5

5.1 Introduzione: Un'illusione ottica **174**

5.2 Una panoramica dell'analisi **174**

5.3 Concetti di analisi **176**

5.3.1 Modelli di oggetti di analisi e modelli dinamici 176
5.3.2 Entità, confini e oggetti di controllo 177
5.3.3 Generalizzazione e specializzazione 178

5.4 Attività di analisi: Dai casi d'uso agli oggetti **179**

5.4.1 Identificare gli oggetti di entità 180
5.4.2 Identificazione degli oggetti di confine 182
5.4.3 Identificazione degli oggetti di controllo 184
5.4.4 Mappatura dei casi d'uso su oggetti con diagrammi di sequenza 185
5.4.5 Modellazione delle interazioni tra oggetti con le carte CRC 189
5.4.6 Identificare le associazioni 190
5.4.7 Identificazione degli aggregati 192
5.4.8 Identificare gli attributi 193
5.4.9 Modellazione del comportamento dei singoli oggetti dipendenti dallo Stato 194
5.4.10 Modellazione delle relazioni di ereditarietà tra gli oggetti 195
5.4.11 Revisione del modello di analisi 196
5.4.12 Riepilogo dell'analisi 197

5.5 Gestione dell'analisi **199**

5.5.1 Analisi della documentazione 199
5.5.2 Assegnazione delle responsabilità 200
5.5.3 Comunicare sull'analisi 201
5.5.4 Iterating sul modello di analisi 203
5.5.5 Segnaletica del cliente 204

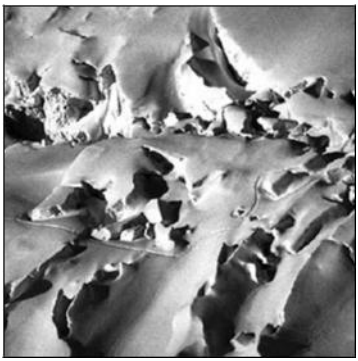
5.6 Caso di studio ARENA **206**

5.7 Ulteriori letture **218**

5.8 Esercizi **219**

Riferimenti **221**

5



Analisi

Sono Foo con un nome, se solo potessi ricordarlo.

-*Un* programmatore di pochissimo cervello

L'*analisi dà come* risultato un modello del sistema che mira ad essere corretto, completo, coerente e privo di ambiguità. Gli sviluppatori formalizzano le specifiche dei requisiti prodotti durante l'elaborazione dei requisiti ed esaminano in modo più dettagliato le condizioni al contorno e i casi eccezionali. Gli sviluppatori convalidano, correggono e chiariscono le specifiche dei requisiti in caso di errori o ambiguità. Il cliente e l'utente sono solitamente coinvolti in questa attività quando la specifica dei requisiti deve essere modificata e quando devono essere raccolte informazioni aggiuntive.

Nell'analisi orientata agli oggetti, gli sviluppatori costruiscono un modello che descrive il dominio dell'applicazione. Ad esempio, il modello di analisi di un orologio descrive come l'orologio rappresenta il tempo: L'orologio conosce gli anni bisestili? Conosce il giorno della settimana? Conosce le fasi lunari? Il modello di analisi viene poi esteso per descrivere come gli attori e il sistema interagiscono per manipolare il modello del dominio applicativo: Come fa il proprietario dell'orologio a resettare l'ora? Come fa il proprietario dell'orologio a resettare il giorno della settimana? Gli sviluppatori utilizzano il modello di analisi, insieme ai requisiti non funzionali, per preparare l'architettura del sistema sviluppato durante la progettazione di alto livello (Capitolo 6, *Progettazione del sistema: Decomposizione del sistema*).

In questo capitolo, si parla più in dettaglio delle attività di analisi. Ci concentriamo sull'identificazione degli oggetti, il loro comportamento, le loro relazioni, la loro classificazione e la loro organizzazione. Descriviamo le questioni di gestione relative all'analisi nel contesto di un progetto di sviluppo multiteam. Infine, discutiamo in modo più dettagliato le questioni di analisi e i compromessi utilizzando il caso studio di ARENA.

5.1 Introduzione: Un'illusione ottica

Nel 1915, Rubin espone un disegno simile alla Figura 5-1 per illustrare il concetto di immagini multistabili. Cosa vedete? Due volti che si guardano l'un l'altro? Se ci si concentra più da vicino sulla zona bianca, si vede invece un vaso. Una volta che si è in grado di percepire entrambe le forme singolarmente, è più facile passare avanti e indietro tra il vaso e i volti.

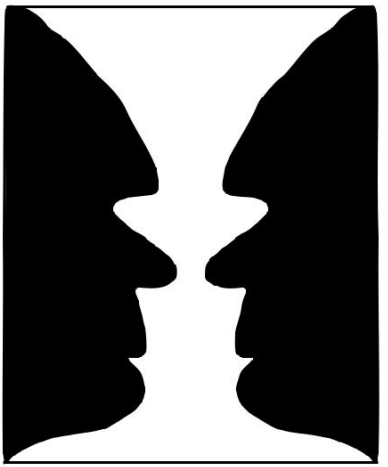


Figura 5-1 Ambiguità: cosa vedi?

Se il disegno in Figura 5-1 fosse stato una specifica dei requisiti, quali modelli avreste dovuto costruire? Le specifiche, come le immagini multistabili, contengono ambiguità causate dalle imprecisioni inerenti al linguaggio naturale e dai presupposti degli autori delle specifiche. Per esempio, una quantità specificata senza unità è ambigua (per esempio, l'esempio "Piedi o miglia?" nella Sezione 4.1), un tempo senza fuso orario è ambiguo (per esempio, la programmazione di una telefonata tra diversi paesi).

La formalizzazione aiuta a identificare le aree di ambiguità, nonché le incoerenze e le omissioni in una specifica dei requisiti. Una volta che gli sviluppatori identificano i problemi con la specifica, li risolvono ottenendo maggiori informazioni dagli utenti e dal cliente. La selezione e l'analisi dei requisiti sono attività iterative e incrementali che si verificano contemporaneamente.

5.2 Una panoramica dell'analisi

L'**analisi si** concentra sulla produzione di un modello del sistema, chiamato modello di analisi, che è corretto, completo, coerente e verificabile. L'analisi è diversa dall'elicitazione dei requisiti in quanto gli sviluppatori si concentrano sulla strutturazione e la formalizzazione dei requisiti richiesti dagli utenti (Figura 5-2). Questa formalizzazione porta a nuove intuizioni e alla scoperta di errori nel modello di analisi.

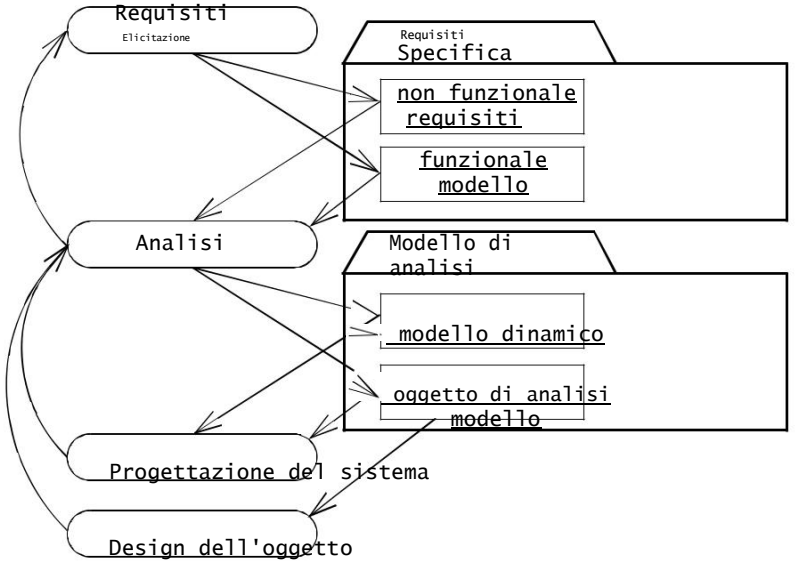


Figura 5-2 Prodotti dell'elicitazione e dell'analisi dei requisiti (diagramma di attività UML).

requisiti. Poiché il modello di analisi potrebbe non essere comprensibile per gli utenti e il cliente, gli sviluppatori devono aggiornare le specifiche dei requisiti per riflettere le intuizioni acquisite durante l'analisi, quindi rivedere i cambiamenti con il cliente e gli utenti. Alla fine, i requisiti, per quanto grandi, dovrebbero essere comprensibili per il cliente e gli utenti.

C'è una tendenza naturale per gli utenti e gli sviluppatori a rimandare le decisioni difficili ad una fase successiva del progetto. Una decisione può essere difficile a causa della mancanza di conoscenza del dominio, della mancanza di conoscenza tecnologica, o semplicemente a causa di disaccordi tra utenti e sviluppatori. Il rinvio delle decisioni consente al progetto di procedere senza intoppi ed evita il confronto con la realtà o con i colleghi. Purtroppo, le decisioni difficili alla fine devono essere prese, spesso a costi più elevati quando vengono scoperti problemi intrinseci durante i test, o peggio, durante la valutazione degli utenti. La traduzione di una specifica dei requisiti in un modello formale o semiformale obbliga gli sviluppatori a identificare e risolvere problemi difficili fin dalle prime fasi dello sviluppo.

Il **modello di analisi** è composto da tre modelli individuali: il **modello funzionale**, rappresentato da casi d'uso e scenari, il modello ad oggetti di **analisi**, **rappresentato da** diagrammi di classe e di oggetto, ed il **modello dinamico**, rappresentato da diagrammi di macchina di stato e di sequenza (Figura 5-3). Nel capitolo precedente, abbiamo descritto come ottenere i requisiti dagli utenti e descriverli come casi d'uso e scenari. In questo capitolo, si descrive come perfezionare il modello funzionale e derivare l'oggetto e il modello dinamico. Questo porta ad una specifica più precisa e completa man mano che i dettagli vengono aggiunti al modello di analisi. Concludiamo il capitolo descrivendo le attività di gestione relative all'analisi. Nella sezione successiva, definiamo i concetti principali dell'analisi.

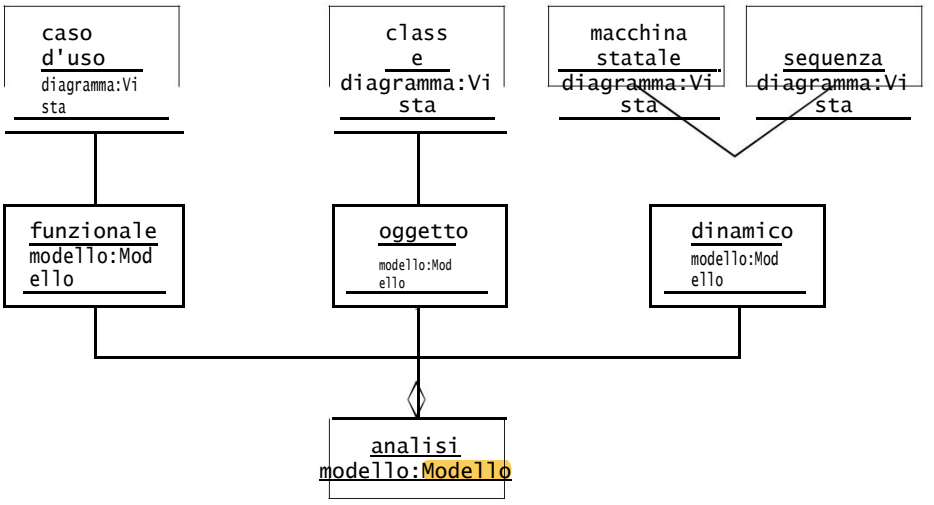


Figura 5-3 Il modello di analisi è composto dal modello funzionale, dal modello ad oggetti e dal modello dinamico. In UML, il modello funzionale è rappresentato con i diagrammi dei casi d'uso, il modello a oggetti con i diagrammi delle classi e il modello dinamico con i diagrammi della macchina di stato e i diagrammi di sequenza.

5.3 Concetti di analisi

In questa sezione descriviamo i principali concetti di analisi utilizzati in questo capitolo. In particolare, descriviamo

- Modelli di oggetti di analisi e modelli dinamici (Sezione 5.3.1)
- Entità, confine e oggetti di controllo (Sezione 5.3.2)
- Generalizzazione e specializzazione (Sezione 5.3.3).

5.3.1 Modelli di oggetti di analisi e modelli dinamici

Il modello di analisi rappresenta il sistema in fase di sviluppo dal punto di vista dell'utente. Il modello ad **oggetti di analisi** è una parte del modello di analisi e si concentra sui singoli concetti che vengono manipolati dal sistema, le loro proprietà e le loro relazioni. Il modello ad oggetto di analisi, rappresentato con diagrammi di classe UML, include classi, attributi e operazioni. Il modello a oggetti di analisi è un dizionario visivo dei concetti principali visibili all'utente.

Il **modello dinamico si** concentra sul comportamento del sistema. Il modello dinamico è rappresentato con diagrammi di sequenza e con macchine a stato. I diagrammi di sequenza rappresentano le interazioni tra un insieme di oggetti durante un singolo caso d'uso. Le macchine a stato rappresentano il comportamento di un singolo oggetto (o di un gruppo di oggetti molto strettamente accoppiati). Il modello dinamico serve ad assegnare responsabilità alle singole classi e, nel processo, ad identificare nuove classi, associazioni ed attributi da aggiungere al modello di oggetti di analisi.

Quando si lavora sia con il modello ad oggetti di analisi che con il modello dinamico, è essenziale ricordare che questi modelli rappresentano concetti a livello utente, non vere e proprie classi di software o

componenti. Per esempio, classi come `Database`, `Sottosistema`, `SessionManager`, `Network`, non dovrebbero apparire nel modello di analisi in quanto l'utente è completamente schermato da questi concetti. Si noti che la maggior parte delle classi nel modello ad oggetti di analisi corrisponderà ad una o più classi di software nel codice sorgente. Tuttavia, le classi di software includeranno molti più attributi e associazioni rispetto alle loro controparti di analisi. Di conseguenza, le classi di analisi dovrebbero essere considerate come astrazioni di alto livello che saranno realizzate in modo molto più dettagliato in seguito. La Figura 5-4 mostra esempi buoni e cattivi di oggetti di analisi per l'esempio di `SatWatch`.

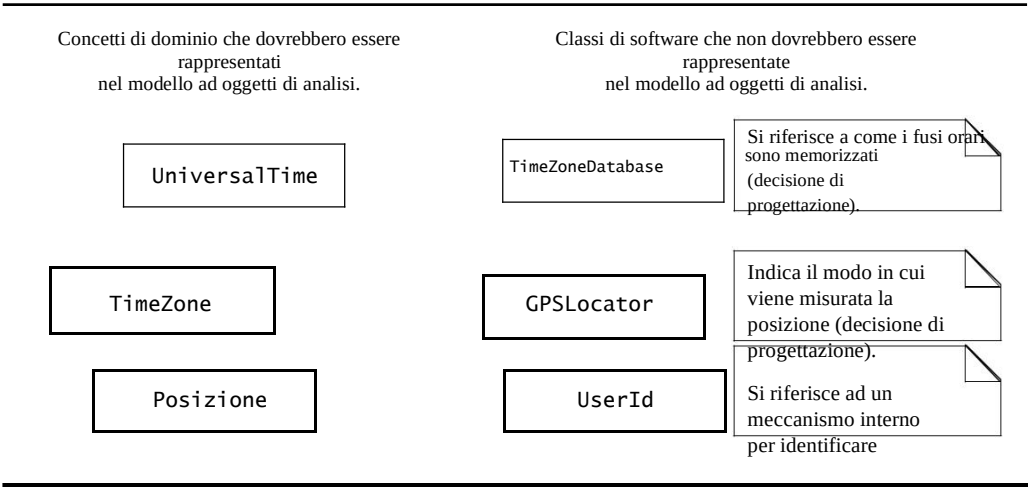


Figura 5-4 Esempi e controesempi di classi nel modello ad oggetti di analisi di `SatWatch`.

5.3.2 Entità, confine e oggetti di controllo

Il modello degli oggetti di analisi è costituito da oggetti entità, confine e controllo [Jacobson et al., 1999]. **Gli oggetti entità** rappresentano le informazioni persistenti tracciate dal sistema. **Gli oggetti limite** rappresentano le interazioni tra gli attori e il sistema. **Gli oggetti di controllo** sono incaricati di realizzare i casi d'uso. Nell'esempio di `2Bwatch`, `Anno`, `Mese` e `Giorno` sono oggetti entità; `Pulsante` e `LCDDisplay` sono oggetti limite; `ChangeDateControl` è un oggetto di controllo che rappresenta l'attività di cambiare la data premendo combinazioni di pulsanti.

La modellazione del sistema con entità, confini e oggetti di controllo fornisce agli sviluppatori una semplice euristica per distinguere concetti diversi, ma correlati. Per esempio, l'ora che viene tracciata da un orologio ha proprietà diverse rispetto al display che rappresenta l'ora. La differenziazione tra gli oggetti limite e gli oggetti entità costringe a tale distinzione: L'ora che viene tracciata dall'orologio è rappresentata dall'oggetto `Tempo`. Il display è rappresentato dall'`LCDDisplay`. Questo approccio con tre tipi di oggetti risulta in oggetti più piccoli e più specializzati. L'approccio con tre tipi di oggetti porta anche a modelli più resistenti ai cambiamenti: l'interfaccia con il sistema (rappresentata dagli oggetti limite) ha più probabilità di cambiare rispetto alla sua funzionalità di base (rappresentata dall'entità e dagli

oggetti di controllo). Separando l'interfaccia dalla funzionalità di base, siamo in grado di mantenere intatta la maggior parte di un modello quando, per esempio, l'interfaccia utente cambia, ma gli oggetti entità non lo fanno.

