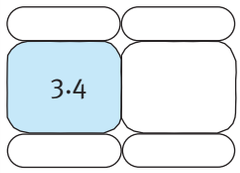
ARCHITETTURE INFORMATICHE
L'ORIENTAMENTO AI SERVIZI

Massimo Bartoletti
Vincenzo Ciancia
Gian Luigi Ferrari
Roberto Guanciale
Daniele Strollo
Roberto Zunino

Le architetture a servizi costituiscono il paradigma emergente per la progettazione e implementazione di applicazioni di rete. I servizi sono strumenti indipendenti dalla piattaforma, che possono essere descritti, pubblicati e composti ottenendo reti di applicazioni distribuite. Questo lavoro si propone di analizzare sia le sfide tecnologiche poste a chi progetta applicazioni distribuite con il paradigma orientato ai servizi, che il problema della definizione di nuovi modelli computazionali che favoriscono la realizzazione di strumenti software innovativi superando i limiti delle soluzioni tecnologiche oggi disponibili.



1. INTRODUZIONE

Il crescente sviluppo delle reti di telecomunicazioni e delle applicazioni informatiche di rete ha portato l'industria IT a confrontarsi con nuove possibilità e nuove sfide. Da un lato la grande diffusione della rete ha permesso lo sviluppo di applicazioni distribuite basate su protocolli "aperti" per fornire servizi agli utenti (si pensi ai portali web per avere un esempio significativo), dall'altro è aumentata la complessità delle applicazioni e la necessità di integrare sistemi informativi e piattaforme eterogenee. In particolare, le aziende in un mercato globalizzato sentono sempre più spesso la necessità di rendere accessibili i loro servizi **globalmente**. Questa esigenza comporta la necessità di avere qualcosa in più di un *front-end* di rete (il portale web) visto come l'interfaccia di rete del sistema informatico aziendale. Se un'azienda vuole essere un **attore attivo** del mercato globale, deve necessariamente integrarsi con l'esterno, con le applicazioni di partner, fornitori e clienti. Il problema di come integrare in modo efficace servizi e applicazioni eterogenee non è solo un problema di natura tecnologica, ma necessita di un approccio interdisciplinare accompagnato da una forte azione di formazione di nuove competenze. Nel contesto di questo articolo non intediamo affrontare le problematiche legislative, gli aspetti di organizzazione aziendali (modelli di business innovativo, la strutturazione dei processi ecc.), di formazione, messe in essere dalla globalizzazione dell'economia. Il nostro punto di vista sarà prettamente scientifico e tecnologico. In particolare, intendiamo evidenziare i limiti e le potenzialità delle soluzioni informatiche basate sulla nozione di servizio.

Le *Service Oriented Architecture* (SOA) [9, 14] sono state introdotte per affrontare le sfide delineate in precedenza. Il paradigma SOA suggerisce una metodologia di progettazione in cui applicazioni complesse sono realizzate attraverso l'interazione di entità chiamate *servizi*. I servizi sono componenti computazionali autonomi, indipendenti dalla piattaforma, che possono essere descritti, pubblicati, individuati, composti ottenendo reti di applicazioni distribuite in grado di

operare all'interno di una singola organizzazione e attraversarne i confini. L'applicazione così ottenuta diviene a sua volta un nuovo servizio disponibile sulla rete. È importante notare che:

- fornitori non vendono più software da installare su sistemi proprietari del cliente, ma forniscono un servizio;
- l'integrazione di più servizi è trasparente al cliente, che vede la collaborazione di più applicazioni come un unico servizio. Questo permette sia al cliente che al fornitore di poter cambiare i loro rispettivi fornitori, indipendentemente da come questi abbiano sviluppato le loro funzionalità.

Nell'ambito di un'architettura SOA è quindi possibile modificare, in maniera relativamente semplice, le modalità di interazione tra i servizi, oppure la combinazione nella quale i servizi vengono utilizzati all'interno di un processo di business. Allo stesso modo risulta più agevole aggiungere nuovi servizi e modificare le soluzioni software per rispondere alle specifiche esigenze di business. In altri termini, il processo di business specifico di un'azienda non è più vincolato da una particolare piattaforma o da un'applicazione ma può essere considerato come un componente di un processo più ampio e quindi può essere riutilizzato o modificato. Concettualmente, il paradigma SOA può essere descritto come il paradigma computazionale in cui i servizi interoperano mediante un meccanismo di descrizione autonomo, il contratto del servizio, indipendente dalla piattaforma sottostante e dalle tecnologie di sviluppo. Pertanto, i servizi in esecuzione su una piattaforma possono anche utilizzare servizi in esecuzione su altre facilitando quindi la riusabilità.

I servizi web (*web services*), sono probabilmente la tecnologia maggiormente sviluppata per la progettazione e realizzazione di applicazioni SOA. In letteratura, esistono definizioni differenti di cosa sia un servizio web. La nozione di servizio web definita dal *World Wide Web Consortium* (W3C) evidenzia bene gli aspetti caratterizzanti dei servizi web. "*A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards*" (cfr http://www.w3.org/TR/ws-gloss/).

La definizione riportata in precedenza evidenzia bene le caratteristiche essenziali di un paradigma computazionale basato sui servizi web. Le operazioni dei servizi (descrizione, pubblicazione, ricerca e l'invocazione) sono basate sullo scambio di messaggi codificati in XML. XML è uno standard di rappresentazione dei dati indipendente dalle diverse piattaforme applicative. Il protocollo SOAP (*Simple Object Access Protocol*) definisce i meccanismi di interazione tra servizi. WSDL (*Web Service Description Language*) è il linguaggio di descrizione delle operazioni del servizio e delle sue modalità di interazione. Infine, i servizi web utilizzano i protocolli standard di internet quali HTTP (*Hyper Text Transfer Protocol*), SMTP (*Simple Mail Transfer Protocol*), e MIME (*Multipurpose Internet Mail Extension*). Una volta definiti e resi disponibili i servizi si avverte la necessità di disporre di un registro dei servizi disponibili e di un protocollo di ricerca dei servizi all'interno della propria rete. Il protocollo UDDI (*Universal Description, Discovery and Integration*) è la risposta standard a questa esigenza. Per una presentazione dettagliata della struttura e delle caratteristiche dei servizi web e per un'introduzione agile e ragionata alle problematiche dei servizi si rimanda alla bibliografia [1, 15].

