittiva della notazione includendo "reminescenze" dei linguaggi di programmazione tradizionali.

BPEL (il nome originario è *BPEL4WS*), è un linguaggio nato dall'unione delle esperienze su *XLANG* e *WSFL (Web Service Flow Language)* e si è guadagnato il ruolo di principale "tecnologia" per la *Web Service Orchestration* e tiene conto, tra i suoi obiettivi, delle caratteristiche e delle problematiche sopra esposte. BPEL permette di specificare un modello di comportamento di Web Service durante un processo di business ed esso assume che questi servizi siano descritti mediante WSDL ed eventualmente memorizzati in un UDDI Registry.
BPEL è basato su una grammatica XML interpretabile da un *orchestration engine* il

cui compito è quello di coordinare i vari Web Service che compongono il processo. BPEL permette di considerare un Web Service come un processo di collaborazione composto da altri Web Service, definendone una natura di tipo ricorsivo e questo "processo di processi" così definito avrà una sua propria interfaccia WSDL.
Un processo definito mediante BPEL è composto da attività che rappresentano sia richieste di tipo client, sia risposte al client medesimo; come specificato in precedenza, il processo si basa su una serie di di Web Service che vengono chiamati *partner* invocati attraverso l'attività di *invoke* che può essere di tipo sincrono o asincrono.

Il seguente listato XML e la figura illustrano rispettivamente un documento espresso in BPEL e la relativa rappresentazione grafica del processo.