Per distinguere tra i diversi tipi di oggetti, UML fornisce il meccanismo dello stereotipo per consentire allo sviluppatore di allegare tali meta-informazioni agli elementi di modellazione. Ad esempio, nella Figura 5-5, si allega lo stereotipo `"controllo"` all'oggetto `ChangeDateControl`. `Oltre` agli stereotipi, possiamo anche usare le convenzioni di denominazione per chiarezza e consigliamo di distinguere i tre diversi tipi di oggetti su base sintattica: gli oggetti di controllo possono avere il suffisso `Control` allegato al loro nome; gli oggetti limite possono essere denominati per indicare chiaramente una caratteristica di interfaccia (ad esempio, includendo il suffisso `Form`, `Button`, `Display` o `Boundary`); gli oggetti entità di solito non hanno alcun suffisso allegato al loro nome. Un altro vantaggio di questa convenzione di denominazione è che il tipo di classe è rappresentato anche quando lo stereotipo UML non è disponibile, ad esempio, quando si esamina solo il codice sorgente.

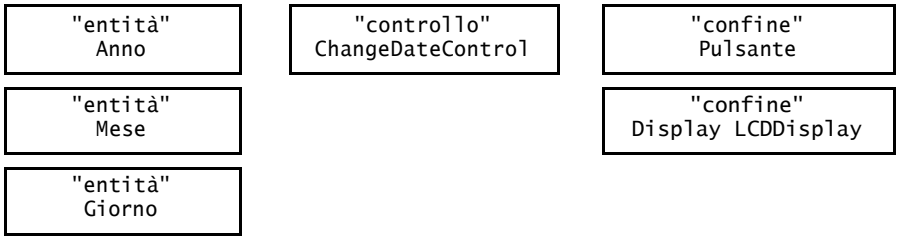


Figura 5-5 Classi di analisi per l'esempio `2Bwatch`.

5.3.3 Generalizzazione e specializzazione

Come abbiamo visto nel capitolo 2, *Modellazione con UML*, l'**ereditarietà** ci permette di organizzare i concetti in gerarchie. In cima alla gerarchia c'è un concetto generale (per esempio, un `Incidente`, Figura 5-6), e in fondo alla gerarchia ci sono i concetti più specializzati (per esempio, `CatInTree`, `TrafficAccident`, `BuildingFire`, `EarthQuake`, `ChemicalLeak`). Ci può essere un numero qualsiasi di livelli intermedi nel mezzo, che coprono concetti più o meno generalizzati (ad esempio, `LowPriorityIncident`, `Emergency`, `Disaster`). Tali gerarchie ci permettono di fare riferimento a molti concetti in modo preciso. Quando usiamo il termine `Incidente`, intendiamo tutti i casi di tutti i tipi di `Incidenti`. Quando usiamo il termine `Emergenza`, ci riferiamo solo a un `Incidente` che richiede una risposta immediata.

La generalizzazione è l'attività di modellazione che identifica concetti astratti da quelli di livello inferiore. Supponiamo, ad esempio, di stare realizzando il reverse-engineering di un sistema di gestione delle emergenze e di scoprire schermi per la gestione degli incidenti stradali e degli incendi. Notando le caratteristiche comuni di questi tre concetti, creiamo un concetto astratto chiamato `Emergenza` per descrivere le caratteristiche comuni (e generali) degli incidenti stradali e degli incendi.

La specializzazione è l'attività che identifica concetti più specifici a partire da uno di alto livello. Per esempio, supponiamo di costruire un sistema di gestione delle emergenze partendo da zero e di discuterne la funzionalità con il cliente. Il cliente ci presenta prima di tutto il

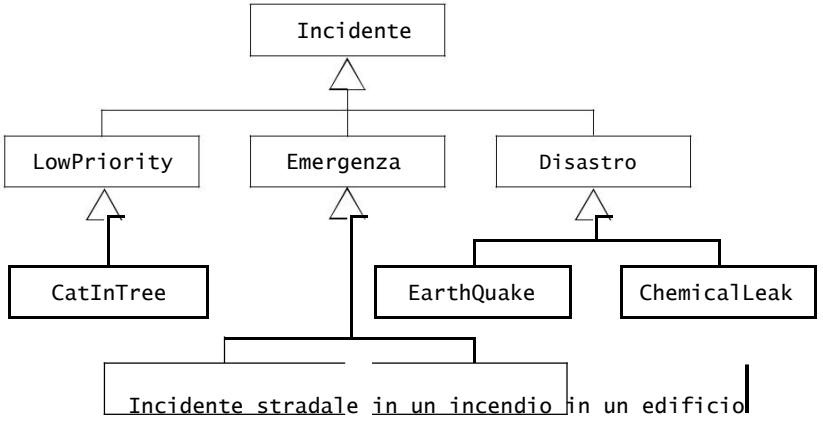


Figura 5-6 Un esempio di gerarchia di generalizzazione (diagramma di classe UML). La parte superiore della gerarchia rappresenta il concetto più generale, mentre i nodi inferiori rappresentano i concetti più specializzati.

concetto di incidente, quindi descrive tre tipi di `incidenti: Disastri`, che richiedono la collaborazione di diverse agenzie, `Emergenze`, che richiedono una gestione immediata ma possono essere gestiti da un'unica agenzia, e `Incidenti a Bassa Priorità`, che non devono essere gestiti se sono necessarie risorse per altri `Incidenti a` più alta priorità.

In entrambi i casi, la generalizzazione e la specializzazione portano alla specificazione di relazioni ereditarie tra i concetti. In alcuni casi, i modellisti chiamano le relazioni ereditarie relazioni di **generalizzazione-specializzazione.** In questo libro, usiamo il termine "ereditarietà" per indicare la relazione e i termini "generalizzazione" e "specializzazione" per indicare le attività che trovano relazioni ereditarie.

5.4 Attività di analisi: Dai casi d'uso agli oggetti

In questa sezione descriviamo le attività che trasformano i casi d'uso e gli scenari prodotti durante l'evocazione dei requisiti in un modello di analisi. Le attività di analisi includono:

- Identificare gli Oggetti Entità (Sezione 5.4.1)
- Identificazione degli oggetti di confine (Sezione 5.4.2)
- Identificazione degli oggetti di controllo (cap. 5.4.3)
- Mappatura dei casi d'uso su oggetti con diagrammi di sequenza (Sezione 5.4.4)
- Modellazione delle interazioni tra gli oggetti con le carte CRC (Sezione 5.4.5)
- Identificazione delle associazioni (Sezione 5.4.6)
- Identificazione degli aggregati (Sezione 5.4.7)
- Identificazione degli attributi (paragrafo 5.4.8)
- Modellazione del comportamento dei singoli oggetti dipendenti dallo Stato (Sezione 5.4.9)

- Modellizzazione dei rapporti di successione (Paragrafo 5.4.10)
- Revisione del modello di analisi (Sezione 5.4.11).

Illustriamo ogni attività concentrandoci sul caso d'uso `ReportEmergency` di `FRIEND` descritto nel Capitolo 4, *Requisiti Elicitazione*. Queste attività sono guidate dall'euristica. La qualità del loro risultato dipende dall'esperienza dello sviluppatore nell'applicazione di queste euristiche e metodi. I metodi e le euristiche presentati in questa sezione sono adattati da [De Marco, 1978], [Jacobson et al., 1999], [Rumbaugh et al., 1991], e [Wirfs-Brock et al., 1990].

5.4.1 Identificare gli oggetti di entità

Gli oggetti partecipanti (vedi Sezione 4.4.6) costituiscono la base del modello di analisi. Come descritto nel Capitolo 4, *Elicitazione dei requisiti*, gli oggetti partecipanti si trovano esaminando ogni caso d'uso e identificando gli oggetti candidati. L'analisi del linguaggio naturale [Abbott, 1983] è un insieme intuitivo di euristiche per identificare oggetti, attributi e associazioni a partire da una specifica dei requisiti. L'euristica di Abbott mappa parti del discorso (ad es. nomi, avere verbi, essere verbi, essere verbi, aggettivi) per modellare componenti (ad es. oggetti, operazioni, relazioni ereditarie, classi). La Tabella 5-1 fornisce esempi di tali mappature esaminando il caso d'uso `ReportEmergency` (Figura 5-7).

L'analisi del linguaggio naturale ha il vantaggio di concentrarsi sui termini degli utenti. Tuttavia, essa soffre di diversi limiti. In primo luogo, la qualità del modello a oggetti dipende molto dallo stile di scrittura dell'analista (ad esempio, la coerenza dei termini utilizzati, la verbalizzazione dei sostantivi). Il linguaggio naturale è uno strumento impreciso, e un modello a oggetti derivato letteralmente dal testo rischia di essere impreciso. Gli sviluppatori possono affrontare questa limitazione riformulando e chiarendo le specifiche dei requisiti mentre identificano e standardizzano oggetti e termini. Una seconda limitazione del linguaggio naturale

Tabella 5-1 L'euristica di Abbott per la mappatura di parti del discorso su componenti del modello [Abbott, 1983].

Parte del discorso	Componente del modello	Esempi
Nome proprio	Istanza	Alice
Nome comune	Classe	Ufficiale di campo
Fare verbo	Operazione	Crea, presenta, seleziona
Essere verbo	Eredità	E' una specie di, è uno dei due
Avere il verbo	Aggregazione	Ha, consiste di, include

Verbo modale	Vincoli	Deve essere
Aggettivo	Attributo	Descrizione dell'incidente

Utilizzare il nome del caso	`ReportEmergency`
Condizione di ingresso	1. Il `FieldOfficer` attiva la funzione "Segnala Emergenza" del suo terminale.
Flusso di eventi	2. 2. `L'AMICIENTE` risponde presentando un modulo all'ufficiale. Il modulo include un menu di tipo di emergenza (emergenza generale, incendio, trasporto), una località, una descrizione dell'incidente, la richiesta di risorse e i campi dei materiali pericolosi. 3. Il `FieldOfficer` compila il modulo specificando minimamente il tipo di emergenza e i campi di descrizione. Il `FieldOfficer` può anche descrivere le possibili risposte alla situazione di emergenza e richiedere risorse specifiche. Una volta completato il modulo, il `FieldOfficer` invia il modulo premendo il pulsante "Invia rapporto", a quel punto il `Dispatcher` viene avvisato. 4. Il `Dispatcher` esamina le informazioni inviate dal `FieldOfficer` e crea un `Incident` nel database invocando il caso d'uso `OpenIncident`. Tutte le informazioni contenute nel modulo del `FieldOfficer` sono automaticamente incluse nell'Incident. Il `Dispatcher` seleziona una risposta assegnando risorse all'incidente (con il caso d'uso `AllocateResources`) e riconosce il rapporto di emergenza inviando un `FRIENDgram` al `FieldOfficer`.
Condizione di uscita	5. Il `FieldOfficer` riceve il riconoscimento e la risposta selezionata.

Figura 5-7 Un esempio di caso d'uso, `ReportEmergency` (formato a una colonna).

L'analisi linguistica è che ci sono molti più sostantivi rispetto alle classi rilevanti. Molti nomi corrispondono ad attributi o sinonimi di altri nomi. L'ordinamento di tutti i sostantivi per una grande specificazione dei requisiti è un'attività che richiede tempo. In generale, l'euristica di Abbott funziona bene per generare una lista di oggetti candidati iniziali a partire da brevi descrizioni, come il flusso di eventi di uno scenario o un caso d'uso. Le seguenti euristiche possono essere utilizzate in combinazione con l'euristica di Abbott:

Euristica per l'identificazione di oggetti di entità

- Termini che gli sviluppatori o gli utenti devono chiarire per comprendere il caso d'uso
- Sostantivi ricorrenti nei casi d'uso (ad esempio, `Incidente`)
- Entità del mondo reale che il sistema deve monitorare (ad esempio `FieldOfficer`, `Dispatcher`, `Resource`)
- Attività del mondo reale che il sistema deve tenere sotto controllo (ad esempio, `EmergencyOperationsPlan`)
- Fonti di dati o lavelli (ad es. `stampante`).

Gli sviluppatori nominano e descrivono brevemente gli oggetti, i loro attributi e le loro responsabilità man mano che vengono identificati. Nominare gli oggetti in modo univoco promuove una terminologia standard. Per gli entity objects si consiglia di iniziare *sempre* con i nomi utilizzati dagli utenti finali e dagli specialisti del dominio delle applicazioni. Descrivere gli oggetti, anche brevemente, permette agli sviluppatori di chiarire i concetti che utilizzano ed evitare malintesi (ad

esempio, utilizzando un oggetto per due concetti diversi ma correlati). Gli sviluppatori non devono tuttavia dedicare molto tempo al dettaglio degli oggetti o

attributi, dato che il modello di analisi è ancora in corso. Gli sviluppatori dovrebbero documentare gli attributi e le responsabilità se non sono evidenti; altrimenti sono sufficienti un nome provvisorio e una breve descrizione per ogni oggetto. Ci saranno molte iterazioni durante le quali gli oggetti potranno essere rivisti. Tuttavia, una volta che il modello di analisi è stabile, la descrizione di ogni oggetto dovrebbe essere dettagliata quanto necessario (si veda la Sezione 5.4.11).

Ad esempio, dopo un primo esame del caso d'uso `ReportEmergency` (Figura 5-7), utilizziamo la conoscenza del dominio applicativo e le interviste con gli utenti per identificare gli oggetti `Dispatcher`, `EmergencyReport`, `FieldOfficer` e `Incident`. Si noti che l'oggetto `EmergencyReport` non è menzionato esplicitamente per nome nel caso d'uso `ReportEmergency`. Il passo 4 del caso d'uso si riferisce al report di emergenza come "informazioni inviate dal `FieldOfficer`". Dopo aver esaminato con il cliente, scopriamo che queste informazioni sono di solito indicate come "rapporto di emergenza" e decidiamo di nominare l'oggetto corrispondente `EmergencyReport`.

La definizione di oggetti entità porta al modello di analisi iniziale descritto nella Tabella 5-2. Si noti che questo modello è lontano da una descrizione completa del sistema che implementa il caso d'uso `ReportEmergency`. Nella sezione successiva, descriviamo l'identificazione degli oggetti limite.

Tabella 5-2 Oggetti entità per il caso d'uso `ReportEmergency`.

`Spedizioniere`	Agente di polizia che gestisce gli `Incidenti`. Un `dispacciatore` apre, documenta e chiude gli `incidenti` in risposta ai `rapporti di emergenza` e ad altre comunicazioni con `gli agenti sul campo`. `I dispacciatori` sono identificati da numeri di badge.
`EmergencyReport`	Rapporto iniziale su un `incidente` da un `FieldOfficer` a un `Dispatcher`. Un `EmergencyReport di` solito innesca la creazione di un `Incidente` da parte del `Spedizioniere`. Un `EmergencyReport` è composto da un livello di emergenza, un tipo (incendio, incidente stradale, altro), un luogo e una descrizione.
`FieldOfficer`	Polizia o vigili del fuoco in servizio. Un `FieldOfficer` può essere assegnato a, al massimo, un Un `incidente` alla volta. `Gli agenti sul campo` sono identificati da numeri di distintivo.
`Incidente`	Situazione che richiede attenzione da parte di un `agente sul campo`. Un `incidente` può essere segnalati nel sistema da un `FieldOfficer` o da chiunque altro esterno al sistema. Un `Incidente` è composto da una descrizione, una risposta, uno stato (aperto, chiuso, documentato), una località e un certo numero di `agenti sul campo`.

5.4.2 Identificazione degli oggetti di confine

Gli oggetti di confine rappresentano l'interfaccia del sistema con gli attori. In ogni caso d'uso, ogni attore interagisce con almeno un oggetto di confine. L'oggetto confine raccoglie le informazioni dall'attore e le traduce in una forma che può essere utilizzata sia dall'entità che dagli oggetti di controllo.

Gli oggetti di confine modellano l'interfaccia utente ad un livello grossolano. Non descrivono in dettaglio gli aspetti visivi dell'interfaccia utente. Ad esempio, gli oggetti limite come "voce di menu" o "barra di scorrimento" sono troppo dettagliati. In primo luogo, gli sviluppatori possono discutere più facilmente i dettagli dell'interfaccia utente con

schizzi e bozzetti. In secondo luogo, il design dell'interfaccia utente continua ad evolvere come conseguenza dei test di usabilità, anche dopo che le specifiche funzionali del sistema diventano stabili. L'aggiornamento del modello di analisi per ogni cambiamento dell'interfaccia utente richiede tempo e non produce alcun beneficio sostanziale.

Euristica per l'identificazione degli oggetti di confine

- Identificare i controlli dell'interfaccia utente di cui l'utente ha bisogno per avviare il caso d'uso (ad esempio, `ReportEmergencyButton`).
- Identificare i moduli di cui gli utenti hanno bisogno per inserire i dati nel sistema (ad es. `EmergencyReportForm`).
- Identificare gli avvisi e i messaggi che il sistema utilizza per rispondere all'utente (ad es,

- Quando più attori sono coinvolti in un caso d'uso, identificare i terminali degli attori (ad esempio, `DispatcherStation`) per fare riferimento all'interfaccia utente in esame.
- Non modellare gli aspetti visivi dell'interfaccia con gli oggetti di confine (i modelli utente sono più adatti a questo scopo).
- Utilizzare *sempre i* termini dell'utente finale per descrivere le interfacce; non utilizzare termini della soluzione o dei domini di implementazione.

Troviamo gli oggetti limite della Tabella 5-3 esaminando il caso d'uso `ReportEmergency`.

Tabella 5-3 Oggetti di confine per il caso d'uso `ReportEmergency`.

