UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

tesi di laurea

Casi di studio sulla migrazione di applicazioni web verso servizi REST

Anno Accademico 2008/2009

relatore
Ch.mo prof. Porfirio Tramontana

candidato
Marco Chimenti
Matr. 534/1940

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

OBBIETTIVI DELLA TESI

- Studio della tecnologia REST
  - Tecniche e best practice per la realizzazione di risorse RESTlet
  - Principali differenze con SOAP

- Studio delle tecniche di migrazione da sistemi legacy verso SOA

- Proposta e valutazione di tecniche per la migrazione di Web application verso servizi REST
  - Attraverso casi di studio che hanno coinvolto la migrazione di due applicazioni.

SMART: Service Migration and Reuse Technique

- **Definisce una metodologia per supportare gli analisti a valutare la migrazione e il riuso di funzioni di un sistema legacy come servizi.**

  - ✓ **Vantaggi di una migrazione verso SOA:**
    Riutilizzo del sistema per ottenere un ritorno di investimento sfruttando le caratteristiche su cui si basa l'ambiente SOA. La migrazione, se gestita bene, potrà portare ad un ottimizzazione dei costi e a maggiore adattabilità e agilità del sistema

  - ❖ **Elementi che costituiscono SMART**
    - Processo SMART: viene utilizzato allo scopo di raccogliere informazioni sul servizio legacy, sui servizi candidati alla migrazione e sul target SOA;
    - Service Migration Interview Guide (SMIG): E' composto da più di 60 categorie di domande che permettono di ottenere informazioni circa il contesto di migrazione.
    - Modelli per artefatti: Fornisce dei modelli collaudati per raccogliere le informazioni derivanti dal processo SMART in modo ordinato (es. StakeHolder List)
    - SMART Tool: l'utilizzo di un tool rende più semplice la raccolta delle informazioni.

REST: REpresentational State Transfer

- **Serie di principi e stili architetturali per costruire sistemi di scambio dati client/server ereditando molte caratteristiche che hanno reso famoso il Web**

  - **Principi cardine:**
    - **Stateless**: Non esiste la gestione dello stato applicativo, ogni richiesta è a sé stante;
    - **Indicizzazione**: Ogni risorsa deve essere identificata univocamente da un proprio URI;
    - **Interfaccia constante e uniforme:** I metodi di accesso sono uguali per ogni tipo di risorsa e vengono definiti mediante i verbi HTTP.
    - **Orientata alle rappresentazioni:** Lo stato di una risorsa deve essere descritto mediante un tipo di rappresentazione che dovrà farsi carico di collegare le risorse tra di loro tramite link.

- **NOTA**

  L'uso di un'interfaccia constante ci costringe ad implementare per ogni risorsa al più operazioni di creazione, modifica, creazione e lettura. E a mappare ognuna di queste per mezzo di un apposito verbo http. In particolare:
  - PUT – Aggiornamento o creazione
  - POST – Creazione
  - DELETE – Cancellazione
  - GET - Lettura