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="SynchronousSample"
targetNamespace="http://www.mycomp.org/SynchronousSample"
xmlns="http://schemas.xmlsoap.org/ws/2004/03/business-process/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2004/03/business-process/"
xmlns:wsdlNS="http://www.mycomp.org/SynchronousSample"
xmlns:xs="http://www.mycomp.org/SynchronousSampleSchemaNamespace">
      <import namespace="http://www.mycomp.org/SynchronousSample"
location="SynchronousSample.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
      <partnerLinks>
            <partnerLink name="partnerLinkA"
partnerLinkType="wsdlNS:SynchronousSamplePartnerLinkType"
myRole="SynchronousSampleProvider"/>
      </partnerLinks>
      <variables>
            <variable name="inputVar" messageType="wsdlNS:requestMessage"/>
            <variable name="outputVar" messageType="wsdlNS:responseMessage"/>
      </variables>
      <sequence>
            <receive name="start" partnerLink="partnerLinkA"
portType="wsdlNS:MyPortType" operation="operationA" variable="inputVar"
createInstance="yes"/>
            <assign name="assign">
                  <copy>
                        <from variable="inputVar" part="inputType"/>
                        <to variable="outputVar" part="resultType"/>
                  </copy>
            </assign>
            <reply name="end" partnerLink="partnerLinkA"
portType="wsdlNS:MyPortType" operation="operationA" variable="outputVar"/>
      </sequence>
</process>
```

Figura 8: Esempio di documento di descrizione di un workflow espresso in BPEL

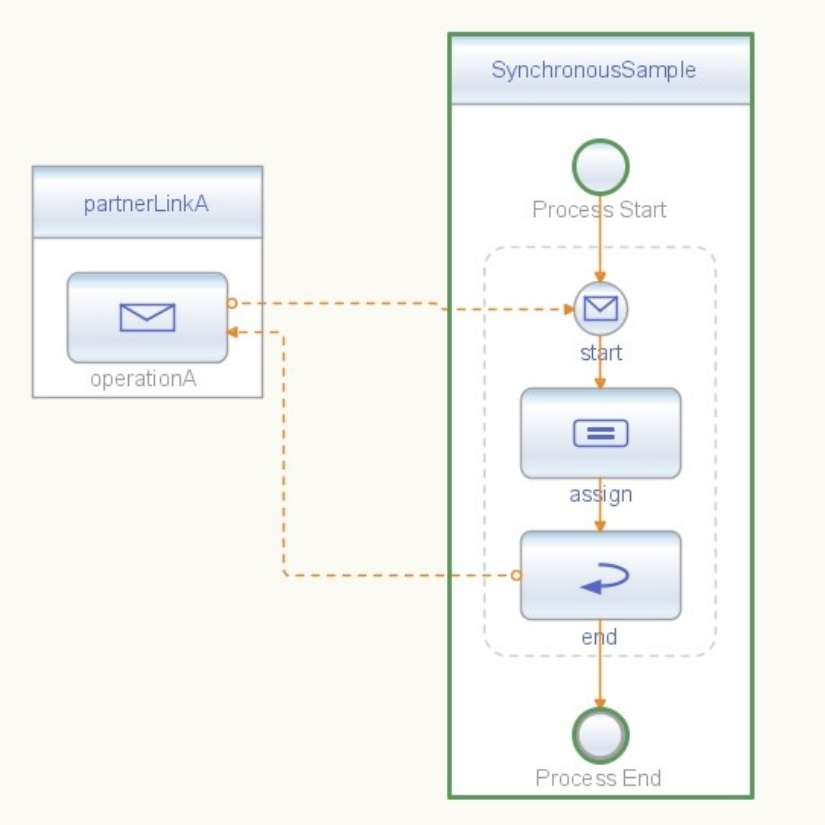

Figura 9: Rappresentazione grafica di un processo BPEL (creato con Netbeans, un popolare ambiente di sviluppo per Java)

Per la definizione vera e propria di questi processi si utilizzano dei costrutti che provengono da quelli definiti per modelli di workflow:

- *sequence,* per le attività in sequenza;
- *flow,* per l'esecuzione di attività in parallelo;
- *switch* e *pick* per le alternative tra attività;

- *while,* come costrutto iterativo.

BPEL permette anche di utilizzare delle variabili allo scopo di permettere un mantenimento dello stato tra chiamate diverse.

2.3. Ambienti e tool di sviluppo per i Web Service in Java

Tutti i moderni linguaggi di programmazione prevedono, oramai, un set più o meno ricco, di librerie per l'implementazione dei Web Service e dei relativi client. In questa sezione verranno descritte alcune tecnologie utilizzate per l'implementazione di Web Service, basati sul linguaggio Java. Per questo linguaggio i framework disponibili sono svariati, in questo lavoro verranno analizzati il framework *Apache Axis* per l'implementazione dei Web Service e il *web container Apache Tomcat* per la loro pubblicazione.

Il primo è stato scelto per la sua semplicità di installazione, per la versatilità dei tool messi a disposizione dello sviluppatore e per la velocità di creazione di un Web Service a partire da un semplice *JavaBean*. Inoltre la sua buona diffusione ed utilizzo costituisce una eccellente alternativa al nuovo framework, recentemente incluso da Sun Microsystems nelle ultime distribuzioni del linguaggio Java, chiamato *JAX-WS*.

Il secondo, Apache Tomcat, è probabilmente il più popolare Web Container per le *Web Application* scritte in Java, nonché piattaforma di riferimento di alcune connesse tecnologie.

2.3.1. Apache Tomcat

Come descritto nella relativa home page del progetto, Tomcat è un *Servlet* container usato per l'implementazione di riferimento delle tecnologie *Java Servlet* e *JavaServer Pages* di Sun. Tomcat è completamente open-source e viene rilasciato sotto la *Apache Software License* [18].
Considerando Tomcat come un sistema di tipo *blackbox*, possiamo riassumere le funzionalità di base nella figura che segue.

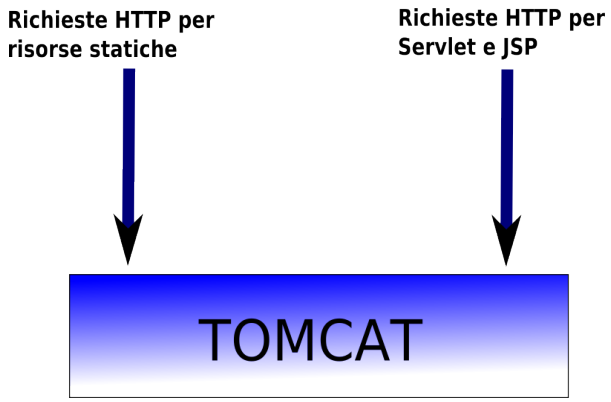

Figura 10: Apache Tomcat visto come una "blackbox"

Tomcat è in grado di agire sia da HTTP server "classico" per contenuti statici, come pagine HTML, immagini, risorse, sia da Servlet container, in quanto mette a disposizione il *runtime* per eseguire, mantenere e rispondere a richieste indirizzate a web application dinamiche realizzate attraverso Java Servlet e JavaServer Pages. Una delle potenzialità più sfruttate di Tomcat è la possibilità di accoppiarlo ad un HTTP Server più performante, per le richieste a risorse statiche, come Apache HTTP Server.

L'architettura interna di Tomcat può essere riassunta nella seguente figura:

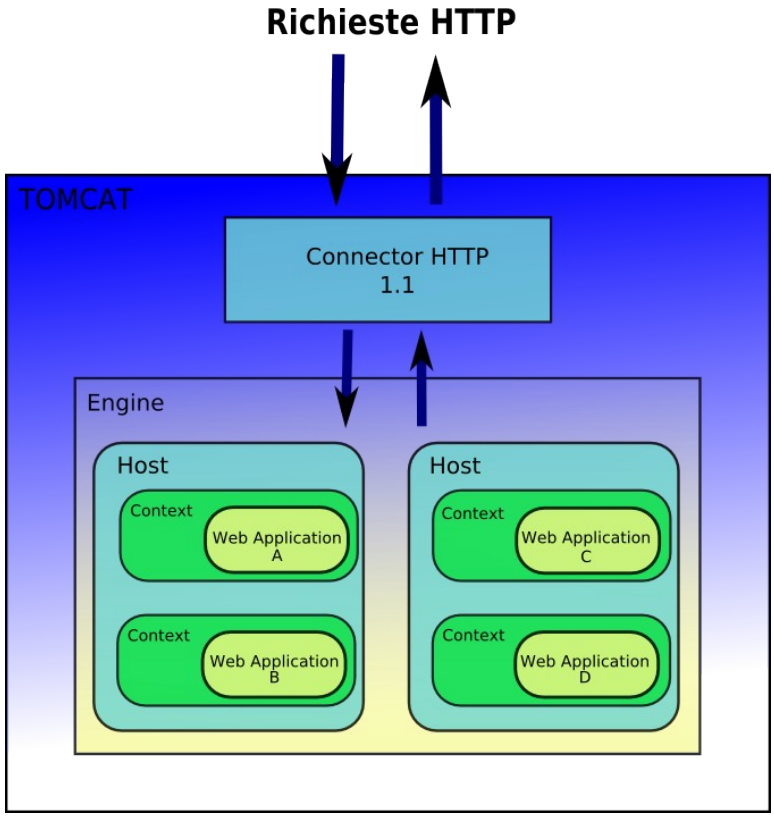

Figura 11: L'architettura interna di Apache Tomcat

Il *Connector* è il modulo di Tomcat che si occupa del suo interfacciamento con l'esterno e gestisce, perciò, le comunicazioni con il client. Esistono vari Connector disponibili per Tomcat: per esempio il *Coyote Connector* , utilizzato per il traffico HTTP e il *JK2 Connector* il quale implementa il protocollo AJP, utilizzato per eseguire Tomcat in coppia con *Apache HTTP Server*.

Un *Engine* rappresenta la *pipeline* della gestione delle richieste per un servizio

specifico; poiché ogni servizio potrebbe avere Connector multipli, l'Engine riceve e processa tutte le richieste provenienti dai Connector associati, fornendo la risposta indietro all'appropriato Connector, il quale provvederà alla trasmissione al client.

Un *Host* è un'associazione di un nome di rete, per esempio: www.ilmiohost.org, a Tomcat. Un singolo Engine può contenere più di un Host, a un singolo componente Host possono essere associati diversi alias di rete.

Un *Context* rappresenta una *Web Application*. Un Host, naturalmente, può contenere più di un Context, ciascuno di essi raggiungibile con un *path* univoco.

Le *Web Application* sono le applicazioni vere e proprie scritte in Java con la tecnologia Servlet, JSP e derivate.

Il container Tomcat viene utilizzato anche per il deployment di Web Service scritti in Java poiché, questi ultimi, solitamente vengono tecnicamente realizzati mediante la creazione (più o meno automatica grazie a numerosi tool disponibili) di Servlet che implementano di fatto il servizio e permettono la sua pubblicazione su di un server Java enabled, come Tomcat appunto.

2.3.2. Apache Axis

Axis è un *SOAP Engine*, ovvero un framework open-source per la costruzione di processori SOAP e Web Service, sia parte server che client consumer.
Il framework è disponibile sia per il linguaggio Java che per C++ [17].
Per quanto riguarda la versione per il linguaggio della Sun, Axis mette a disposizione:

- un semplice server stand-alone;
- l'implementazione di SOAP versione 1.1 e 1.2;
- il supporto per la serializzazione/deserializzazione;
- un server installabile come web application su un Servlet Container;
- supporto per il *Web Service Description Language (WSDL)*;
- i tool di generazione automatica del codice Java a partire da un file WSDL e viceversa;
- un TCP/IP monitor e un SOAP Monitor;
- possibilità di tre modalità di invocazione dei Web Service: mediante *Dynamic Invocation Interface*, mediante *stub* generato automaticamente a partire dal WSDL e *Dynamic Proxy*.

Una caratteristica, spesso molto utile, per creare dei semplici web service con Axis è rappresentata dall'opportunità di crearne uno a partire da una classe Java (JavaBean), semplicemente rinominando il file da *.java* a *.jws; c*opiando il file rinominato all'interno dell'installazione della web application server di Axis, tutti i necessari *SOAP handler* e classi di utilità verranno automaticamente generati dal framework e il Web Wervice così generato sarà automaticamente raggiungibile mediante URL ben definito. L'utilità di questa particolare

caratteristica si riduce, comunque, alla creazione di semplici Web Service, in quanto è possibile solamente l'utilizzo dei soli tipi primitivi nell'invocazione delle operazioni e sono presenti anche altre limitazioni che ne rendono, comunque, limitato l'utilizzo in campo *production*. In ogni caso, questa potenziale opzione, costituisce un valido metodo per fare del *rapid prototyping* di applicazioni orientate ai servizi.

Il framework Axis consente anche la creazione delle classi client a partire dal WSDL che descrive il servizio.

3. Sistemi di gestione di Workflow

Le organizzazioni attuali, sia che esse operino nel campo più prettamente business, sia che operino nel campo della ricerca, un esempio su tutti è rappresentato dalla *bioinformatica*, tipicamente necessitano della gestione di una grande mole di flussi di dati e di risorse computazionali, sempre più decentralizzate e distribuite, a partire da risorse distribuite nello stesso dipartimento/edificio sino ad arrivare a quelle distribuite su Internet e quindi a livello planetario. Ora più che mai, trovandoci in piena epoca di realizzazione di architetture orientate ai servizi (SOA), trovano applicazione e vita nuova i sistemi di workflow.

I *Workflow Management System* vengono definiti come "*sistemi di ausilio per le organizzazioni per specificare, eseguire e coordinare il flusso di processi di lavoro all'interno di un ambiente lavorativo distribuito*"[8].

Ancora, in generale, un sistema di workflow riguarda l'automazione delle procedure nelle quali i documenti, le informazioni o le singole unità di lavoro vengono passate tra i partecipanti in accordo ad un ben definito insieme di regole per il raggiungimento di un determinato obiettivo di business [9].

In questo modo possiamo dare una definizione più coincisa di un workflow:

"Un workflow è un meccanismo computerizzato per l'automazione di un processo di business, nella sua interezza o in parte"

Da queste definizioni segue che un *Workflow Management System* è un sistema che fornisce delle automazioni in forma procedurale di un processo di business, mediante la gestione di una sequenza di attività di lavoro e l'invocazione di appropriate risorse computazionali associate ai vari passi delle attività.

Quindi, più formalmente:

> *"Un Workflow Management System è un sistema che definisce, gestisce ed esegue completamente dei workflow attraverso l'esecuzione di componenti software il cui ordine di esecuzione è guidato da una rappresentazione computerizzata della logica di workflow"*

Le definizioni precedenti sono state elaborate dalla *Workflow Management Coalition (WfMC)*, organizzazione di cui fanno parte aziende come *Oracle, BEA, IBM, Sun Microsystems* e molte altre, il cui scopo è quella di gestire i lavori e collaborare all'evoluzione dei sistemi di workflow.

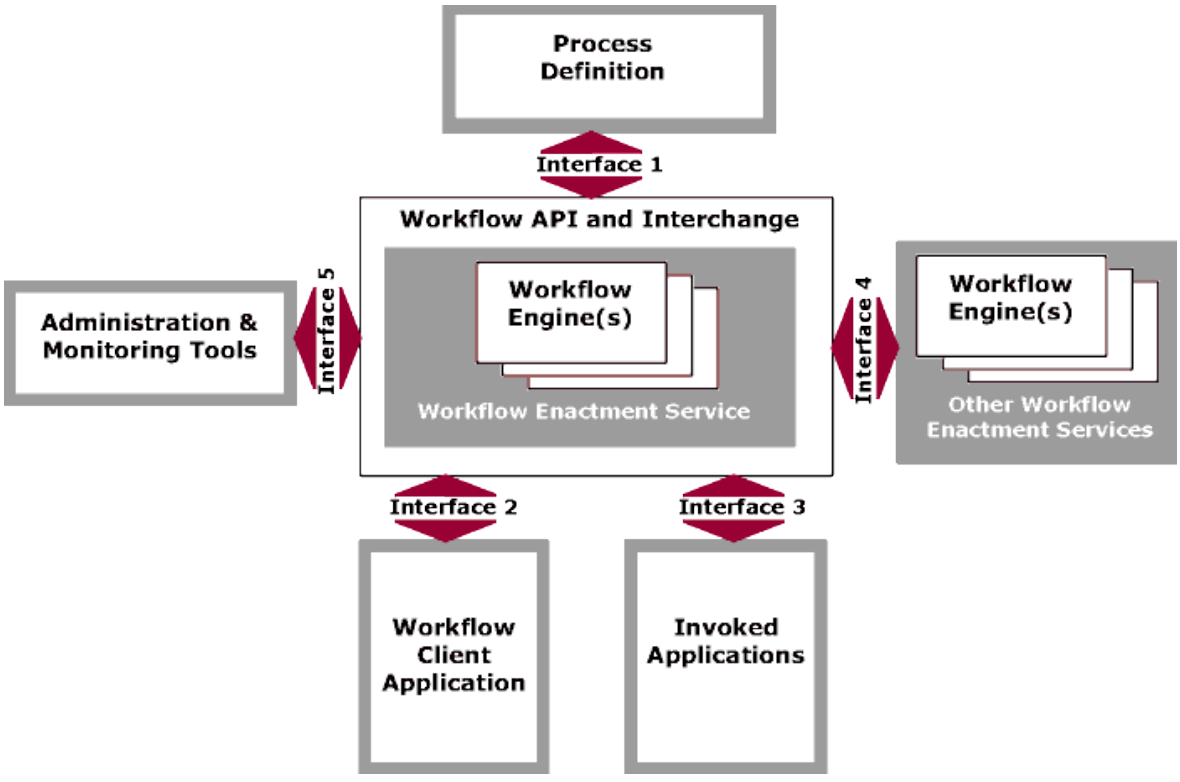

Figura 12: Il Workflow Reference Model secondo la Workflow Management Coalition (WfMC)

L'evoluzione dei sistemi di workflow ha visto il loro impiego in differenti aree di applicazione, ricordiamo tra le tante: elaborazione di immagini, *document management*, posta elettronica e sistemi di directory, applicazioni di *groupware*, gestione dei cicli di sviluppo del software.

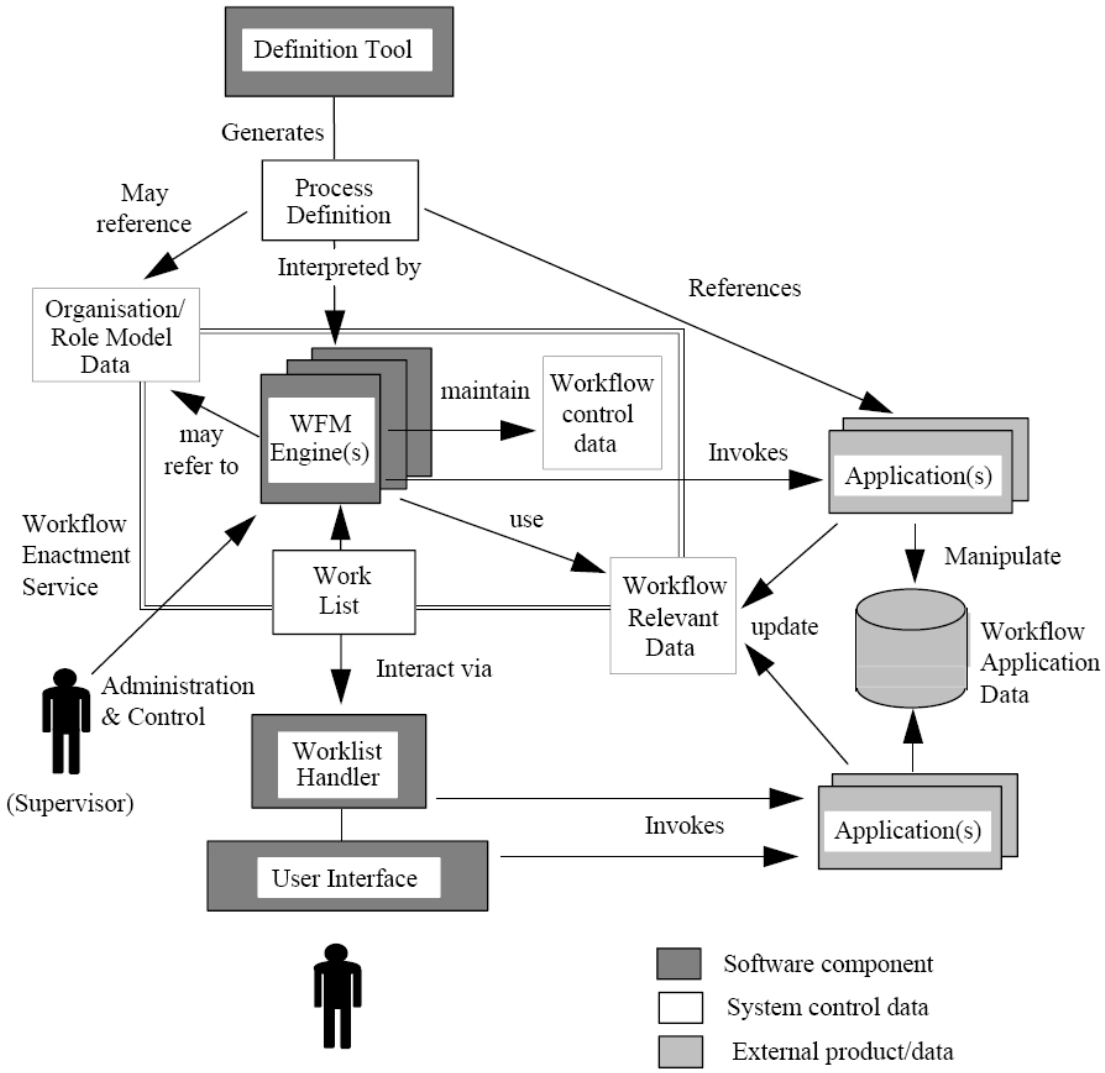

Figura 13: Architettura generica della definizione di un processo di workflow secondo la WfMC

Nella figura precedente viene illustrata la generica architettura per la definizione di un processo di workflow. Mediante un tool di composizione (*Definition Tool*) è possibile creare una definizione di processo il quale si serve di applicazioni "esterne" per il suo completamento. La definizione del processo viene interpretata da un *enactor*, chiamato in figura *Workflow Management Engine (WFM Engine)* il quale esegue il processo così come definito nella prima fase. In ogni fase delle elaborazioni, l'utente (appartenente a diverse tipologie: supervisore, utente del workflow, ecc...) ha una vista sul progresso delle operazioni.

La figura precedente potrebbe essere facilmente estesa per rappresentare un sistema di workflow distribuito, nel senso che, se dislocassimo diversi *Workflow Management Engine* su diverse macchine (remote) e facessimo in modo che un workflow definito su un host facesse uso di un altro definito su secondo host, potremmo avere un sistema di workflow di tipologia distribuita.

Da quanto introdotto sinora, si evince che, per la definizione di un sistema di workflow, sono fondamentali *il modello del workflow e la sua rappresentazione mediante qualche formalismo (process definition)* e la presenza di un *Workflow Engine*.

Il secondo di questi, si occupa di fornire essenzialmente un ambiente di *run-time* per una istanza di workflow definita mediante un formalismo (un linguaggio dotato di una ben definita grammatica).

Un *Workflow Engine* dovrebbe possedere le seguenti funzionalità:

- interpretazione della definizione del processo ;
- controllo delle istanze del processo: creazione, attivazione, sospensione, terminazione, ecc...;
- definizione di una interfaccia per l'invocazione di servizi ed applicazioni esterne;

- *facility* di supervisione per il controllo, l'amministrazione e l'auditing dei processi.

La definizione del processo implica la definizione di un qualche formalismo o *meta-modello* per la descrizione dell'intero workflow; nella stesura del suo modello di riferimento, la WfMC definì un meta-modello di base per la definizione dei flussi, includendo anche l'introduzione delle entità partecipanti al processo.
Il meta-modello di base definito è illustrato nella Figura 14

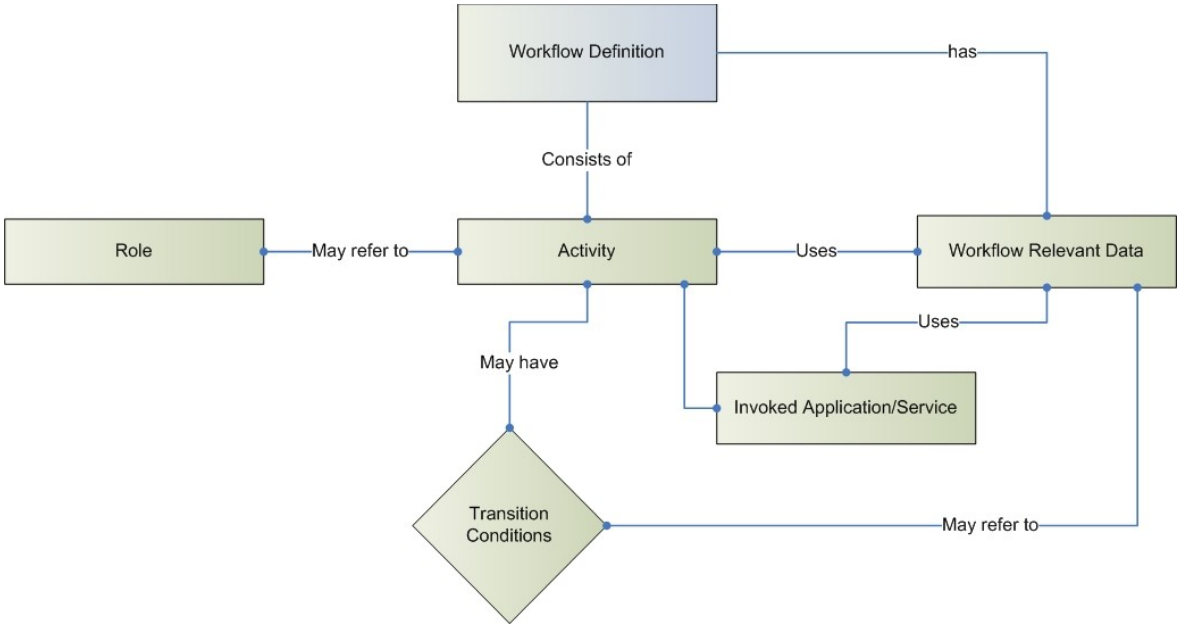

Figura 14: Il meta-modello di base per la definizione di un workflow così come definito dalla WfMC

La figura precedente introduce un meta-modello costituito dalle seguenti entità e le relazioni tra di essi per la definizione di un workflow:

Workflow Definition: per la definizione del processo, il meta-modello deve possedere i seguenti attributi:

- nome del processo di workflow;
- numero della versione;
- condizioni di start e di terminazione del processo;
- sicurezza ed altri dati di controllo;

Activity: ciascuna attività di cui è composto un workflow deve possedere i seguenti attributi:

- nome dell'attività
- tipo (subflow, atomic flow, ecc..)
- condizioni di pre e post esecuzione dell'attività
- altri vincoli di scheduling

Transition Conditions: questa entità rappresenta ed esprime le condizioni di esecuzione e transizione del workflow (costrutti condizionali e di iterazione).

Workflow relevant data: i dati rilevanti per il flusso all'interno della definizione:

- nome dei dati e percorso
- tipi di dato

Role: il nome e l'entità dell'organizzazione

Invoked Application/Service: applicazioni esterne al workflow richiamate per l'elaborazione del processo di workflow, esse sono definite da:

- tipo generico o nome
- parametri d'esecuzione
- localizzazione del servizio o percorso

Particolare importanza riveste la definizione di una interfaccia comune per l'invocazione di applicazioni remote. Per questo motivo un meta-modello per un sistema di workflow deve definire quali sono gli standard d'interfaccia per perseguire questa funzionalità. Nei moderni sistemi di workflow, tra i quali quelli che verranno esaminati più avanti in questo lavoro, l'interfaccia principale per le applicazioni remote è rappresentata dal *Web Service Description Language (WSDL).*

Un'altra caratteristica chiave che un sistema di workflow dovrebbe possedere è quella della *interoperabilità* tra diversi sistemi e diverse piattaforme (intesi come sistemi operativi, linguaggi di programmazione, ecc...). Ancora una volta, gli intenti originari nella definizione formale dei sistemi di workflow sono stati raggiunti facendo uso delle tecnologie basate su Web Service.
Come abbiamo visto, la definizione di un processo di workflow, implica quindi la definizione di una serie di "passi" e coordinamento di attività per la persecuzione di un determinato obiettivo di *business*.
Introduciamo ora brevemente la terminologia utilizzata durante la definizione del processo e la sua esecuzione per descrivere la natura del suo flusso e delle sue interazioni [10].

3.1. Process

Il processo è una rappresentazione formalizzata di un processo di business, rappresentata come un insieme coordinato (parallelo e/o seriale) di attività le quali sono interconnesse per perseguire uno scopo ben preciso. Le attività possono essere considerate come le singole "unità" di lavoro (funzioni) oppure come dei blocchi di attività stesse.

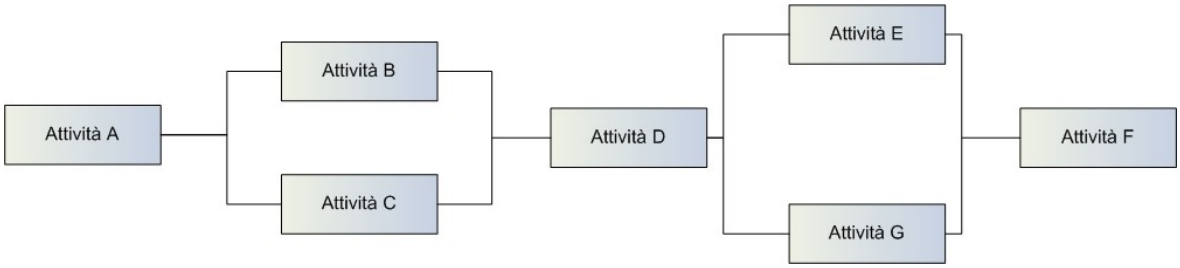

Figura 15: Un processo come sequenza di attività

3.2. Parallel Routing

Segmento di un processo in fase di esecuzione durante il quale due o più attività sono in esecuzione parallelamente all'interno del workflow dando origine a molteplici thread di controllo. Il routing parallelo solitamente ha origine da un *AND-Split* e termina con un *AND-Join*.

3.3. Sequential Routing

Segmento di un processo in fase di esecuzione nel quale diverse attività vengono

eseguite in ordine sequenziale sotto un unico thread di esecuzione. In questo caso non vengono usati nessun *AND-Split* e nessun *AND-Join*.

3.4. AND-Split

Un punto, all'interno del workflow, nel quale un singolo thread di controllo si divide in due o più thread i quali vengono eseguiti in parallelo consentendo ad attività multiple di essere eseguite simultaneamente.

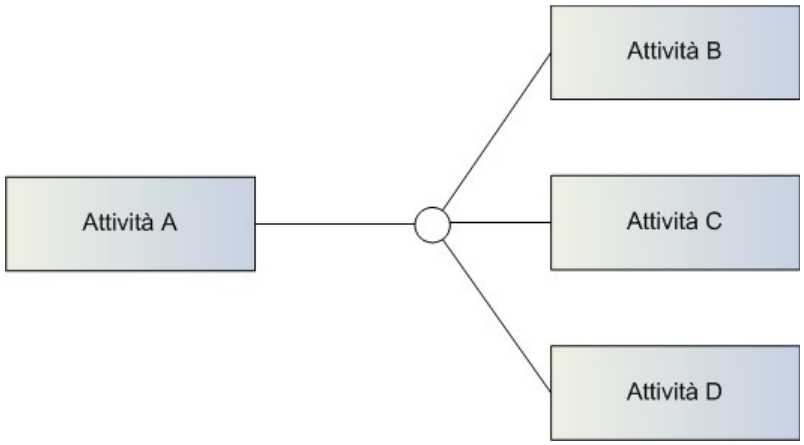

Figura 16: AND-Split

3.5. AND-Join

Un punto, all'interno del workflow, nel quale due o più attività ad esecuzione parallela convergono in un unico thread di controllo.

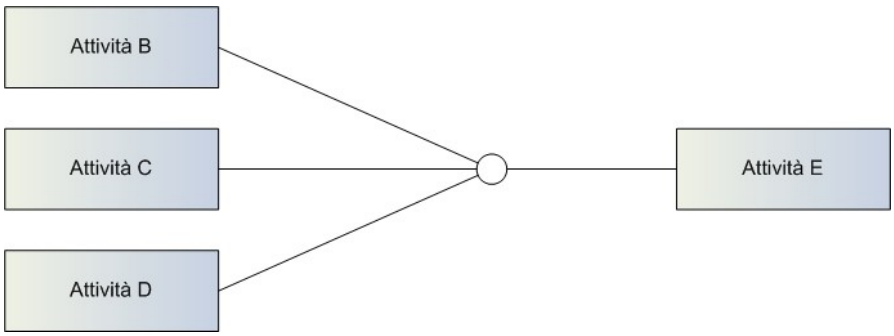

Figura 17: AND-Join

3.6. OR-Split

Un punto, all'interno del workflow, nel quale un singolo thread di controllo prende la decisione di scegliere un determinato ramo d'esecuzione quando incontra un insieme di differenti alternative nel workflow.

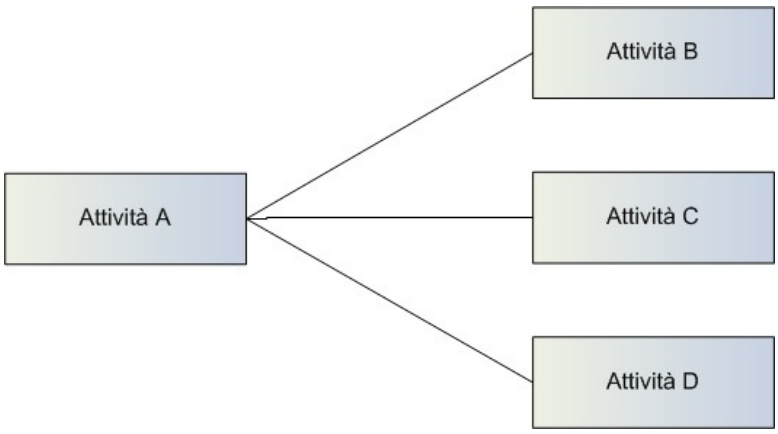

Figura 18: OR-Split

3.7. OR-Join

Un punto, all'interno del workflow, nel quale due o più attività alternative convergono verso una singola attività comune al prossimo passo di esecuzione nel workflow. Poiché non è stata eseguita nessuna esecuzione parallela delle attività al punto di join, non è necessaria nessuna sincronizzazione tra thread.

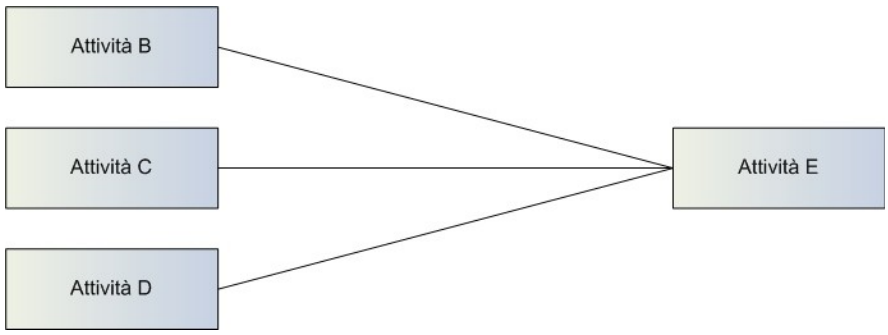

Figura 19: OR-Join

3.8. Iteration

Un ciclo di attività, all'interno del workflow, che implica l'esecuzione ripetitiva di una o più attività del workflow sino a quando una determinata condizione viene soddisfatta. Equivalente ad un loop di tipo while.

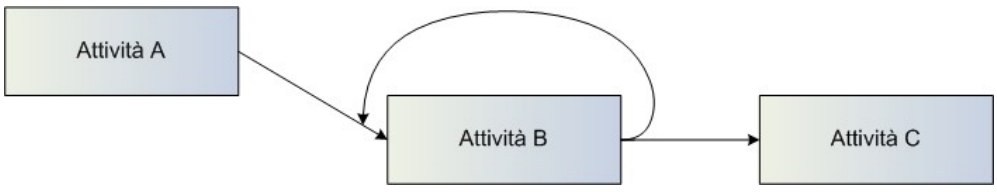

Figura 20: Iteration

3.9. Pre-Condition

Espressione logica che può essere valutata da un workflow engine per decidere se una istanza di processo, o un'attività all'interno di un processo, può essere iniziata.

3.10. Post-Condition

Espressione logica che può essere valutata da un workflow engine allo scopo di decidere se un'istanza di processo, o un'attività all'interno di un processo, è da ritenere completata.

3.11. Transition

Un punto, durante l'esecuzione di un processo, nel quale un'attività completa le sue operazioni e il thread di controllo passa ad un'altra, la quale inizia le sue operazioni.

3.12. Transition Condition

Espressione logica che può essere valutata da un workflow engine per determinare la sequenza di esecuzione delle attività all'interno di un processo.

4. Analisi di due Workflow Management System per l'e-Science

In questo capitolo verranno esaminati due Workflow Management System open-source orientati all'*e-Science*: *Taverna* e *Triana*. Il primo particolarmente orientato alla bioinformatica, il secondo divenuto, con la versione attuale, indipendente da un particolare dominio di applicazione ma sempre orientato all'ambito delle sperimentazioni scientifiche.
Come accennato nell'introduzione, non vengono considerati WfMS basati su BPEL, ma sono stati scelti i due sistemi citati per il loro particolare dominio di orientamento: e-Science anziché generici processi di business.

4.1. Taverna

Gli scopi del progetto Taverna sono quelli di fornire un linguaggio e un toolkit software per facilitare l'utilizzo dei workflow e delle applicazioni distribuite all'interno della comunità *eScience*. Taverna è gratuitamente disponibile sotto licenza GNU Lesser General Public License (LGPL).
Il progetto Taverna nasce da una collaborazione tra l'European Bioinformatics Institute (EBI), IT Innovation, la School of Computer Science, University of Newcastle, Newcastle Centre for Life, School of Computer Science at the University of Manchester e la Nottingham University Mixed Reality Lab . Altre collaborazioni includono il Biomoby project, Seqhound, Biomart e molte altre persone dislocate geograficamente [19].

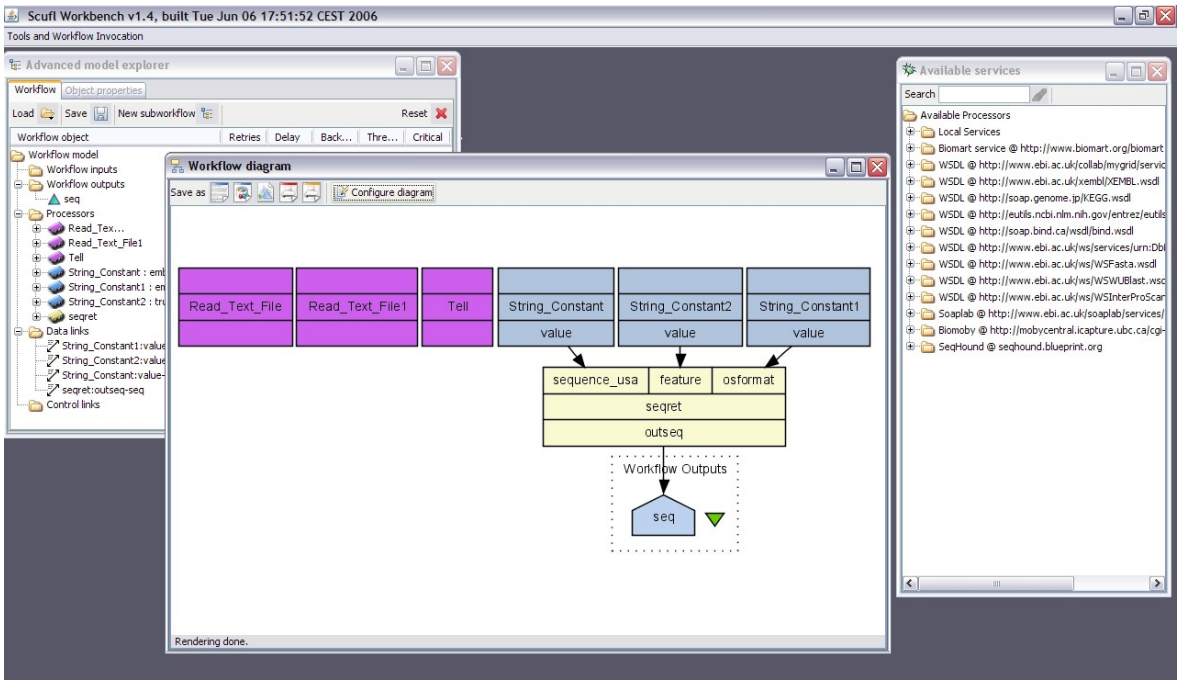

Figura 21: Il workbench di Taverna con un workflow in fase di design

Il toolkit Taverna si presenta come un *workbench* completamente visuale il quale permette agli utilizzatori di costruire complessi workflow a partire da componenti (Web Service) localizzati sia su macchine remote che su macchine locali; permette l'esecuzione del workflow e la visualizzazione dei risultati.

Taverna permette anche di effettuare svariate operazioni sui componenti, come, per esempio, il discovery, la descrizione e la selezione di librerie personalizzate a partire dai componenti disponibili in modo che si possano adattare all'utilizzo in una particolare applicazione.

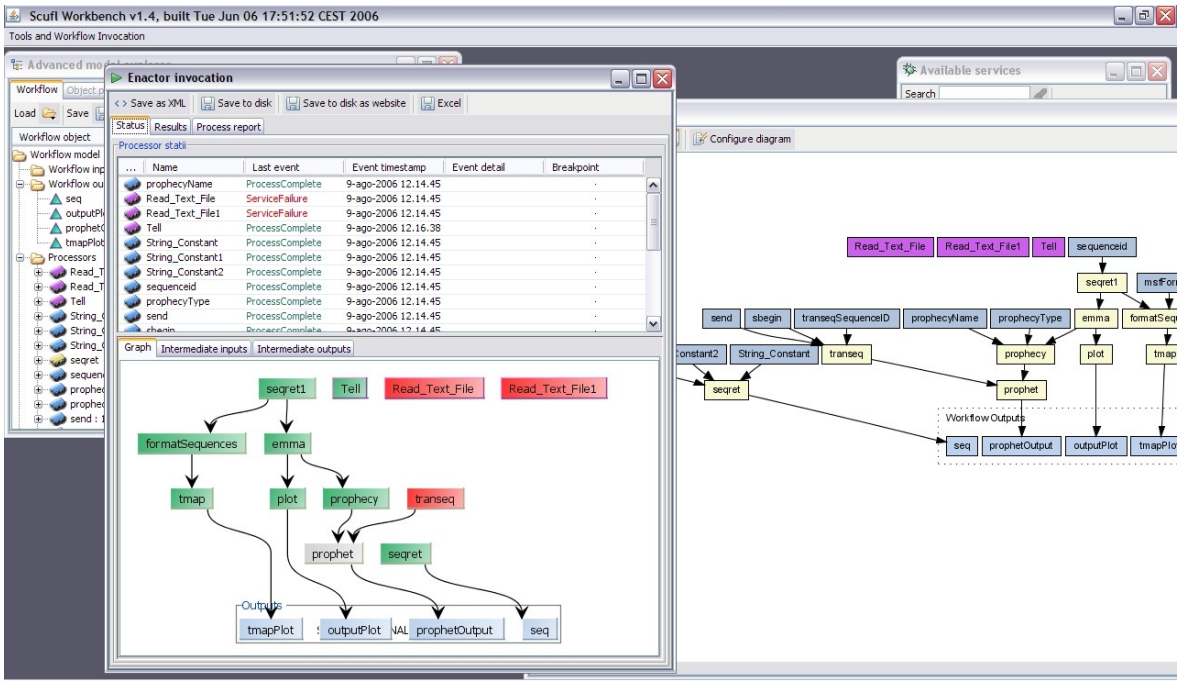

Figura 22: Esecuzione di un workflow in Taverna, la finestra dell'Enactor mostra il flusso d'esecuzione e gli eventuali problemi, come in questo caso

Taverna definisce alcuni concetti principali:

Workflow: definito come un set di componenti e relazioni tra essi utilizzati per definire un processo di una certa complessità a partire da blocchi costruttivi più semplici. Le relazioni posso essere di tipo *dati*, le quali trasferiscono l'output da un componente all'input di un altro componente; possono essere di tipo *controllo* le quali specificano alcune condizioni sull'esecuzione dei componenti. Un esempio di una relazione di tipo controllo può essere il classico ordine temporale tipo: "non eseguire il componente C1 finchè non si è completata l'esecuzione del componente C2".

Taverna esprime un workflow in un formato XML chiamato *XSCUFL (XML Simple Conceptual Unified Flow Language);* la Figura 23, illustra il documento XSCUFL riguardante un workflow creato con Taverna.

Componenti: un componente è un blocco costruttivo "riusabile" il quale esegue una ben definita funzione all'interno del processo di esecuzione. Naturalmente, trattandosi di applicazioni distribuite, i componenti possono trovarsi fisicamente su qualsiasi risorsa computazionale su Internet oppure in locale sulla macchina dell'utente di Taverna.

Service: tutti i servizi sono anche Componenti. Un servizio viene definito come un componente remoto. Taverna nasconde i protocolli di comunicazione con i servizi, per esempio SOAP, rendendoli trasparenti all'utilizzatore del workbench.

Enactor: Un *Workflow Enactor* è l'entità responsabile della coordinazione dell'invocazione dei *Component* costituenti un *Workflow*. L'Enactor governa l'intero processo di esecuzione del workflow includendo una visualizzazione della fase di esecuzione e dei trasferimenti di dati tra i componenti.