`RiconoscimentoNotifica`	Avviso utilizzato per la visualizzazione della conferma di ricezione del `Dispatcher` al `Agente sul campo`.
`DispatcherStation`	Computer utilizzato dal `Dispatcher`.
`ReportEmergencyButton`	Pulsante utilizzato da un `FieldOfficer` per avviare l'uso di `ReportEmergency` caso.
`EmergencyReportForm`	Modulo utilizzato per l'inserimento del `ReportEmergency`. Questo modulo è presentato al `FieldOfficer` sul `FieldOfficerStation` quando viene selezionata la funzione "Segnala emergenza". Il sito `EmergencyReportForm` contiene campi per specificare tutti gli attributi di un rapporto di emergenza e un pulsante (o altro controllo) per l'invio del modulo compilato.
`FieldOfficerStation`	Computer mobile utilizzato dal `FieldOfficer`.
`IncidentForm`	Modulo utilizzato per la creazione di `Incidenti`. Questo modulo viene presentato al

`DispatcherStation` sulla `DispatcherStation` quando l'`EmergencyReport`
è ricevuto. Il `Dispatcher` utilizza questo modulo anche per l'allocazione delle risorse
e di riconoscere il rapporto del `FieldOfficer`.

Si noti che l'`IncidentForm` non è esplicitamente menzionato da nessuna parte nel caso d'uso di `ReportEmergency`. Abbiamo identificato questo oggetto osservando che il `Dispacciatore ha` bisogno di un'interfaccia per visualizzare il rapporto di emergenza presentato dal `FieldOfficer` e per inviare un riconoscimento. I termini usati per descrivere gli oggetti di confine nel modello di analisi dovrebbero seguire la terminologia dell'utente, anche se si è tentati di usare termini del dominio di implementazione.

Abbiamo fatto progressi nella descrizione del sistema. Ora abbiamo incluso l'interfaccia tra l'attore e il sistema. Ci mancano però ancora alcuni pezzi significativi della descrizione, come l'ordine in cui avvengono le interazioni tra gli attori e il sistema. Nella sezione successiva, descriviamo l'identificazione degli oggetti di controllo.

5.4.3 Identificazione degli oggetti di controllo

Gli oggetti di controllo sono responsabili del coordinamento degli oggetti di confine e degli oggetti entità. Gli oggetti di controllo di solito non hanno una controparte concreta nel mondo reale. Spesso esiste una stretta relazione tra un caso d'uso e un oggetto di controllo; un oggetto di controllo viene solitamente creato all'inizio di un caso d'uso e cessa di esistere alla sua fine. È responsabile della raccolta di informazioni dagli oggetti di confine e della loro distribuzione agli oggetti entità. Per esempio, gli oggetti di controllo descrivono il comportamento associato alla sequenza delle forme, alle code di annullamenti e di storia e alla distribuzione delle informazioni in un sistema distribuito.

Inizialmente modelliamo il flusso di controllo del caso d'uso `ReportEmergency` con un oggetto di controllo per ogni attore: `ReportEmergencyControl` per il `FieldOfficer` e `ManageEmergency-Control` per il `Dispatcher`, rispettivamente (Tabella 5-4).

La decisione di modellare il flusso di controllo del caso d'uso `ReportEmergency` con due oggetti di controllo deriva dalla consapevolezza che la `FieldOfficerStation` e la `DispatcherStation` sono in realtà due sottosistemi che comunicano su un collegamento asincrono. Questa decisione avrebbe potuto essere rinviata fino all'attività di progettazione del sistema. D'altra parte, rendere visibile questo concetto nel modello di analisi ci permette di concentrarci su comportamenti di eccezione come la perdita di comunicazione tra le due stazioni.

Euristica per l'identificazione degli oggetti di controllo

- Identificare un oggetto di controllo per ogni caso d'uso.
- Identificare un oggetto di controllo per ogni attore nel caso d'uso.
- La durata di vita di un oggetto di controllo dovrebbe coprire l'estensione del caso d'uso o l'estensione di una sessione utente. Se è difficile identificare l'inizio e la fine dell'attivazione di un oggetto di controllo, il caso d'uso corrispondente probabilmente non ha condizioni di entrata e di uscita ben definite.

Nella modellazione del caso d'uso `ReportEmergency`, abbiamo modellato la stessa funzionalità utilizzando oggetti entità, confine e controllo. Passando dalla prospettiva del flusso

di eventi a quella strutturale, abbiamo aumentato il livello di dettaglio della descrizione e selezionato termini standard per fare riferimento alle principali entità del dominio applicativo e del sistema. Nella sezione successiva, abbiamo

Tabella 5-4 Oggetti di controllo per il caso d'uso ReportEmergency.

`ReportEmergencyControl`	Gestisce la funzione di reporting `ReportEmergency` sul `FieldOfficerStation`. Questo oggetto viene creato quando il `FieldOfficerStation` seleziona il pulsante "Segnala emergenza". Poi crea un `EmergencyReportForm` e lo presenta al `FieldOfficer`. Dopo inviando il modulo, questo oggetto raccoglie poi le informazioni dall'oggetto forma, crea un `rapporto di emergenza e` lo inoltra al `Dispatcher`. L'oggetto di controllo attende quindi il ritorno di un riconoscimento dalla `DispatcherStation`. Quando si riceve il riconoscimento, l'oggetto `ReportEmergencyControl` crea un `AcknowledgmentNotice` e lo visualizza al `FieldOfficer`.
`ManageEmergencyControl`	Gestisce la funzione di reporting `ReportEmergency` sul `DispatcherStation`. Questo oggetto viene creato quando un `EmergencyReport` è stato ricevuto. Crea quindi un `IncidentForm` e lo visualizza al `Dispatcher`. Una volta che il `Dispatcher` ha creato un `Incidente`, `risorse` assegnate, e ha presentato un riconoscimento, `ManageEmergencyControl` inoltra il riconoscimento al `FieldOfficerStation`.

costruire un diagramma di sequenza utilizzando il caso d'uso `ReportEmergency` e gli oggetti che abbiamo scoperto per garantire la completezza del nostro modello.

5.4.4 Mappatura dei casi d'uso su oggetti con diagrammi di sequenza

Un **diagramma di sequenza** lega le custodie con gli oggetti. Esso mostra come il comportamento di un caso d'uso (o scenario) è distribuito tra gli oggetti partecipanti. I diagrammi di sequenza non sono di solito un buon mezzo di comunicazione con l'utente come lo sono i casi d'uso, poiché i diagrammi di sequenza richiedono più background sulla notazione. Per i clienti esperti di computer, sono intuitivi e possono essere più precisi dei casi d'uso. In tutti i casi, tuttavia, i diagrammi di sequenza rappresentano un altro spostamento di prospettiva e permettono agli sviluppatori di trovare oggetti mancanti o aree grigie nelle specifiche dei requisiti.

In questa sezione modelliamo la sequenza delle interazioni tra gli oggetti necessari per realizzare il caso d'uso. Le figure da 5-8 a 5-10 sono diagrammi di sequenza associati al caso d'uso `ReportEmergency`. Le colonne di un diagramma di sequenza rappresentano gli oggetti che partecipano al caso d'uso. La colonna più a sinistra è l'attore che avvia il caso d'uso. Le frecce orizzontali sulle colonne rappresentano messaggi, o stimoli, che vengono inviati da un oggetto all'altro. Il tempo procede verticalmente dall'alto verso il basso. Per esempio, la prima freccia in Figura 5-8 rappresenta il messaggio di `stampa` inviato da un `FieldOfficer` ad un `ReportEmergencyButton`. La ricezione di un messaggio fa scattare l'attivazione di un'operazione. L'attivazione è rappresentata da un rettangolo verticale da cui possono provenire altri messaggi. La lunghezza del rettangolo rappresenta il tempo di attivazione dell'operazione. Nella Figura 5-

8, l'operazione attivata dal messaggio di `stampa` invia un messaggio di `creazione alla` classe `ReportEmergencyControl`. Un'operazione può essere pensata come un servizio che

l'oggetto fornisce ad altri oggetti. I diagrammi di sequenza rappresentano anche la durata di vita degli oggetti. Gli oggetti che esistono già prima dei primi stimoli nel diagramma di sequenza sono rappresentati in cima al diagramma. Gli oggetti che vengono creati durante l'interazione sono rappresentati con il messaggio `"crea"` che punta all'oggetto. Le istanze che vengono distrutte durante l'interazione hanno una croce che indica quando l'oggetto cessa di esistere. Tra il rettangolo che rappresenta l'oggetto e la croce (o la parte inferiore del diagramma, se l'oggetto sopravvive all'interazione), una linea tratteggiata rappresenta l'intervallo di tempo in cui l'oggetto può ricevere messaggi. L'oggetto non può ricevere messaggi al di sotto del segno della croce. Per esempio, in Figura 5-8 un oggetto della classe `ReportEmergencyForm` viene creato quando l'oggetto di `ReportEmergencyControl` invia il messaggio `"crea"` e viene distrutto una volta che l'`EmergencyReportForm è stato` inviato.

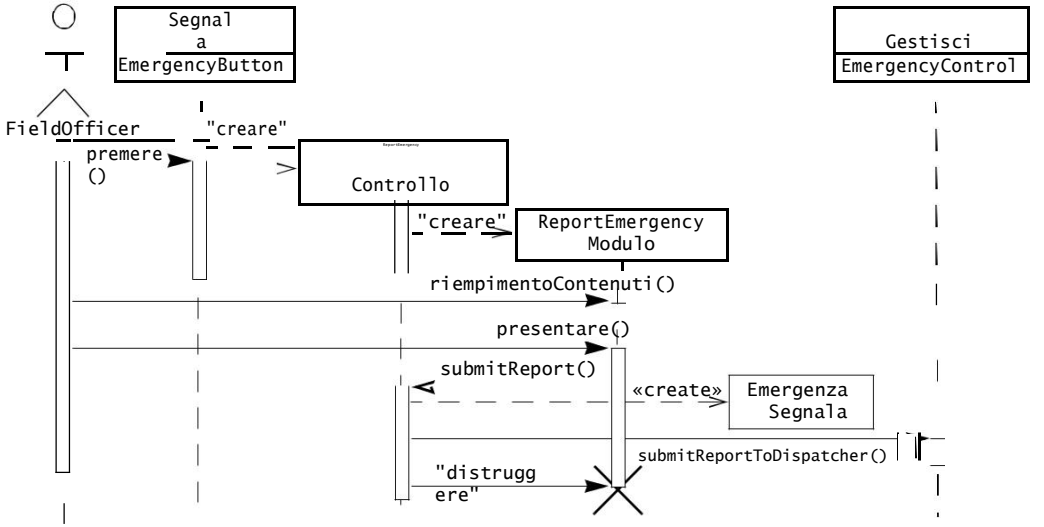


Figura 5-8 Diagramma di sequenza per il caso d'uso `ReportEmergency`.

In generale, la seconda colonna di un diagramma di sequenza rappresenta l'oggetto limite con cui l'attore interagisce per avviare il caso d'uso (ad esempio, `ReportEmergencyButton`). La terza colonna è un oggetto di controllo che gestisce il resto del caso d'uso (ad es. `ReportEmergency- Control`). Da quel momento in poi, l'oggetto di controllo crea altri oggetti di confine e può interagire anche con altri oggetti di controllo (ad esempio, `ManageEmergencyControl`).

Nella Figura 5-9, scopriamo l'oggetto entità `Acknowledgment` che abbiamo dimenticato durante l'esame iniziale del caso d'uso `ReportEmergency` (nella Tabella 5-2). L'oggetto `Acknowledgment` è diverso da un `AcknowledgmentNotice`: L'`Acknowledgment` contiene le informazioni associate ad un `Acknowledgment` e viene creato prima dell'oggetto limite `AcknowledgmentNotice`. Quando si descrive l'oggetto `Acknowledgment`, ci si rende conto anche che il caso d'uso `ReportEmergency` originale (descritto nella Figura 5-7) è incompleto. Esso menziona solo l'esistenza di un `Acknowledgment` e non descrive le informazioni ad esso associate.

In questo caso, gli sviluppatori hanno bisogno di un chiarimento da parte del cliente per definire quali informazioni sono necessarie nel

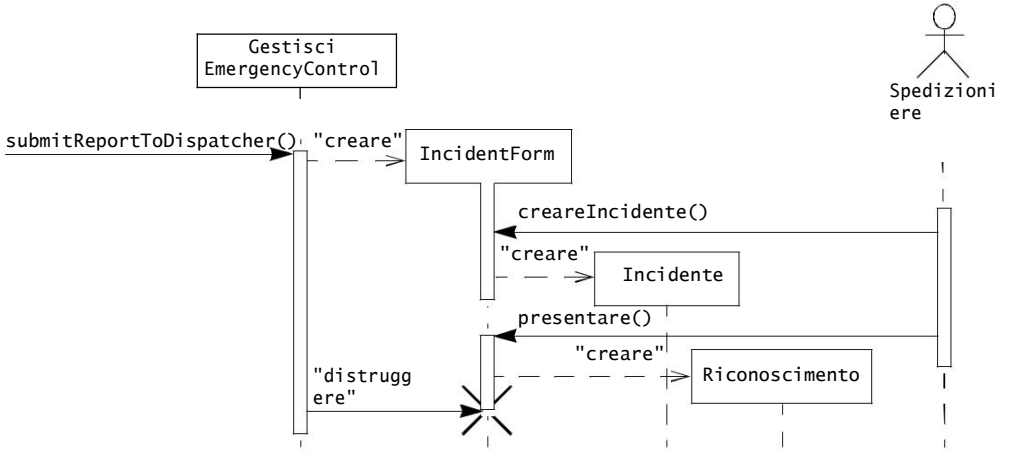


Figura 5-9 Diagramma di sequenza per il caso d'uso `ReportEmergency` (continua dalla Figura 5-8).

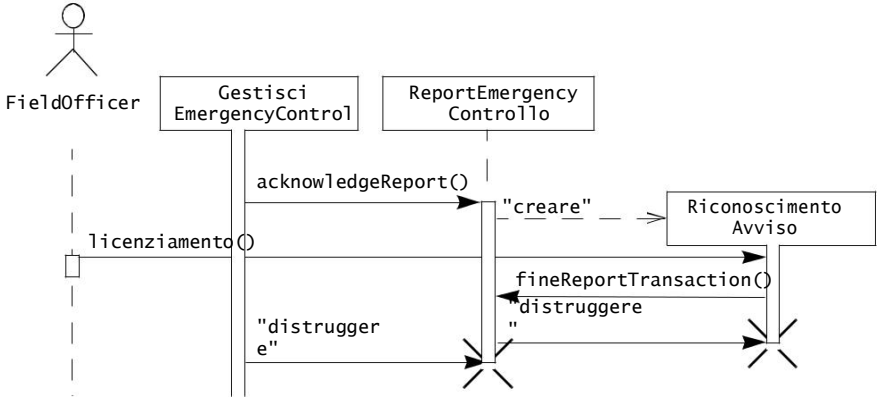


Figura 5-10 Diagramma di sequenza per il caso d'uso `ReportEmergency` (continua dalla Figura 5-9).

`Riconoscimento`. Dopo aver ottenuto tale chiarimento, l'oggetto `Acknowledgment` viene aggiunto al modello di analisi (Tabella 5-5), e il caso d'uso `ReportEmergency` viene chiarito per includere le informazioni aggiuntive (Figura 5-11).

Costruendo diagrammi di sequenza, non solo modelliamo l'ordine dell'interazione tra gli oggetti, ma distribuiamo anche il comportamento del caso d'uso. Cioè, assegniamo le responsabilità ad ogni oggetto sotto forma di un insieme di operazioni. Queste operazioni possono essere condivise da qualsiasi caso d'uso a cui un dato oggetto partecipa. Si noti che la definizione di un oggetto che è condiviso in due o più casi d'uso dovrebbe essere identica; cioè, se un'operazione appare in più di un diagramma di sequenza, il suo comportamento dovrebbe essere lo stesso.

Utilizzare il nome del caso	`ReportEmergency`
Condizione di ingresso	1. Il `FieldOfficer` attiva la funzione "Segnala Emergenza" del suo terminale.
Flusso di eventi	2. 2. `L'AMICIENTE` risponde presentando un modulo all'ufficiale. Il modulo include un menu del tipo di emergenza (emergenza generale, incendio, trasporto), un luogo, descrizione dell'incidente, richiesta di risorse e campi di materiali pericolosi. 3. Il `FieldOfficer` compila il modulo specificando minimamente il campi del tipo di emergenza e della descrizione. Il `FieldOfficer` può anche descrivere le possibili risposte alla situazione di emergenza e richiedere specifiche risorse. Una volta completato il modulo, il `FieldOfficer` invia il formulari premendo il pulsante "Invia rapporto", a quel punto il `Dispatcher` viene notificato. 4. Il `Dispacciatore` esamina le informazioni presentate dal `FieldOfficer` e crea un `Incidente` nel database invocando l'`OpenIncident` caso d'uso. Tutte le informazioni contenute nel modulo del `FieldOfficer` sono automaticamente incluso nell'`Incidente`. Il `Dispatcher` seleziona un risposta assegnando risorse all'`Incidente` (con la `AllocateResources` use case) e prende atto del rapporto di emergenza da parte di invio di un `FRIENDgram` al `FieldOfficer`. **Il `Riconoscimento` indica al `FieldOfficer` che è stato ricevuto il `rapporto di emergenza`,** **un `Incidente` creato, e le risorse assegnate all'`Incidente`. Il sito `Il riconoscimento` comprende le risorse (ad esempio, un camion dei pompieri) e le loro** **orario di arrivo previsto.**
Condizione di uscita	5. Il `FieldOfficer` riceve l'`Acknowledgment` e il selezionato risposta.

Figura 5-11 Caso d'uso di Refined `ReportEmergency`. La scoperta e l'aggiunta dell'oggetto `Acknowledgment` al modello di analisi ha rivelato che il caso d'uso `ReportEmergency` originale non descriveva accuratamente le informazioni associate agli `Acknowledgments`. I miglioramenti sono indicati in **grassetto**.

Tabella 5-5 Oggetto di `riconoscimento` per il caso d'uso `ReportEmergency`.

