Artifact Centric Business Processes (I)

Introduzione

Autore: Piero Cangialosi
Docente: Prof. Giuseppe De Giacomo

Dipartimento di Informatica e Sistemistica
SAPIENZA - Universitá di Roma

16 Novembre 2008

La modellazione dei processi di business *artifact-centric* nasce a metà degli anni novanta all'interno di IBM, presentandosi come la risposta al crescente bisogno delle aziende di poter applicare rapidamente i cambiamenti ai propri processi di business e poter così superare i propri competitori sul mercato.

La metodologia che presenteremo si basa sul concetto fondamentale di *artefatto*. I principali vantaggi di ragionare secondo quest'ottica sono:

- maggiore flessibilità nella rapresentazione
- maggiore facilità ad analizzare i cambiamenti
- maggiore capacità di gestire i sistemi di implementazione che supportano il business

Un artefatto è un pezzo di informazione concreto, identificabile e auto-esplicativo che può essere utilizzato da una persona coinvolta nel processo di business per portare avanti il processo stesso, e di esso raprresenta l'unica informazione esplicita.
Le proprietà formali di un artefatto sono catturate dai seguenti assiomi:

- un artefatto è caratterizzato da un'identità unica, valida all'interno dell'intero enteprise, e da un contenuto auto-esplicativo
- l'identità di un artefatto non può essere cambiata
- il contenuto di un artefatto può essere modificato arbitrariamente

Il processamento degli artefatti (che possono essere creati, modificati, archiviati - anche se non ancora completi) viene modellato attraverso altri due concetti: task e repository.

- l'obiettivo di un task è quello di effettuare un'azione e memorizzare il risultato in uno o più artefatti in suo possesso
- un task viene eseguito mediante l'arrivo di un artefatto oppure tramite un opportuno meccanismo di attivazione
- gli artefatti entrano in un task spontaneamente oppure attraverso una esplicita richiesta
- se un task ha terminato la sua esecuzione tutti gli artefatti in esso contenuti vengono espulsi
- gli artefatti possono essere inseriti in un repository
- un repository può rispondere ad una richiesta per un artefatto in esso contenuto ma non può inviare artefatti spontaneamente