Uno degli aspetti innovativi delle architetture SOA e dei servizi web in particolare riguarda la composizione e il coordinamento dei servizi. La definizione delle politiche di coordinamento dei servizi avviene attualmente in maniera pragmatica, risolvendo gli aspetti cruciali del coordinamento caso per caso. In questo modo, si garantisce l'interoperabilità della soluzione ma alcune proprietà significative come il rispetto dei vincoli di sicurezza o l'utilizzo efficiente delle risorse non sono necessariamente assicurate. A seconda del modello utilizzato per realizzare le politiche di coordinamento si parla di **orchestrazione** o di **coreografia**. La differenza sostanziale tra queste due modelli di coordinazione consiste nella visione che i servizi hanno l'uno del-

l'altro. Nell'orchestrazione l'attenzione è incentrata su ogni singolo partecipante mentre, nella coreografia, ad ogni partecipante è offerta una visione parziale o addirittura globale del sistema. La babele di proposte di standard per orchestrazione e coreografia di servizi sviluppati negli ultimi anni (WSCI, BPML, WSFL, BPEL4WS, BTP, WS-CDL) testimonia la necessità di linguaggi per il coordinamento dei servizi. Rimanendo sul versante tecnologico, la *Service Component Architecture* (SCA) [8] è molto probabilmente l'esempio maggiormente significativo di modello di programmazione per servizi che pone una particolare enfasi sugli aspetti di definizione e realizzazione delle politiche di coordinamento dei servizi nel progetto di soluzioni business (gli *assembly* dei servizi nella terminologia SCA).

Realizzare politiche di coordinamento dei servizio non è solamente un problema tecnologico. Nella letteratura scientifica sono presenti diverse proposte di natura fondazionale che cercano di risolvere alcuni aspetti essenziali del problema. Altrettanto forte è l'interesse del mondo accademico e della ricerca di base alle problematiche relative alla definizione di linguaggi per il coordinamento dei servizi [13, 14, 20]. Le diverse proposte forniscono strumenti matematici per esprimere e certificare in modo formale proprietà generali sul comportamento di politiche di coordinamento di servizi. Un esempio significativo di politica di coordinamento riguarda le "transazioni di lunga durata" (*long running transactions*). Le transazioni di lunga durata sono un meccanismo fondamentale per aumentare la robustezza delle applicazioni di business realizzate tramite servizi. Le transazioni *long running* perdono la proprietà di serializzabilità e possiedono una nozione meno restrittiva di atomicità garantita da *rollback ad-hoc* chiamati "compensazioni" [11]. Un altro aspetto importante riguarda la definizione di politiche di sicurezza. I meccanismi forniti dagli standard di WS-security non sono sufficienti a garantire la sicurezza di una soluzione business, anche nel caso in cui i protocolli adoperati siano sicuri. La sicurezza di un sistema progettato con un'architettura a servizi può essere violata a causa di comportamenti inattesi e indesiderati che si rilevano solamente coordinando i servizi.

La definizione di modelli formali di certificazione di proprietà di politiche di coordinamento per architetture a servizi è essenziale per comprendere e garantire che una specifica soluzione di business soddisfi i requisiti richiesti e, in caso contrario, per comprendere quando e perchè si è verificato un problema. Più in generale la capacità di analizzare e certificare proprietà delle politiche di coordinamento dei servizi consente di progettare soluzioni software robuste in grado di prevedere ed evitare i problemi prima che si verifichino. Chi progetta un sistema di norma conosce un servizio solamente tramite le informazioni presenti nella descrizione WSDL del servizio stesso. È difficile comprendere quale servizio ha causato un problema e perchè. La definizione di meccanismi di certificazione delle proprietà di coordinamento costituisce, a nostro parere, il valore aggiunto delle soluzioni SOA basate su servizi web.

Questo articolo si pone l'obiettivo di presentare e analizzare alcune delle problematiche tecnologiche e scientifiche poste dalla necessità di definire compiutamente politiche per il cordinamento di servizi. Il primo aspetto che prenderemo in esame in questo nostro percorso riguarda le problematiche relative alla progettazione di servizi con transazioni long-running. La parte finale dell'articolo sarà dedicata alla presentazione di una metodologia che si propone di aiutare il designer nella fase di progettazione degli orchestratori di servizi in modo tale che l'applicazione nella sua interezza rispetti tutti i contratti di sicurezza posti in essere. Il nostro obiettivo è quello di illustrare l'idea (provocatoria) che la sinergia tra lo sviluppo tecnologico e la ricerca di base possa favorire la progettazione e lo sviluppo di soluzioni innovative per le aziende che operano nel mercato globale. In particolare, intendiamo mostrare come l'introduzione di metodologie basate su tecniche e strumenti di certificazione di proprietà favorisca la realizzazione di nuovi strumenti software a valore aggiunto in una prospettiva di *internetworked enterprise* superando i limiti delle soluzioni software oggi disponibili.

2. LE POLITICHE DI COORDINAMENTO DEI SERVIZI

La programmazione di applicazioni SOA richiede di affrontare nuove problematiche [18, 19]. La composizione di servizi e la descrizione del loro flusso di esecuzione non può essere arbitraria. Le politiche che regolano l'esecuzione di servizi sono spesso indicate con il termine generico di politiche di *coordinazione*. A seconda del modello utilizzato per implementare la coordinazione, si parla di *orchestrazione* o di *coreografia*. La differenza sostanziale tra i due modelli consiste nella visione che i servizi hanno l'uno dell'altro. Nell'orchestrazione l'attenzione è incentrata su ogni singolo partecipante mentre, nella coreografia, ad ogni partecipante è offerta una visione parziale o addirittura globale del sistema. Nel seguito verranno meglio evidenziate queste differenze cercando di enfatizzare quelli che sono i punti di forza e gli svantaggi dei due modelli.