`Riconoscimento`	Risposta di un dispatcher al `FieldOfficer`'s `EmergencyReport`. Da inviando un `Riconoscimento`, il `Dispacciatore` comunica al `FieldOfficer` che ha ricevuto l'`EmergencyReport`, ha creato un `Incidente`, e gli sono state assegnate risorse. Il `Riconoscimento` contiene il

risorse assegnate e il loro orario di arrivo previsto.

La condivisione delle operazioni tra i vari casi d'uso consente agli sviluppatori di eliminare le ridondanze nelle specifiche dei requisiti e di migliorarne la coerenza. Si noti che la chiarezza dovrebbe sempre avere la precedenza sull'eliminazione della ridondanza. La frammentazione del comportamento in molte operazioni complica inutilmente le specifiche dei requisiti.

Nell'analisi, i diagrammi di sequenza sono usati per aiutare ad identificare nuovi oggetti partecipanti e comportamenti mancanti. Poiché i diagrammi di sequenza si concentrano sul comportamento di alto livello, l'implementazione

questioni come la performance non dovrebbero essere affrontate a questo punto. Dato che i diagrammi di interazione degli edifici possono richiedere molto tempo, gli sviluppatori dovrebbero prima di tutto concentrarsi su funzionalità problematiche o sotto specificate. Disegnare diagrammi di interazione per parti del sistema che sono semplici o ben definite potrebbe non sembrare un buon investimento di risorse di analisi, ma dovrebbe anche essere fatto per evitare di trascurare alcune decisioni chiave.

Euristica per il disegno di diagrammi di sequenza

- La prima colonna dovrebbe corrispondere all'attore che ha avviato il caso d'uso.
- La seconda colonna dovrebbe essere un oggetto di confine (che l'attore ha utilizzato per avviare il caso d'uso).
- La terza colonna dovrebbe essere l'oggetto di controllo che gestisce il resto del caso d'uso.
- Gli oggetti di controllo vengono creati da oggetti limite che avviano i casi d'uso.
- Gli oggetti limite sono creati da oggetti di controllo.
- Gli oggetti entità sono accessibili tramite oggetti di controllo e di confine.
- Gli oggetti entità *non* accedono *mai* agli oggetti di confine o di controllo; ciò rende più facile la condivisione degli oggetti entità tra i casi d'uso.

5.4.5 Modellazione delle interazioni tra oggetti con le carte CRC

Un'alternativa per identificare le interazioni tra gli oggetti sono le **carte CRC** [Beck & Cunningham, 1989]. Le schede CRC (CRC sta per classe, responsabilità e collaboratori) sono state inizialmente introdotte come strumento per insegnare concetti orientati agli oggetti a principianti e a sviluppatori esperti che non hanno familiarità con l'orientamento agli oggetti. Ogni classe è rappresentata con una scheda di indice (chiamata scheda CRC). Il nome della classe è rappresentato in alto, le sue responsabilità nella colonna di sinistra, e i nomi delle classi di cui ha bisogno per svolgere le sue responsabilità sono rappresentati nella colonna di destra. La Figura 5-12 mostra due schede per le classi `ReportEmergencyControl` e `Incident`.

Le schede CRC possono essere utilizzate durante le sessioni di modellazione con i team. I partecipanti, tipicamente un mix di sviluppatori ed esperti di dominio delle applicazioni, passano attraverso uno scenario e identificano le classi che sono coinvolte nella realizzazione dello scenario. Viene messa sul tavolo una scheda per ogni istanza. Responsabilità

ReportEmergencyControl

Responsabilità	**Collaboratori**
Raccoglie gli input dal campo-	EmergencyReportForm
ufficiale	EmergencyReport
Sequenza di controlli di moduli durante l'emergenza segnalazione	RiconoscimentoNotifica

Incidente

Responsabilità	**Collaboratori**
Traccia tutte le informazioni relativo ad un singolo inci- ammaccatura.	Risorsa

Figura 5-12 Esempi di schede CRC per le classi `ReportEmergencyControl` e `Incident`.

sono poi assegnati ad ogni classe man mano che lo scenario si sviluppa e i partecipanti negoziano le responsabilità di ogni oggetto. La colonna dei collaboratori viene riempita man mano che si identificano le dipendenze con altre carte. Le carte vengono modificate o spostate di lato mentre si esplorano nuove alternative. Le carte non vengono mai gettate via, perché i blocchi per le alternative del passato possono essere riutilizzati quando nuove idee vengono messe sul tavolo.

Le schede CRC e i diagrammi di sequenza sono due rappresentazioni diverse per supportare lo stesso tipo di attività. I diagrammi di sequenza sono uno strumento migliore per un singolo modellatore o per documentare una sequenza di interazioni, perché sono più precisi e compatti. Le schede CRC sono uno strumento migliore per un gruppo di sviluppatori che affinano e iterano su una struttura di oggetti durante una sessione di brainstorming, perché sono più facili da creare e modificare.

5.4.6 Identificazione delle associazioni

Mentre i diagrammi di sequenza permettono agli sviluppatori di rappresentare le interazioni tra gli oggetti nel tempo, i diagrammi di classe permettono agli sviluppatori di descrivere le interdipendenze degli oggetti. Abbiamo descritto la notazione del diagramma di classe UML nel capitolo 2, *Modellazione con UML,* e la usiamo in tutto il libro per rappresentare vari artefatti del progetto (ad esempio, attività, deliverable). In questa sezione, discutiamo l'uso dei diagrammi di classe per rappresentare le associazioni tra gli oggetti. Nella Sezione 5.4.8, si discute l'uso dei diagrammi di classe per rappresentare gli attributi degli oggetti.

Un'**associazione** mostra una relazione tra due o più classi. Per esempio, un `FieldOfficer` scrive un `rapporto di emergenza` (vedi Figura 5-13). Identificare le associazioni ha due vantaggi. In primo luogo, chiarisce il modello di analisi rendendo esplicite le relazioni tra gli oggetti (ad esempio, un `EmergencyReport` può essere creato da un `FieldOfficer` o da un `Dispatcher`). In secondo luogo, permette allo sviluppatore di scoprire i casi limite associati ai collegamenti. I casi limite sono eccezioni che devono essere chiarite nel modello. Per esempio, è intuitivo supporre che la maggior parte dei `rapporti EmergencyReport` siano scritti da un `FieldOfficer`. Tuttavia, il sistema dovrebbe supportare `EmergencyReports` scritti da più di un FieldOfficer? Il sistema dovrebbe consentire l'anonimato dei `rapporti EmergencyReports`? Queste domande dovrebbero essere analizzate durante l'analisi, discutendone con il cliente o con gli utenti finali.

Le associazioni hanno diverse proprietà:

- Un **nome** per descrivere l'associazione tra le due classi (ad esempio, `Writes` in Figura 5-13). I nomi delle associazioni sono facoltativi e non devono essere necessariamente unici a livello globale.

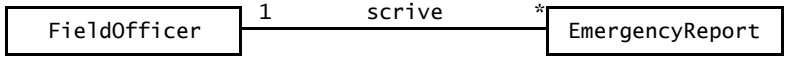


Figura 5-13 Un esempio di associazione tra le classi `EmergencyReport` e `FieldOfficer`.

- Un **ruolo** ad ogni estremità, identificando la funzione di ogni classe rispetto alle associazioni (ad esempio, l'`autore` è il ruolo svolto dal `FieldOfficer` nell'associazione `Writes`).
- Una **molteplicità** ad ogni estremità, che identifica il possibile numero di istanze (ad esempio, `*` indica che un `FieldOfficer` può scrivere zero o più `rapporti EmergencyReport`, mentre `1` indica che ogni rapporto `EmergencyReport` ha esattamente un `FieldOfficer` come autore).

Inizialmente, le associazioni tra gli oggetti entità sono le più importanti, in quanto rivelano maggiori informazioni sul dominio dell'applicazione. Secondo l'euristica di Abbott (vedi Tabella 5-1), le associazioni possono essere identificate esaminando i verbi e le frasi verbali che denota uno stato (ad esempio, *ha*, *è parte di*, *gestisce*, riferisce, è *attivato da, è contenuto in, parla con, include*). Ogni associazione dovrebbe essere nominata e i ruoli dovrebbero essere assegnati ad ogni fine.

Euristica per l'identificazione delle associazioni

- Esaminare le frasi dei verbi.
- Nominare le associazioni e i ruoli in modo preciso.
- Utilizzare i qualificatori il più spesso possibile per identificare i namespace e gli attributi chiave.
- Eliminare qualsiasi associazione che possa derivare da altre associazioni.
- Non preoccupatevi della molteplicità fino a quando l'insieme delle associazioni non è stabile.
- Troppe associazioni rendono illeggibile un modello.

Il modello a oggetti includerà inizialmente troppe associazioni se gli sviluppatori includeranno tutte le associazioni identificate dopo aver esaminato le frasi verbali. Nella Figura 5-14, per esempio, identifichiamo due relazioni: la prima tra un `Incidente` e l'`EmergencyReport` che ha innescato la sua creazione; la seconda tra l'`Incidente` e il `FieldOfficer di` segnalazione. Dato che `EmergencyReport` e `FieldOfficer` hanno già un'associazione che modella la paternità, l'associazione tra `Incident` e `FieldOfficer` non è necessaria. L'aggiunta di associazioni non necessarie complica il modello, portando a modelli incomprensibili e a informazioni ridondanti.

La maggior parte degli oggetti di entità hanno una caratteristica identificativa utilizzata dagli attori per accedervi. `I FieldOfficers e i Dispatchers` hanno un numero di badge. Gli `Incident e i Reports hanno un numero` assegnato e sono archiviati per data. Una volta che il modello di analisi include la maggior parte delle classi e delle associazioni, gli sviluppatori dovrebbero esaminare ogni classe e verificare come viene identificata dagli attori e in quale contesto. Ad esempio, i numeri dei distintivi `FieldOfficer` sono unici nell'universo? In tutta la città? In una stazione di polizia? Se sono unici in tutte le città, il sistema `FRIEND` può conoscere i `FieldOfficer di` più città? Questo approccio può essere formalizzato esaminando ogni singola

classe e identificando la sequenza di associazioni che devono essere attraversate per accedere a una specifica istanza di quella classe.

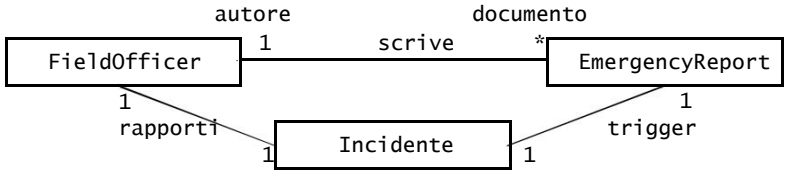


Figura 5-14 Eliminazione dell'associazione ridondante. La ricezione di un `EmergencyReport` fa scattare la creazione di un `Incident` da parte di un `Dispatcher`. Dato che `EmergencyReport` ha un'associazione con il `FieldOfficer` che l'ha scritto, non è necessario mantenere un'associazione tra `FieldOfficer` e `Incidente`.

5.4.7 Identificazione degli aggregati

Le **aggregazioni** sono tipi speciali di associazioni che denota un rapporto di tipo integrale. Per esempio, una `FireStation` è composta da un certo numero di `Vigili del Fuoco`, `FireEngine`, `Ambulanze` e una `LeadCar`. Uno `Stato` è composto da un certo numero di `Contee` che sono, a loro volta, composte da un certo numero di `Comuni` (Figura 5-15). Un'aggregazione è mostrata come un'associazione con un diamante sul lato dell'intera parte.

Ci sono due tipi di aggregazione, la composizione e la condivisione. Un diamante solido denota la composizione. Un'aggregazione di **composizione** indica che l'esistenza delle parti dipende dall'insieme. Per esempio, una `Contea fa` sempre parte esattamente di uno `Stato`, un `Comune` fa sempre parte di una `Contea`. Poiché i confini politici non cambiano spesso, un `Comune` non sarà parte di un'altra `Contea` o non sarà condiviso con un'altra Contea (almeno, nella vita del sistema di risposta alle emergenze).

Un diamante cavo denota una relazione di **aggregazione condivisa**, indicando il tutto e la parte possono esistere indipendentemente. Ad esempio, anche se un `FireEngine` fa parte di una sola `FireStation` al massimo, può essere riassegnato ad un'altra `FireStation` nel corso della sua vita.

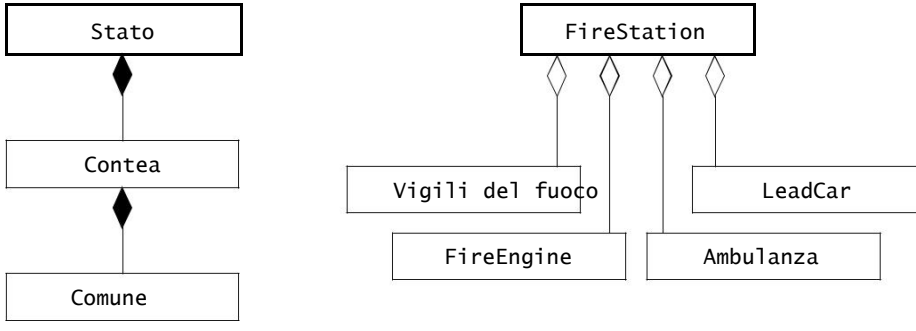


Figura 5-15 Esempi di aggregazioni e composizioni (diagramma di classe UML). Uno `Stato` è composto da molte `Contee`, che a loro volta sono composte da molti `Comuni`. Una `FireStation` include `FireFighters`, `FireEngines`, `Ambulanze` e una `LeadCar`.

Le associazioni di aggregazione sono utilizzate nel modello di analisi per indicare i concetti di parte intera. Le associazioni di aggregazione aggiungono informazioni al modello di analisi su come i concetti di contenimento nel dominio applicativo possono essere organizzati in una gerarchia o in un grafico diretto. Le aggregazioni sono spesso usate nell'interfaccia utente per aiutare l'utente a navigare attraverso molte istanze. Per esempio, nella Figura 5-15, `FRIEND` potrebbe offrire una rappresentazione ad albero per `i Dispatchers` per trovare le `Contee` all'interno di uno `Stato` o di un `Comune` con una specifica `Contea`. Tuttavia, come per molti concetti di modellazione, è facile sovrastrutturare il modello. Se non si è sicuri che l'associazione che si sta descrivendo sia un concetto completo, è meglio modellarla come associazione uno-a-molti, e rivisitarla in seguito quando si ha una migliore comprensione del dominio applicativo.

5.4.8 Identificare gli attributi

Gli attributi sono proprietà di singoli oggetti. Per esempio, un `EmergencyReport`, come descritto nella Tabella 5-2, ha un tipo di emergenza, una posizione e una proprietà di descrizione (vedi Figura 5-16). Questi sono inseriti da un `FieldOfficer` quando segnala un'emergenza e sono successivamente tracciati dal sistema. Quando si identificano le proprietà degli oggetti, si devono considerare solo gli attributi rilevanti per il sistema. Per esempio, ogni `FieldOfficer` ha un numero di sicurezza sociale che non è rilevante per il sistema informativo di emergenza. Invece, `i FieldOfficer` sono identificati da un numero di badge, che è rappresentato dalla proprietà `badgeNumber`.

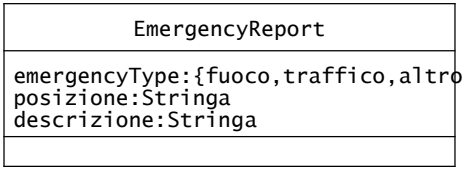


Figura 5-16 Attributi della classe `EmergencyReport`.

Le proprietà che sono rappresentate da oggetti non sono attributi. Per esempio, ogni `EmergencyReport` ha un autore che è rappresentato da un'associazione alla classe `FieldOfficer`. Gli sviluppatori dovrebbero identificare il maggior numero possibile di associazioni prima di identificare gli attributi per evitare di confondere gli attributi e gli oggetti. Gli attributi hanno:

- Un **nome che** li identifica all'interno di un oggetto. Per esempio, un `EmergencyReport` può avere un attributo `reportType` e un attributo `emergencyType`. Il `reportType` descrive il tipo di rapporto che viene archiviato (ad esempio, rapporto iniziale, richiesta di risorsa, rapporto finale). L'`emergencyType descrive` il tipo di emergenza (ad es. incendio, traffico, altro). Per evitare confusione, questi attributi non dovrebbero essere chiamati entrambi `tipi`.

- Una breve descrizione.

- Un **tipo che** descrive i valori legali che può assumere. Ad esempio, l'attributo di `descrizione di` un `EmergencyReport` è una stringa. L'attributo `emergencyType` è un'enumerazione che può assumere uno dei tre valori: `incendio`, `traffico`, `altro`. I tipi di attributo si basano su tipi di base predefiniti in UML.

Gli attributi possono essere identificati utilizzando l'euristica di Abbott (vedi Tabella 5-1). In particolare, si dovrebbe esaminare una frase sostantiva seguita da una frase possessiva (ad esempio, la descrizione di un'emergenza) o da una frase aggettivale (ad esempio, la descrizione dell'emergenza). Nel caso di oggetti di entità, qualsiasi proprietà che deve essere memorizzata dal sistema è un attributo candidato.

Si noti che gli attributi rappresentano la parte meno stabile del modello a oggetti. Spesso, gli attributi vengono scoperti o aggiunti in ritardo nello sviluppo quando il sistema viene valutato dagli utenti. A meno che gli attributi aggiunti non siano associati a funzionalità aggiuntive, gli attributi aggiunti non comportano grandi cambiamenti nella struttura dell'oggetto (e del sistema). Per queste ragioni, gli sviluppatori non hanno bisogno di spendere risorse eccessive per identificare e dettagliare gli attributi che rappresentano aspetti meno importanti del sistema. Questi attributi possono essere aggiunti in seguito, quando il modello di analisi o gli schizzi dell'interfaccia utente vengono convalidati.