```
<?xml version="1.0" encoding="UTF-8"?>
<s:scufl xmlns:s="http://org.embl.ebi.escience/xscufl/0.1alpha" version="0.2"
log="0">
	<s:workflowdescription
lsid="urn:lsid:www.mygrid.org.uk:operation:MQ30X8Z9EX2" author="" title=""/>
	<s:processor name="Fail_if_false">

	<s:local>org.embl.ebi.escience.scuflworkers.java.FailIfFalse</s:local>
	</s:processor>
	<s:processor name="if_script">
		<s:beanshell>
			<s:scriptvalue>
if(Double.valueOf(a)&gt;=Double.valueOf(b))
{
= "true";
}
{
= "false";
}res;</s:scriptvalue>
			<s:beanshellinputlist>
				<s:beanshellinput
s:syntactictype="'text/plain'">a</s:beanshellinput>
				<s:beanshellinput
s:syntactictype="'text/plain'">b</s:beanshellinput>
			</s:beanshellinputlist>
			<s:beanshelloutputlist>
				<s:beanshelloutput
s:syntactictype="'text/plain'">res</s:beanshelloutput>
			</s:beanshelloutputlist>
		</s:beanshell>
	</s:processor>
	<s:processor name="Fail_if_true">

	<s:local>org.embl.ebi.escience.scuflworkers.java.FailIfTrue</s:local>
	</s:processor>
	<s:processor name="generateRandomNumber1">
		<s:arbitrarywsdl>

	<s:wsdl>http://localhost:8088/axis/UtilitiesWebService.jws?wsdl</s:wsdl>
			<s:operation>generateRandomNumber</s:operation>
		</s:arbitrarywsdl>
	</s:processor>
	<s:processor name="generateRandomNumber">
		<s:arbitrarywsdl>

	<s:wsdl>http://localhost:8088/axis/UtilitiesWebService.jws?wsdl</s:wsdl>
			<s:operation>generateRandomNumber</s:operation>
		</s:arbitrarywsdl>
	</s:processor>
	<s:processor name="mul">
		<s:arbitrarywsdl>

	<s:wsdl>http://localhost:8088/axis/SimpleMathWebService.jws?wsdl</s:wsdl>
			<s:operation>mul</s:operation>
		</s:arbitrarywsdl>
	</s:processor>
	<s:processor name="div">
		<s:arbitrarywsdl>

	<s:wsdl>http://localhost:8088/axis/SimpleMathWebService.jws?wsdl</s:wsdl>
			<s:operation>div</s:operation>
		</s:arbitrarywsdl>
```