Il termine orchestrazione si rifà alla metafora di un'orchestra in cui i singoli musicisti hanno conoscenza della propria attività da svolgere e attendono dal direttore di orchestra (*orchestrator*) le direttive prima di eseguire la prossima attività. Il maggiore svantaggio che questo modello comporta riguarda la centralizzazione del punto di controllo da parte dell'orchestratore. Seppure esistano soluzioni che prevedono la distribuzione del flusso di controllo tra diversi orchestratori resta di fatto che le informazioni di controllo siano gestite solamente da orchestratori. Il vantaggio che ne deriva da questo modello riguarda il totale isolamento dei servizi stessi, i quali non sanno in quale flusso di lavoro sono coinvolti e si limitano a svolgere le proprie attività comportandosi sempre allo stesso modo indipendentemente dal contesto di esecuzione.

Il linguaggio per la specifica di orchestrazione di servizi web più diffuso attualmente, risulta essere il *Business Process Execution Language* (BPEL) [6,12]. BPEL definisce dei costrutti ad alto livello (ad esempio condizionale, cicli, join di flussi paralleli) che vengono utilizzati dall'orchestratore per controllo di flusso. BPEL è un linguaggio eseguibile in quanto, una volta date le specifiche del *workflow* all'engine di orchestrazione, questo si occupa di eseguire le opportune attività.

Nella metafora della coreografia, ogni partecipante, in funzione delle azioni intraprese dagli altri partecipanti (non necessariamente da tutti) decide quale sia l'operazione da intraprendere. Di solito questo paradigma di coordinamento viene descritto dalla metafora dei ballerini "*Dancers dance following a global scenario without a single point of control*". Il maggior vantaggio di questa strategia risiede nella totale distribuzione delle informazioni di *workflow* ai componenti. Tuttavia, si deve evidenziare il fatto che i servizi coinvolti in una coreografia sono ben consapevoli del contesto in cui si trovano ad agire e, pertanto, risultano meno isolati rispetto a servizi utilizzati nell'orchestrazione. I servizi devono essere progettati preventivando di essere utilizzati all'interno di coreografia per cui devono esportare alcune funzionalità di supporto per poter gestire lo stato del flusso. Solitamente talune strategie che implementano coreografia, impongono l'utilizzo di servizi con stato (servizi *statefull*) al fine di mantenere consistenti i dati relativi alla gestione del flusso.

Uno standard per la descrizione di coreografie è WS-CDL (*Web Service Choreography Description Language*) [16]. La specifica WS-CDL prevede la stipula un contratto che contiene la definizione globale di vincoli necessari a garantire il normale svolgimento di tutte le attività. Questa visione globale viene successivamente proiettata nei singoli partner i quali devono garantire di assumere un comportamento conforme con quanto loro richiesto.

3. LA CERTIFICAZIONE DEI SERVIZI

I linguaggi attualmente usati per descrivere le operazioni che i servizi web offrono agli utenti sono basati su XML e, pertanto, puramente sintattici, cioè non specificano alcuna proprietà comportamentale associata all'invocazione del servizio. Tipicamente è la documentazione semi-formale del servizio a spiegare quali sono gli effetti che ci si aspetta quando si utilizza una determinata funzionalità.

Il problema di superare i vincoli sintattici della suite dei protocolli dei servizi web è stato affrontato facendo ricorso a metodi differenti.

Una delle soluzioni presenti in letteratura è quella del *Semantic Web* [7]. L'idea di base di questa soluzione consiste nel formalizzare relazioni fra simboli (le *ontologie*) in modo da guidare l'attività di ricerca di un servizio. Se si descrive tramite una ontologia che "regione" è allo stesso tempo un sottoinsieme di "stato" e un sovrainsieme di "città", un cliente che cerca servizi web legati ad aziende della Lombardia può avere come risposta non solo quelli che identificano esattamente, la regione, ma anche altri che coprono quella più ampia dell'Italia, o quella più ristretta di Milano. Sono in fase di sviluppo alcuni standard per specificare ontologie e servizi. Tra queste proposte citiamo WSML (*Web Service Modeling Language*), OWL (*Ontology Web Language*), WSMO (*Web Service Modeling Ontology*).
Una differente linea di ricerca affronta questo problema mediante l'uso di metodi formali per la certificazione e verifica di proprietà [20]. Quando si parla di verifica di componenti software, ci si riferisce all'uso di un meccanismo che consenta di dimostrare matematicamente l'aderenza del componente a un insieme di requisiti, tipicamente espressi usando un linguaggio formale, derivato dalla logica matematica. Nel contesto di questo lavoro considereremo le tecniche di verifica di *model-checking*. Le tecniche di model-checking permettono di verificare proprietà di un modello, sia esso specificato a tempo di progettazione, oppure estrapolato dal codice stesso del programma dopo la fase di implementazione. Gli strumenti automatici che sono in grado di eseguire il processo di verifica sono chiamati *model-checker*. Se applicato a modelli creati durante la fase di progettazione, il *model-checking* consente di evitare di propagare eventuali errori alle fasi successive dello sviluppo, a differenza di testing che invece opera sul prodotto finito. Inoltre, il model checking ha un ruolo importante anche all'interno di una metodologia *agile*, dato che è possibile integrare il testing con esecuzioni dello strumento di verifica, e segnalare immediatamente allo sviluppatore eventuali problemi non catturati dai casi di test. Una delle caratteristiche che rendono interessante l'uso del model-checking nella progettazione software è che, nel caso in cui un requisito venga violato, il model-checker restituisce un percorso di esecuzione che rappresenta tutti gli stati attraversati dal sistema fino al fallimento. Questo rende più facile evitare gli errori di progettazione, che influenzano a volte in maniera determinante l'implementazione, e se scoperti troppo tardi possono costringere a costosi *re-factoring*.
Le tecniche di model-checking si applicano in modo naturale a problemi in cui ciò che conta non è quali dati passino nel sistema, ma piuttosto il **flusso di controllo**, o il **flusso dei dati**. Per estrapolare il flusso di controllo o dei dati di un programma o di un progetto software, si possono usare tecniche raffinate come l'astrazione, che consiste in una esecuzione "simbolica" del programma stesso, che genera via via un modello non completo, ma sufficiente a fare una verifica di molte proprietà interessanti. Un esempio significativo è fornito da Java PathFinder. Java PathFinder è un ambiente di sviluppo per programmi Java sviluppato dalla NASA in cui sono disponibili un meccanismo di astrazione e un *model-checking* che analizza bytecode Java. Java PathFinder consente di verificare automaticamente proprietà di assenza di deadlock e trovare eccezioni non gestite, generando una traccia dell'esecuzione del software che porta alla condizione di errore.
Storicamente, il model-checking è stato proposto in seguito ai lavoro pionieristici di E. M. Clarke, E. A. Emerson nel 1981, e di J. P. Queille, J. Sifakis nel 1982. Clarke, Emerson e Sifakis sono stati recentemente insigniti del *Turing Award* per il 2008 per i risultati ottenuti nel campo del *model-checking*. Il *Turing Award* è il principale riconoscimento scientifico assegnato a ricercatori per il loro contribuito allo sviluppo dell'informatica. Nella direzione del *model-checking* si stanno muovendo anche le grandi imprese fornitrici di software e servizi.