Euristica per l'identificazione degli attributi
- Esaminare le frasi possessive.
- Rappresentare lo stato memorizzato come un attributo dell'oggetto entità.
- Descrivere ogni attributo.
- Non rappresentare un attributo come oggetto; utilizzare invece un'associazione (vedi Sezione 5.4.6).
- Non perdere tempo a descrivere i dettagli prima che la struttura dell'oggetto sia stabile.

a.Adattato da [Rumbaugh et al., 1991].

5.4.9 Modellazione del comportamento dei singoli oggetti dipendenti dallo Stato

I diagrammi di sequenza sono utilizzati per distribuire il comportamento tra gli oggetti e per identificare le operazioni. I diagrammi di sequenza rappresentano il comportamento del sistema dal punto di vista di un singolo caso d'uso. I diagrammi di macchina di stato rappresentano il comportamento dal punto di vista di un singolo oggetto. La visualizzazione del comportamento dal punto di vista di ogni oggetto permette allo sviluppatore di costruire una descrizione più formale del comportamento dell'oggetto e, di conseguenza, di identificare i casi d'uso mancanti. Concentrandosi sui singoli stati, gli sviluppatori possono identificare nuovi comportamenti. Per esempio, esaminando ogni transizione nel diagramma della macchina di stato che è innescata da un'azione dell'utente, lo sviluppatore dovrebbe essere in grado di identificare un passo di flusso in un caso d'uso che descrive l'azione dell'attore che innesca la transizione. Si noti che non è necessario costruire macchine a stato per ogni classe del sistema. Vale la pena

considerare solo oggetti con una durata di vita estesa e un comportamento dipendente dallo stato. Questo è quasi sempre il caso degli oggetti di controllo, meno spesso degli oggetti entità, e quasi mai degli oggetti di confine.

La Figura 5-17 mostra una macchina di stato per la classe `Incidente`. L'esame di questa macchina di stato può aiutare lo sviluppatore a verificare se ci sono casi d'uso per documentare, chiudere e archiviare gli `Incidenti`. Raffinando ulteriormente ogni stato, lo sviluppatore può aggiungere dettagli alle diverse azioni dell'utente che cambiano lo stato di un incidente. Per esempio, durante lo stato `Attivo` di un indicatore, `i FieldOfficers` dovrebbero essere in grado di richiedere nuove risorse, e `i Dispatchers` dovrebbero essere in grado di allocare risorse agli incidenti esistenti.

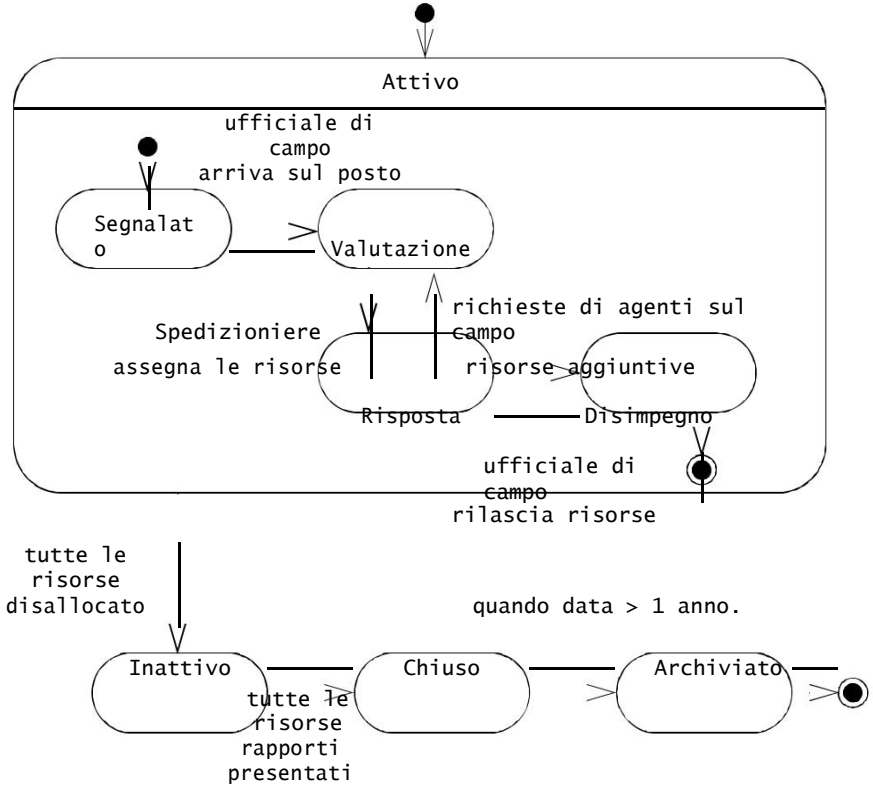


Figura 5-17 Macchina a stato UML per `Incident`.

5.4.10 Modellazione delle relazioni di ereditarietà tra gli oggetti

La generalizzazione è utilizzata per eliminare la ridondanza dal modello di analisi. Se due o più classi condividono attributi o comportamenti, le somiglianze si consolidano in una superclasse. Ad esempio, sia i `Dispatcher` che `i FieldOfficer` hanno un attributo `badgeNumber` che serve ad

identificarli all'interno di una città. `FieldOfficers` e `Dispatchers` sono entrambi `PoliceOfficers` a cui vengono assegnate funzioni diverse. Per modellare esplicitamente questa somiglianza, introduciamo una classe astratta *`PoliceOfficer`* da cui le classi `FieldOfficer` e `Dispatcher` ereditano (vedi Figura 5-18).

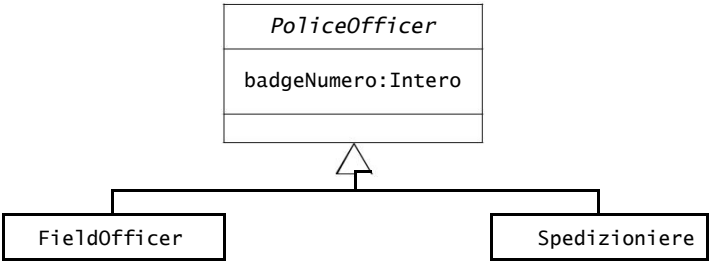


Figura 5-18 Un esempio di relazione ereditaria (diagramma di classe UML).

5.4.11 Revisione del modello di analisi

Il modello di analisi è costruito in modo incrementale e iterativo. Il modello di analisi è raramente corretto o addirittura completo al primo passaggio. Sono necessarie diverse iterazioni con il cliente e l'utente prima che il modello di analisi converga verso una corretta specifica utilizzabile dagli sviluppatori per la progettazione e l'implementazione. Per esempio, un'omissione scoperta durante l'analisi porterà all'aggiunta o all'estensione di un caso d'uso nella specifica dei requisiti, che può portare ad ottenere maggiori informazioni dall'utente.

Una volta che il numero di modifiche al modello è minimo e la portata delle modifiche localizzate, il modello di analisi diventa stabile. Poi il modello di analisi viene rivisto, prima dagli sviluppatori (cioè le revisioni interne), poi congiuntamente dagli sviluppatori e dal cliente. L'obiettivo della revisione è di assicurarsi che le specifiche dei requisiti siano corrette, complete, coerenti e non ambigue. Inoltre, gli sviluppatori e il cliente verificano anche se i requisiti sono realistici e verificabili. Si noti che gli sviluppatori devono essere pronti a scoprire gli errori a valle e ad apportare modifiche alle specifiche. Tuttavia, vale la pena di investire per individuare il maggior numero possibile di errori di requisiti a monte. La revisione può essere facilitata da una lista di controllo o da una lista di domande. Di seguito sono riportate domande di esempio adattate da [Jacobson et al., 1999] e [Rumbaugh et al., 1991].

Per garantire che il modello sia *corretto, è* necessario porre le seguenti domande:

- Il glossario degli oggetti entità è comprensibile per l'utente?
- Le classi astratte corrispondono ai concetti a livello utente?
- Tutte le descrizioni sono conformi alle definizioni degli utenti?
- Tutte le entità e gli oggetti di confine hanno come nomi frasi sostantivi significativi?
- Tutti i casi d'uso e gli oggetti di controllo hanno come nomi frasi verbali significative?
- Tutti i casi di errore sono descritti e gestiti?

Per garantire la *completezza del* modello è necessario porre le seguenti domande:

- Per ogni oggetto: È necessario per ogni caso d'uso? In quale caso d'uso viene creato? modificato? distrutto? È possibile accedervi da un oggetto di confine?

- Per ogni attributo: Quando è impostato? Qual è il suo tipo? Dovrebbe essere un qualificatore?
- Per ogni associazione: Quando viene attraversata? Perché è stata scelta la molteplicità specifica? Si possono qualificare le associazioni con una a molte e molteplici molteplicità?
- Per ogni oggetto di controllo: Ha le associazioni necessarie per accedere agli oggetti che partecipano al suo corrispondente caso d'uso?

Per garantire la *coerenza del* modello, è necessario porre le seguenti domande:

- Ci sono più classi o casi d'uso con lo stesso nome?
- Le entità (ad es. casi d'uso, classi, attributi) con nomi simili indicano concetti simili?
- Ci sono oggetti con attributi e associazioni simili che non si trovano nella stessa gerarchia di generalizzazione?

Le seguenti domande dovrebbero essere poste per garantire che il sistema descritto dal modello di analisi sia *realistico*:

- Ci sono novità nel sistema? Sono stati costruiti studi o prototipi per garantirne la fattibilità?
- È possibile soddisfare i requisiti di prestazioni e affidabilità? Questi requisiti sono stati verificati da eventuali prototipi in esecuzione sull'hardware selezionato?

5.4.12 Riepilogo dell'analisi

L'attività di sollecitazione dei requisiti è altamente iterativa e incrementale. I pezzi di funzionalità vengono abbozzati e proposti agli utenti e al cliente. Il cliente aggiunge requisiti, critica le funzionalità esistenti e modifica i requisiti esistenti. Gli sviluppatori studiano i requisiti non funzionali attraverso la prototipazione e gli studi tecnologici e sfidano ogni requisito proposto. Inizialmente, l'evocazione dei requisiti assomiglia ad un'attività di brainstorming. Man mano che la descrizione del sistema cresce e i requisiti diventano più concreti, gli sviluppatori devono estendere e modificare il modello di analisi in modo più ordinato per gestire la complessità delle informazioni.

La Figura 5-19 mostra una tipica sequenza delle attività di analisi. Gli utenti, gli sviluppatori e il cliente sono coinvolti nello sviluppo di un modello di caso d'uso iniziale. Essi identificano una serie di concetti e costruiscono un glossario degli oggetti partecipanti. Queste prime due attività sono state discusse nel capitolo precedente. Le altre attività sono state trattate in questa sezione. Gli sviluppatori classificano gli oggetti partecipanti in entità, confine e oggetti di controllo (in *Definire gli oggetti entità,* Sezione 5.4.1, *Definire gli oggetti confine*, Sezione 5.4.2, e *Definire gli oggetti di controllo, Sezione 5.4.*3). Queste attività si svolgono in un anello stretto fino a quando la maggior parte delle funzionalità del sistema non è stata identificata come casi d'uso con nomi e brevi descrizioni. Poi gli sviluppatori costruiscono diagrammi di sequenza per identificare eventuali oggetti mancanti (*Definire le interazioni*, Sezione 5.4.4). Quando tutti gli oggetti entità sono stati nominati e descritti brevemente, il modello di analisi dovrebbe rimanere abbastanza stabile man mano che viene perfezionato.

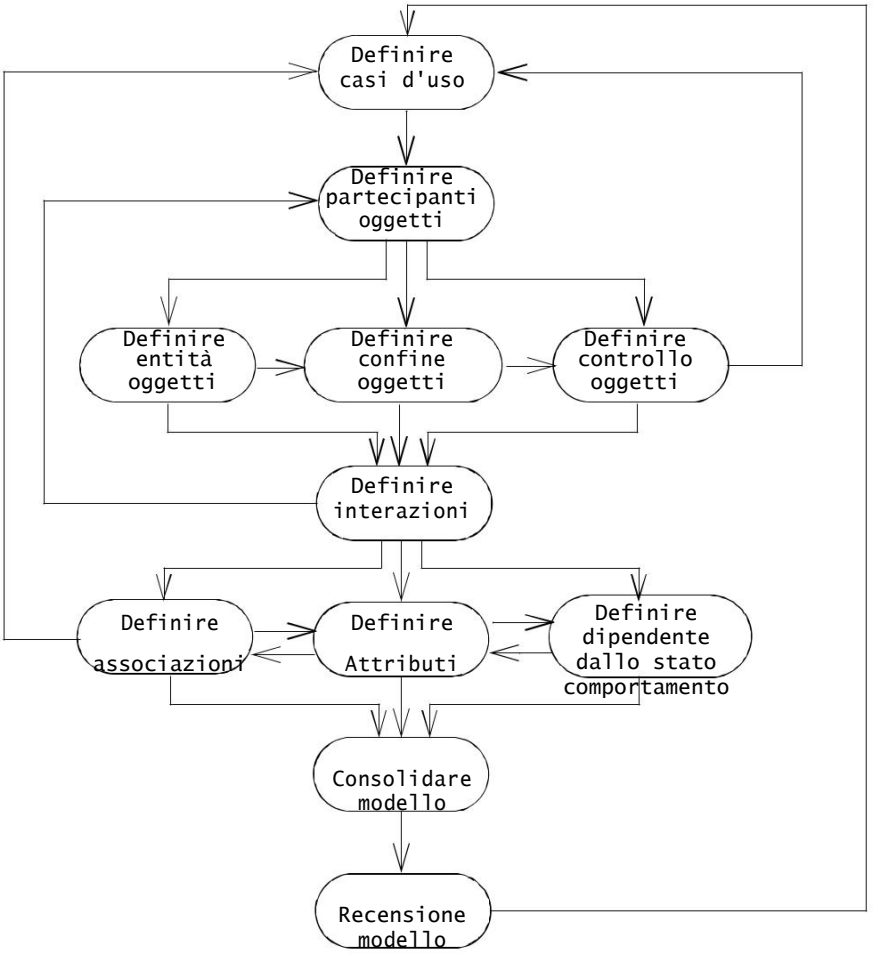


Figura 5-19 Attività di analisi (diagramma delle attività UML).

Definire le associazioni (Sezione 5.4.6), Definire *gli attributi* (Sezione 5.4.8) e *Definire il comportamento dipendente dallo stato* (Sezione 5.4.9) costituiscono il perfezionamento del modello di analisi. Queste tre attività si svolgono in un ciclo serrato durante il quale lo stato degli oggetti e le loro associazioni vengono estratti dai diagrammi di sequenza e dettagliati. I casi d'uso vengono poi modificati per tener conto di eventuali cambiamenti di funzionalità. Questa fase può portare all'identificazione di un'ulteriore porzione di funzionalità sotto forma di ulteriori casi d'uso. Il processo complessivo viene poi ripetuto incrementalmente per questi nuovi casi d'uso.

Durante il *modello Consolidate* (Paragrafo 5.4.10), gli sviluppatori consolidano il modello introducendo qualificatori e relazioni di generalizzazione e sopprimendo le

ridondanze. Durante il *modello Review* (Sezione 5.4.11), il cliente, gli utenti e gli sviluppatori esaminano il modello per

correttezza, coerenza, completezza e realismo. Il programma del progetto dovrebbe prevedere revisioni multiple per garantire requisiti di alta qualità e per fornire opportunità di apprendere l'attività sui requisiti. Tuttavia, una volta che il modello raggiunge il punto in cui la maggior parte delle modifiche sono di tipo estetico, la progettazione del sistema dovrebbe procedere. Ci sarà un punto durante i requisiti in cui non si possono prevedere ulteriori problemi senza ulteriori informazioni dalla prototipazione, studi di usabilità, indagini tecnologiche o progettazione del sistema. Ottenere ogni dettaglio corretto diventa un esercizio dispendioso: alcuni di questi dettagli diventeranno irrilevanti al prossimo cambiamento. Il management dovrebbe riconoscere questo punto e avviare la fase successiva del progetto.

5.5 Gestione dell'analisi

In questa sezione vengono affrontate le questioni relative alla gestione delle attività di analisi in un progetto di sviluppo multiteam. La sfida principale nella gestione dei requisiti in un progetto di questo tipo è quella di mantenere la coerenza, pur utilizzando così tante risorse. Alla fine, il documento di analisi dei requisiti dovrebbe descrivere un unico sistema coerente e comprensibile per una singola persona.

Per prima cosa descriviamo un modello di documento che può essere utilizzato per documentare i risultati dell'analisi (Sezione 5.5.1). Successivamente, descriviamo l'assegnazione del ruolo all'analisi (Sezione 5.5.2). Poi affrontiamo i problemi di comunicazione durante l'analisi. Successivamente, si affrontano le questioni di gestione relative alla natura iterativa e incrementale dei requisiti (Sezione 5.5.4).

5.5.1 Analisi della documentazione

Come abbiamo visto nel capitolo precedente, l'attività di selezione e di analisi dei requisiti è documentata nel documento di analisi dei requisiti (RAD, Figura 5-20). Le sezioni RAD da 1 a 3.5.2 sono già state scritte durante l'elaborazione dei requisiti. Durante l'analisi, rivediamo queste sezioni come ambiguità e nuove funzionalità vengono scoperte. Lo sforzo principale, tuttavia, si concentra sulla scrittura delle sezioni che documentano il modello oggetto di analisi (sezioni RAD 3.5.3 e 3.5.4).

RAD Sezione 3.5.3, *Modelli di oggetti*, documenta in dettaglio tutti gli oggetti che abbiamo identificato, i loro attributi e, quando abbiamo usato i diagrammi di sequenza, le operazioni. Poiché ogni oggetto è descritto con definizioni testuali, le relazioni tra gli oggetti sono illustrate con diagrammi di classe.