```
        </s:processor>
        <s:link source="generateRandomNumber1:generateRandomNumberReturn"
sink="div:b"/>
        <s:link source="generateRandomNumber1:generateRandomNumberReturn"
sink="if_script:b"/>
        <s:link source="generateRandomNumber1:generateRandomNumberReturn"
sink="mul:b"/>
        <s:link source="generateRandomNumber:generateRandomNumberReturn"
sink="div:a"/>
        <s:link source="generateRandomNumber:generateRandomNumberReturn"
sink="if_script:a"/>
        <s:link source="generateRandomNumber:generateRandomNumberReturn"
sink="mul:a"/>
        <s:link source="if_script:res" sink="Fail_if_false:test"/>
        <s:link source="if_script:res" sink="Fail_if_true:test"/>
        <s:coordination name="mul_BLOCKON_Fail_if_false">
                <s:condition>
                        <s:state>Completed</s:state>
                        <s:target>Fail_if_false</s:target>
                </s:condition>
                <s:action>
                        <s:target>mul</s:target>
                        <s:statechange>
                                <s:from>Scheduled</s:from>
                                <s:to>Running</s:to>
                        </s:statechange>
                </s:action>
        </s:coordination>
        <s:coordination name="div_BLOCKON_Fail_if_true">
                <s:condition>
                        <s:state>Completed</s:state>
                        <s:target>Fail_if_true</s:target>
                </s:condition>
                <s:action>
                        <s:target>div</s:target>
                        <s:statechange>
                                <s:from>Scheduled</s:from>
                                <s:to>Running</s:to>
                        </s:statechange>
                </s:action>
        </s:coordination>
</s:scufl>
```

Figura 23: Documento in formato XSCUFL per la descrizione di un workflow in Taverna

L'ambiente di Taverna mette a disposizione dell'utente altre caratteristiche degne di nota, mediante le quali è possibile creare dei workflow davvero complessi e robusti, tra quelle più interessanti, riportiamo:

- *Retries*: la possibilità di specificare, per ogni processore (e quindi anche per i Web Service), il numero di tentativi che l'*enactor* del workflow deve

effettuare nel caso si verifichi un fallimento nell'invocazione dei servizi o dei componenti;

- *Delay*: la possibilità di specificare un ritardo, in millisecondi, tra un tentativo di invocazione dei processori e il successivo, naturalmente questa proprietà può essere legata alla precedente;

- *Backoff*: la possibilità di specificare un fattore per determinare di quanto il Delay deve essere incrementato per retries successive dopo il primo tentativo di invocazione dei processori;

- *Threads*: la possibilità di specificare, per ogni processore, il numero di istanze concorrenti che dovrebbero essere utilizzate dall'enactor durante una iterazione;

- *Critical*: valore booleano che indica la possibilità di definire il processore come "critico". Se impostato come "critico", al fallire dell'invocazione del processore segue un fallimento dell'intero workflow con conseguente interruzione della sua esecuzione; se non impostato, l'esecuzione del workflow, se possibile, continua, escludendo tutti i processori in qualche modo correlati a quello la cui invocazione è fallita;

- *Alternate Processor*: è possibile specificare, per un processore, un processore alternativo nel caso tutte le sue invocazioni da parte dell'enactor dovessero fallire, in accordo con le proprietà di Retries, Delay e Critical; naturalmente il processore alternativo specificato deve essere coerente con l'interfaccia esposta da quello "principale", ovvero il processore

alternativo presumibilmente deve essere in grado di effettuare lo stesso lavoro del "principale".

4.2. Triana

Triana, creato e distribuito secondo modello open-source dall' Università di Cardiff, è un altro Workflow Management System, scritto interamente in Java e quindi eseguibile su diverse piattaforme. Triana mette a disposizione un ambiente completamente grafico che permette di creare potenti costrutti di alto livello con uno sforzo "minimo" da parte dello sviluppatore, questo mediante creazione di workflow e composizione di servizi. Secondo quanto riportato dagli stessi creatori di Triana, esso permette di sviluppare rapidamente potenti programmi e applicazioni software senza avere a che fare con i dettagli e i costrutti dei linguaggi di programmazione classici. Triana può agevolmente essere usato per trattare una grossa varietà di tipi di dato: dati numerici, audio, immagini, file di testo. A questo scopo il suo ambiente mette a disposizione un toolkit di analisi dei segnali, di manipolazione di immagini, di desktop-publishing ad altre risorse software.

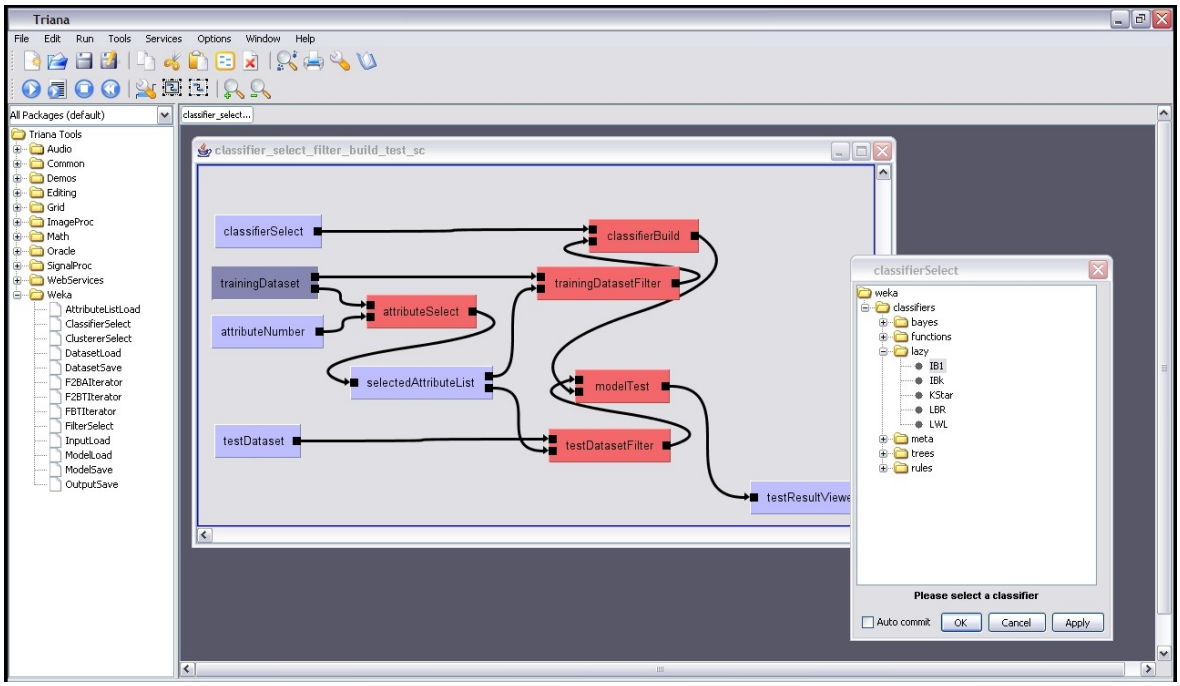

Figura 24: L'ambiente di Triana con la composizione e la personalizzazione di un workflow

Triana dimostra la sua potenza nell'automazione di task ripetitivi e il conseguente monitoraggio degli *spettri* di dati provenienti da eventuali esperimenti che possono durare giorni o mesi.

Come altresì detto, Triana viene sviluppato da scienziati all'Università di Cardiff, nel Regno Unito e viene abitualmente utilizzato in progetti europei come: *GEO600 (gravitational wave experiment)* [24].

4.2.1. Distributed Computing con Triana

All'interno del contesto di Triana, il toolkit distingue due tipi di componenti distribuiti: componenti G*rid-Oriented* e componenti S*ervice-Oriented* [25].

I componenti Servive-Oriented, come Web Service, sono componenti software di computazione accessibili da remoto; questi componenti possono essere

invocati da Triana e costituiscono gli eventuali moduli che compongono un workflow.

I componenti Grid-Oriented rappresentano dei lavori, avviati da Triana, su macchine remote utilizzando un *grid resource manager*. Diversamente dai componenti Service-Oriented, questi ultimi espongono interfacce accessibili da remoto perciò la comunicazione con essi è disponibile solamente mediante input/output su file. Come illustrato in Figura 25, Triana utilizza diverse interfacce per accedere a queste due diverse categorie di componenti, in breve:

per i componenti Service-Oriented viene usata la *GAP* (*Grid Application Prototype*) Interface. Per i componenti Grid-Oriented vengono utilizzate le *GridLab GAT (Grid Application Toolkit)* API.

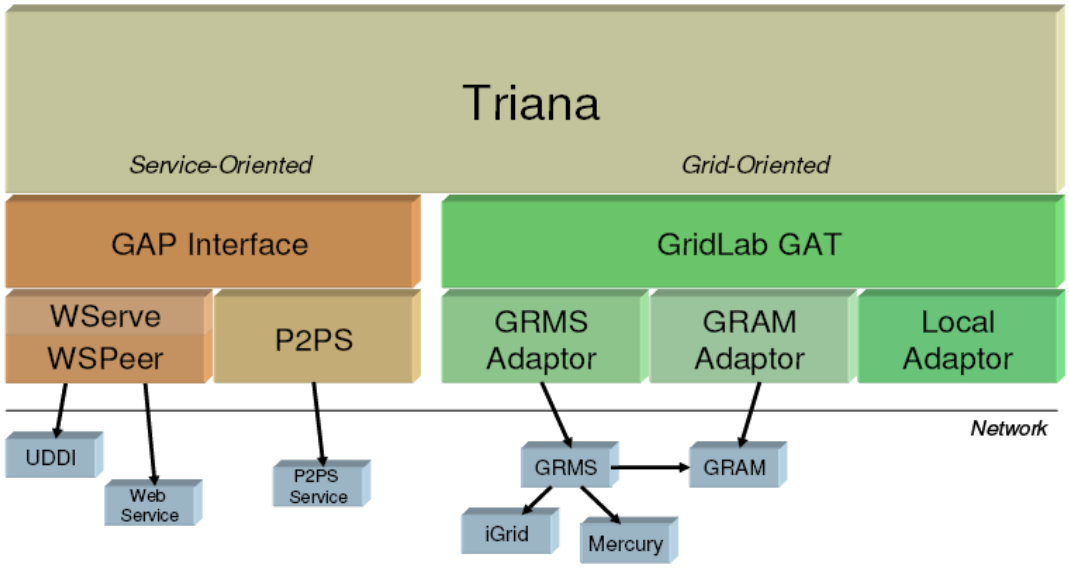

Figura 25: Architettura generale di Triana

In aggiunta a questi componenti, esiste anche una buona gamma di componenti

Java "locali" utilizzabili per la composizione dei workflow o per lo sfruttamento di facilities comunemente utilizzate: dalla semplice concatenazione di stringhe alla visualizzazione di grafici ottenuti dall'elaborazione dei segnali.

Grazie alla sua architettura e alle caratteristiche, Triana, originalmente creato per l'elaborazione dei segnali è diventato indipendente da un particolare dominio d'applicazione e quindi può essere usato con profitto in svariati campi d'applicazione come, citandone alcuni: bioinformatica, elaborazione dei segnali come l'individuazione delle onde gravitazionali e radio astronomia, data-mining, elaborazione di immagini mediche, elaborazione audio distribuita.

Come accennato precedentemente, anche Triana è un tool completamente scritto in Java ed ha la possibilità di essere esteso con la creazione di nuovi componenti locali, questo avviene grazie al fatto che sono messe a disposizione delle API (Application Program Interface) ben documentate per l'implementazione di processori personalizzati. Naturalmente essi possono essere redistribuiti ed utilizzati da altri utenti che necessitano di essi.

Un'altra caratteristica molto interessante è che Triana mette a disposizione delle procedure per pubblicare i workflow creati sotto forma di Web Service in un UDDI Registry specificato; in questo modo, un potenziale utente remoto può includere nel suo workflow, in via di composizione, un altro workflow completamente già definito e disponibile remotamente come Web Service, ergo descritto da un WSDL.

In Triana, un workflow è rappresentato mediante un formato XML simile al formato *WSFL (Web Service Flow Language)* il quale è un linguaggio basato su XML per la composizione e la coreografia di flussi di Web Service; WSFL viene utilizzato per la rappresentazione XML di un processo.

La figura seguente illustra la definizione di un workflow con Triana mediante il suo formato XML nativo.

```
<?xml version="1.0" encoding="UTF-8"?>
<tool>
 <toolname>triana-loop</toolname>
 <package />
 <inportnum>0</inportnum>
 <outportnum>0</outportnum>
 <inparam />
 <outparam />
 <parameters>
   <param name="popUpDescription" type="unknown">
     <value>No description for tool</value>
   </param>
 </parameters>
 <tasks>
   <task>
     <toolname>generateRandomNumber</toolname>
     <package>WebServices.UtilitiesWebService</package>
     <proxy type="WebService">
       <param paramname="portName">
         <value>UtilitiesWebService</value>
       </param>
       <param paramname="wsdlLocation">
         <value>http://localhost:8088/axis/UtilitiesWebService.jws?wsdl</value>
       </param>
       <param paramname="serializedPipe">
         <value>http://localhost:8088/axis/UtilitiesWebService.jws?wsdl:Utilitie
sWebService:generateRandomNumber</value>
       </param>
       <param paramname="operation">
         <value>generateRandomNumber</value>
       </param>
     </proxy>
     <renderingHints>
       <renderingHint hint="TaskGraphFactory" proxyDependent="true">
         <param paramname="factory">
           <value>WebServices (GAP)</value>
         </param>
       </renderingHint>
       <renderingHint hint="WebService" proxyDependent="true" />
     </renderingHints>
     <inportnum>0</inportnum>
     <outportnum>1</outportnum>
     <inparam />
     <outparam />
     <input>
       <type>Unknown Type</type>
     </input>
     <output>
       <type>Unknown Type</type>
     </output>
     <parameters>
       <param name="minOut" type="unknown">
         <value>1</value>
       </param>
       <param name="helpFile" type="unknown">
         <value>http://localhost:8088/axis/UtilitiesWebService.jws?wsdl</value>
       </param>
       <param name="popUpDescription" type="unknown">
         <value>No description for tool</value>
       </param>
       <param name="maxIn" type="unknown">
         <value>0</value>
```

```
      </param>
      <param name="maxOut" type="unknown">
        <value>1</value>
      </param>
      <param name="guiY" type="gui">
        <value>2.660377358490566</value>
      </param>
      <param name="guiX" type="gui">
        <value>0.11949685534591195</value>
      </param>
      <param name="outputType0" type="unknown">
        <value>java.lang.Double</value>
      </param>
      <param name="defaultOut" type="unknown">
        <value>1</value>
      </param>
      <param name="minIn" type="unknown">
        <value>0</value>
      </param>
      <param name="defaultIn" type="unknown">
        <value>0</value>
      </param>
    </parameters>
  </task>
  <task>
    <toolname>ConstView</toolname>
    <package>Common.Const</package>
    <proxy type="Java">
      <param paramname="unitPackage">
        <value>Common\Output</value>
      </param>
      <param paramname="unitName">
        <value>triana.tools.ConstView</value>
      </param>
    </proxy>
    <renderingHints>
      <renderingHint hint="TaskGraphFactory" proxyDependent="true">
        <param paramname="factory">
          <value>Default</value>
        </param>
      </renderingHint>
    </renderingHints>
    <inportnum>1</inportnum>
    <outportnum>0</outportnum>
    <inparam />
    <outparam />
    <input>
      <type>java.lang.Number</type>
      <type>triana.types.Const</type>
    </input>
    <parameters>
      <param name="minOut" type="internal">
        <value>0</value>
      </param>
      <param name="helpFile" type="internal">
        <value>ConstView.html</value>
      </param>
      <param name="popUpDescription" type="internal">
        <value>Displays a Const in a Window</value>
      </param>
      <param name="maxIn" type="internal">
        <value>1</value>
```

```
        </param>
        <param name="maxOut" type="internal">
          <value>2147483647</value>
        </param>
        <param name="guiY" type="gui">
          <value>2.660377358490566</value>
        </param>
        <param name="guiX" type="gui">
          <value>3.729559748427673</value>
        </param>
        <param name="constValue" type="unknown">
          <value>76.54988118591083</value>
        </param>
        <param name="TRIGGER" type="userAccessible">
          <value>Active</value>
        </param>
        <param name="defaultOut" type="internal">
          <value>0</value>
        </param>
        <param name="paramPanelClass" type="internal">
          <value>triana.tools.ConstViewPanel</value>
        </param>
        <param name="minIn" type="internal">
          <value>1</value>
        </param>
        <param name="toolVersion" type="internal">
          <value>3</value>
        </param>
        <param name="defaultIn" type="internal">
          <value>1</value>
        </param>
      </parameters>
    </task>
    <task>
      <toolname>inc</toolname>
      <package>WebServices.SimpleMathWebService</package>
      <proxy type="WebService">
        <param paramname="portName">
          <value>SimpleMathWebService</value>
        </param>
        <param paramname="wsdlLocation">
          <value>http://localhost:8088/axis/SimpleMathWebService.jws?wsdl</value>
        </param>
        <param paramname="serializedPipe">
          <value>http://localhost:8088/axis/SimpleMathWebService.jws?wsdl:SimpleM
athWebService:inc</value>
        </param>
        <param paramname="operation">
          <value>inc</value>
        </param>
      </proxy>
      <renderingHints>
        <renderingHint hint="TaskGraphFactory" proxyDependent="true">
          <param paramname="factory">
            <value>WebServices (GAP)</value>
          </param>
        </renderingHint>
        <renderingHint hint="WebService" proxyDependent="true" />
      </renderingHints>
      <inportnum>1</inportnum>
      <outportnum>1</outportnum>
      <inparam />
```

```
    <outparam />
    <input>
      <type>Unknown Type</type>
    </input>
    <output>
      <type>Unknown Type</type>
    </output>
    <parameters>
      <param name="minOut" type="unknown">
        <value>1</value>
      </param>
      <param name="helpFile" type="unknown">
        <value>http://localhost:8088/axis/SimpleMathWebService.jws?wsdl</value>
      </param>
      <param name="popUpDescription" type="unknown">
        <value>No description for tool</value>
      </param>
      <param name="maxIn" type="unknown">
        <value>1</value>
      </param>
      <param name="maxOut" type="unknown">
        <value>1</value>
      </param>
      <param name="guiY" type="gui">
        <value>5.188679245283019</value>
      </param>
      <param name="guiX" type="gui">
        <value>1.949685534591195</value>
      </param>
      <param name="outputType0" type="unknown">
        <value>java.lang.Double</value>
      </param>
      <param name="defaultOut" type="unknown">
        <value>1</value>
      </param>
      <param name="minIn" type="unknown">
        <value>1</value>
      </param>
      <param name="defaultIn" type="unknown">
        <value>1</value>
      </param>
    </parameters>
  </task>
  <task>
    <toolname>Loop</toolname>
    <package>Common.Control</package>
    <proxy type="Java">
      <param paramname="unitPackage">
        <value>Common\Logic</value>
      </param>
      <param paramname="unitName">
        <value>Common.Logic.Loop</value>
      </param>
    </proxy>
    <renderingHints>
      <renderingHint hint="ControlTool" proxyDependent="false">
        <param paramname="description">
          <value />
        </param>
        <param paramname="validStandalone">
          <value>true</value>
        </param>
```

```
        <param paramname="controlName">
          <value>Loop</value>
        </param>
        <param paramname="validTopLevel">
          <value>false</value>
        </param>
      </renderingHint>
      <renderingHint hint="TaskGraphFactory" proxyDependent="true">
        <param paramname="factory">
          <value>Default</value>
        </param>
      </renderingHint>
    </renderingHints>
    <inportnum>2</inportnum>
    <outportnum>2</outportnum>
    <inparam />
    <outparam />
    <input>
      <type>java.lang.Object</type>
    </input>
    <output>
      <type>java.lang.Object</type>
    </output>
    <parameters>
      <param name="comp0" type="internal">
        <value>=</value>
      </param>
      <param name="conditionUnit" type="userAccessible">
        <value>Common.Logic.DefaultExitCondition</value>
      </param>
      <param name="maxOut" type="internal">
        <value>2147483647</value>
      </param>
      <param name="popUpDescription" type="internal">
        <value>A conditional looping unit</value>
      </param>
      <param name="$var3" type="userAccessible">
        <value>0</value>
      </param>
      <param name="$var2" type="userAccessible">
        <value>0</value>
      </param>
      <param name="$var1" type="userAccessible">
        <value>0</value>
      </param>
      <param name="defaultIn" type="internal">
        <value>2</value>
      </param>
      <param name="$var0" type="userAccessible">
        <value>0</value>
      </param>
      <param name="outputType1" type="unknown">
        <value>java.lang.Double</value>
      </param>
      <param name="toolVersion" type="internal">
        <value>3</value>
      </param>
      <param name="minIn" type="internal">
        <value>2</value>
      </param>
      <param name="outputType0" type="unknown">
        <value>java.lang.Double</value>
```

```
      </param>
      <param name="eq0" type="internal">
        <value>10</value>
      </param>
      <param name="param0" type="internal">
        <value>iterations</value>
      </param>
      <param name="enabled" type="userAccessible">
        <value>true</value>
      </param>
      <param name="exitCondition" type="userAccessible">
        <value>(iterations =  10)</value>
      </param>
      <param name="defaultOut" type="internal">
        <value>2</value>
      </param>
      <param name="defaultNodeRequirement" type="internal">
        <value>optional</value>
      </param>
      <param name="paramUpdatePolicy" type="internal">
        <value>processUpdate</value>
      </param>
      <param name="minOut" type="internal">
        <value>2</value>
      </param>
      <param name="maxIn" type="internal">
        <value>2147483647</value>
      </param>
      <param name="totalIterations" type="userAccessible">
        <value>10</value>
      </param>
      <param name="iterations" type="userAccessible">
        <value>10</value>
      </param>
      <param name="init$0" type="internal">
        <value />
      </param>
      <param name="paramPanelClass" type="internal">
        <value>Common.Logic.LoopPanel</value>
      </param>
      <param name="helpFile" type="internal">
        <value>Loop.html</value>
      </param>
      <param name="guiY" type="gui">
        <value>2.811320754716981</value>
      </param>
      <param name="guiX" type="gui">
        <value>1.9056603773584906</value>
      </param>
    </parameters>
  </task>
  <connections>
    <connection>
      <source taskname="generateRandomNumber" node="0" />
      <target taskname="Loop" node="0" />
    </connection>
    <connection>
      <source taskname="Loop" node="0" />
      <target taskname="ConstView" node="0" />
    </connection>
    <connection>
      <source taskname="Loop" node="1" />
```