4. LE TRANSAZIONI DI LUNGA DURATA

Un aspetto importante e innovativo delle politiche di coordinamento dei servizi riguarda gli aspetti transazionali. Spesso l'esecuzione di una politica di coordinamento richiede numerose interazioni con l'utente o con processi intermedi che vanno anche al di fuori dell'applicazione. Si pensi per esempio ad un servizio di vendita online che accetta pagamenti tramite servizi che effettuano bonifico

bancario. In questa situazione, il pagamento potrebbe essere notificato alcuni giorni dopo che l'attività di prenotazione dell'acquisto del bene ha iniziato il suo ciclo. Per questi motivi, le tecniche tradizionali dei sistemi distribuiti non sono sufficienti a garantire la correttezza di comportamenti transazionali. Infatti non è possibile riutilizzare le tecniche tradizionale per la gestione dellle transazioni (transazioni ACID) dato che i tempi di esecuzione delle operazioni coinvolte sono molto lunghi ed è impensabile attuare meccanismi tradizionali di *locking*, *rollback* e isolamento. Le *Transazioni Long Running*, transazioni di lunga durata, sono lo strumento principale che è stato adottato per rispondere alle richieste di atomicità di servizi. Il meccanismo computazionale adottato per operare agilmente con transazioni di lunga durata è la compensazione (*compensation*). La nozione di compensazione è stata introdotta nei processi SAGAS [11]. Le compensazioni sono quelle attività che permettono di programmare quelle azioni da eseguire per rispristinare lo stato dell'esecuzione parziale di un processo di business. Intuitivamente, le compensazioni possono essere viste come un meccanismo di gestione delle eccezioni programmabile. Si consideri, per esempio, il seguente processo transazionale:

P := [A1 << B1] ; [A2 << B2] ; [A3<< B3]

Il processo transazionale P è costituito dalla composizione sequenziale di tre attività (A1, A2 e A3). Ad ogni attività è associata una compensazione specifica. Per comprendere il comportamento del processo, supponiamo che l'attività A1 completi con successo la propria esecuzione, mentre l'esecuzione dell'attività A2 generi una situazione di fallimento. In seguito alla segnalazione del fallimento di A2, viene mandato in esecuzione la compensazione B1 (quella associata all'attività A1 terminata con successo) per ripristinare lo stato dell'esecuzione del processo. In altri termini, il fallimento dell'attività ha comportato il fallimento dell'intero processo transazionale. Si noti che la compensazione B2 non deve essere mandata in esecuzione dato che A2 non ha completato la propria esecuzione con successo. Similmente, nel caso in cui A1 e A2 abbiano completato con successo l'esecuzione il processo transazionale termina con succcesso solamente nel caso in cui A3 abbia successo. In caso di fallimento di A3 saranno mandate in esecuzione le compensazioni B2 e B3. Infine, le compensazioni B3, B2 e B1 saranno eseguite in quest'ordine nel caso in cui P sia un sottoprocesso di un processo transazionale e che P abbia ricevuto una notifica di fallimento dopo aver concluso con successo la propria esecuzione.
Questo semplice esempio bene evidenzia l'idea che le compensazioni sono quelle azioni che devono essere eseguite localmente per ripristinare la consistenza dello stato locale del servizio nel caso di notifica di fallimento.
Il diagramma illustrato nella figura 1 descrive la specifica ad alto livello della componente transazionale di un servizio che ha il compito di modificare in parallelo lo stato di un certo numero di oggetti distinti tra loro. Si suppone che gli oggetti siano accessibili dal servizio e che siano memorizzati in un sistema *legacy* che garantisce il *roll-back* perfetto tramite le azioni standard di *Commit e Rollback*.