REST vs SOAP

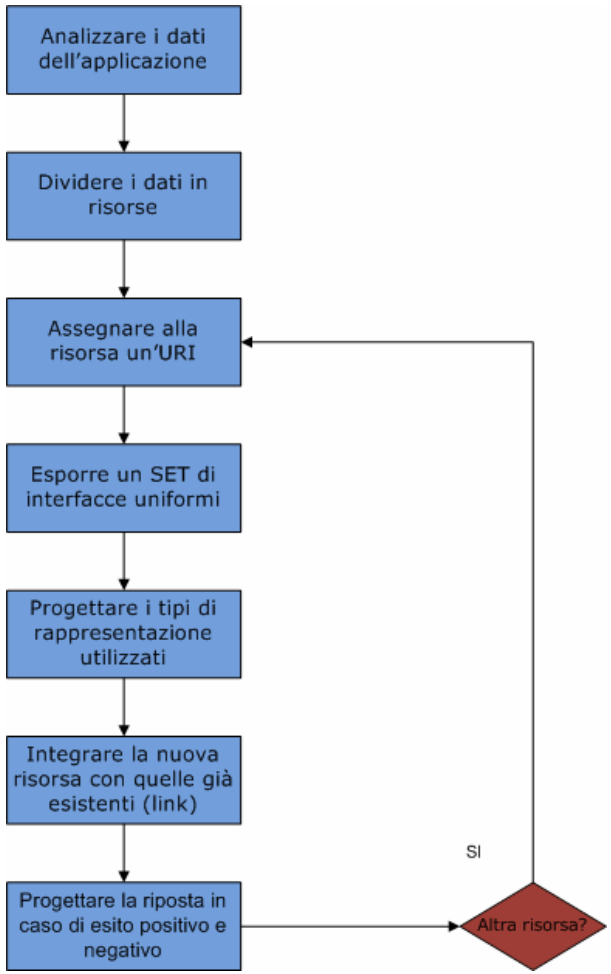
- **Vantaggi**
  - ✓ Semantica già descritta per ogni risorsa dai metodi HTTP che sono applicabili su di essa, evitando la necessità di ricorrere a documenti di descrizione;
  - ✓ Minore appesantimento dei messaggi dovuto alla mancanza di envelope SOAP;
  - ✓ Maggiore controllo sul tipo di operazione richiesta da parte di sistemi di sicurezza che possono riconoscere semplicemente guardando l'header http se l'operazione e sicura o meno;
  - ✓ Semplice da implementare perché basato su concetti già ampiamente affermati utilizzati;

- **Svantaggi**:
  - Manca il supporto standard al "type check" o per la gestione della sicurezza, ciò potrebbe rendere lo sviluppo di applicazioni complesse più difficile;
  - In alcuni casi i verbi HTTP non sono in grado di definire in maniera completa la semantica di un'operazione.

- **Diffusione**
  - ➢ Per capire quanto i web services con architettura REST si stiano diffondendo, basta fare riferimento alle parole di Jeff Barr, creatore di syndic8 e attualmente Amazon's chief web services evangelist. Barr affermò che Amazon offriva dei web services basati sia su soap che su REST, ma solo il 15% del loro traffico era SOAP.

Costruzione di una risorsa REST

Per la creazione di una generica risorsa RESTful abbiamo adottato questo tipo di metodologia:



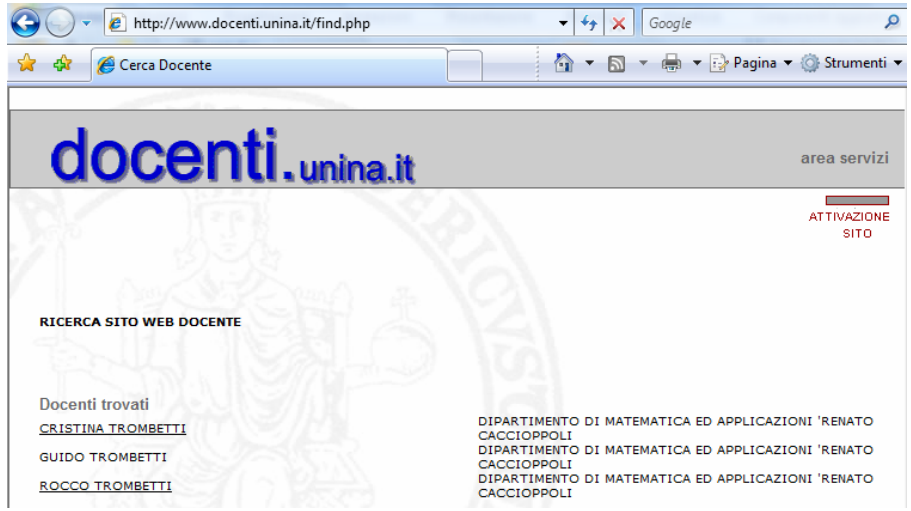
- Le prime due fasi ci aiutano nella costruzione del dominio dei dati e nell'individuazione delle risorse;
- La fase iterativa si applica ad ogni risorsa e serve per definirla completamente;