```
        <target taskname="inc" node="0" />
      </connection>
      <connection>
        <source taskname="inc" node="0" />
        <target taskname="Loop" node="1" />
      </connection>
    </connections>
  </tasks>
</tool>
```

Figura 26: Formato XML nativo di Triana per la descrizione dei workflow

4.3. Funzionalità di Taverna e Triana a confronto

Continuando l'analisi dei due Workflow Management System (*WfMS*) orientati all'e-Science, descritti nella sezione precedente, non possiamo non effettuare un'analisi più accurata delle funzionalità messe a disposizione dai due tool.

Cominciamo dicendo che Taverna, già in configurazione standard, effettua il discovery di molti Web Service per la bioinformatica disponibili in rete, citandone qualcuno abbiamo servizi, immediatamente utilizzabili per costruire dei workflow, mediante processori per *Soaplab, registri BioMOBY, Biomart, SeqHound* ed altri. Triana si pone, invece, in una posizione che possiamo definire più "neutra", infatti, pur mettendo a disposizione funzionalità specifiche all'e-Science, non effettua nessun particolare discovery di servizi. Tralasciando questi aspetti, concentreremo la nostra attenzione sui costrutti messi a disposizione e sul loro tipo di implementazione; in particolare, faremo riferimento ai principali costrutti generici dei sistemi di Workflow, così come teorizzati e descritti nel capitolo relativo in questo documento. Entrambi i tool, permettono la costruzione e l'esecuzione di workflow "assemblando" tra loro componenti Web Service e componenti locali, come costanti, visualizzatori di messaggi, immagini, ecc...

Per la nostra analisi ci serviremo di due semplici Web Service, realizzati in *Java* con *Apache Axis* e *Apache Tomcat,* che espongono le seguenti funzionalità:

SimpleMathWebService			
Operazioni	**Parametri**	**Risultato**	**Descrizione**
sum	Due numeri in virgola mobile con precisione doppia	Somma dei valori passati come parametri	L'operazione effettua la somma tra due numeri
div	Due numeri in virgola mobile con precisione doppia	Risultato della divisione del numero passato come primo parametro quello passato come secondo parametro	L'operazione effettia la divisione tra i due numeri passati come parametri
dec	Un numero in virgola mobile con precisione doppia	Il numero passato come parametro decrementato di 1	Decrementa di 1 il numero passato come parametro
isGreater	Due numeri in virgola mobile con precisione doppia	Valore booleano true se e solo se il primo parametro è maggiore del secondo, restituisce false altrimenti	Confronta i due numeri passati come parametri e verifica se il primo è maggiore del secondo
inc	Un numero in virgola mobile con precisione doppia	Il numero passato come parametro incrementato di 1	Incrementa di 1 il valore numerico passato come parametro
getVersion	nessuno	Il numero di	Restituisce il

SimpleMathWebService			
		versione del servizio come stringa	numero di versione attuale del servizio
sub	Due numeri in virgola mobile con precisione doppia	Il risultato della sottrazione del secondo parametro dal primo	Sottrae i due numeri passati come parametri
mul	Due numeri in virgola mobile con precisione doppia	Il risultato della moltiplicazione del primo parametro per il secondo	Moltiplica i valori numerici passati come parametri

Tabella 2: Operazioni esposte dal SimpleMathWebService, Web Service di test

UtilitiesWebService			
Operazioni	**Parametri**	**Risultato**	**Descrizione**
generateTimestampString	nessuno	Una stringa con il timestamp corrente	Restituisce la data e l'ora al momento dell'invocazione del servizio
generateRandomNumber	nessuno	Un numero in precisione doppia casuale tra 0.0 e 100.0	L'operazione genera e restituisce un numero casuale in virgola mobile con precisione doppia compreso tra 0.0 e 100.0
generateDoubleArray	nessuno	Un array di 10 numeri in virgola mobile con precisione doppia	L'operazione genera e restituisce un array di 10 numeri casuali in virgola mobile con precisione doppia compresi tra 0.0 e 100.0
getVersion	nessuno	Il numero di versione del servizio come stringa	Restituisce il numero di versione attuale del servizio

Tabella 3: Le operazione esposte dal UtilitiesWebService, Web Service di test

Per effettuare l'analisi, i due Web Service precedenti vengono entrambi importati nei due WfMS attraverso le loro specifiche operazioni di *discovery* e *import* a partire dalla locazione del descrittore WSDL e successivamente le loro operazioni vengono combinate in modo da ottenere e testare una certa logica applicativa come illustrata nelle sezioni successive.

4.3.1. Sequential Routing

Questa funzionalità di base dei sistemi di workflow è trattata in maniera pressoché identica da entrambi i WfMS; entrambi infatti permettono, in via del tutto grafica, la selezione delle componenti e il collegamento per la creazione di un flusso di esecuzione sequenziale.
Considerando i due Web Service di test illustrati nelle tabelle precedenti, un esempio di elaborazione sequenziale potrebbe essere la seguente:

Genera due numeri casuali e sommali, dopodiché incrementa di uno il risultato della somma

Le figure seguenti illustrano i workflow appositamente creati con Taverna e Triana per ottenere l'elaborazione precedente.

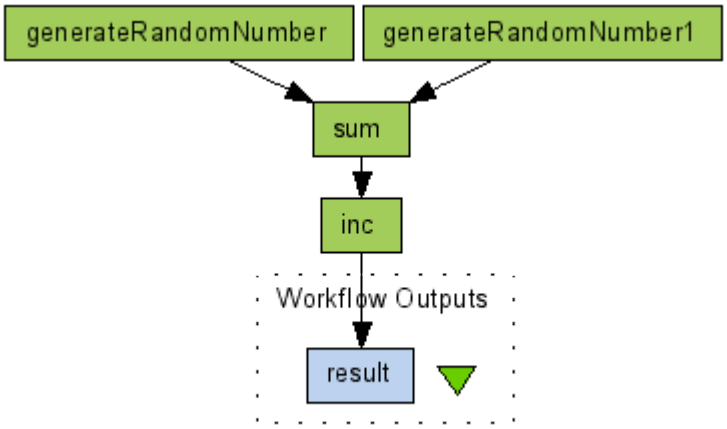

Figura 27: Workflow realizzato con Taverna per incrementare di uno la somma di due numeri casuali

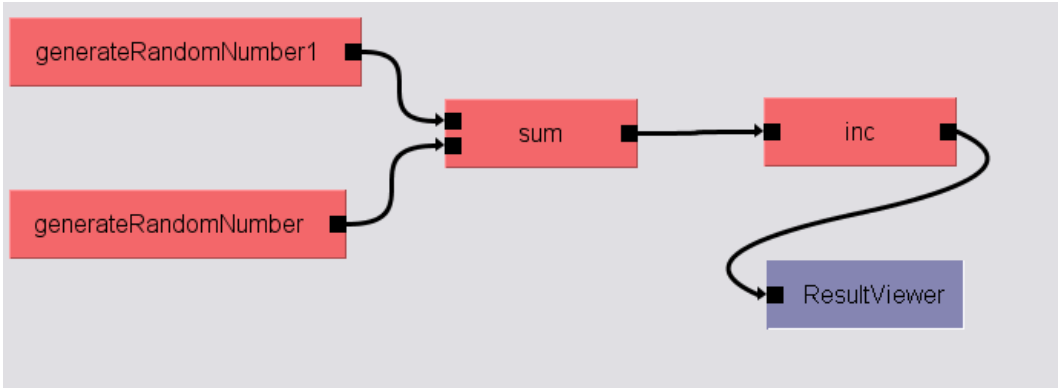

Figura 28: Workflow realizzato con Triana per incrementare di uno la somma di due numeri casuali

4.3.2. Parallel Routing

Per testare questa funzionalità, nei due WfMS, cerchiamo di creare un workflow

che effettui le seguenti operazioni:

Genera, contemporaneamente, due coppie di numeri casuali e calcola la loro somma per ciascuna coppia, dopodiché effettua la moltiplicazione delle due somme.

Le figure seguenti illustrano i workflow ottenuti.

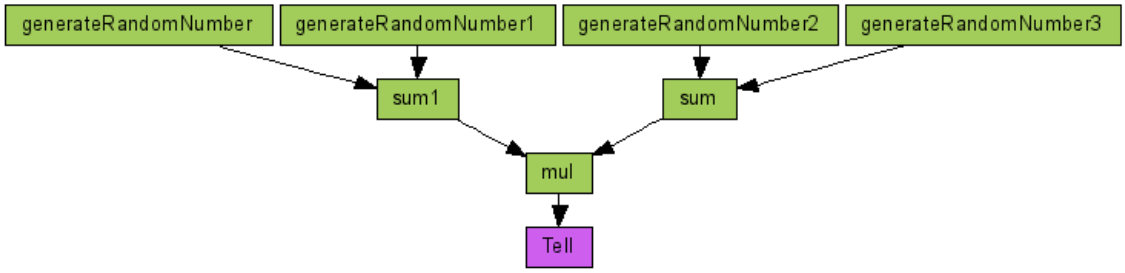

Figura 29: Parallel Routing in Taverna

In Triana l'elaborazione parallela è ottenuta consentendo all'utente di specificare il numero nodi di input-output in una unità del workflow. In pratica i nodi di *AND-Split* e di *AND-Join* sono implementati da un qualsiasi blocco nel diagramma;

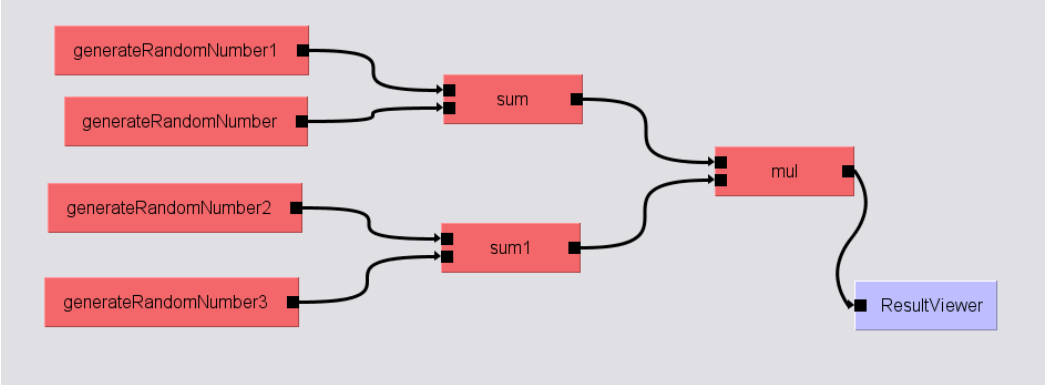

Figura 30: Parallel Routing in Triana

4.3.3. Conditional Processing

L'elaborazione condizionata viene usata, all'uscita da un nodo del workflow, per consentire la scelta di un percorso di esecuzione tra quelli disponibili. Naturalmente il "ramo" scelto dipende da un particolare attributo del workflow o da una *condizione*. Nella letteratura, spesso, si fa riferimento, teoricamente, a questo tipo di elaborazione, introducendo due nodi particolari: *IF-Split*, *IF-Join*, equivalenti, rispettivamente a *OR-Split* e a *OR-Join*.
Mentre Triana mette a disposizione dei componenti appositi per il costrutto di controllo di tipo *if*, Taverna mette a disposizione dell'utente la possibilità di creare degli script per codificare qualsiasi tipo di controllo. Gli script devono essere scritti in Java e sono interpretati mediante l'integrazione (già presente nella distribuzione di Taverna) di BeanShell, un interprete, di dimensioni ridotte e integrabile, di sorgenti Java con caratteristiche di un linguaggio di scripting ad oggetti [14].

Per illustrare un salto condizionato in un workflow utilizzando entrambi i tool, supponiamo di voler costruire un flusso che esegua le seguenti operazioni:

Genera due numeri casuali a e b e confrontali, se a>=b allora dividi a per b, altrimenti moltiplica a per b

Come abbiamo detto, per realizzare questo workflow, ci serviamo di un piccolo script Java che effettua il confronto di due numeri a e b e che restituisce il booleano *true* se e solo se a>=b, il booleano *false* altrimenti. La figura seguente mostra il listato.