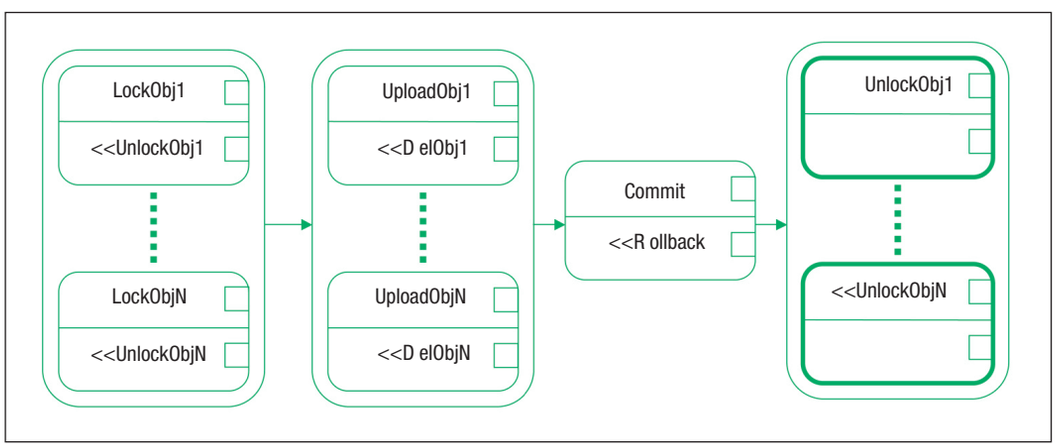


FIGURA 1
Transazioni di lunga durata in Sagas

La maggior parte dei modelli di *workflow* presenti in letteratura adotta l'idea di compensazione per la corretta gestione delle transazioni di lunga durata. Le diverse proposte differiscono tra loro per la presenza di compensazioni aventi caratteristiche differenti (per esempio compensazioni gerarchiche). Si rimanda alla nota [17] per un'analisi dettagliata delle diverse proposte.
Per comprendere come affrontare il problema della progettazione transazioni *long running*, consideriamo un caso di studio piccolo ma significativo. Consideriamo il caso di un veicolo equipaggiato con un sistema di diagnostica digitale. Il sistema controlla periodicamento lo stato dell'automobile, e quando si presentano situazioni di guasto (surriscaldamento del motore, liquidi sotto livello di soglia ecc.) il sistema attiva automaticamente un servizio di emergenza. Il servizio di emergenza mediante una interazione con l'utente determina e seleziona il servizio necessario per la gestione della situazione di emergenza. Per semplicità, supponiamo che il sistema di emergenza possa selezionare e richiedere un servizio di carro–attrezzi e un servizio di officina per la riparazione del quasto rilevato. Supponiamo, inoltre, che possa venire richiesto opzionalmente un servizio di macchina a noleggio.
La figura 2 illustra il design del *workflow* del caso di studio descritto in prccedenza. La rappresentazione grafica del workflow utilizza la notazione standard denominata BPMN (*Business Process Modeling Notation*). Per maggiori dettagli su BPMN si consiglia di fare riferimento al sito http://www.bpmn.org/.
Nel *workflow* di figura 2, la sottotransazione che gestisce l'operazione di noleggio dell'autoveicolo è isolata dal resto del processo. Questo comporta che il fallimento della sua esecuzione non determina automaticamente il fallimento del workflow nella sua interezza.

5. UNA METODOLOGIA PER L'ORCHESTRAZIONE SICURA DI SERVIZI

La sicurezza dei sistemi software è una questione cruciale in un mondo sempre più permeato dall'Information Technology. Anche le applicazioni basate sui servizi web presentano forti requisiti di sicurezza visto che operano nel contesto distribuito e insicuro di Internet. Autorevoli enti, come il CERT della *Carnegie Mellon University*, riportano un numero sempre crescente di segnalazioni relative