La sezione RAD 3.5.4, *Modelli dinamici*, documenta il comportamento del modello ad oggetti in termini di diagrammi macchina di stato e diagrammi di sequenza. Sebbene queste informazioni siano ridondanti con il modello dei casi d'uso, i modelli dinamici ci permettono di rappresentare in modo più preciso i comportamenti complessi, compresi i casi d'uso che coinvolgono molti attori.

Il RAD, una volta completato e pubblicato, sarà basato e messo sotto gestione della configurazione. La sezione cronologia delle revisioni del RAD fornirà una cronologia delle modifiche, con l'indicazione dell'autore responsabile di ogni modifica, la data della modifica e una breve descrizione della modifica.

Documento di analisi dei requisiti

1. Introduzione
2. Sistema attuale
3. 3. Sistema
   proposto 3.1
   Panoramica
   3.2 Requisiti funzionali
   3.3 Requisiti non funzionali
   3.4 Modelli di sistema
      3.4.1 Scenari
      3.4.2 Modello del caso d'impiego
      3.4.3 Modello dell'oggetto
         3.4.3.1 Dizionario dei dati
         3.4.3.2 Diagrammi di classe
      3.4.4 Modelli dinamici
      3.4.5 Interfaccia utente - Percorsi di navigazione e mock-up dello schermo
4. Glossario

Figura 5-20 Schema generale del documento di analisi dei requisiti (RAD). Vedere la Figura 4-16 per uno schema dettagliato.

5.5.2 Assegnazione delle responsabilità

L'analisi richiede la partecipazione di una vasta gamma di individui. L'utente target fornisce la conoscenza del dominio dell'applicazione. Il cliente finanzia il progetto e coordina il lato utente. L'analista ottiene la conoscenza del dominio applicativo e la formalizza. Gli sviluppatori forniscono un feedback sulla fattibilità e sui costi. Il project manager coordina lo sforzo sul lato dello sviluppo. Per i sistemi di grandi dimensioni, molti utenti, analisti e sviluppatori possono essere coinvolti, introducendo ulteriori sfide durante l'integrazione e i requisiti di comunicazione del progetto. Queste sfide possono essere affrontate assegnando ruoli e scopi ben definiti ai singoli individui. Ci sono tre tipi principali di ruoli: generazione di informazioni, integrazione e revisione.

- L'**utente finale** è l'esperto del dominio delle applicazioni che genera informazioni sul sistema attuale, sull'ambiente del sistema futuro e sui compiti che dovrebbe supportare. Ogni utente corrisponde a uno o più attori e aiuta a identificare i casi d'uso associati.
- Il **cliente**, un ruolo di integrazione, definisce l'ambito del sistema in base alle esigenze dell'utente. Utenti diversi possono avere opinioni diverse sul sistema, sia perché beneficeranno di diverse parti del sistema (ad esempio, un dispatcher contro un ufficiale di campo), sia perché gli utenti hanno opinioni o aspettative diverse sul sistema futuro. Il cliente funge da integratore di informazioni sul dominio delle applicazioni e risolve le incoerenze nelle aspettative degli utenti.

- L'**analista** è l'esperto del dominio applicativo che modella il sistema attuale e genera informazioni sul sistema futuro. Ogni analista è inizialmente responsabile del dettaglio di uno o più casi d'uso. Per un insieme di casi d'uso, l'analisi identificherà un certo numero di oggetti, le loro associazioni e i loro attributi utilizzando le tecniche descritte nella Sezione 5.4. L'analista è tipicamente uno sviluppatore con un'ampia conoscenza del dominio applicativo.
- L'**architetto**, ruolo di integrazione, unifica il caso d'uso e i modelli di oggetti dal punto di vista del sistema. Analisti diversi possono avere stili di modellazione diversi e visioni diverse delle parti dei sistemi di cui non sono responsabili. Anche se gli analisti lavorano insieme e molto probabilmente risolveranno le differenze man mano che progrediscono attraverso l'analisi, il ruolo dell'architetto è necessario per fornire una filosofia di sistema e per identificare le omissioni nei requisiti.
- L'**editor di documenti** è responsabile dell'integrazione di basso livello del documento e del formato complessivo del documento e del suo indice.
- Il responsabile della **configurazione** è responsabile del mantenimento di una storia di revisione del documento, nonché delle informazioni di tracciabilità relative alla RAD con altri documenti (come il documento di progettazione del sistema; si veda il capitolo 6, *Progettazione del sistema: Decomposizione del sistema*).
- Il **revisore** convalida il RAD per correttezza, completezza, coerenza e chiarezza. Utenti, clienti, sviluppatori o altri individui possono diventare revisori durante la convalida dei requisiti. Le persone che non sono ancora state coinvolte nello sviluppo rappresentano revisori eccellenti, perché sono più in grado di identificare le ambiguità e le aree che necessitano di chiarimenti.

La dimensione del sistema determina il numero di diversi utenti e analisti che sono necessari per suscitare e modellare i requisiti. In tutti i casi, ci dovrebbe essere un ruolo di integrazione dal lato del cliente e uno dal lato dello sviluppo. Alla fine, i requisiti, per quanto grandi siano le dimensioni del sistema, dovrebbero essere comprensibili per un singolo individuo che abbia una conoscenza individuale nel dominio applicativo.

5.5.3 Comunicare sull'analisi

Il compito di comunicare le informazioni è più impegnativo durante la selezione e l'analisi dei requisiti. I fattori che contribuiscono a ciò sono

- *Diversi background dei partecipanti*. Utenti, clienti e sviluppatori hanno diversi domini di competenza e utilizzano diversi vocabolari per descrivere gli stessi concetti.
- *Diverse aspettative delle parti interessate*. Gli utenti, i clienti e le direzioni hanno obiettivi diversi nella definizione del sistema. Gli utenti vogliono un sistema che supporti i loro attuali processi di lavoro, senza interferenze o minacce alla loro posizione attuale (ad esempio, un sistema migliorato si traduce spesso nell'eliminazione delle posizioni attuali). Il cliente

vuole massimizzare il ritorno dell'investimento. Il management vuole consegnare il sistema in tempo. Le diverse aspettative e le diverse poste in gioco nel progetto possono portare a una riluttanza a condividere le informazioni e a segnalare i problemi in modo tempestivo.

- *Nuove squadre*. L'individuazione e l'analisi dei requisiti segna spesso l'inizio di un nuovo progetto. Ciò si traduce in nuovi partecipanti e nuovi incarichi di squadra e, quindi, in un periodo di ramp-up durante il quale i membri del team devono imparare a lavorare insieme.
- *Sistema in evoluzione*. Quando un nuovo sistema viene sviluppato da zero, i termini e i concetti relativi al nuovo sistema sono in flusso durante la maggior parte dell'analisi e della progettazione del sistema. Un termine può avere un significato diverso domani.

Nessun metodo o meccanismo di comunicazione può affrontare i problemi legati alla politica interna e al nascondimento delle informazioni. Obiettivi contrastanti e concorrenza saranno sempre parte di grandi progetti di sviluppo. Poche semplici linee guida, tuttavia, possono aiutare a gestire la complessità di visioni conflittuali del sistema:

- *Definire territori chiari*. La definizione dei ruoli come descritto nella Sezione 5.5.2 fa parte di questa attività. Ciò include anche la definizione di forum di discussione pubblici e privati. Per esempio, ogni team può avere un database di discussione come descritto nel Capitolo 3, *Organizzazione e comunicazione del progetto*, e la discussione con il cliente avviene su un database di clienti separato. Il cliente non dovrebbe avere accesso al database interno. Allo stesso modo, gli sviluppatori non dovrebbero interferire con le politiche interne del cliente/utente.
- *Definire obiettivi chiari e criteri di successo*. La definizione in codice di obiettivi e criteri di successo chiari, misurabili e verificabili sia da parte del cliente che degli sviluppatori facilita la risoluzione dei conflitti. Si noti che definire un obiettivo chiaro e verificabile è un compito non banale, dato che è più facile lasciare aperti gli obiettivi. Gli obiettivi e i criteri di successo del progetto devono essere documentati nella sezione 1.3 del RAD.
- *Brainstorming*. Mettere tutti gli interessati nella stessa stanza e generare rapidamente soluzioni e definizioni può rimuovere molte barriere nella comunicazione. Condurre le revisioni come attività reciproca (cioè, rivedere i deliverable sia del cliente che degli sviluppatori durante la stessa sessione) ha un effetto simile.

Il brainstorming, e più in generale lo sviluppo cooperativo dei requisiti, può portare alla definizione di notazioni condivise e ad hoc a supporto della comunicazione. Storyboard, schizzi dell'interfaccia utente e diagrammi di flusso di dati di alto livello appaiono spesso spontaneamente. Man mano che le informazioni sul dominio applicativo e sul nuovo sistema maturano, è fondamentale che venga utilizzata una notazione precisa e strutturata. In UML, gli sviluppatori utilizzano casi d'uso e scenari per comunicare con il cliente e gli utenti, e utilizzano diagrammi a oggetti, diagrammi di sequenza e macchine di stato per comunicare con altri sviluppatori (cfr.

sezioni 4.4 e 5.4). Inoltre, l'ultima versione dei requisiti dovrebbe essere disponibile per tutti i partecipanti. Mantenere un live

La versione online del documento di analisi dei requisiti con una cronologia aggiornata delle modifiche facilita la propagazione tempestiva dei cambiamenti in tutto il progetto.

5.5.4 Iterazione sul modello di analisi

L'analisi avviene in modo iterativo e incrementale, spesso in parallelo con altre attività di sviluppo come la progettazione e l'implementazione di sistemi. Si noti, tuttavia, che la modifica e l'estensione senza restrizioni del modello di analisi può portare solo al caos, specialmente quando è coinvolto un gran numero di partecipanti. Le iterazioni e gli incrementi devono essere gestiti con attenzione e le richieste di modifiche devono essere monitorate una volta che i requisiti sono stati definiti. L'attività dei requisiti può essere vista come una serie di passi (brainstorming, solidificazione, maturità) che convergono verso un modello stabile.

Brainstorming

Prima di iniziare qualsiasi altra attività di sviluppo, i requisiti sono un processo di brainstorming. Tutto - i concetti e i termini usati per riferirsi ad essi - cambia. L'obiettivo di un processo di brainstorming è quello di generare il maggior numero possibile di idee senza necessariamente organizzarle. Durante questa fase, le iterazioni sono rapide e di vasta portata.

Solidificazione

Una volta che il cliente e gli sviluppatori convergono su un'idea comune, definiscono i confini del sistema e si accordano su un insieme di termini standard, inizia la solidificazione. La funzionalità è organizzata in gruppi di casi d'uso con le loro corrispondenti interfacce. I gruppi di funzionalità sono assegnati a diversi team che sono responsabili del dettaglio dei loro casi d'uso corrispondenti. Durante questa fase, le iterazioni sono rapide ma localizzate.

Maturità

I cambiamenti al livello superiore sono ancora possibili, ma più difficili, e quindi vengono effettuati con maggiore attenzione. Ogni team è responsabile dei casi d'uso e dei modelli di oggetti relativi alle funzionalità che sono stati assegnati. Un team interfunzionale, il team di architettura, composto da rappresentanti di ogni team, è responsabile dell'integrazione dei requisiti (ad esempio, la denominazione).

Una volta che il cliente ha firmato i requisiti, la modifica del modello di analisi dovrebbe affrontare le omissioni e gli errori. Gli sviluppatori, in particolare il team di architettura, devono garantire che la coerenza del modello non venga compromessa. Il modello dei requisiti è in fase di gestione della configurazione e le modifiche dovrebbero essere propagate ai modelli di progettazione esistenti. Le iterazioni sono lente e spesso localizzate.

Il numero di caratteristiche e funzioni di un sistema aumenta sempre di più con il tempo. Ogni cambiamento, tuttavia, può minacciare l'integrità del sistema. Il rischio di introdurre più problemi con le modifiche tardive deriva dalla perdita di informazioni nel progetto. Le dipendenze tra le funzioni non sono tutte catturate; molte ipotesi possono essere implicite e dimenticate nel momento in cui il

il cambiamento è stato fatto. Spesso il cambiamento risponde ad un problema, nel qual caso c'è molta pressione per attuarlo, con il risultato di un esame solo superficiale delle conseguenze del cambiamento. Quando vengono aggiunte nuove caratteristiche e funzioni al sistema, queste devono essere contestate con le seguenti domande: Sono state richieste dal cliente? Sono necessarie o sono abbellimenti? Dovrebbero far parte di un programma di utilità separato e mirato invece che del sistema di base? Le modifiche sono requisiti fondamentali o caratteristiche opzionali? Qual è l'impatto delle modifiche alle funzioni esistenti in termini di coerenza, interfaccia, affidabilità?

Quando sono necessarie delle modifiche, il cliente e lo sviluppatore definiscono la portata della modifica e il suo risultato desiderato e modificano il modello di analisi. Dato che esiste un modello di analisi completo per il sistema, specificare nuove funzionalità è più facile (anche se la sua implementazione è più difficile).

5.5.5 Segnaletica del cliente

La firma del cliente rappresenta l'accettazione del modello di analisi (come documentato dal documento di analisi dei requisiti) da parte del cliente. Il cliente e gli sviluppatori convergono su un'unica idea e concordano sulle funzioni e le caratteristiche che il sistema avrà. Inoltre, sono d'accordo:

- un elenco di priorità
- un processo di revisione
- un elenco di criteri che saranno utilizzati per accettare o rifiutare il sistema
- un programma e un budget.

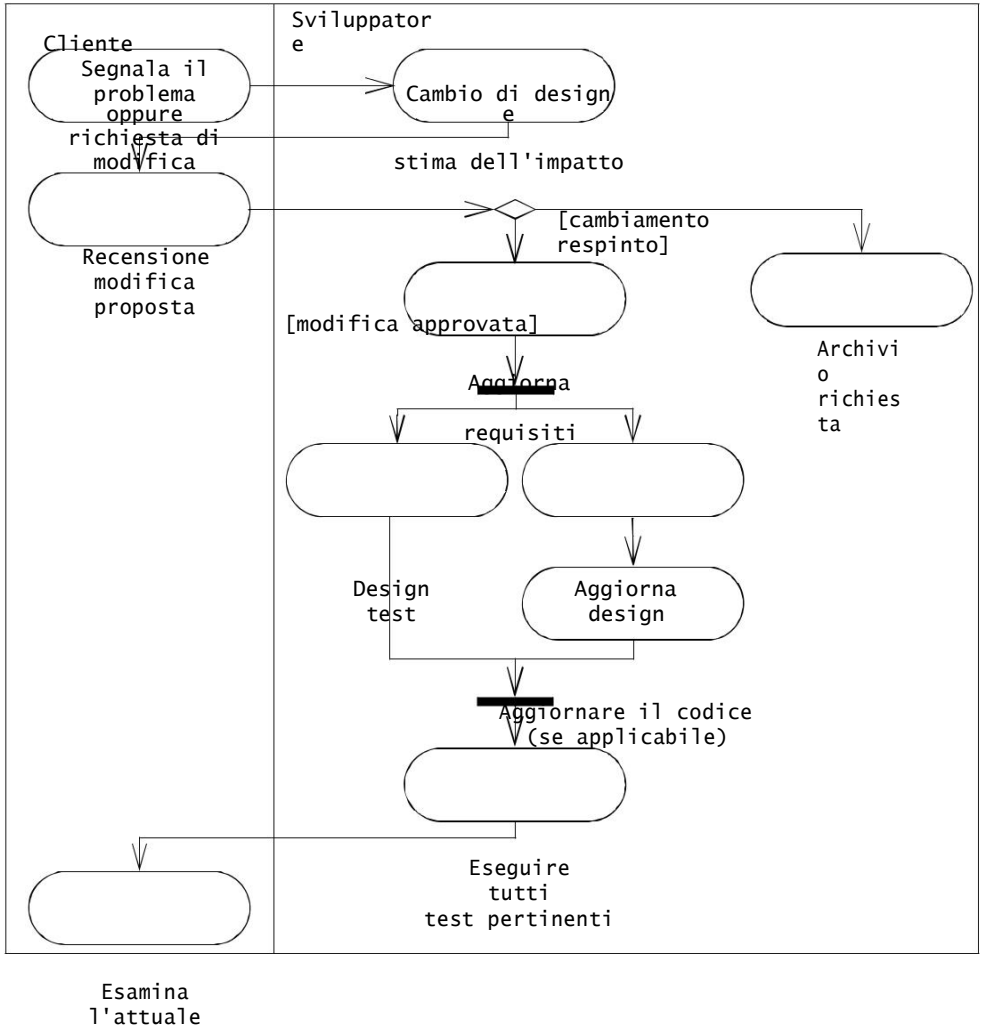
La prioritizzazione delle funzioni del sistema permette agli sviluppatori di comprendere meglio le aspettative del cliente. Nella sua forma più semplice, permette agli sviluppatori di separare i campanelli e i fischietti dalle caratteristiche essenziali. Permette inoltre agli sviluppatori di fornire il sistema in pezzi incrementali: le funzioni essenziali vengono fornite per prime, mentre i pezzi aggiuntivi vengono forniti a seconda della valutazione del pezzo precedente. Anche se il sistema deve essere consegnato come un unico pacchetto completo, la prioritizzazione delle funzioni permette al cliente di comunicare chiaramente ciò che è importante per lei e dove dovrebbe essere l'enfasi dello sviluppo. La Figura 5-21 fornisce un esempio di schema di priorità.

Ad ogni funzione è assegnata una delle seguenti priorità

- **Alta priorità: una** caratteristica di alta priorità deve essere dimostrata con successo durante l'accettazione del cliente.
- **Priorità media - Una** caratteristica di priorità media deve essere presa in considerazione nella progettazione del sistema e nella progettazione dell'oggetto. Sarà implementata e dimostrata nella seconda iterazione dello sviluppo del sistema.
- **Bassa priorità - Una** caratteristica a bassa priorità illustra come il sistema possa essere esteso a lungo termine.