```
if(Double.valueOf(a)>=Double.valueOf(b))
{

      res= "true";
}
else
{
      res= "false";
}
return res;
```

Figura 31: Lo script in Java, per il confronto di due numeri in virgola mobile, che viene eseguito da BeanShell all'interno di Taverna

Taverna mette a disposizione due semplici componenti per effettuare test di tipo booleano: *Fail_if_true* e *Fail_if_false*; il primo interrompe il flusso d'esecuzione se gli viene passato il booleano *true* in input, il secondo se gli viene passato il booleano *false,* altrimenti il flusso continua normalmente.

Il diagramma del workflow così ottenuto è il seguente:

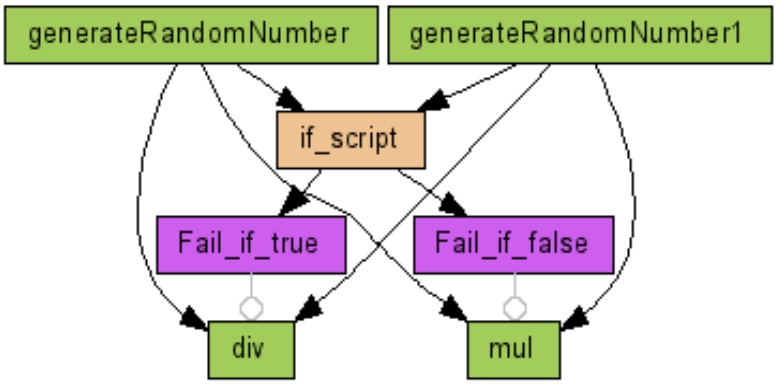

Figura 32: Workflow con salto condizionato in Taverna

Il funzionamento è il seguente: lo script Java effettua il confronto e il risultato (o *true* o *false*) viene passato come input ad entrambi i componenti di test booleano;

a seconda del valore passato uno dei due interrompe il flusso e l'esecuzione continua solo seguendo un ramo all'interno del workflow, eseguendo o la divisione o la moltiplicazione dei valori numerici generati.
Nella figura precedente i connettori in grigio tra i test booleani e le operazioni di *div* e *mul* sono dipendenze speciali di Taverna del tipo *Coordinate from...* le quali indicano che quanto segue verrà eseguito solo in dipendenza dal componente origine del connettore.
In Triana esiste già un componente "interno" per la creazione di salti condizionati ma ha la proprietà di accettare come input solo delle costanti. D'altra parte, osservando il codice della sua implementazione in Java (accessibile internamente all'ambiente di Triana stesso) è facilmente personalizzabile per qualsiasi esigenza.

4.3.4. Iteration

Per le iterazioni, i due *WfMS* adottano approcci decisamente diversi; questo avviene perché Taverna è un WfMS fondamentalmente di tipo *data-flow oriented* (basato sulla logica del flusso delle informazioni) anziché *control-flow oriented* (basato sulla logica di controllo) e quindi molti dei *pattern* definiti per i Workflow Management System, non sono applicabili, un esempio su tutti è, appunto, l'iterazione: non ci sono infatti costrutti o processori espliciti di *loop* in Taverna. Nonostante questo modello di flusso, Taverna, comunque offre un minimale controllo del flusso, adottando una semplice tecnica di *control flow*, utilizzata, ad esempio, per il salto condizionato nei workflow, come descritto più avanti in questa sezione.
Taverna usa un tipo di gestione dei cicli, denominato *Implicit Iteration*. Prendiamo

in considerazione un componente (un processore) *f* il quale accetta un elemento a come parametro di input; assumiamo che nel workflow gli venga passata una lista di elementi, il componente verrà invocato tante volte quanti sono gli elementi nella lista passando come parametro elemento per elemento.

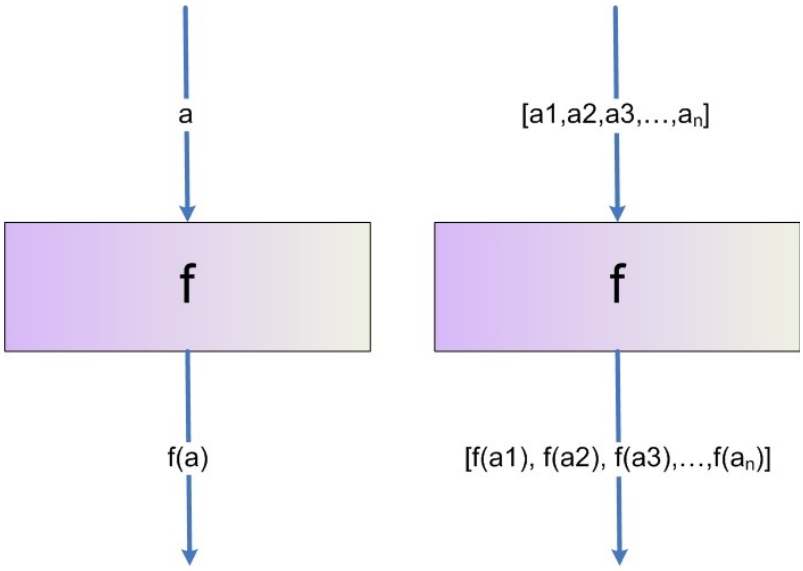

Figura 33: Implicit Iteration in Taverna; se ad un processore che accetta un parametro a in input viene passata una lista di elementi, Taverna applica automaticamente un iterazione invocando il processore per ciascun elemento in lista

Se prendiamo in considerazione un processore che accetta più di un parametro come input, e passiamo delle liste di elementi, Taverna può essere configurato per adottare due tipi diversi di strategie d'iterazione: *cross product* o *dot product*; nel primo caso viene eseguito il prodotto cartesiano degli elementi nelle liste ed invocato il processore con ciascun risultato, nel secondo caso, gli elementi vengono presi ad *n-ple* seguendo il loro ordine. L'utente è libero di cambiare questa impostazione configurando graficamente l'iterazione.

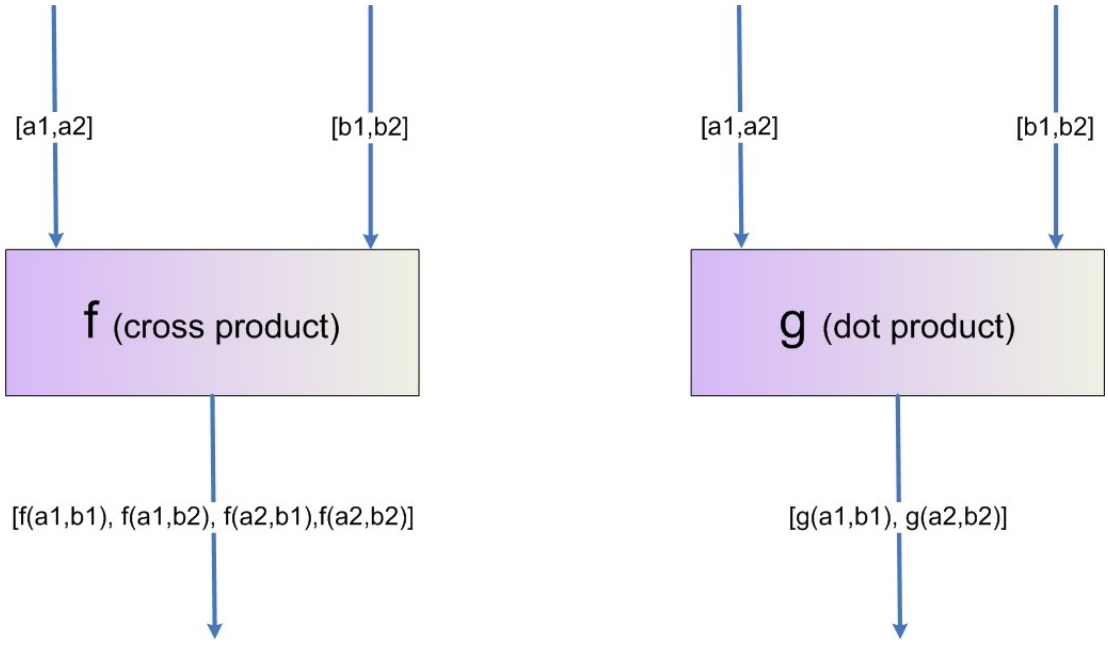

Figura 34: Le due diverse configurabili strategie di iterazione in Taverna nel caso di parametri multipli: strategie di cross e dot product

Facendo riferimento a quanto detto all'inizio di questa sezione, anche Triana presenta fondamentalmente caratteristiche di un *data-flow oriented WfMS* anche se offre anche delle funzionalità di un *control-flow oriented WfMS*. Triana supporta il *control flow* anche tra attività tra le quali non esiste un esplicito collegamento di *data flow*, queste funzionalità vengono chiamate *Trigger* all'interno dell'ambiente dell'applicazione.

Ritornando al discorso delle *Iteration,* in Triana esiste, ad esempio, un componente apposito per effettuare iterazioni con costrutti tipo *for* e *while* con condizioni d'uscita basate su variabili, compreso il controllo del numero di iterazioni.

Immaginiamo di voler costruire un workflow per eseguire le seguenti operazioni:

Genera un numero casuale ed incrementalo di uno per 10 volte

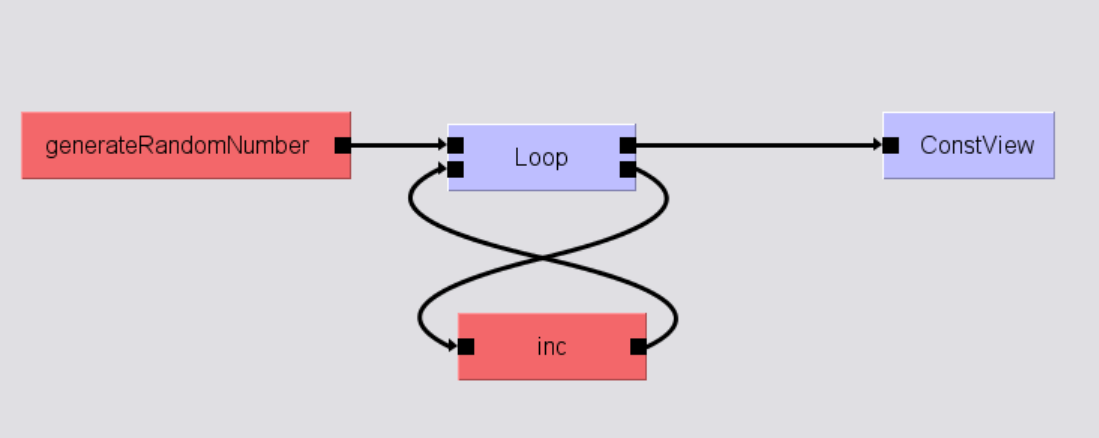

Figura 35: Implementazione di un loop in Triana: utilizzando il componente Loop ed impostando la condizione d'uscita tra le sue proprietà possiamo creare un ciclo condizionato. In questo esempio viene invocato un Web Service che genera un numero casuale il quale viene incrementato (sempre mediante l'invocazione di un servizio remoto) per 10 volte

Utilizzando la funzionalità di raggruppamento dei processori del workflow in Triana si possono realizzare, naturalmente, loop più complessi e articolati.

5. Bioinformatica e servizi per la bioinformatica

Con il *The Human Genome Project (Progetto Genoma Umano)*, ovvero la costituzione di un progetto internazionale per la mappatura completa del *DNA* umano, si è assistito ad una notevole evoluzione nella ricerca biologica. Le sfide con cui essa ora deve rapportarsi sono la gestione delle enormi basi di dati in crescita, l'elaborazione di questi dati, la sperimentazione mediante algoritmi e tecniche miste, l'analisi e la visualizzazione dei risultati.

La convergenza della biologia verso l'informatica e la disponibilità pubblica dei dati, degli strumenti e degli algoritmi sulla rete Internet hanno portato al sorgere di alcune problematiche, già esistenti in campo prettamente informatico, ma ora sempre più in evidenza, come la notevole eterogeneità dei software e una loro integrazione per la costituzione di un insieme distribuito di applicazioni che possano veramente aiutare le ricerche nel campo della biologia. E' sorta quindi la necessità di adottare delle tecnologie abilitanti per la costruzione o l'integrazione di componenti software in ambito distribuito e decentralizzato i quali consentano di integrare su larga scala i dati disponibili e consentire, quindi, al ricercatore di estrarre e manipolare questi dati in una maniera decisamente più agevole. Con la nascita della bioinformatica ed il livello della combinazione software-potenza di calcolo raggiunto al giorno d'oggi, alcuni standard e strumenti dell'informatica cominciano a rivelarsi fondamentali per l'accesso ai dati e alla loro elaborazione in una maniera semplificata anche se basata su sistemi e piattaforme eterogenei. Tecnologie come Web Services e Workflow Management System costituiscono una buona piattaforma da cui partire per arrivare alla costruzione di un "progetto" globale attivo per la ricerca biomolecolare distribuita a livello internazionale. Parecchi progetti sono da qualche tempo attivi e in continua evoluzione ed inoltre alcune importanti basi

dati di biologia molecolare sono ora accessibili mediante Web Service creati e mantenuti da, per citarne qualcuno, l'European Bioinformatic Institute (EBI) e il National Center for Biotechnology Information (NCBI). Contemporaneamente sono disponibili, anche open-source, dei sistemi di Workflow Management per la creazione di flussi di procedure automatiche che possono contribuire collaborativamente al lavoro incessante della ricerca sul genoma umano.

5.1. Un esempio di applicazione dei Web Service e sistemi di Workflow alla bioinformatica

Per illustrare le potenzialità dei sistemi di workflow e dei web service nell'ambito delle applicazioni distribuite, consideriamo come particolare campo d'applicazione la bioinformatica, costruendo un esperimento "distribuito" per la classificazione di dati genomici, provenienti da analisi "a *microarray" e* riguardanti alcune forme tumorali, facendo uso di tecniche di *data-mining*. L'esperimento si conclude con dei report indicanti l'errore di classificazione al variare del numero di attributi genomici selezionati. I dataset utilizzati sono quelli relativi alla *Acute Lymphoblastic Leukemia* [26], ottenuti effettuando una pre-selezione degli attributi realizzata mediante la funzione di *attribute importance* del pacchetto di *Oracle* e portando il loro numero dagli originali *12558* a *400,* allo scopo di rendere meno computazionalmente pesanti le prove. Naturalmente nessun problema sorgerebbe nell'applicare l'elaborazione al dataset completo, risulterebbe solamente un aumento dei tempi e un maggiore utilizzo di memoria RAM nella macchina host di esecuzione del workflow.

I *microarray*, comunemente conosciuti anche come *gene chip, DNA chip* o *gene array* sono una collezione di microscopici DNA *spot,* "appiccicati" ad una superficie solida, come vetro, plastica o chip di silicio, che formano un *array* allo scopo di consentire la profilazione e il monitoraggio dei livelli d'espressione per migliaia di geni contemporaneamente.

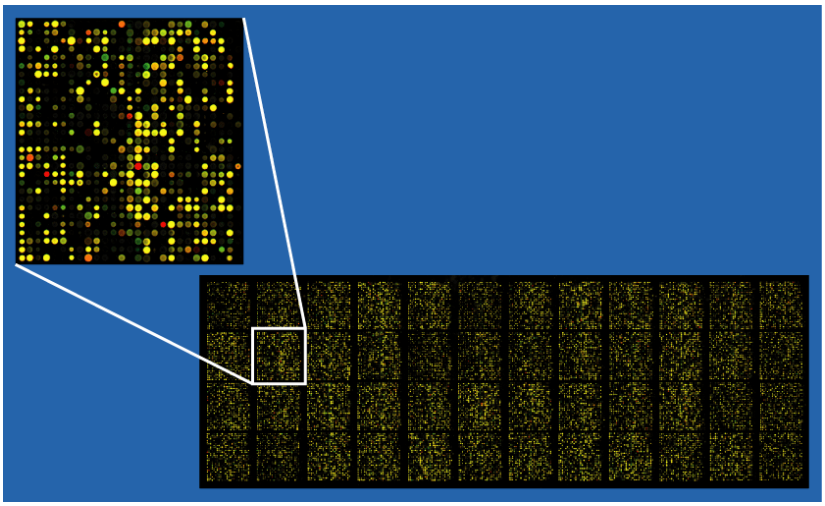

Figura 36: Esempio di un microarray con circa 40000 DNA spot

In questo modo possono essere ottenute delle grosse basi di dati genomici la cui analisi solleva un notevole numero di problematiche di tipo statistico e di data-mining tra cui la normalizzazione dei dati stessi.

E' a partire da questi dati che vengono effettuati gli esperimenti illustrati di seguito.

Le applicazioni implementate e messe a disposizione in rete, mediante *deployment* su alcune macchine, sono costituite da una serie di Web Service, creati con *Apache Axis* e *Apache Tomcat*, i quali consentono di effettuare le seguenti operazioni [15][16]:

Feature Selection	Estrae un sottoinsieme di attributi da un dataset fornito come input, mantenendolo possibilmente piccolo; gli attributi ritenuti significanti vengono mantenuti e inclusi nella lista restituita come risultato, mentre quelli ritenuti irrilevanti non vengono inclusi in tale lista. La rilevanza di ogni singolo attributo viene determinata dalla misura ottenuta tramite alcuni criteri statistici (per esempio $\chi 2$) oppure basati sulla teoria dell'informazione (per esempio la minimum description lenght). La Feature Selection è necessaria per rendere le operazioni di classificazione fattibili in termini di risorse computazionali dal momento che i dataset generati dall'analisi a micro-array sono caratterizzati da un grande numero di attributi e non tutti possono essere necessari e rilevanti per il tipo di ricerca che si vuole effettuare.
Filtering	Riduce la dimensione del dataset originali, fornito in input, mantenendo soltanto un sottoinsieme di tutti gli attributi (per esempio mantenendo solo quelli selezionati tramite Feature Selection).
Classification	Costruisce modello di classificazione (*classifier*), per un determinato dataset, utilizzando degli algoritmi di classificazione come le *Reti Bayesiane* [20], *Support Vector Machine (SVM)* o *K-nearest neighbor* ed altri.

Testing	Effettua il test del modello di classificazione o clusterizzazione in esame utilizzando un insieme indipendente di dati ed effettua le misurazioni del numero di dati correttamente classificati rapportato a quelli non correttamente classificati.

Tabella 4: Operazioni remote messe a disposizione dai Web Service implementati per esperimenti di bioinformatica

Gli esperimenti illustrati effettuano una classificazione di dati genomici in base ad una *feature selection* utilizzando un insieme di dati per il training dell'algoritmo di classificazione scelto in fase di esecuzione ed un altro insieme di dati per il test del classificatore addestrato. In base a questi, possiamo ottenere un report del variare dell'errore di classificazione al variare del numero di attributi via via scelto.

L'esperimento è realizzato mediante composizione di un workflow, utilizzando i due WfMS illustrati in precedenza: Triana e Taverna; il workflow a sua volta è ottenuto mediante composizione di Web Service e componenti locali ai due WfMS. I primi fanno uso di algoritmi di data-mining forniti dalle librerie del software *Weka*, un popolare tool, open-source e multipiattaforma, per il *data-mining* scritto interamente in Java[21].

La figura seguente illustra l'esperimento creato utilizzando Triana.

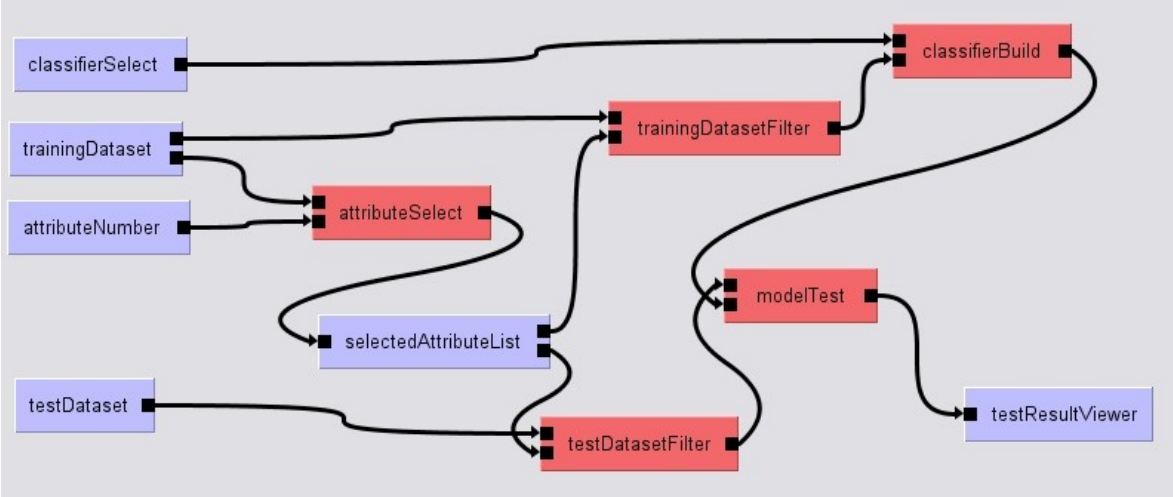

Figura 37: Esperimento di classificazione di attributi genomici mediante workflow composto con Triana: i componenti in celeste sono i componenti locali all'ambiente, in rosso i componenti distribuiti, ovvero i Web Service dislocati su macchine remote

La figura seguente illustra lo stesso esperimento realizzato con Taverna.

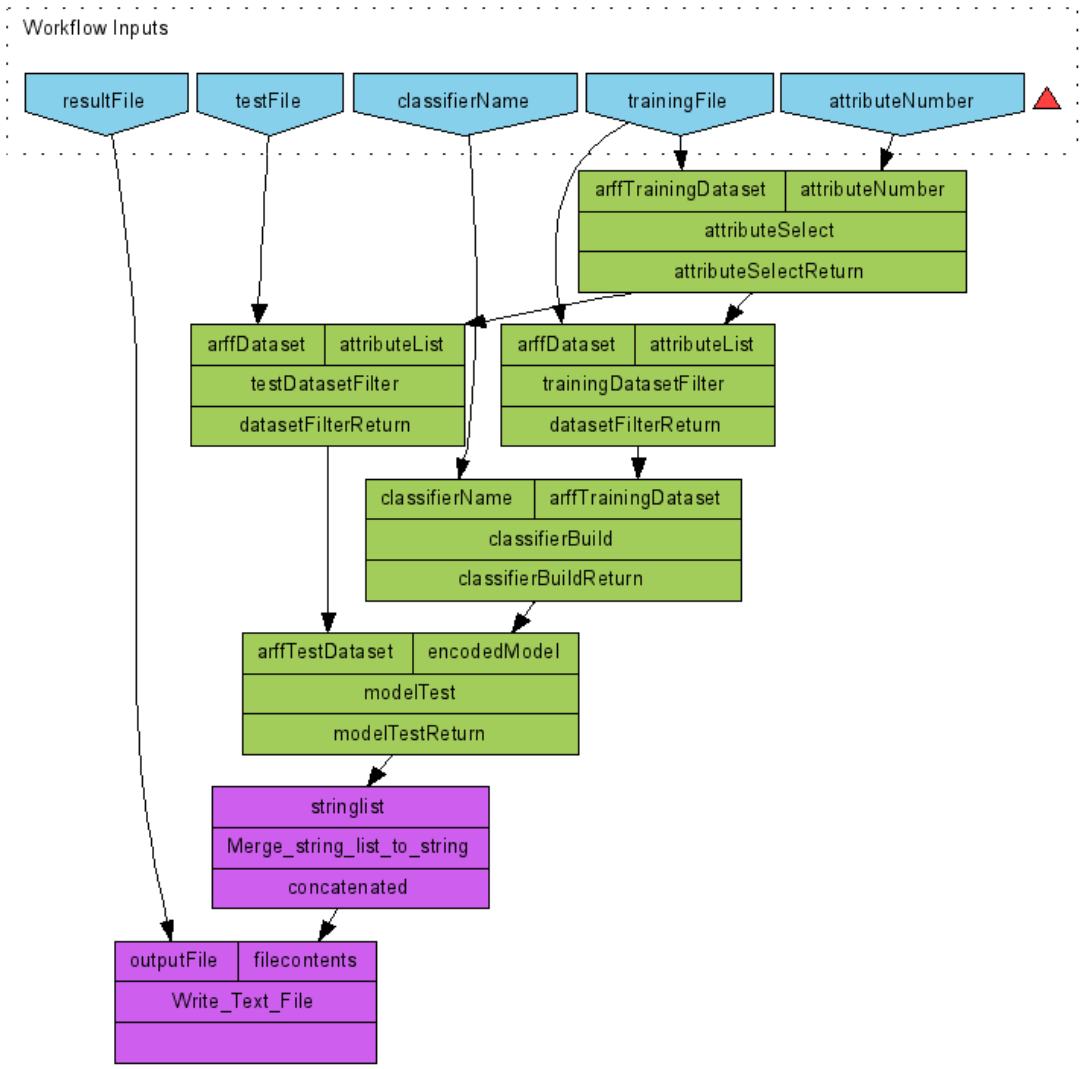

Figura 38: Esperimento di classificazione di attributi genomici mediante workflow composto con Taverna: i componenti in violetto sono i componenti locali all'ambiente, in verde i componenti distribuiti, ovvero i Web Service dislocati su macchine remote; in figura sono messi in evidenza anche le operazioni invocate

I due workflow definiscono, in input:

- il *trainingDataset* mediante il *trainingFile*, che rappresenta l'insieme di dati utilizzato per l'addestramento del modello di classificatore;

- il *testDataset* mediante il *testFile*, che rappresenta l'insieme di dati utilizzato per il test e la validazione del modello creato;

- l'*attributeNumber*, il numero di attributi genomici da utilizzare nella costruzione del modello;

- *classifierName*, nome dell'algoritmo di classificazione da utilizzare, mediante notazione puntata messo a disposizione dalle libreria di Weka, per esempio: *weka.classifiers.lazy.IB1*;

e in output i risultati dell'esperimento sotto la forma di un report dell'errore di classificazione ottenuto mediante l'esecuzione del workflow e specifica dei parametri di input richiesti.

I Web Service coinvolti, forniscono le seguenti funzionalità:

- *AttributeSelect*: esegue una feature selection sul dataset fornito in input e restituisce una lista contenente gli attributi ritenuti più significativi basando la rilevanza sul *ranking chi-quadro (χ2)*. Il numero massimo di attributi per la feature selection viene specificato mediante l'input attributeNumber, come illustrato in precedenza;

- *DatasetFilter:* esegue un filtraggio del dataset, passato come parametro in input, eliminando tutti gli attributi non contenuti nella *attributeList* e restituendo il nuovo dataset così ottenuto;
- *ClassifierBuild*: costruisce un algoritmo di classificazione specificato dal

parametro di input *classifierName* ed utilizzando il *trainingDataset* per l'addestramento del classificatore;

- *ModelTest*: Effettua il test del (modello di) classificatore, specificato in input, utilizzando il *testDataset* come insieme di dati di test. In output fornisce la matrice di confusione o altre misure dell'accuratezza del modello.

La realizzazione di questo esperimento mediante workflow mette in evidenza le capacità di entrambi i Workflow Management System di mettere a disposizione dell'utente i principali costrutti teorici esaminati nel Capitolo *3. Sistemi di Workflow*, in particolare:

- *Sequential Routing*: per esempio la sequenza di operazioni: *modelTest, testResultViewer* nel workflow creato con Triana e la sequenza: *modelTest, Merge_string_list_to_string, Write_Text_File* nel workflow creato con Taverna;

- *Parallel Routing:* in un punto particolare dei due workflow (*selectedAttributeList*) ha origine un *AND-Split* che sfocia in un Parallel routing; in quel punto, infatti inizia un fork di thread i quali eseguono, parallelamente, le seguenti sequenze di operazioni: *trainingDatasetFilter, classifierBuild* e *testDatasetFilter*. Un *AND-Join*, rappresentato dal processore *modelTest*, effettua una convergenza (Join) dei due flussi di esecuzione parallela verso la ripresa dell'elaborazione sequenziale;

- *AND-Split:* dopo l'esecuzione del processore *selectedAttributeList*;

- *AND-Join:* prima dell'esecuzione del processore *modelTest*;

La figura seguente illustra i risultati ottenuti dall'esecuzione del workflow in entrambi i WfMS per un *attributeNumber* pari a *10*.