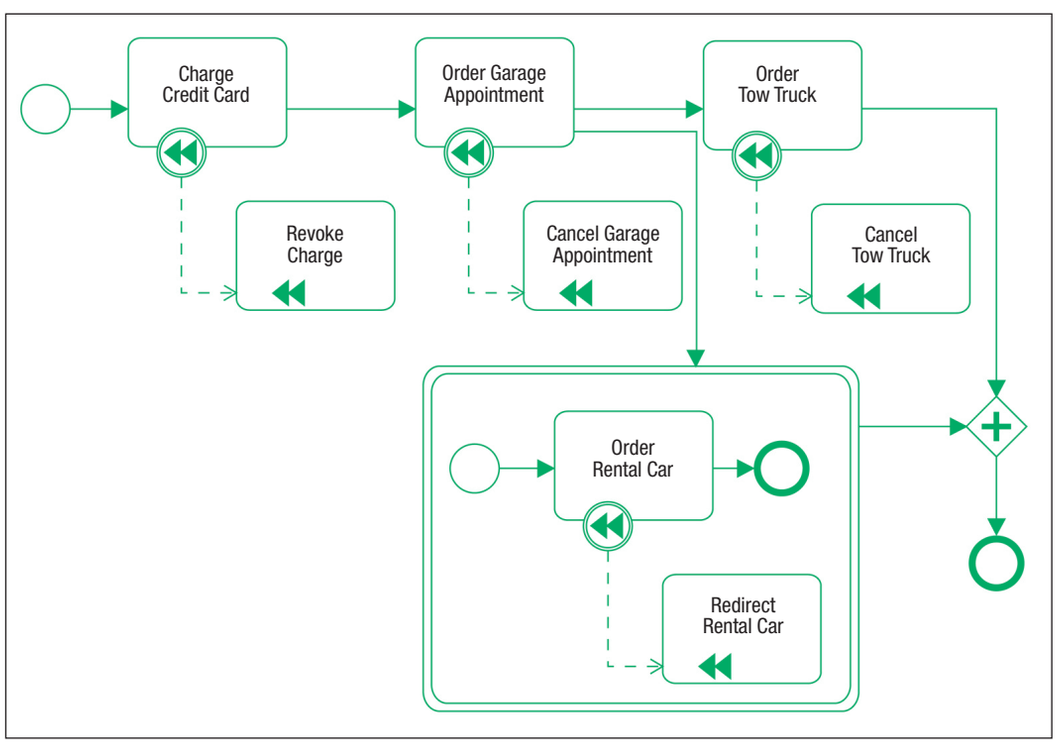


FIGURA 2
Un processo di business con compensazione nella notazione BPMN

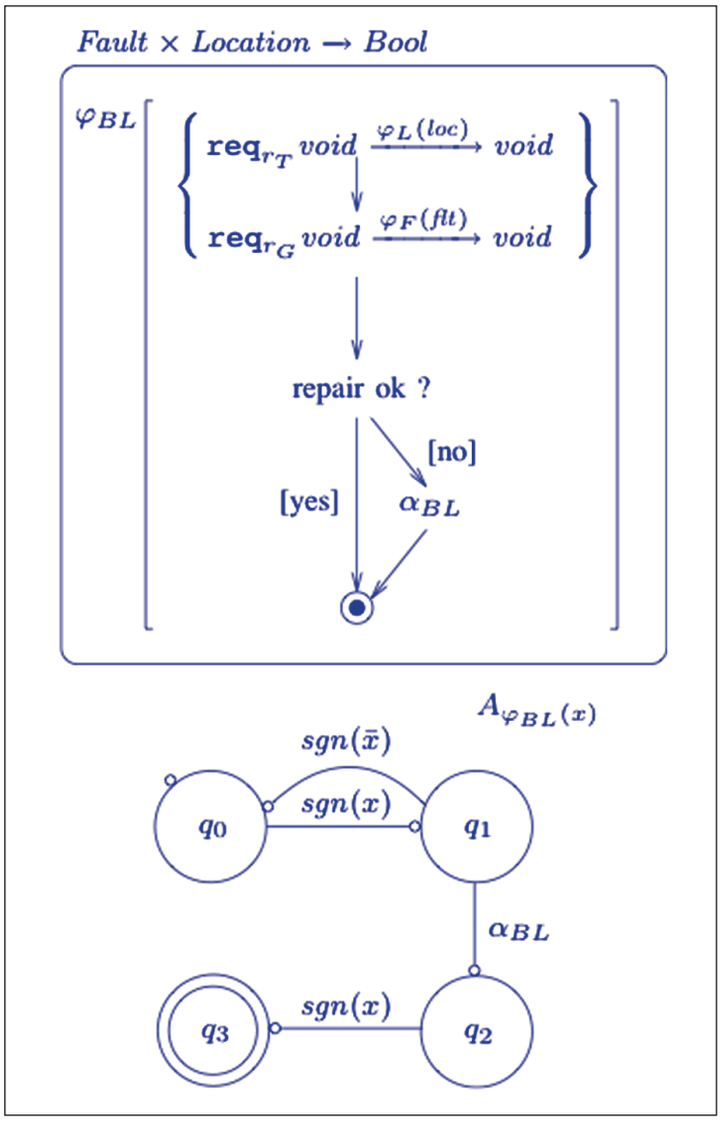


FIGURA 3
Car-emergency service

alla vulnerabilità dei sistemi informatici, spesso dovute a problemi nella progettazione e realizzazione del software. La soluzione tipica è quella di riparare tali "falle" rendendone più difficile l'utilizzo da parte degli attaccanti, ma questo approccio non va certamente alla radice del problema. Un approccio complementare è, invece, quello di modellare e verificare i requisiti di sicurezza fin dalla specifica di un sistema software, al fine di ridurre al minimo le vulnerabilità sul prodotto finale. L'adozione di tecniche formali può quindi giocare un ruolo molto importante per individuare possibili problemi di sicurezza fin dalle prime fasi di progettazione per capirne a fondo le cause e per rimuoverli prima che altre scelte progettuali costringano a "inventare" rimedi a posteriori, non sempre possibili e soddisfacenti. I meccanismi forniti dagli standard di WS-security non sono sufficienti a garantire la sicurezza di una soluzione business, anche nel caso che i protocolli adoperati siano sicuri. Il problema principale è come orchestrare servizi, garantendo di non abortire l'esecuzione dell'orchestrazione a causa di azioni che tentano di violare la sicurezza. I servizi che soddisfano localmente la proprietà richiesta non sono sempre buoni candidati, dato che il loro comportamento può altresì invalidare la sicurezza dell'intera orchestrazione.

Per comprendere come affrontare il problema della sicurezza, consideriamo una piccola estensione del caso di studio presentato in precedenza. In particolare assumiamo che la selezione di un servizio possa tenere conto delle preferenze del conducente del veicolo. Per semplicità, supponiamo che il sistema di emergenza possa selezionare e richiedere un servizio di carro–attrezzi e un servizio di officina per la riparazione del guasto rilevato. Il nostro obiettivo è quello di illustrare le principali caratteristiche di una metodologia di supporto progettazione della orchestrazione dei servizi. Per fare questo utilizzeremo una notazione grafica a la UML (diagrammi dell'attività, e diagrammi dei casi di studio).

La metodologia per la definizione dell'orchestrazione sicura dei servizi è stata definita in [2, 3, 4] e sfrutta una tecnica di *model-checking* che a partire dai contratto di servizio determina tutte le possibili orchestrazioni che soddisfano i contratti in opera. Tecnicamente, i contratti di servizi sono una estensione dei documenti WSDL e sono formalmente definiti mediante un'opportuno sistema di tipo.

Per introdurre in modo semplice la notazione, presentiamo il servizio di *car-emergency*, ovvero il servizio invocato dal sistema di diagnistica del veicolo per la gestione delle situazioni di emergenza. Il diagramma nella figura 3 descrive la specifica del comportamento del servizio di *car-emergency*. Il servizio è definito mediante una forma particolare di diagramma delle attività in cui sono descritti l'interfaccia del servizio (la parte superiore del diagramma) ed il flusso

del suo comportamento. Un aspetto importante della metodologia è che devono essere definite anche le politiche che vincolano il comportamento del servizio. In particolare, la politica φ_{BL} indica che il servizio memorizza, localmente, le officine che non hanno soddisfatto l'utente. L'idea è che al termine della riparazione il servizio di officina deve firmare digitalmente un certificato di qualità. Se l'utente non è soddisfatto della qualità del servizio offerto inserisce l'officina nella lista nera (black list - BL) delle officine. Le officine inserite nella black-list non saranno più considerate per la gestione di emergenze. La politica di qualità è definita dall'automa a stati finiti nella parte inferiore della figura. Analogamente al servizio di emergenza, tutti i servizi presenti possono specificare la loro politica di qualità. Per esempio, il servizio di officina assume la presenza di una politica di raggiungibilità da parte del carro attrezzi. La poltica φ_{GZ} controlla che il carro attrezzi sia effettivamente in grado di raggiungere l'officina.