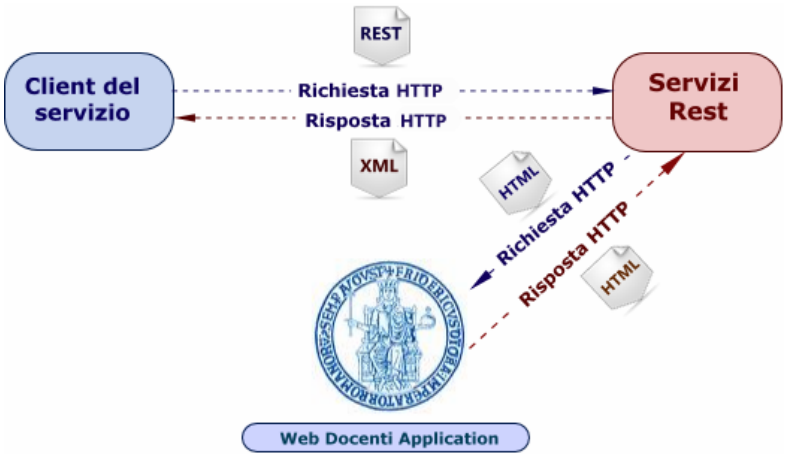
- La fase di divisione dei dati in risorse avviene in modo simile a come avviene nel mondo object-oriented cioè individuando i sostantivi dal documento di specifica. Ma a causa dell'utilizzo di un'interfaccia unificata le responsabilità della risorsa sono ridotte. Per questo si rende necessario individuare risorse associate per ampliare le azioni sulla risorsa.

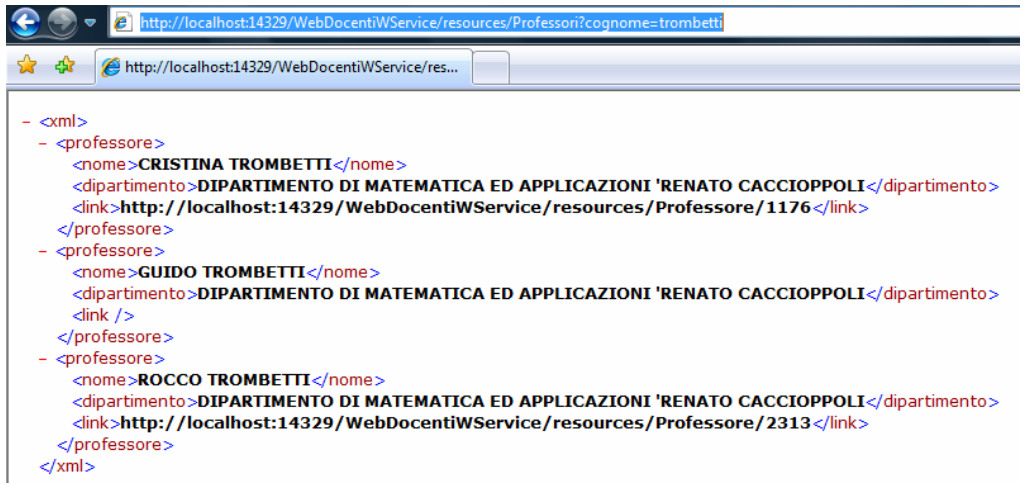
UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

Caso d'uso: Migrazione di una web application accessibile solo tramite interfaccia web

- **Nome Applicazione:** Web Docenti
- **Tipologia servizio da migrare:** Read – only
- **Meccanismo di controllo per accesso delle risorse:** Nessuno
- **Casi d'uso da migrare:** Lettura dati docenti e ricerca docenti per nome e/o cognome