```
--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  4  0  0  0  1  0  1 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  1  0 16  0  4  0  1 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  4  0  8  3 12  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  1  0  0  0 26 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.667     0.047      0.444     0.667     0.533    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  0.727     0.1        0.64      0.727     0.681    Hyperdip>50
  1         0.028      0.667     1         0.8      MLL
  0.444     0.059      0.706     0.444     0.545    OTHERS
  1         0          1         1         1        T-ALL
  0.963     0.024      0.929     0.963     0.945    TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances          88               78.5714 %
Incorrectly Classified Instances        24               21.4286 %
Kappa statistic                          0.7407
Mean absolute error                      0.0612
Root mean squared error                  0.2474
Relative absolute error                 26.2115 %
Root relative squared error             72.5071 %
Total Number of Instances              112
```

Figura 39: I risultati dell'esperimento effettuato vengono sia scritti su file in locale, sia mostrato a video come richiesto dai due workflow, creati ed eseguiti in Triana e Taverna, per un attributeNumber uguale a 10

Nel workflow realizzato con Taverna (Figura 38), è semplice realizzare un esperimento ripetuto per vari valori dell'*attributeNumber*, semplicemente sfruttando la caratteristica di *implicit loop* del WfMS in questione, ossia, fornendo

come valore in input per questo parametro una lista di numeri, anziché uno solo.

La figura seguente illustra i risultati dell'esperimento ripetuto fornendo, come input al workflow in Taverna, l'*attributeNumber* come lista di valori. Il diagramma illustra la variazione dell'errore di classificazione all'aumentare delle *features* selezionate. In Appendice A sono riportati i risultati completi dell'esperimento.

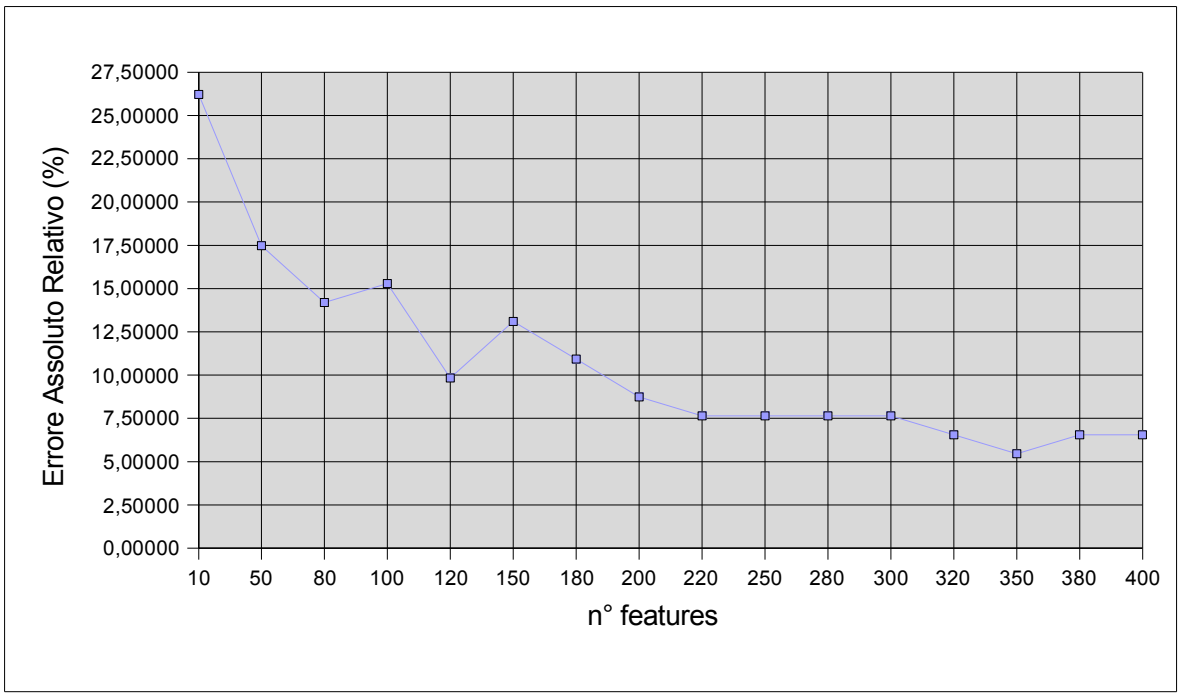

Figura 40: Risultati per l'esperimento ripetuto facendo uso dell'implicit loop di Taverna, in input la lista di valori 10, 50, 80, 100, 120, 150, 180, 200, 220, 250, 280, 300, 320, 350, 380, 400 come attributeNumber, ovvero il numero di feature da selezionare di volta in volta.

Con l'esperimento precedente si è ottenuto un grafico il quale illustra l'andamento dell'errore nel test del classificatore costruito (mediante algoritmo IB1) variando di volta in volta il numero di feature selezionate. Si evince che, relativamente alla lista di numeri fornita in input al workflow, la scelta migliore del numero di attributi, e cioè la scelta risultante in una migliore accuratezza dei

risultati, ricade sul *350*.

Per ottenere l'esperimento ripetuto in Triana, è stato necessario implementare un componente locale, scritto in linguaggio Java e facente uso delle API messe a disposizione per gli sviluppatori dallo stesso WfMS, chiamato *F2BTIterator*, il quale, al suo interno, mediante loop di tipo *while* consente di ripetere l'esperimento con i valori passati nel parametro *inputList*.

La figura seguente illustra il workflow iterativo così ottenuto in Triana.

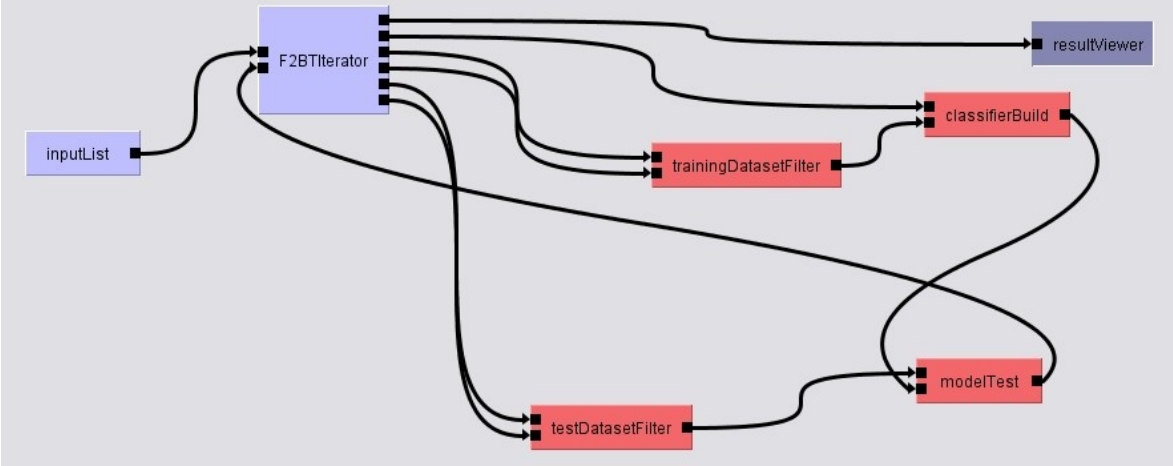

Figura 41: Il nuovo workflow, realizzato con Triana, per ripetere ciclicamente l'esperimento con una lista di attributeNumber passata in input mediante il parametro inputList

5.2. Un possibile scenario applicativo

Esperimenti del tipo di quello appena presentato stanno diventando sempre più diffusi in ambito scientifico ed in particolare modo quelli applicati alla bioinformatica, dove, spesso, come abbiamo visto nell'esperimento presentato, essi consistono in una applicazione ripetitiva e ciclica di tecniche di data-mining a modelli e basi di dati fisicamente dislocati e di dimensioni rilevanti. Attraverso l'unione di architetture orientate ai servizi, Web Service e sistemi di orchestrazione di questi ultimi, mediante sistemi di workflow, si prospettano interessanti e reali scenari applicativi in tutte le branche della bioinformatica (e dell'*e-Science* in generale), dove, con l'unione degli intenti e la condivisione delle tecniche, delle conoscenze e competenze intellettive dei ricercatori e, non per ultimi, dei dati, possono essere raggiunti i risultati sperati nella diagnosi delle malattie e lo studio, in genere, delle correlazioni genomiche.

Uno scenario verosimile, applicabile e sostenuto dalla presenza di svariati progetti in corso, ampiamente documentati nella letteratura scientifica ([15][28][29] tanto per citarne qualcuno), scaturisce dall'analisi della tendenza di queste sperimentazioni collaborative e distribuite e può essere rappresentato come nella figura seguente.

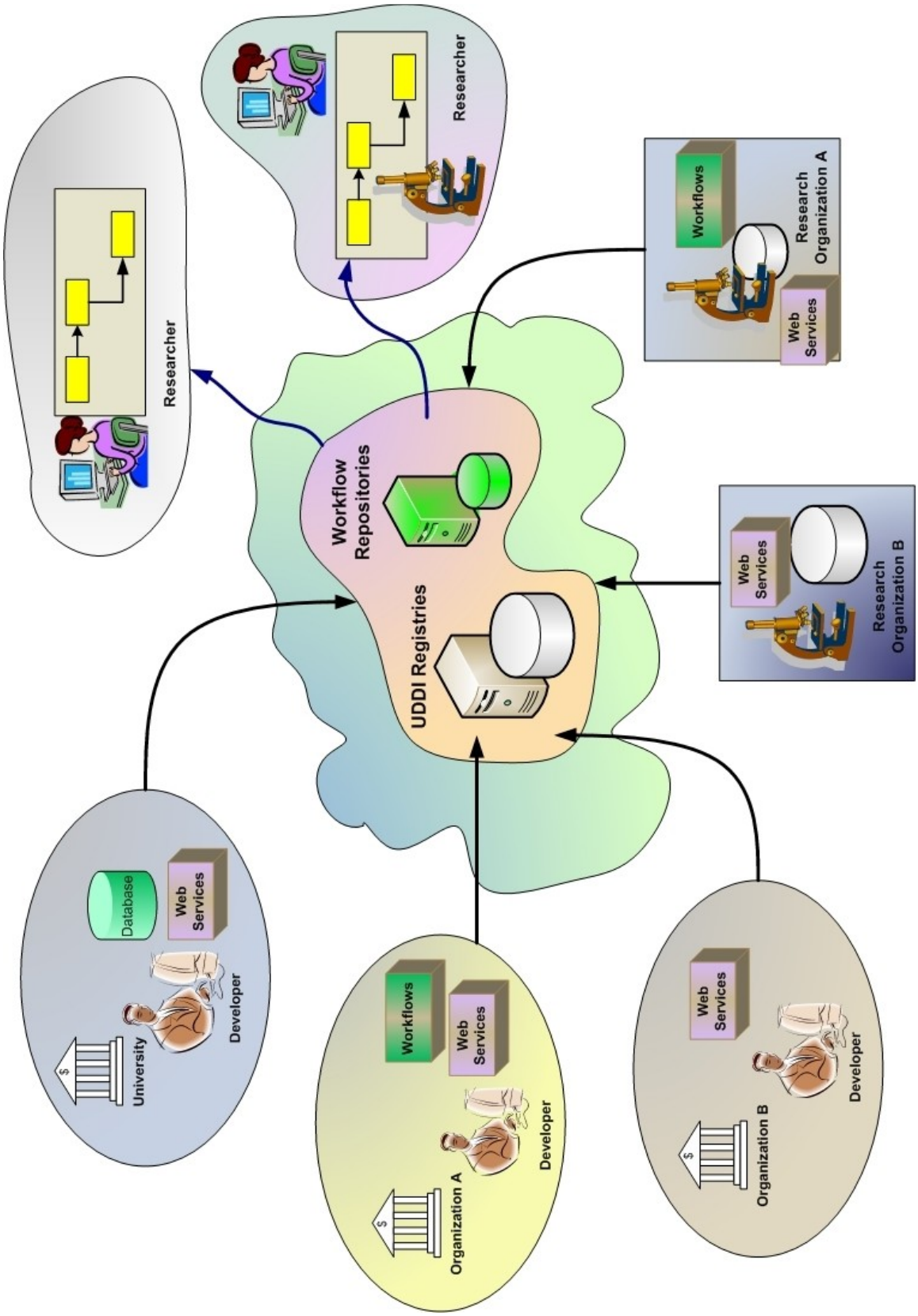

Figura 42: Scenario collaborativo per esperimenti scientifici, reso possibile dall'utilizzo congiunto di SOA, Web Service e di Workflow Management System

La figura precedente illustra uno scenario collaborativo distribuito per la realizzazione di esperimenti scientifici. Come si può vedere, diverse organizzazioni, quali Università, centri di ricerca, organizzazioni scientifiche, sviluppano una molteplicità di servizi, algoritmi di elaborazione di dati, algoritmi di data-mining e utilità di fruizione dei dati, tutti esposti come Web Service, descritti mediante WSDL e resi disponibili in registri UDDI; allo stesso modo esse possono sviluppare direttamente dei workflow che realizzano determinati esperimenti (anch'essi distribuiti o meno) mediante composizione di servizi e anche questa tipologia di risorsa viene condivisa e resa pubblica: o mediante creazione di Web Service (per esempio Triana, come abbiamo visto, permette la pubblicazione di un workflow direttamente come un Web Service su di un UDDI Registry) o mediante apposito repository o portale Web [15]. Ancora, diverse organizzazioni mettono a disposizione, per la consultazione, grosse basi di dati contenenti dati relativi ai domini scientifici trattati (per esempio dati genomici per esperimenti di bioinformatica), anch'esse consultabili mediante interrogazione di Web Service, appositamente creati, che permettono di avere determinate viste sui dati da remoto. In questa sorta di distribuzione e pubblicazione di risorse, qualunque utente-ricercatore, interessato alla realizzazione di un determinato esperimento, può comporre il proprio workflow, attraverso un Workflow Management System del tipo di quelli esaminati in questo lavoro, sia facendo un discovery di ogni singolo Web Service, mediante i registri UDDI, e quindi componendoli, sia integrando workflow già disponibili includendoli nel loro esperimento come sub-workflow.

In questo modo la ricerca in determinanti campi può sfruttare la carta vincente rappresentata da una reale architettura di servizi che permette la collaborazione tra organizzazioni scientifiche e la condivisione di risorse, sia costituite da preziosi dati sia da non meno preziose risorse di calcolo.

6. Conclusioni

Con l'introduzione, e la progressiva realizzazione, delle Service Oriented Architecture, si assiste ad una sostanziale crescita del livello di astrazione delle applicazioni distribuite: non più solamente sistemi eterogenei che, in qualche modo, comunicano tra loro, ma piuttosto sistemi e *applicazioni distribuite* che forniscono *servizi,* fruibili attraverso l'utilizzo di tecnologie standardizzate per la pubblicazione, per il *discovery* in rete da parte dei *service consumer* e per lo scambio di messaggi. Una parte rilevante a questo processo di standardizzazione, e un grosso ruolo nella integrazione tra i sistemi, sono ricoperti dalla tecnologia dei Web Service che può essere, e a ragione, ritenuta come la tecnologia abilitante per la reale costruzione e definitiva adozione della Service Oriented Architecture. Come scritto nell'introduzione a questo lavoro, a partire da questa nuova "tendenza" delle tecnologie per i sistemi distribuiti, trovano linfa nuova e finalmente fattibile applicazione, i concetti teorici e pratici dei sistemi di *workflow,* teorizzati e definiti formalmente a partire dagli anni '70. Essi permettono di incrementare il livello di astrazione delle applicazioni, della loro progettazione e del loro modello di programmazione, fornendo uno strumento, utilizzabile anche da utenti non programmatori, per la costruzione di applicazioni distribuite semplicemente mediante composizione, semplice o complessa, di applicazioni remote e locali, funzionalità, servizi e definizione dei flussi di dati e di controllo tra di esse. A dimostrazione di questi fatti, sono stati presentati due *Workflow Management System, Triana* e *Taverna*, orientati all'ambito scientifico (*e-Science*), liberamente fruibili, e, dopo aver descritto le funzionalità da essi messe a disposizione, sono stati utilizzati per la realizzazione di un esperimento di bioinformatica facente uso di algoritmi di data-mining su basi di dati di tipo genomico. L'esperimento è stato condotto creando due workflow, ottenuti

componendo processori remoti, rappresentati da Web Service realizzati in Java, i quali espongono servizi per l'elaborazione di dati genomici.

L'esperimento, oltre ad aver mostrato la possibilità di una reale applicazione dei sistemi di workflow inseriti in un nuovo contesto di applicazioni distribuite, è stato anche il pretesto per esaminare lo stato dell'implementazione dei concetti, teorizzati sinora, per la definizione dei sistemi di workflow, i modelli di applicazione, i limiti e le problematiche affrontate e da affrontare, in particolare, da due validi progetti di creazione di un Workflow Management System, quali Taverna e Triana. In questo lavoro non sono state trattate le problematiche relative alla definizione di una s*emantica* sia dei servizi che dei workflow; Taverna in particolare, offre la possibilità di annotare semanticamente (mediante documenti *RDF*) i componenti utilizzati per definire il processo di workflow in composizione. Attualmente vi sono diversi filoni di ricerca per la definizione di *ontologie* sia per i Web Service, sia per i workflow, per la verifica, il discovery e la composizione di queste tecnologie [30][31][32]. Alla luce di quanto detto, possiamo concludere dicendo che la strada verso la realizzazione di applicazioni distribuite orientate ai servizi è stata oramai aperta e segnata e i sistemi di workflow possono veramente rappresentare una carta vincente per la realizzazione di sistemi complessi con una loro concreta applicazione, rivolta a domini che vanno da processi di business generici sino a processi di sperimentazione e ricerca scientifica, come nel caso della bioinformatica.

7. Bibliografia

[1] Hao He, *What is Service-Oriented Architecture,* XML.com, 2003

[2] Ethan Cerami, *Web Services Essentials. Distribuited Applications with XML-RPC, SOAP, UDDI & WSDL*, O'REILLY

[3] Gottshalk K., Graham S., Kreger H., Snell, J., *Introduction to Web Service Architecture*, IBM Systems Journal, vol.41, No. 2, 2002

[4] Pernici B., Plebani P., *Un'introduzione ragionata al mondo dei Web Service*, Mondo Digitale, n.1, Marzo 2004

[5] Leymann F., Roller D., Thatte S.: *Goals of the BPEL4WS Specification*, xml.coverpages.org/BPEL4WS-DesignGoals.pdf

[6] Thatte S. (Editor): *Business Process Execution Language for Web Services. Version 1.1*, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[7] Ellis Clarence A., *Workflow Technology*, Computer Supported Cooperative Work, John Wiley & Sons Ltd, 1999

[8] Bull Corporation, *Flowpath Functional Specification,* Bull S. A. Paris, France, 1992

[9] Workflow Management Coalition, *The Workflow Reference Model*, 1995

[10] Workflow Management Coalition, *Terminology & Glossary,* 1999

[11] W3C, *Web Services Description Language (WSDL), 1.1*, 2001, http://www.w3.org/TR/wsdl

[12] OASIS, *UDDI Executive Overview: Enabling Service-Oriented Architecture*, 2004, http://uddi.org/pubs/uddi-exec-wp.pdf

[13] W3C, *SOAP Version 1.2 Part 1: Messaging Framework,* 2003, http://www.w3.org/TR/soap12-part1/

[14] BeanShell, http://www.beanshell.org/

[15] Bosin A., Dessì N., Fugini M. G., Liberati D., Pes B., *The Future of Portals in e-Science*

[16] Bioinformatics Web Services, Università di Cagliari, http://www.dsf.unica.it/~andrea/webservices.html

[17] Apache Axis Web Services, http://ws.apache.org/axis/

[18] Apache Tomcat, http://tomcat.apache.org/

[19] Taverna Project, *Taverna User Manual*, http://taverna.sourceforge.net

[20] Bosin A, Dessì N, Liberati D, Pes B., *Learning Bayesian Classifiers from Gene-Expression MicroArray Data*, 6th International Workshop, WILF2005, Lecture

Notes in Computer Science, vol. 3849, pp. 297-304, Springer-Verlag

[21] Weka, *Weka 3: Data Mining Software in Java*, http://www.cs.waikato.ac.nz/~ml/weka/index.html

[22] Baldoni, Marchetti, Tucci-Piergiovanni, *Appunti Integrativi su Sistemi Distribuiti*, 2001-2002, Università degli Studi di Roma "La Sapienza"

[23] Channabasavaiah K., Holley K., Tuggle Edward M., *Migrating to a service-oriented architecture*, IBM White Paper, Aprile 2004

[24] The Triana Project, *Triana User Guide*, www.trianacode.org/docs/index.html

[25] GridLab, *Triana Workflow Specification*, www.gridlab.org/WorkPackages/wp-3/D3.3.pdf

[26] Kent Ridge Bio-medical Data Set Repository, *Classification, Subtype Discovery, and Prediction of Outcome in Pediatric Acute Lymphoblastic Leukemia by Gene Expression Profiling,* Cancer Cell, 1:133-143, March, 2002, http://sdmc.lit.org.sg/GEDatasets/Datasets.html#Leukemia

[27] Web Service Interoperability, *WS-I*, http://www.ws-i.org/

[28] Gaizauskas R., Davis N., Demetriou G., Guo Y., Roberts I., *Integrating Biomedical Text Mining Services into a Distribuited Workflow Environment*

[29] Digiampietri L. A., Medeiros C. B., Setubal J. C., *A framework based on Web Service orchestration for bioinformatics workflow management,* Genetics and Molecular

Research, 2005

[30] Mikko Laukkanen, Heikki Helin, *Composing Workflows of Semantic Web Services*

[31] A. Ankolenkar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne, and K. Sycara, *DAML-S: Web Service Description for the Semantic Web,* in Proceedings of The First International Semantic Web Conference (ISWC), Sardinia (Italy), June 2002.

[32] Narayanan S., McIlraith S., *Simulation, Verification and Automated Composition of Web Services*, WWW2002, Hawaii, USA

Riferimenti sparsi

Wikipedia, *Service Oriented Architecture*, http://en.wikipedia.org/wiki/Service-orien ted_architecture

Java Technology, http://java.sun.com

Mokabyte, *Service Oriented Architecture: dalla teoria alla pratica*, mokabyte n° 100, ottobre 2005, http://www2.mokabyte.it/cms/article.run?articleId=i_4BR-KK6-OGH-R5B_7f000001_11024688_5b5b3c3e

Christoph Schittko, *Web Service Orchestration with BPEL,* Momentum Software

Chris Peltz, *Web Service Orchestration, a review of emerging technologies, tools, and standards,* Hewlett Packard, Co.

Sriram Krishnan, Kim K. Baldridge, Jerry P. Greenberg, Brent Stearn and Karan Bhatia, *An End-to-end Web Services-based Infrastructure for Biomedical Applications*

Ian Wang, *WSRF and Triana,* Tutorial, 2006, www.trianacode.org

Gopalan Suresh Raj, Binod PG, Keith Babo, Palkovich R., *Implementing Service-Oriented Architectures (SOA) with the Java EE 5 SDK*, Sun Microsystems, maggio 2006

Netbeans Software, www.netbeans.org

W3C, *Web Services Activity*, http://www.w3.org/2002/ws

Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, *Quality Driven Web Services Composition*, WWW2003, Budapest

F. Leymann., *Web Service Flow Language (WSFL1.0), May 2001,* http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Kelly P. M., Coddington P. D., Wendelborn A. L., *A Simplified Approach to Web Service Development*, Fourth Australasian Workshop on Grid Computing and e-Research (AusGrid 2006), Australia

Fielding R. T., *Architectural Styles and the Design of Network-based Software Architectures*, Università della California – Irvine, USA, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

W3C, *Web Service Semantics, WSDL-S*, W3C Member Submission 7 November 2005, http://www.w3.org/Submission/WSDL-S/

W3C, *XML Schema Part 0: Primer Second Edition*, W3C Recommendation 28 October 2004, http://www.w3.org/TR/xmlschema-0/

8. APPENDICE A

In questa sezione sono riportati, per intero, i risultati ottenuti eseguendo in modo ripetuto l'esperimento di bioinformatica mediante workflow, come illustrato nel paragrafo 5.1. L'esperimento è stato ripetuto effettuando una *feature selection* con un numero variabile di attributi genomici; per l'esattezza il numero di attributi ha seguito la sequenza: 10, 50, 80, 100, 120, 150, 180, 200, 220, 250, 280, 300, 320, 350, 380, 400.