La parte innovativa del diagramma delle attività riguarda l'introduzione del blocco di orchestrazione, ovvero quella parte della sequenza del flusso di esecuzione compresa tra le parentesi grafe. In questo caso, il blocco permette di specificare la richiesta di orchestrare due chiamate di servizio, una per il carro attrezzi e l'altra per l'officinia di riparazione. La caratteristica originale della metodologia è che l'orchestrazione è definite mediante un meccanismo di chiamata dei servizi per contratto. La chiamata per contratto prevede che la selezione dei servizi sia basata sulle proprietà di cui essi godono, piuttosto che sull'identità del produttore del servizio, come accade normalmente. Il blocco di orchestrazione ha il compito di individuare una orchestrazione valida per entrambe l'invocazioni di servizio. Infatti non ha significato alcuno prenotare il carro attrezzi quando l'officina non è disponibile. L'orchestrazione viene individuata mediante l'invocazione di uno strumento di model checking che determina la lista delle orchestrazioni valide.

La figura 4 descrive un diagramma di caso di studio (*Use Case Diagram*) in cui sono presenti un cliente e quattro servizi (due officine e due servizi di carro attrezzi). Ipotizziamo che l'automobile abbia una emergenza ai pneumatici nella zona di Pisa (ZIP_{PI}) e che l'officina in Lucca sia stata inserita nella lista nera (sgn_{LU}) in precedenza. Inoltre, un servizio di carro attrezzi opera nelle provincie di Pisa, e Firenze, mentre l'altro servizio opera nelle provincie di Pisa, Siena e Lucca. Infine, entrambe le officine offrono il servi-

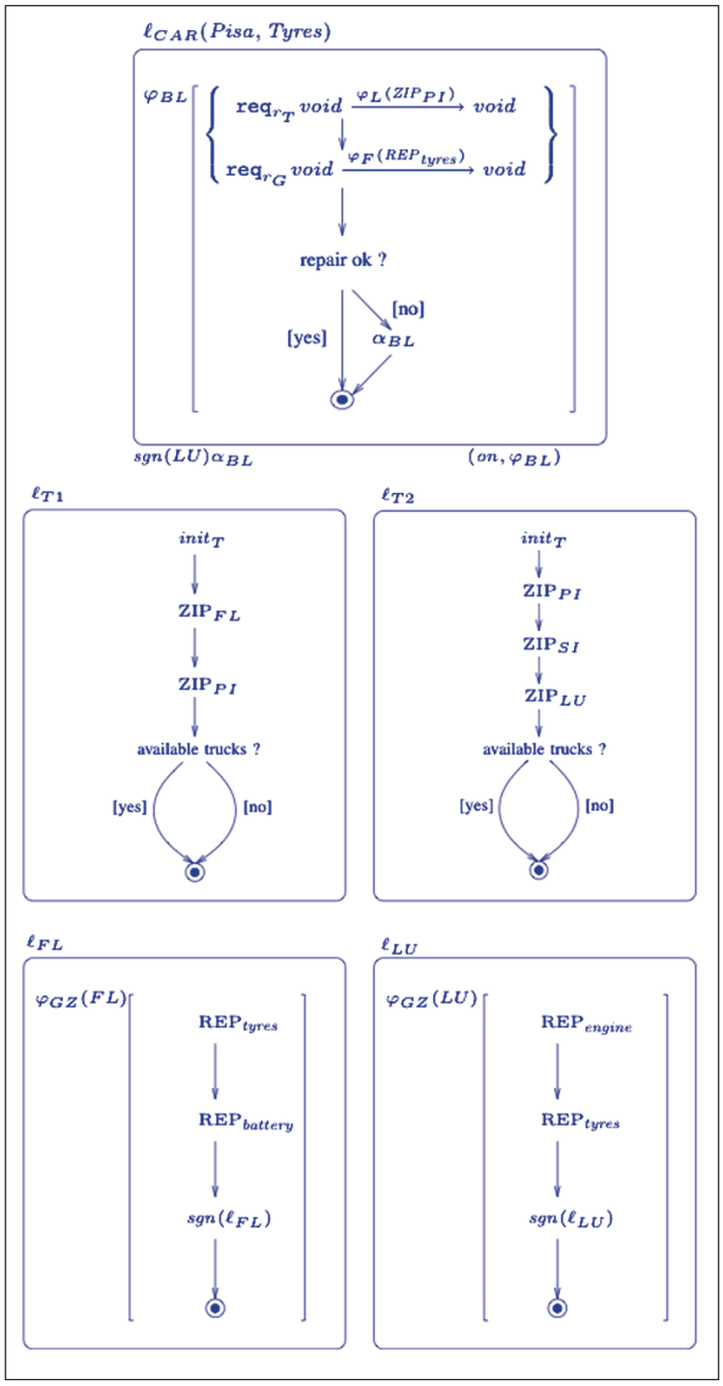


FIGURA 4
Diagramma caso di studio

zio di riparazione e sostituzione dei pneumatici. Si può facilmente notare che l'orchestrazione che invoca il servizio di carro attrezzi T2 e l'officina che opera a Lucca, viola la politica definite dall'utente. L'orchestrazione che invoca il servizio di carro attrezzi T1 e il servizio di officina di Firenze rispetta tutti i contratti posti in opera.
La metodologia descritta in precedenza è supportata da un *middleware* di programmazione denominato JSCL (*Java Signal Core Layer*) [10]. JSCL è equipaggiato con un ambiente di sviluppo grafico (tecnicamente un Plug-in di Eclipse) che permette di generare automaticamente il codice della orchestrazione a partire dalla descrizione del flusso delle attività. Infine, JSCL fornisce un supporto automatico alla progettazione ed implementazione di transazioni long-running [5].

6. CONCLUSIONI

In questo lavoro abbiamo cercato di individuare alcune delle sfide poste a chi progetta applicazioni distribuite con il paradigma orientato ai servizi. Il nostro obiettivo è stato quello di illustrare l'idea che la giusta miscela tra innovazione tecnologica e ricerca fondazionale di base favorisca la realizzazione di strumenti software innovativi superando i limiti delle soluzioni tecnologiche oggi disponibili.
Le problematiche descritte in questo articolo sono state affrontate in progetti di ricerca sia in ambito nazionale che europeo. In Italia è attualmente in corso il progetto di ricerca di base *FIRB* denominato TOCAI.IT, *Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet* (http://www.dis.uniroma1.it/~tocai/index.php). In particolare nel contesto di questo progetto sono studiate e sperimentati i servizi di base forniti dal middleware JSCL per la progettazione di comportamenti transazionali per servizi. Nel contesto del 6th – (FP6) è attivo il progetto SENSORIA, *Software Engineering for Service-Oriented Overlay Computers* (http://www.sensoria-ist.eu/) che si pone l'obiettivo di definire un insieme di metodologie innovative per la progettazione e sviluppo di architetture software orientate ai servizi. La specifica e l'implementazione di JSCL, la metodologia per l'orchestrazione sicura di componenti sono stati realizzati all'interno di questo progetto.