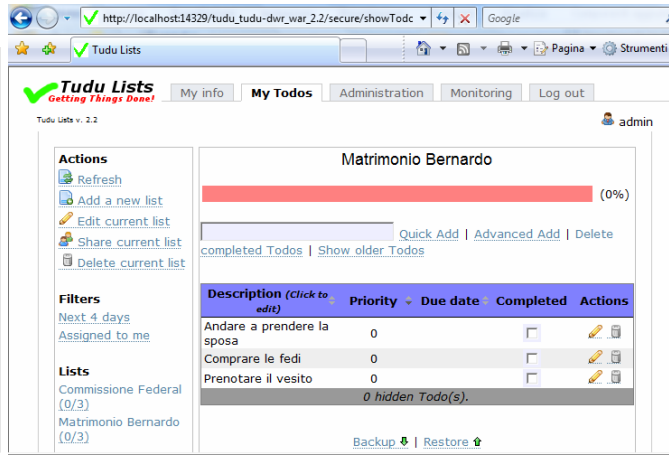




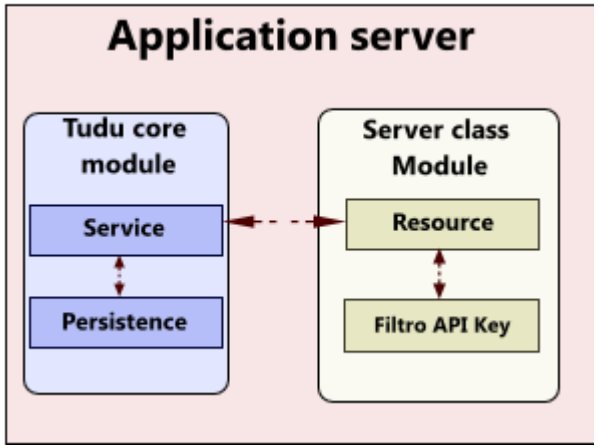
UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

Caso di studio: Migrazione di un'applicazione con accesso alle classi di Business

- **Nome applicazione da migrare:** Tudu
- **Tipologia del servizio da migrare:** Stateful
- **Meccanismo di controllo per l'accesso alle risorse:** API Key
- **Casi d'uso da migrare:** Elenco todo List, Inserimento nuova todo List; modifica, eliminazione e inserimento nuovo elemento todo;



- **Schema architetturale Tudu application**
  - Noi utilizzeremo il modulo CORE che contiene le classi per la persistenza dei dati e quelle che costituiscono il dominio

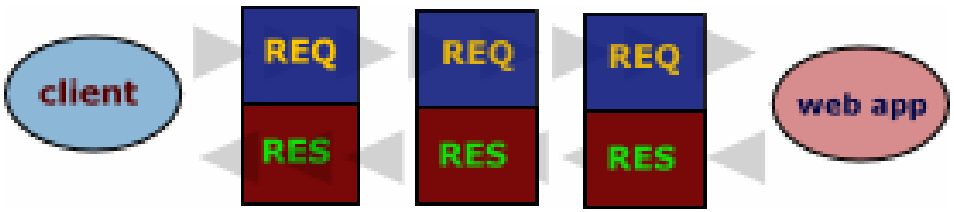


Problematica controllo accesso alle risorse: API Key

- L'api key è un codice univoco, solitamente alfanumerico, che accoppiato alla richiesta ad un web service ne permette la fruizione. Se abbinato ad un contatore che si incrementa ogni volta che viene utilizzato il servizio fornisce indicazione sull'utilizzo dello stesso da parte di un utente.

- E' possibile implementare un meccanismo per la gestione della API Key tramite i filtri, che vengono messi a disposizione l'ambiente Java. Questi sono utili ogni qual'volta ci sia bisogno di eseguire il Pre-processing o il Post-processing delle richieste.