Esempio

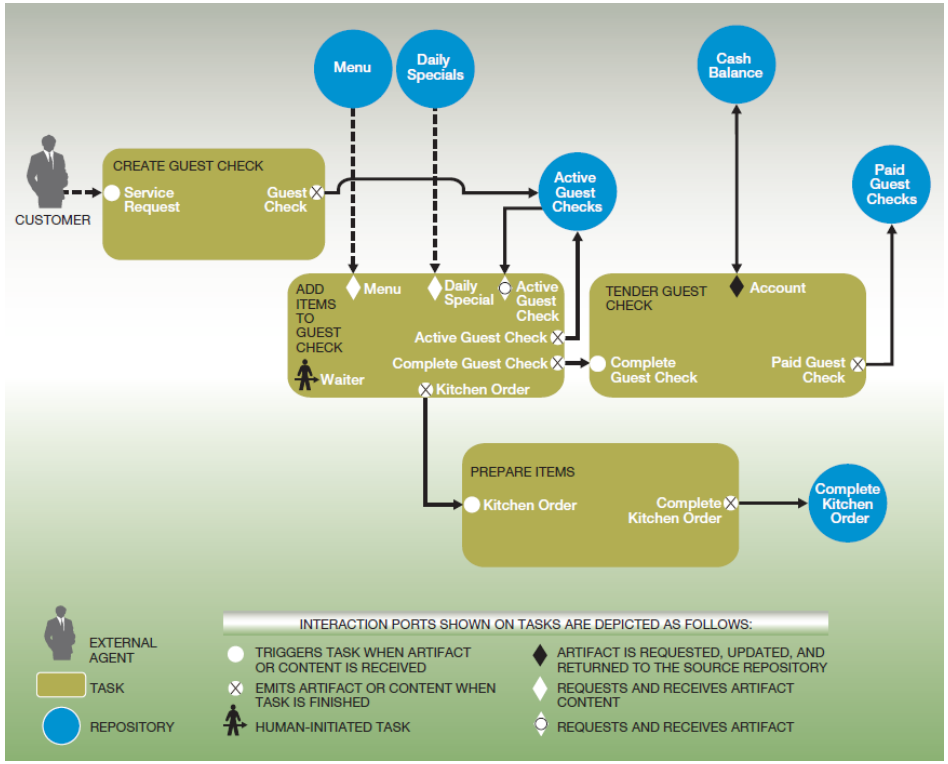

Figura: Ciclo di vita dell'artefatto `Guest Check`

Il primo passo dell'approccio data-centric consiste nell'individuare gli artefatti chiave assieme ai rispettivi cicli di vita. Successivamente viene creata una specifica logica dettagliata per ogni classe di artefatto, per i servizi che vi opereranno, e per le associazioni tra i task e i servizi (che, come vedremo, assumono la forma di regole di business). L'ultimo passo consiste nel trasformare questa specifica *dichiarativa* in una specifica *procedurale*, che può messere ottimizzata e successivamente mappata in una implementazione fisica.

La metodologia si basa su quattro concetti fondamentali:

- **ARTEFATTI**
  Un artefatto, come accennato nell'introduzione, contiene informazioni pertinenti al processo di business
- **(MACRO) CICLI DI VITA**
  La scoperta di nuovi artefatti procede mano a mano che si analizza il ciclo di vita degli artefatti stessi.
  Questi cicli di vita sono generalmente definiti come macchine a stati finiti
- **SERVIZI**
  Un servizio modifica uno o più artefatti (la modifica si riflette nel valore degli attributi di questi utlimi)
- **ASSOCIAZIONI**
  Un servizio applica delle modifiche ad una serie di artefatti in maniera ristretta da una serie di vincoli.

I passi da compiere per arrivare ad una specifica finale sono i seguenti:

1. *Individuazione degli Artefatti*
   - Identificare gli Artefatti principali
   - Identificare gli stadi principali del ciclo di vita degli Artefatti tramite l'analisi dello scenario
2. *Design del Business Operation Model (BOM)*
   - Specifica logica dello schema degli Artefatti (ad esempio modello ER)
   - Specifica dei Servizi che fanno avanzare gli Artefatti attrvaerso i loro cicli di vita
   - Specifica delle regole ECA che abilitano tali servizi
3. *Design del diagramma del flusso di lavoro concettuale*
4. *Implementazione di ogni singolo componente*