Bibliografia

[1] Alonso G., Casati F., Kuno H., Machiraju V.: *Web Services: Concepts, Architecture and Applications*. Springer Verlag 2004.

[2] Bartoletti M., Degano P., Ferrari G.: *Types and Effects for Secure Service Orchestration*. Proc. 19th IEEE Computer Security Foundations Workshop (CSFW'06), IEEE Press, 2006.

[3] Bartoletti M., Degano P., Ferrari G., Zunino R.: *Secure Service Orchestration*. Foundations of Security Analysis and Design IV, FOSAD 2006/2007 Tutorial Lectures, Lecture Notes in Computer Science 4677, Springer 2007.

[4] Bartoletti M., Degano P., Ferrari G., Zunino R.: Semantics-based design for Secure Web Services. *IEEE Transactions on Software Engineering*, TSE, Vol. 34, n. 1, 2008.

[5] Bruni R., Ferrari G., Melgratti H., Montanari U., Strollo D., Tuosto E.: *From Theory to Practice in Transactional Composition of Web Services*. Formal Techniques for Computer Systems and Business Processes, Lecture Notes in Computer Science 3670, Springer 2005.

[6] *Business Process Execution Language* (BPEL). Disponibile on line all'indirizzo http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[7] Cardoso J., Sheth A.: *Amit. Semantic Web Services, Processes and Applications*. Springer 2006.

[8] Chappel D.: *Introducing SCA*. Disponibile on line all'indirizzo (http://www.davidchappell.com/-articles/Introducing_SCA.pdf), July 2007.

[9] Erl T.: *SOA Principles of Service Design*. The Prentice Hall Service-Oriented Computing Series, Prentice Hall 2007

[10] Ferrari G., Guanciale R., Strollo D.: *JSCL: A Middleware for Service Coordination*. 26th IFIP WG 6.1, Formal Techniques for Networked and Distributed Systems – FORTE 2006, Lecture Notes in Computer Science 4229, Springer 2007.

[11] Garcia-Molina H., Salem K.: *Sagas*. International Conference on Management of Data, ACM SIGMOD, ACM Press, 1987.

[12] Juric M., Mathew B., Sarang P.: *Business Process Execution Language for Web Services*. PACKT Publishing, 2006.

[13] Kitchin J., Cook W.R., Misra J.: *A Language for Task Orchestration and Its Semantic Properties*. CONCUR 2006, Lecture Notes in Computer Science, 4137 Springer 2006.

[14] Papazoglou M., Georgakopoulos D.: Special issue on service oriented computing. *Communications of the ACM*, Vol. 46, n. 10, 2003.

[15] Pernici B., Plebani P.: Introduzione ragionata al mondo dei Web Service. *Mondo Digitale*, marzo 2004, AICA.

[16] Ross-Talbot S., Fletcher T.: *Web Services Choreography Description Language*. Primer, W3C Working Draft, 2006 (http://www.w3.org/TR/2006/WD-ws-cdl-10-primer-20060619/)

[17] Sheth A., Worah D.: *Transactions in Transactional Workflows*. Advanded Transaction Models and Architectures, Kluwer, 1997.

[18] Stal M.: Web services: Beyond component-based computing. *Communications of the ACM*, Vol. 55, n. 10, 2002.

[19] Vogels W.: Web services are not distributed objects. *IEEE Internet Computing*, Vol. 7, n. 6, 2003.

[20] Wirsing M., et~al: *Semantic-based development of service-oriented systems*. 26th IFIP WG 6.1, Formal Techniques for Networked and Distributed Systems – FORTE 2006, Lecture Notes in Computer Science 4229, Springer 2007.

MASSIMO BARTOLETTI ha ottenuto il Dottorato di Ricerca in Informatica, Scuola Galilei, Università d Pisa. Titolare di un assegno di ricerca presso il Dipartimento di Informatica Università di Pisa. I suoi interessi riguardano le problematiche relative alla specifica e verifica di proprietà di sicurezza in sistemi distribuiti.
E-mail: bartolet@di.unipi.it

VINCENZA CIANCIA sta terminando il Dottorato di Ricerca in Informatica, Scuola Galilei, Università di Pisa. I suoi interessi riguardano l'utilizzo di metodi formali nella certificazione di programmi.
E-mail: ciancia@di.unipi.it

GIAN LUIGI FERRARI professore associato presso il Dipartimento di Informatica Università di Pisa. I suoi interessi più recenti riguardano i metodi e gli strumenti per la progettazione e la realizzazione di architetture software orientate ai servizi. È autore di molte pubblicazioni internazionali, tra cui oltre cento articoli su riviste e atti di convegni.
E-mail: g.ferrari@di.unipi.it

ROBERTO GUANCIALE sta terminando il Dottorato di Ricerca in Scienze e Tecnologie dell'Informatica, Institutes for Advances Studies, IMT, Lucca. I suoi interessi riguardano il progetto e la realizzazione di architetture software orientate ai servizi.
E-mail: roberto.guanciale@imtlucca.it.

DANIELE STROLLO sta terminando il Dottorato di Ricerca in Scienze e Tecnologie dell'Informatica, Institutes for Advances Studies, IMT, Lucca. I suoi interessi riguardano il progetto e la realizzazione di architetture software orientate ai servizi.
E-mail: daniele.strollo@imtlucca.it.

ROBERTO ZUNINO Ricercatore Universitario presso il Dipartimento di Informatica e Telecomunicazioni, Università di Trento. I suoi interessi riguardano le problematiche relative alla specifica e verifica di proprietà di sicurezza in sistemi distribuiti.
E-mail: zunino@di.unipi.it