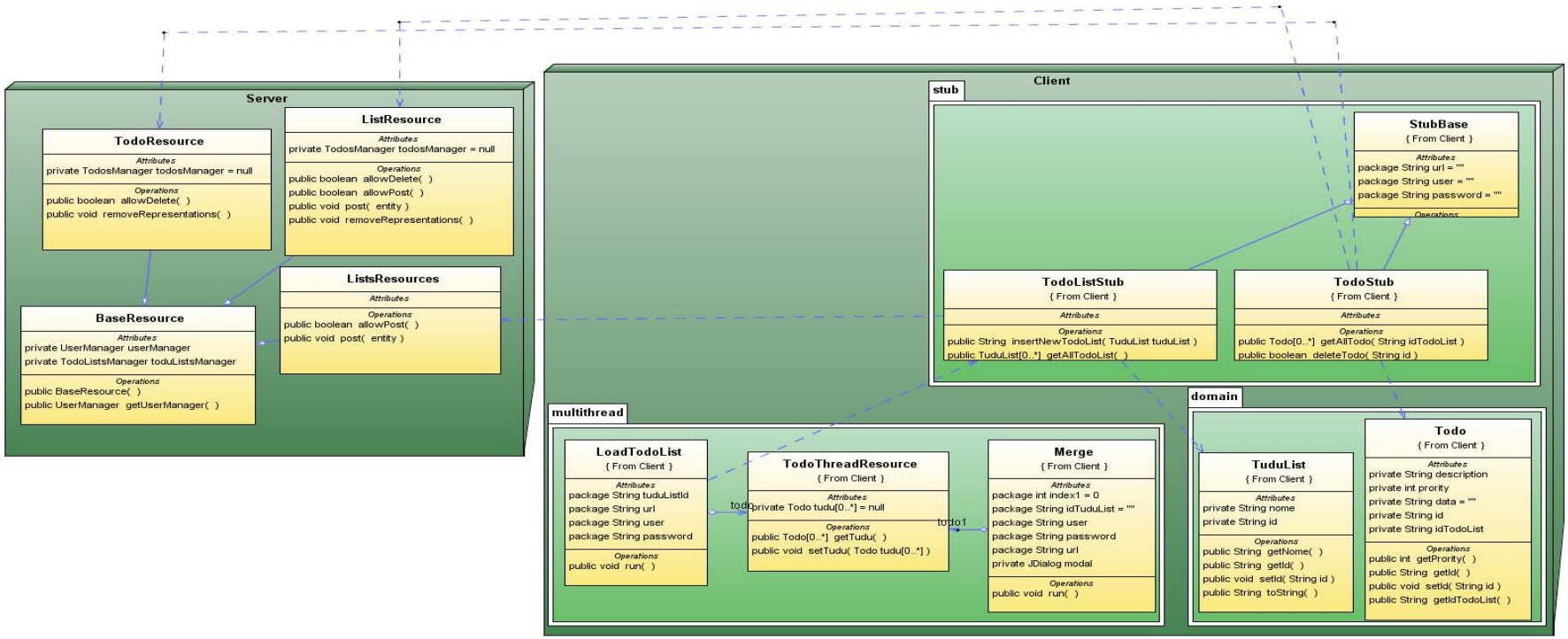
- Esaminando, prima di inoltrare la richiesta alla servlet, la api key associata all'invocazione del servizio possiamo valutare la sua correttezza ed eventualmente rifiutare la richiesta.



Realizzazione di casi d'uso complessi che integrano servizi migrati

- Attraverso un applicativo client che componendo i vari casi d'uso riesce ad implementarne uno del tutto nuovo
- Il nuovo caso d'uso è il merging di due liste

Class diagram client/server



Conclusioni

- **Obbiettivi**

√ L'obbiettivo che si è raggiunto con la tesi è stato quello di utilizzare un processo di migrazione verso servizi RESTful di web application esistenti, approfondendo e ampliando metodologie, best practice e frameworks ancora troppo poco documentati.

- **Sviluppi Futuri:**

❖ Progettazione e implementazione di un programma wrapper che supporti il processo di migrazione