Figura 5-21 Un esempio di schema di priorità per i requisiti.

Dopo l'approvazione del cliente, i requisiti vengono stabiliti a livello di base e vengono utilizzati per affinare la stima dei costi del progetto. I requisiti continuano a cambiare dopo l'approvazione, ma questi cambiamenti sono soggetti a un processo di revisione più formale. I requisiti cambiano, sia a causa di errori, omissioni, cambiamenti nell'ambiente operativo, cambiamenti nel dominio applicativo o cambiamenti tecnologici. La definizione di un processo di revisione in anticipo incoraggia la comunicazione dei cambiamenti in tutto il progetto e riduce il numero di sorprese nelle fasi successive. Si noti che un processo di cambiamento non deve necessariamente essere burocratico o richiedere spese generali eccessive. Può essere semplice come nominare una persona responsabile della ricezione delle richieste di modifica, dell'approvazione delle modifiche e del monitoraggio della loro attuazione. La Figura 5-22 illustra un esempio più complesso in cui le modifiche sono progettate



modifica

Figura 5-22 Un esempio di processo di revisione (diagramma di attività UML).

e revisionati dal cliente prima di essere implementati nel sistema. In tutti i casi, il riconoscimento del fatto che i requisiti non possono essere congelati (ma solo basati) andrà a beneficio del progetto.

L'elenco dei criteri di accettazione viene rivisto prima della firma. L'attività di selezione e di analisi dei requisiti chiarisce molti aspetti del sistema, compresi i requisiti non funzionali ai quali il sistema deve conformarsi e l'importanza relativa di ogni funzione. Riaffermando i criteri di accettazione al momento dell'approvazione, il cliente assicura che gli sviluppatori siano aggiornati su eventuali cambiamenti nelle aspettative del cliente.

Il budget e il programma vengono rivisti dopo che il modello di analisi diventa stabile. Descriviamo nel Capitolo 14, *Project Management*, le questioni relative alla stima dei costi.

Sia che la firma del cliente sia un accordo contrattuale o che il progetto sia già regolato da un contratto precedente, si tratta di una pietra miliare importante del progetto. Rappresenta la convergenza tra cliente e sviluppatore su un unico insieme di definizioni funzionali del sistema e su un unico insieme di aspettative. L'accettazione del documento di analisi dei requisiti è più critica di qualsiasi altro documento, dato che molte attività dipendono dal modello di analisi.

5.6 Caso di studio ARENA

In questa sezione applichiamo al sistema `ARENA` i concetti e i metodi descritti in questo capitolo. Iniziamo con il modello dei casi d'uso e con il glossario sviluppato nel capitolo precedente. Identifichiamo le entità partecipanti, i confini e gli oggetti di controllo e li perfezioniamo aggiungendo attributi e associazioni al modello di oggetti di analisi. Infine, identifichiamo le relazioni di eredità e consolidiamo il modello degli oggetti di analisi. In questa sezione, ci concentriamo principalmente sul caso d'uso `AnnounceTournament`.

5.6.1 Identificazione degli oggetti di entità

Gli oggetti entità rappresentano concetti nel dominio dell'applicazione che sono tracciati dal sistema. Utilizziamo il glossario prodotto durante l'elicitazione come punto di partenza per identificare gli oggetti entità in `ARENA`. Identifichiamo altri entity objects e i loro attributi applicando l'euristica di Abbott sui casi d'uso. Inizialmente ci concentriamo solo sulle frasi dei sostantivi che indicano i concetti del dominio di applicazione. La Figura 5-23 illustra il caso d'uso `AnnounceTournament` con la prima occorrenza di frasi sostantive che abbiamo identificato in **grassetto**.

Si noti che identifichiamo gli oggetti di entità corrispondenti agli attori nel modello del caso d'uso. Gli attori sono concetti nel dominio dell'applicazione e sono rilevanti per il sistema (ad es. per il controllo dell'accesso o per la documentazione delle responsabilità o della paternità). In `ARENA`, ogni `LeagueOwner` legittimo è rappresentato con un oggetto che viene utilizzato per memorizzare dati specifici di quel `LeagueOwner`, `come le` sue informazioni di contatto, le leghe che gestisce, e così via.

Si noti, inoltre, che non tutte le frasi dei sostantivi che abbiamo identificato corrispondono alle classi. Per esempio, `il nome di un torneo` è un sostantivo che si riferisce ad un attributo della classe del `torneo`. `Elenco degli inserzionisti` è un'associazione, in questo

caso, tra la classe della `Lega` e la classe dell'`inserzionista`. Possiamo usare alcune semplici euristiche per distinguere tra le frasi dei nomi che corrispondono a oggetti, attributi e associazioni:

Nome	AnnuncioTournament

Flusso di eventi

1. La **LegaOwner** richiede la creazione di un **torneo**.
2. Il sistema controlla se il LeagueOwner ha superato il **numero di tornei** in **campionato** o nell'**arena**. In caso contrario, il sistema presenta al LeagueOwner un modulo.
3. Il LeagueOwner specifica un **nome**, le **date di inizio e fine dell'applicazione** durante che i Giocatori possono richiedere per il torneo, le **date di inizio e fine del** torneo e un **numero massimo di Giocatori**.
4. Il sistema chiede al LigaverbandOwner se si debba cercare una sponsorizzazione esclusiva e, in caso affermativo, presenta una **lista di inserzionisti** che hanno espresso il desiderio di essere **sponsor esclusivi**.
5. Se il LeagueOwner decide di cercare uno sponsor esclusivo, seleziona un sottoinsieme di **i nomi** degli **sponsor proposti**.
6. Il sistema notifica agli sponsor selezionati il prossimo torneo e la **quota fissa** per le sponsorizzazioni esclusive.
7. Il sistema comunica le loro **risposte** al proprietario della Lega.
8. Se ci sono sponsor interessati, la LeagueOwner ne seleziona uno.
9. Il sistema registra il **nome** dello sponsor esclusivo e addebita la quota fissa per le sponsorizzazioni sul **conto dell'Inserzionista**. D'ora in poi, tutti i **banner pubblicitari** associati al torneo sono forniti solo dallo sponsor esclusivo.
10. Se non sono stati selezionati sponsor (o perché nessun inserzionista era interessato o perché il proprietario della Lega non ne ha selezionato nessuno), i banner pubblicitari vengono selezionati a caso e addebitati sul conto di ogni inserzionista per unità.
11. Una volta che i problemi di sponsorizzazione sono chiusi, il sistema richiede il LeagueOwner con un **elenco di gruppi di Giocatori, Spettatori e Gli inserzionisti** che potrebbero essere interessati al nuovo torneo.
12. La LeagueOwner seleziona i gruppi da notificare.
13. Il sistema crea una home page nell'arena per il torneo. Questa pagina viene utilizzata come punto d'ingresso al torneo (ad es. per fornire ai giocatori interessati un modulo di iscrizione al torneo e per interessare gli spettatori a guardare le **partite**).
14. Alla data di inizio dell'applicazione, il sistema notifica ogni utente interessato inviandogli un link alla pagina principale del torneo. I giocatori possono quindi fare domanda per il torneo con il caso d'uso ApplyForTournament fino alla data di fine dell'applicazione.

Figura 5-23 Applicazione dell'euristica di Abbott per l'identificazione degli oggetti entità nel caso d'uso `AnnounceTournament`. La prima occorrenza di una frase di un sostantivo è sottolineata in **grassetto**.

- *Gli attributi sono proprietà.* Gli attributi rappresentano una singola proprietà di un oggetto. Essi rappresentano un aspetto parziale di un oggetto e sono incompleti. Per esempio, il nome di un `inserzionista` è un attributo che identifica un `inserzionista`. Tuttavia, non include altre informazioni rilevanti sull'`Inserzionista` (ad esempio, il suo saldo del conto corrente, il tipo di banner che pubblicizza, ecc) che sono rappresentati da altri attributi o associazioni della classe dell'`Inserzionista`.
- *Gli attributi sono di tipo semplice. Gli attributi* sono proprietà che spesso hanno tipi come un numero (ad esempio, il numero massimo di `tornei`), una stringa (ad esempio, il nome di un `inserzionista`), le date (ad esempio, la data di inizio e di fine di un `torneo`). Anche le proprietà come l'indirizzo, il numero di previdenza sociale e il numero di identificazione del veicolo sono di solito considerati come tipi semplici (e quindi rappresentati come attributi) perché gli utenti li trattano come concetti semplici e atomici. I concetti complessi sono rappresentati come oggetti che sono in relazione con altri oggetti con associazioni. Per esempio, un `conto` è un oggetto che è correlato all'`inserzionista` corrispondente e può includere un saldo, una storia di transazioni, un limite di credito e altre proprietà simili.
- *I nomi che si riferiscono alle collezioni sono associazioni, spesso con fini impliciti*. Liste, gruppi, tabelle e insiemi sono rappresentati da associazioni. Per esempio, `ARENA` sollecita il `LeagueOwner` con un elenco di `Pubblicitari` potenzialmente interessati a sponsorizzazioni esclusive. Questo concetto può essere rappresentato con un'associazione tra la classe `Arena` e la classe dell'`Inserzionista`, che denota quali sono gli `Inserzionisti` interessati a sponsorizzazioni esclusive. Spesso la fine dell'associazione è implicita. Ad esempio, quando
  questioni di sponsorizzazione sono chiusi, `ARENA` sollecita il `LeagueOwner` con un elenco di gruppi di `giocatori`, `spettatori` e `pubblicitari`. Identifichiamo una nuova classe, `InterestGroup`, che rappresenta collezioni di utenti interessati a nuovi eventi su un campionato o una partita. Poi, identifichiamo un'associazione tra la classe `Arena` e la classe `InterestGroup` (corrispondente alla parola "lista") che rappresenta tutti gli `InterestGroup`. Quindi, identifichiamo un'associazione tra la classe `InterestGroup` e le classi `Player`, `Spectator` e `Advertisers` (corrispondente alla parola "gruppo"). Infine, identifichiamo ulteriori associazioni che hanno origine dalla classe `InterestGroup` ad altre classi che rappresentano l'interesse degli utenti nell'`InterestGroup` (ad esempio, `Lega`, `Gioco`).

La Tabella 5-6 elenca gli oggetti entità, i loro attributi e le loro associazioni che abbiamo identificato finora dal caso d'uso `AnnounceTournament`. Alleghiamo gli attributi e le associazioni alle loro classi rilevanti e scriviamo le definizioni per le nuove classi. La scrittura delle definizioni ha diversi scopi. In primo luogo, un nome non è abbastanza specifico perché tutti gli stakeholder condividano la stessa comprensione del concetto. Ad esempio, termini come `Game` and `Match` possono essere scambiati in molti contesti. In `ARENA`, invece, si riferiscono a concetti distinti (ad esempio, un `Gioco` rappresenta un insieme di regole applicate da un software, una `Partita` rappresenta una competizione tra un gruppo di `Giocatori`). In secondo luogo, gli

oggetti identificati durante l'analisi corrispondono anche ai termini del glossario che abbiamo iniziato durante l'elicitazione. Gli stakeholder utilizzano il glossario durante tutto lo sviluppo per risolvere ambiguità e

stabilire una terminologia standard. Scrivere definizioni brevi man mano che le classi vengono identificate è il modo migliore per evitare ambiguità e malintesi. Rinviare la scrittura delle definizioni comporta la perdita di informazioni e definizioni incomplete.

Tabella 5-6 Oggetti entità che partecipano al caso d'uso di `AnnounceTournament` identificati dalle frasi del sostantivo nel caso d'uso. "(?)" indicano aree di incertezza che portano alle domande della Figura 5-24.

Entità Oggetto	Attributi e associazioni	Definizione
`Conto`	-• bilancio -• storia delle accuse (?) -• storia dei pagamenti (?)	Un `conto` rappresenta l'importo attualmente dovuto da un `inserzionista`, una storia di spese, e pagamenti.
`Inserzionista`	-• nome -• campionati di interesse per sponsorizzazioni esclusive (?) -• tornei sponsorizzati -• conto	Attore interessato ad esporre la pubblicità striscioni durante i `fiammiferi`.
`Pubblicità`	-• gioco associato (?)	Immagine fornita da un `inserzionista` per la visualizzazione durante le partite.
`Arena`	-• numero massimo di tornei - quota fissa per le sponsorizzazioni (?) -• campionati *(implicito)* -• gruppi di interesse *(implicito)*	Un'istanziazione del sistema `ARENA`.
`Gioco`		Un `Gioco` è una competizione tra una serie di `Giocatori` che si svolgono secondo un set di regole. In `ARENA`, il termine `Gioco` si riferisce ad un un software che fa rispettare l'insieme delle regole, tiene traccia dei progressi di ogni `giocatore`, e decide il vincitore.
`InterestGroup`	-• elenco di giocatori, spettatori, o inserzionisti -• giochi e campionati di interessi *(impliciti)*	I `gruppi di interesse` sono elenchi di utenti nel `ARENA` che condividono un interesse (ad esempio, per un gioco o una lega). I `gruppi di interesse` sono utilizzati come mailing list per la notifica di potenziali attori di nuovi eventi.

Lega	-• numero massimo di tornei -• gioco	Una `Lega` rappresenta una comunità per la corsa `Tornei`. Una `Lega` è associata ad un `Gioco` specifico e `TorneoStile`. `Giocatori` iscritti alla `Lega` accumulare punti secondo il `Valutazione` degli `esperti della Lega`.

Tabella 5-6 *Continua*.

Entità Oggetto	Attributi e associazioni	Definizione
`LegaTitolare`	-• nome *(implicito)*	L'attore che crea una `Lega` e responsabile per l'organizzazione di `tornei` all'interno del `Lega`.
`Partita`	-• torneo -• giocatori	Un `Match` è una gara tra due o più `Giocatori` nell'ambito di una `partita`. Il sito risultato di una `partita` può essere un singolo vincitore e un insieme di perdenti o un pareggio (in cui i loro non sono vincitori o perdenti). Alcuni `Stili di Torneo` può non permettere legami.
`Giocatore`	-• nome *(implicito)*	
`Torneo`	-• nome data di inizio -• dell'applicazione data di scadenza della -• domanda -• data di inizio del gioco -• data di fine gioco numero massimo di -• giocatori -• sponsor esclusivo	Un `Torneo` è una serie di `Partite` tra un set di `giocatori`. I `tornei` terminano con un vincitore singolo. Il modo in cui `i giocatori` si accumulano punti e `le partite` sono previste è dettata dalla `Lega` in cui si svolge il `Torneo` organizzato.

L'identificazione degli oggetti entità e dei relativi attributi di solito fa scattare ulteriori domande al cliente. Ad esempio, quando identifichiamo gli attributi impliciti e le associazioni, dovremmo ricontrollare con il cliente per confermare se la nostra intuizione era corretta. In altri casi, le finalità di un'associazione sono ambigue. Raccogliamo tutte le domande generate dall'identificazione degli oggetti e torniamo al cliente (o all'esperto del dominio). La Figura 5-24 illustra le domande che abbiamo dopo aver identificato gli oggetti di entità che partecipano al caso d'uso di `AnnounceTournament`.

Domande per il cliente ARENA

- Quali informazioni devono essere registrate nei conti degli inserzionisti? Ad esempio, si dovrebbe registrare un registro completo della visualizzazione di ogni banner pubblicitario?
- Gli inserzionisti esprimono l'interesse per sponsorizzazioni esclusive per campionati specifici o per l'arena completa?
- I banner pubblicitari devono essere associati ai giochi (per consentire una selezione più intelligente dei banner quando non c'è una sponsorizzazione esclusiva)?
- La quota fissa per la sponsorizzazione esclusiva varia a seconda del campionato o del torneo?

Figura 5-24 Domande scatenate dall'identificazione degli oggetti entità.

5.6.2 Identificazione degli oggetti di confine

Gli oggetti di confine rappresentano l'interfaccia tra il sistema e gli attori. Essi sono identificati dai casi d'uso e di solito rappresentano l'interfaccia utente ad un livello grossolano. Non rappresentano informazioni sul layout o dettagli dell'interfaccia utente come menu e pulsanti. I modelli di interfaccia utente sono molto più adatti a questo tipo di informazioni. Invece, gli oggetti di contorno rappresentano concetti come finestre, forme o artefatti hardware come le postazioni di lavoro. Ciò consente alle parti interessate di visualizzare dove la funzionalità è disponibile nel sistema.

L'euristica di Abbott non identifica molti oggetti di confine, in quanto spesso sono lasciati implicitamente all'inizio. Invece, scandiamo il caso d'uso `AnnounceTournament` use case (Figura 5-23) e identifichiamo dove vengono scambiate le informazioni tra gli attori e il sistema. Ci concentriamo sia sui moduli in cui gli attori forniscono informazioni al sistema (ad esempio, il modulo utilizzato dal `LeagueOwner` per creare un `Torneo`) sia sugli avvisi in cui il sistema fornisce informazioni agli attori (ad esempio, un avviso ricevuto dagli `inserzionisti che` richiedono una sponsorizzazione). Come per altri oggetti, definiamo brevemente ogni classe nel momento in cui la identifichiamo. La tabella 5-7 illustra gli oggetti di confine che abbiamo identificato per l'`AnnuncioTournament` con le loro definizioni. La Figura 5-25 illustra le nostre domande aggiuntive.

Si noti che `AnnounceTournament` è un caso d'uso relativamente complesso che coinvolge diversi attori. Questo produce un numero relativamente elevato di oggetti di confine. In pratica, un caso d'uso può avere ben pochi oggetti limite per rappresentare l'interfaccia tra l'attore iniziatore e il sistema. In tutti i casi, tuttavia, ogni caso d'uso dovrebbe avere almeno un oggetto limite partecipante (possibilmente condiviso con altri casi d'uso).

Tabella 5-7 Oggetti di confine che partecipano al caso d'uso `AnnounceTournament`.