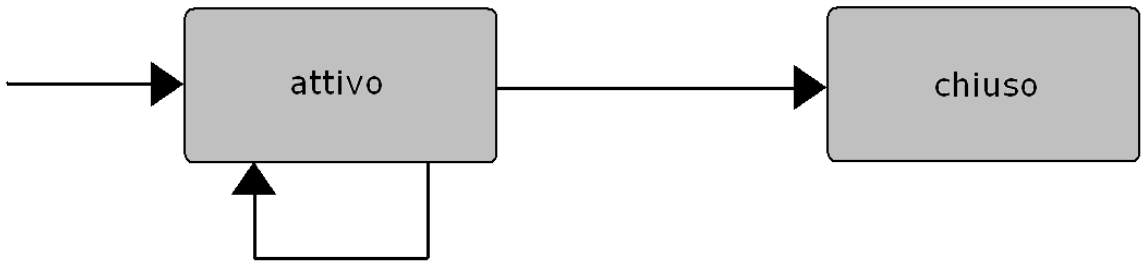

Figura: Ciclo di vita dell'artefatto `Guest Check`

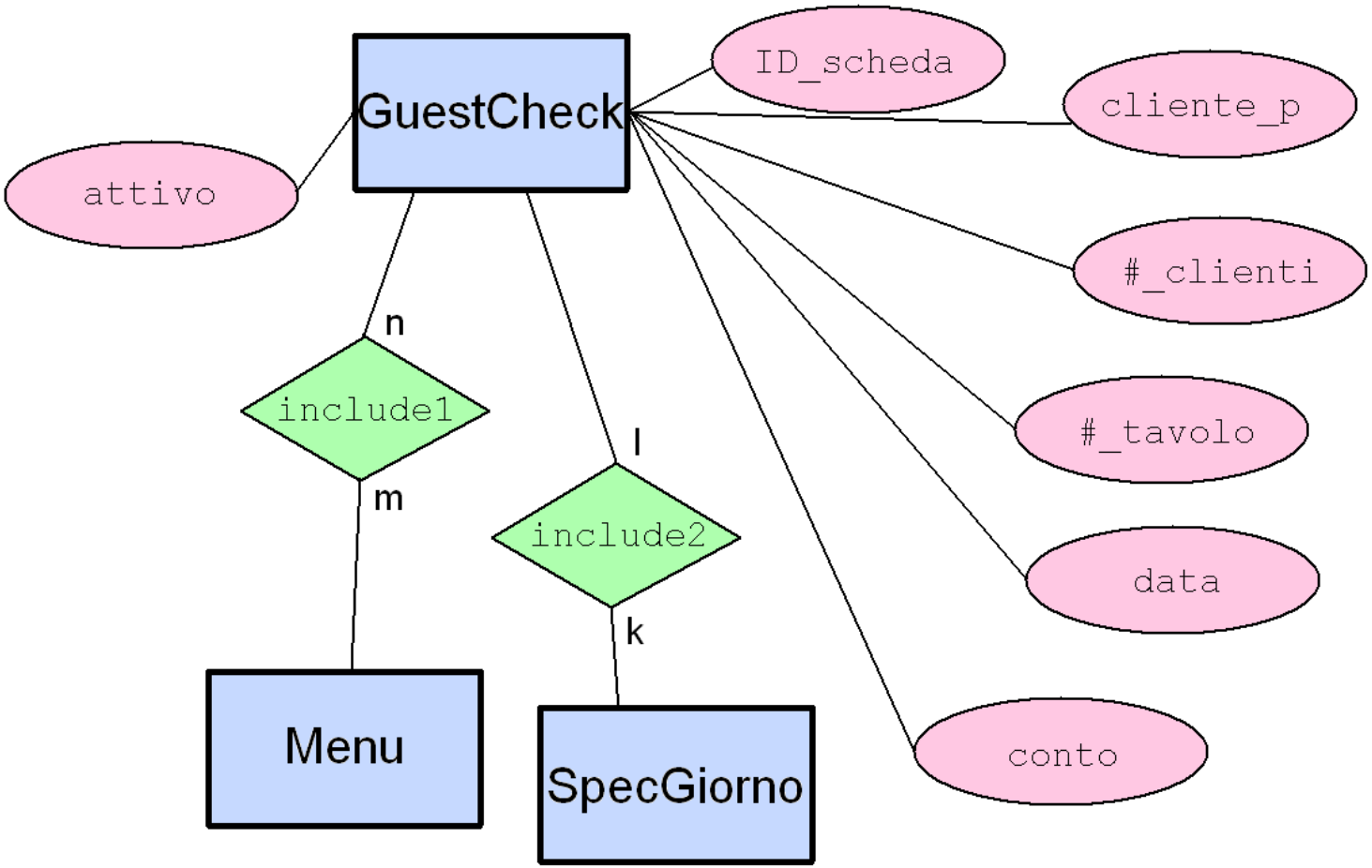

Figura: Schema dell'artefatto `Guest Check`

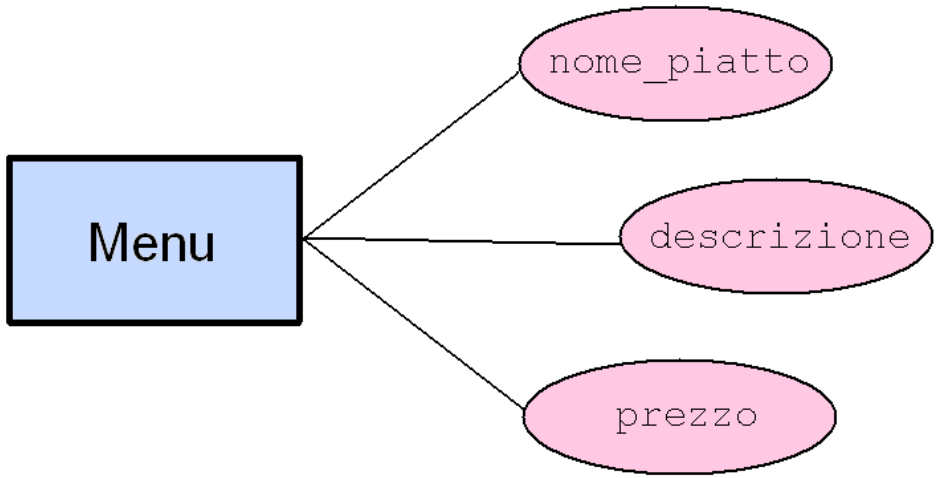

Figura: Schema dell'artefatto `Menu`

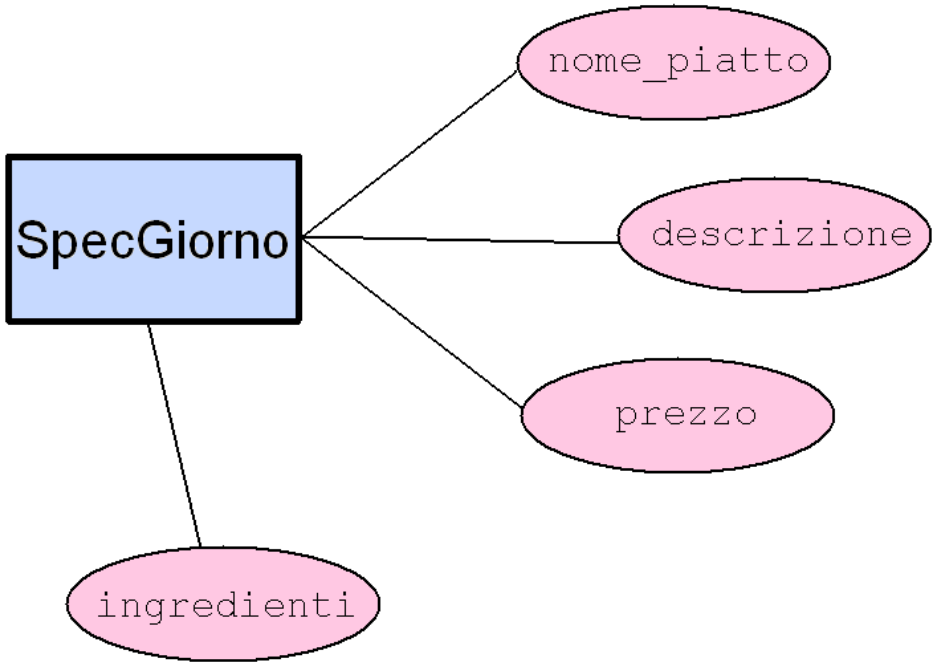

Figura: Schema dell'artefatto `SpecGiorno`

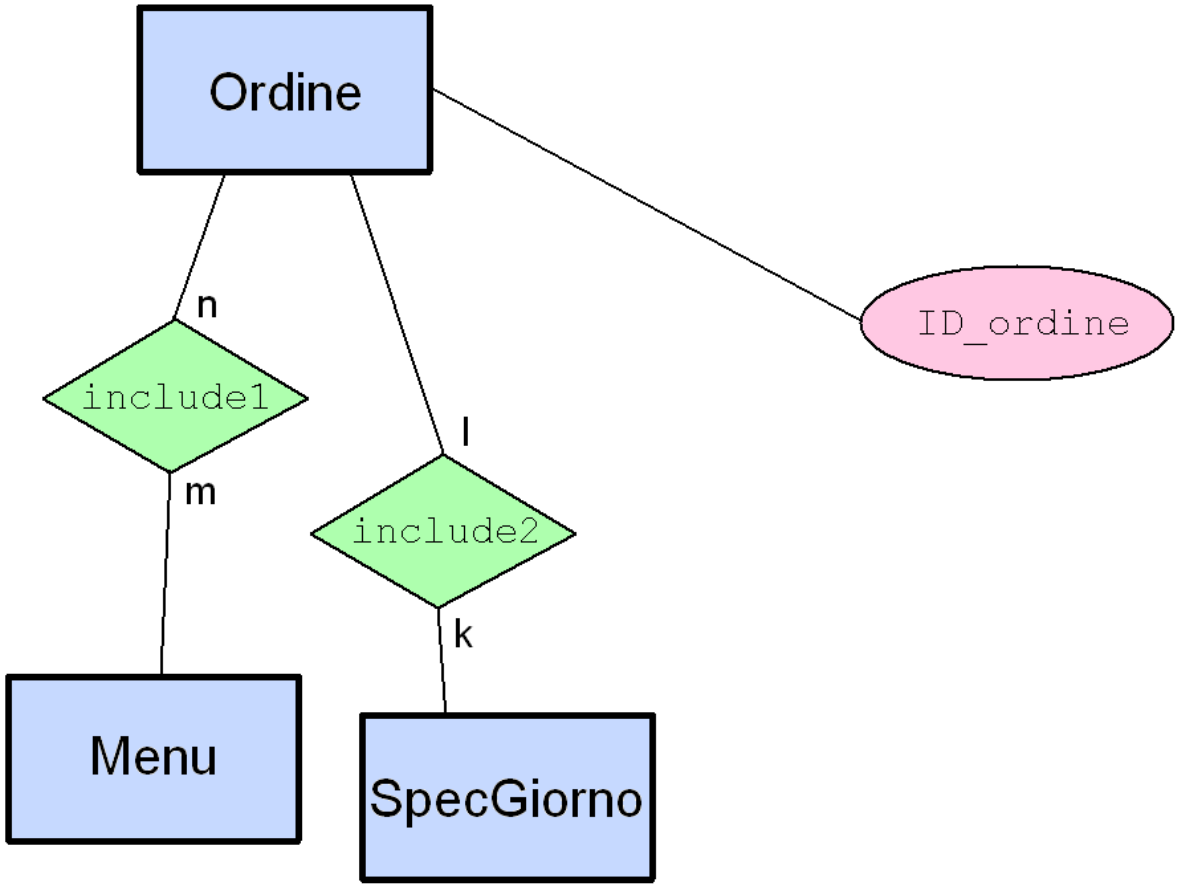

Figura: Schema dell'artefatto `Ordine`

Possibili Servizi relativi all'esempio scelto

crea_Guest_Check(): Questo servizio ha l'effetto di creare un `Guest Check`

crea_Menu(): Questo servizio ha l'effetto di creare un elemento del `Menu`

crea_SpecGiorno(): Questo servizio ha l'effetto di creare una specialità giornaliera

prepara_piatto(): Questo servizio istanzia un ordine per la cucina

aggiungi_piatto_Guest_Check(`Menu`: *m*, `SpecGiorno`: *s*, `Guest Check`: *g*): Questo servizio aggiunge il piatto del menu *m* o la specialità giornaliera *s* a *g*

chiudi_Guest_Check(`Guest Check`: *g*, `Cassa`: *c*): Questo servizio chiude *g* e aggiorna il saldo in *c*

Specifica dei Servizi (IOPE)

- artefatti e attributi di **I**nput,
- artefatti e attributi di **O**utput,
- **P**re-condizioni, e
- **E**ffetti (Condizionali).