```
--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  4  0  0  0  1  0  1 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  1  0 16  0  4  0  1 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  4  0  8  3 12  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  1  0  0  0 26 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.667     0.047      0.444     0.667     0.533    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  0.727     0.1        0.64      0.727     0.681    Hyperdip>50
  1         0.028      0.667     1         0.8      MLL
  0.444     0.059      0.706     0.444     0.545    OTHERS
  1         0          1         1         1        T-ALL
  0.963     0.024      0.929     0.963     0.945    TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances          88               78.5714 %
Incorrectly Classified Instances        24               21.4286 %
Kappa statistic                          0.7407
Mean absolute error                      0.0612
Root mean squared error                  0.2474
Relative absolute error                 26.2115 % (10)
Root relative squared error             72.5071 %
Total Number of Instances              112
```

```
--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  2  0  0  0  3  0  1 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 21  0  1  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  3  0  5  2 16  0  1 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.333     0.028      0.4       0.333     0.364    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  0.955     0.056      0.808     0.955     0.875    Hyperdip>50
  1         0.019      0.75      1         0.857    MLL
  0.593     0.047      0.8       0.593     0.681    OTHERS
  1         0          1         1         1        T-ALL
  1         0.024      0.931     1         0.964    TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances          96               85.7143 %
Incorrectly Classified Instances        16               14.2857 %
Kappa statistic                          0.8254
Mean absolute error                      0.0408
Root mean squared error                  0.202
Relative absolute error                 17.4743 % (50)
Root relative squared error             59.2018 %
Total Number of Instances              112




--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  3  0  0  0  3  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 21  0  1  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  2  0  6  1 18  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1
```

```
--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.5       0.019      0.6       0.5       0.545    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  0.955     0.067      0.778     0.955     0.857    Hyperdip>50
  1         0.009      0.857     1         0.923    MLL
  0.667     0.047      0.818     0.667     0.735    OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances          99               88.3929 %
Incorrectly Classified Instances        13               11.6071 %
Kappa statistic                          0.8579
Mean absolute error                      0.0332
Root mean squared error                  0.1821
Relative absolute error                 14.1979 % (80)
Root relative squared error             53.3638 %
Total Number of Instances              112




--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  3  0  0  0  3  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 19  0  3  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  2  0  3  3 19  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.5       0.019      0.6       0.5       0.545    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  0.864     0.033      0.864     0.864     0.864    Hyperdip>50
  1         0.028      0.667     1         0.8      MLL
  0.704     0.071      0.76      0.704     0.731    OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------
```

```
Correctly Classified Instances          98               87.5    %
Incorrectly Classified Instances        14               12.5    %
Kappa statistic                          0.8473
Mean absolute error                      0.0357
Root mean squared error                  0.189
Relative absolute error                 15.29   % (100)
Root relative squared error             55.3782 %
Total Number of Instances              112




--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  5  0  0  0  1  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 22  0  0  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  3  0  3  2 19  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.833     0.028      0.625     0.833     0.714    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  1         0.033      0.88      1         0.936    Hyperdip>50
  1         0.019      0.75      1         0.857    MLL
  0.704     0.012      0.95      0.704     0.809    OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances         103               91.9643 %
Incorrectly Classified Instances         9                8.0357 %
Kappa statistic                          0.9024
Mean absolute error                      0.023
Root mean squared error                  0.1515
Relative absolute error                  9.8293 %(120)
Root relative squared error             44.4013 %
Total Number of Instances              112
```

```
--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  3  0  0  0  3  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 22  0  0  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  4  0  3  2 18  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.5       0.038      0.429     0.5       0.462    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  1         0.033      0.88      1         0.936    Hyperdip>50
  1         0.019      0.75      1         0.857    MLL
  0.667     0.035      0.857     0.667     0.75     OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances         100               89.2857 %
Incorrectly Classified Instances        12               10.7143 %
Kappa statistic                          0.8695
Mean absolute error                      0.0306
Root mean squared error                  0.175
Relative absolute error                 13.1057 % (150)
Root relative squared error             51.2702 %
Total Number of Instances              112




--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  4  0  0  0  2  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 22  0  0  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  4  0  4  0 19  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1
```

```
--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.667     0.038      0.5       0.667     0.571    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  1         0.044      0.846     1         0.917    Hyperdip>50
  1         0          1         1         1        MLL
  0.704     0.024      0.905     0.704     0.792    OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances         102               91.0714 %
Incorrectly Classified Instances        10                8.9286 %
Kappa statistic                          0.8911
Mean absolute error                      0.0255
Root mean squared error                  0.1597
Relative absolute error                 10.9214 % (180)
Root relative squared error             46.8031 %
Total Number of Instances              112




--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  6  0  0  0  0  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 21  0  1  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  3  0  2  2 20  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  1         0.028      0.667     1         0.8      BCR-ABL
  1         0          1         1         1        E2A-PBX1
  0.955     0.022      0.913     0.955     0.933    Hyperdip>50
  1         0.019      0.75      1         0.857    MLL
  0.741     0.012      0.952     0.741     0.833    OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------
```

```
Correctly Classified Instances         104               92.8571 %
Incorrectly Classified Instances         8                7.1429 %
Kappa statistic                          0.9133
Mean absolute error                      0.0204
Root mean squared error                  0.1429
Relative absolute error                  8.7372 % (200)
Root relative squared error             41.862  %
Total Number of Instances              112




--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  6  0  0  0  0  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 21  0  1  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  3  0  3  0 21  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  1         0.028      0.667     1         0.8      BCR-ABL
  1         0          1         1         1        E2A-PBX1
  0.955     0.033      0.875     0.955     0.913    Hyperdip>50
  1         0          1         1         1        MLL
  0.778     0.012      0.955     0.778     0.857    OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances         105               93.75   %
Incorrectly Classified Instances         7                6.25   %
Kappa statistic                          0.9239
Mean absolute error                      0.0179
Root mean squared error                  0.1336
Relative absolute error                  7.645  % (220)
Root relative squared error             39.1583 %
Total Number of Instances              112
```

```
--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  6  0  0  0  0  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 21  0  1  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  3  0  3  0 21  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  1         0.028      0.667     1         0.8      BCR-ABL
  1         0          1         1         1        E2A-PBX1
  0.955     0.033      0.875     0.955     0.913    Hyperdip>50
  1         0          1         1         1        MLL
  0.778     0.012      0.955     0.778     0.857    OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances         105               93.75   %
Incorrectly Classified Instances         7                6.25   %
Kappa statistic                          0.9239
Mean absolute error                      0.0179
Root mean squared error                  0.1336
Relative absolute error                  7.645  % (250)
Root relative squared error             39.1583 %
Total Number of Instances              112




--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  5  0  1  0  0  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 21  0  1  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  2  0  3  0 22  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1
```

```
--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.833     0.019      0.714     0.833     0.769    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  0.955     0.044      0.84      0.955     0.894    Hyperdip>50
  1         0          1         1         1        MLL
  0.815     0.012      0.957     0.815     0.88     OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances         105               93.75   %
Incorrectly Classified Instances         7                6.25   %
Kappa statistic                          0.9236
Mean absolute error                      0.0179
Root mean squared error                  0.1336
Relative absolute error                  7.645  % (280)
Root relative squared error             39.1583 %
Total Number of Instances              112




--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  4  0  0  0  2  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 21  0  1  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  1  0  3  0 23  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.667     0.009      0.8       0.667     0.727    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  0.955     0.033      0.875     0.955     0.913    Hyperdip>50
  1         0          1         1         1        MLL
  0.852     0.035      0.885     0.852     0.868    OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------
```

```
Correctly Classified Instances         105               93.75   %
Incorrectly Classified Instances         7                6.25   %
Kappa statistic                          0.9232
Mean absolute error                      0.0179
Root mean squared error                  0.1336
Relative absolute error                  7.645  % (300)
Root relative squared error             39.1583 %
Total Number of Instances              112




--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  5  0  0  0  1  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 22  0  0  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  2  0  3  0 22  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.833     0.019      0.714     0.833     0.769    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  1         0.033      0.88      1         0.936    Hyperdip>50
  1         0          1         1         1        MLL
  0.815     0.012      0.957     0.815     0.88     OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances         106               94.6429 %
Incorrectly Classified Instances         6                5.3571 %
Kappa statistic                          0.9345
Mean absolute error                      0.0153
Root mean squared error                  0.1237
Relative absolute error                  6.5529 % (320)
Root relative squared error             36.2535 %
Total Number of Instances              112
```

```
--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  5  0  0  0  1  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 22  0  0  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  1  0  3  0 23  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.833     0.009      0.833     0.833     0.833    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  1         0.033      0.88      1         0.936    Hyperdip>50
  1         0          1         1         1        MLL
  0.852     0.012      0.958     0.852     0.902    OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances         107               95.5357 %
Incorrectly Classified Instances         5                4.4643 %
Kappa statistic                          0.9453
Mean absolute error                      0.0128
Root mean squared error                  0.1129
Relative absolute error                  5.4607 % (350)
Root relative squared error             33.0948 %
Total Number of Instances              112




--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  4  0  0  0  2  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 22  0  0  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  2  0  2  0 23  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1
```

```
--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.667     0.019      0.667     0.667     0.667    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  1         0.022      0.917     1         0.957    Hyperdip>50
  1         0          1         1         1        MLL
  0.852     0.024      0.92      0.852     0.885    OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------

Correctly Classified Instances         106               94.6429 %
Incorrectly Classified Instances         6                5.3571 %
Kappa statistic                          0.9343
Mean absolute error                      0.0153
Root mean squared error                  0.1237
Relative absolute error                  6.5529 % (380)
Root relative squared error             36.2535 %
Total Number of Instances              112




--------------CLASSIFICATORE------
IB1 classifier
--------------MATRICE-------------
=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
  4  0  0  0  2  0  0 |  a = BCR-ABL
  0  9  0  0  0  0  0 |  b = E2A-PBX1
  0  0 22  0  0  0  0 |  c = Hyperdip>50
  0  0  0  6  0  0  0 |  d = MLL
  2  0  2  0 23  0  0 |  e = OTHERS
  0  0  0  0  0 15  0 |  f = T-ALL
  0  0  0  0  0  0 27 |  g = TEL-AML1

--------------DETTAGLI------------
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.667     0.019      0.667     0.667     0.667    BCR-ABL
  1         0          1         1         1        E2A-PBX1
  1         0.022      0.917     1         0.957    Hyperdip>50
  1         0          1         1         1        MLL
  0.852     0.024      0.92      0.852     0.885    OTHERS
  1         0          1         1         1        T-ALL
  1         0          1         1         1        TEL-AML1

--------------SOMMARIO------------
```

Correctly Classified Instances	106	94.6429 %
Incorrectly Classified Instances	6	5.3571 %
Kappa statistic	0.9343	
Mean absolute error	0.0153	
Root mean squared error	0.1237	
Relative absolute error	6.5529 % (400)	
Root relative squared error	36.2535 %	
Total Number of Instances	112	