Specifica IOPE di *aggiungi_piatto_Guest_Check*

- Input:
  - Una lista di elementi `Menu` *m* da aggiungere alla scheda.
  - Una lista di elementi `SpecGiorno` *s* da aggiungere alla scheda.
  - L'artefatto `Guest Check` *g* da modificare.
- Output:
  - Update di *g*
  - un nuovo ordine per la cucina *o*
- Pre-condizioni:
  - `Guest Check` è nello stato *attivo*.
- Effetti condizionali:
  - `true` $\rightarrow$ Le relazioni *include*1 e *include*2 di *g* vengono aggiornate per includere i nuovi piatti e specialità del giorno.
  - `true` $\rightarrow$ l'ordine *o* contiene la lista dei piatti e delle specialità ordinate

Specifica IOPE di *chiudi_Guest_Check*

- Input:
  - L'artefatto `Guest Check` *g* da chiudere.
  - La cassa
- Output:
  - Update del `Guest Check` *g* e della Cassa.
- Pre-condizioni:
  - `Guest Check` è nello stato *attivo*.
- Effetti condizionali:
  - `true` $\rightarrow$ *g* è nello stato `completato` e al bilancio cassa si aggiunge *g.conto*.

Componenti fondamentali di una regola ECA

- Eventi:
  - Un attributo appena assegnato
  - Un passaggio di stato per un artefatto
  - Il lancio o il completamento di un servizio
  - Una esplicita richiesta da parte di un esecutore
- Condizioni
  - Formule scritte in un sottoinsieme della Logica del Primo Ordine o in Algebra Relazionale
- Azioni
  - Invocazione di un Servizio
  - Cambio di stato per un Artefatto

Due esempi di regole

R1: *Aggiungi Piatto*

evento richiesta di aggiungere una lista di elementi `Menu` *m* e una lista di elementi `SpecGiorno` ***s*** al `Guest Check` *g*.

condizione *g* è nello stato `attivo`

azione **invoca** *aggiungi_piatto_Guest_Check*(*m*,*s*,*g*)

R2: *Chiudi Conto*

evento richiesta di chiudere il conto associato al `Guest Check` *g*.

condizione *g* è nello stato `attivo`

azione **invoca** *chiudi_Guest_Check*(*g*)

Semantica dell regole ECA

- **Non-determinismo:** Se più di una regola è candidata ad essere attivata da uno stesso evento, il sistema ne sceglie una non deterministicamente.
- **No starvation:** Una regola non può attendere di essere attivata indefinitamente.
- **Serializzabilità:** L'esecuzione delle regole può essere interleaved e/o parallela, ma l'effetto deve essere equivalente a quello di un qualche ordine seriale.

Verifiche fondamentali sul sistema

- **Raggiungibilità:** Dato un insieme di regole, esiste un predicato raggiungibile tramite un'esecuzione di tali regole?
- **Deadlock:** Il sistema può raggiungere un deadlock? In caso affermativo, la situazione di deadlock può essere evitata?
- **Terminazione:** Supponendo di avere artefatti con un ciclo di vita finito, ogni possibile esecuzione delle regole garantisce che gli artefatti si trovino nello stato finale dopo un numero finito di passi?

Verifiche sull'implementazione delle regole

- **Conformità:** Si può dimostrare che l'implementazione soddisfa tutti i requisiti definiti nella semantica (ad esempio assenza di starvation e serializzabilità)?
- **Ottimizzazione:** Come può essere implementato il sistema di regole affinchè non si testino inutilmente condizioni di attivazione di regole?