Oggetto di confine	Definizione
`TorneoFormulario`	Modulo utilizzato dal `LeagueOwner` per specificare le proprietà di un `Torneo` durante la creazione o l'editing.
`Modulo di richiestaSponsorizzazioneFormulario`	Modulo utilizzato dal `proprietario del campionato` per richiedere le sponsorizzazioni da parte degli `inserzionisti` interessati.
`SponsorizzazioneRichiesta`	Avviso ricevuto dagli `inserzionisti` che richiedono la sponsorizzazione.
`SponsorshipReply`	Avviso ricevuto da `LeagueOwner` che indica se un L'`inserzionista` vuole la sponsorizzazione esclusiva della torneo.
`SelectExclusiveSponsorForm`	Formulario utilizzato dal `proprietario del campionato` per chiudere la sponsorizzazione questione.

`NotifyInterestGroupsForm`	Modulo utilizzato dal `LeagueOwner` per informare gli utenti interessati.
`InterestGroupNotice`	Avviso ricevuto dagli utenti interessati sulla creazione di un nuovo `Torneo`.

Altre domande per il cliente ARENA

- Cosa dobbiamo fare con gli sponsor che non rispondono?
- Come possiamo pubblicizzare un nuovo torneo se non ci sono gruppi di interesse rilevanti?
- Come devono essere informati gli utenti (ad esempio, e-mail, telefono cellulare, casella di notifica ARENA)?

Figura 5-25 Domande scatenate dall'identificazione di oggetti di confine.

5.6.3 Identificazione degli oggetti di controllo

Gli oggetti di controllo rappresentano il coordinamento tra gli oggetti di confine e gli oggetti entità. Nel caso comune, un singolo oggetto di controllo viene creato all'inizio del caso d'uso e accumula tutte le informazioni necessarie per completare il caso d'uso. L'oggetto di controllo viene poi distrutto con il completamento del caso d'uso.

In `AnnounceTournament`, identifichiamo un unico oggetto di controllo chiamato `AnnounceTournamentControl`, che si occupa dell'invio e della raccolta degli avvisi agli `inserzionisti`, della verifica della disponibilità delle risorse e, infine, della notifica agli utenti interessati. Si noti che, nel caso generale, più oggetti di controllo potrebbero partecipare allo stesso caso d'uso, se, ad esempio, ci sono flussi alternativi di eventi da coordinare, più postazioni di lavoro che operano in modo asincrono, o se alcune informazioni di controllo sopravvivono al completamento del caso d'uso.

5.6.4 Modellazione delle interazioni tra oggetti

Abbiamo identificato una serie di entità, confini e oggetti di controllo che partecipano al caso d'uso `AnnounceTournament`. Lungo il percorso, abbiamo anche identificato alcuni dei loro attributi e associazioni. Rappresentiamo questi oggetti in un diagramma di sequenza, che rappresenta le interazioni che si verificano durante il caso d'uso per identificare ulteriori associazioni e attributi.

Nel diagramma di sequenza, disponiamo gli oggetti che abbiamo identificato lungo la fila superiore. Posizioniamo a sinistra l'attore iniziatore (ad esempio, `LeagueOwner`), seguito dall'oggetto limite responsabile dell'avvio del caso d'uso (ad esempio, `TournamentForm`), seguito dall'oggetto di controllo principale (ad esempio, `AnnounceTournamentControl`), e dagli oggetti entità (ad esempio, `Arena`, `League` e `Tournament`). Tutti gli altri attori partecipanti e i loro corrispondenti oggetti limite si trovano a destra del diagramma. Per motivi di spazio, abbiamo diviso il diagramma di sequenza associato all'`AnnounceTournament` in tre figure. La Figura 5-26 mostra la sequenza delle interazioni che portano alla creazione di un torneo. La Figura 5-27 illustra il flusso di lavoro per la richiesta e la selezione di uno sponsor esclusivo. La Figura 5-28 si concentra sulla notifica dei gruppi di interesse.

Il diagramma di sequenza nella Figura 5-26 è semplice. Il `LeagueOwner` richiede la creazione del torneo e ne specifica il parametro iniziale (es. nome, numero massimo di giocatori). Viene creata l'istanza `AnnounceTournamentControl` e, se le risorse lo consentono, viene creata un'istanza di entità `Torneo`.

Il diagramma di sequenza in Figura 5-27 è più interessante in quanto porta all'identificazione di ulteriori associazioni ed attributi. Quando si richiedono sponsorizzazioni, l'oggetto di controllo deve

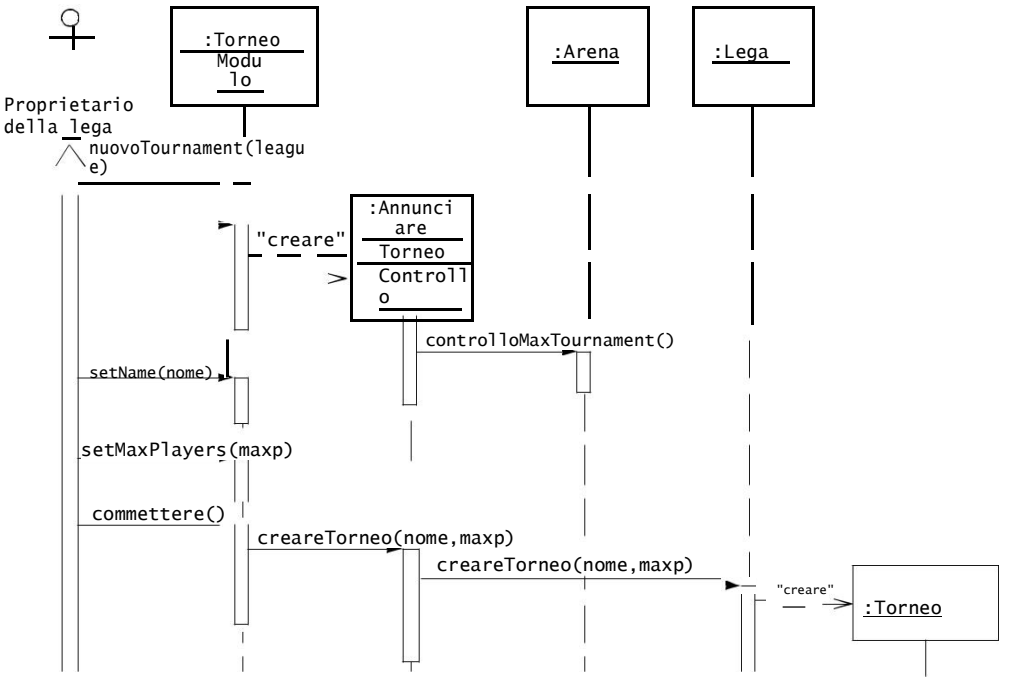


Figura 5-26 Diagramma di sequenza UML per `AnnounceTournament`, flusso di lavoro per la creazione di tornei.

per prima cosa ottenere una lista degli sponsor interessati. Lo richiede alla classe `Arena`, che mantiene l'elenco degli sponsor interessati. Ciò comporta che la classe `Arena` mantenga in ogni momento l'elenco di tutti gli `inserzionisti`, in modo da poter restituire tale elenco all'oggetto `AnnounceTournamentControl` (o oggetti di controllo per altri casi d'uso che richiedono l'elenco di tutti gli `inserzionisti`). Per notificare un `Inserzionista`, potremmo anche avere bisogno di informazioni di contatto, come l'indirizzo e-mail, o potremmo avere bisogno di creare una casella di posta elettronica per gli avvisi all'interno di `ARENA`. Di conseguenza, aggiungiamo un attributo di contatto alla classe dell'Inserzionista, che inizialmente memorizza l'indirizzo e-mail dell'`Inserzionista` fino a quando non vengono supportati altri dispositivi. Anticipando esigenze simili per altri attori, aggiungiamo anche attributi di contatto alle classi `LeagueOwner` e `Player`.

Quando si costruisce il diagramma di sequenza per la notifica dei gruppi di interesse (Figura 5-28), ci si rende conto che il caso d'uso non specifica come lo sponsor selezionato viene notificato. Di conseguenza, aggiungiamo un passo nel caso d'uso per notificare a tutti gli sponsor che hanno risposto sulle decisioni di sponsorizzazione prima che i gruppi d'interesse siano notificati. Questo richiede l'identificazione di un nuovo oggetto di confine, `SponsorNotice`. Il resto dell'interazione non produce alcuna nuova scoperta, poiché abbiamo già anticipato la necessità delle classi `InterestGroup` e `InterestGroupNotice`.

5.6.5 Revisione e consolidamento del modello di analisi

Ora che abbiamo identificato la maggior parte degli oggetti partecipanti, le loro associazioni e i loro attributi, abbiamo disegnato diagrammi di classe UML che documentano i risultati della nostra analisi fino ad ora. Poiché abbiamo identificato molti oggetti, utilizziamo diversi diagrammi di classe per rappresentare il modello degli oggetti di analisi. Noi

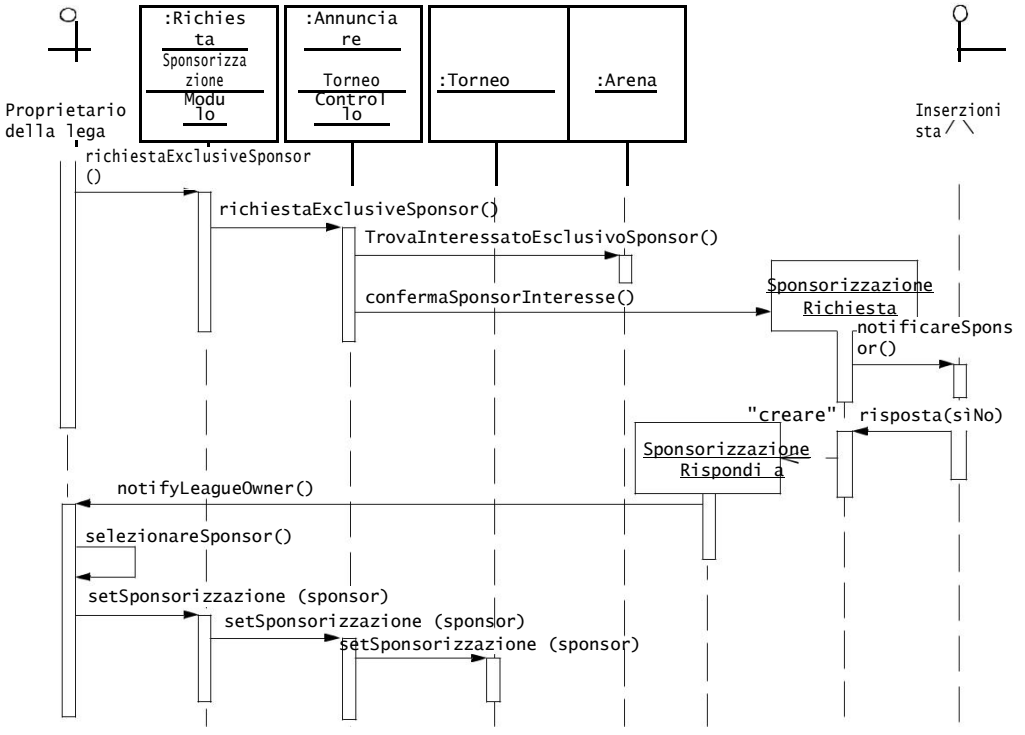


Figura 5-27 Diagramma di sequenza UML per il caso d'uso di `AnnounceTournament`, workflow di sponsorizzazione.

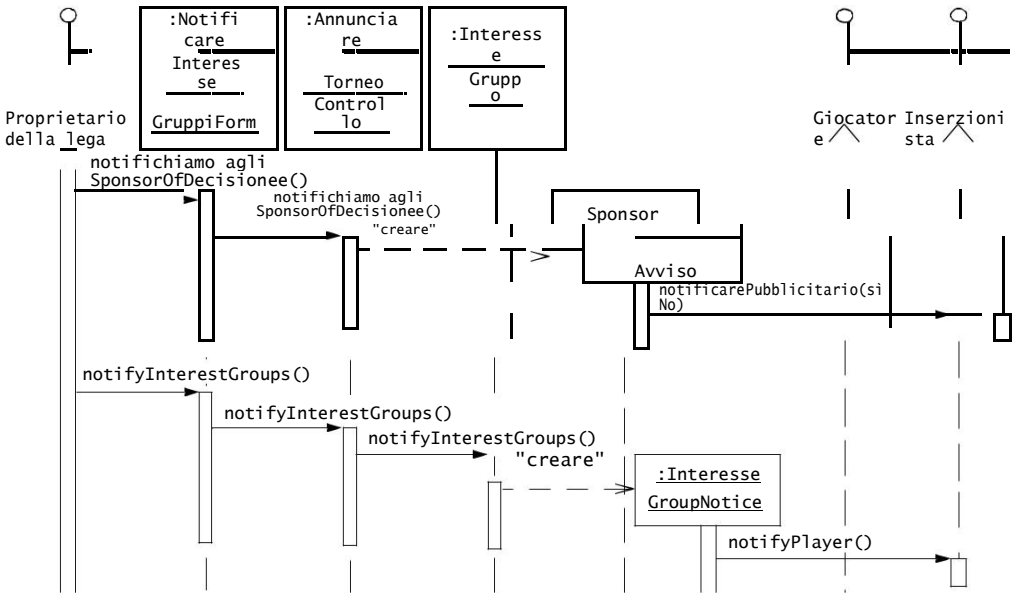
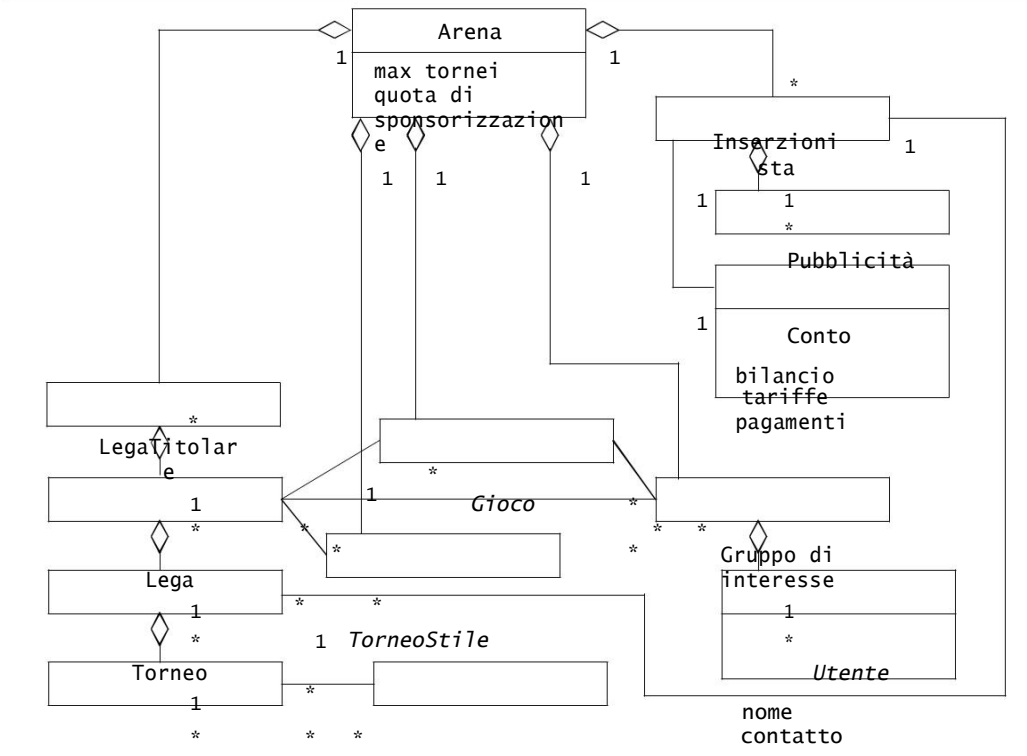


Figura 5-28 Diagramma di sequenza UML per il caso d'uso `AnnounceTournament`, flusso di lavoro del gruppo di interesse.

utilizzare questi diagrammi di classe come indice visivo nel glossario che abbiamo sviluppato. Anche se non dovremmo aspettarci che il cliente o gli utenti siano in grado di rivedere i diagrammi di classe, possiamo usare i diagrammi di classe per generare ulteriori domande per le interviste con il cliente.

Ci concentriamo in primo luogo sugli oggetti entità, poiché questi devono essere attentamente esaminati dal cliente in quanto rappresentano concetti di dominio applicativo (Figura 5-29). Si noti che usiamo la classe `Arena` come oggetto radice del sistema; la classe `Arena` rappresenta una specifica istanziazione. Per esempio, data un'istanziazione, è possibile ottenere un elenco di tutti i `gruppi di interesse`, gli `inserzionisti`, i `proprietari della lega`, i `giochi` e `gli stili di torneo` interrogando la classe `Arena`. Inoltre, si noti che gli oggetti non sono condivisi tra le istanziazioni. Ad esempio, `i LeagueOwners` appartengono esattamente a un'istanziazione del sistema. Se un utente è un `LeagueOwner` in diverse istanziazioni di `ARENA del` sistema, possiede un account `LeagueOwner` in ogni istanziazione. Facciamo questo tipo di scelte durante l'analisi in base alla nostra interpretazione della dichiarazione del problema, in base alla nostra esperienza, e in base alle risorse disponibili per costruire nel sistema. In tutti i casi, queste decisioni devono essere riviste e confermate dal cliente.

Successivamente, si disegna un diagramma di classe che rappresenta le gerarchie ereditarie (Figura 5-30). Sebbene UML consenta la coesistenza di relazioni e associazioni ereditarie nello stesso diagramma, è buona pratica durante l'analisi disegnare due diagrammi separati per rappresentare ogni tipo di



Partita Giocatore

Figura 5-29 Oggetti entità identificati dopo l'analisi del caso d'uso `AnnounceTournament`.

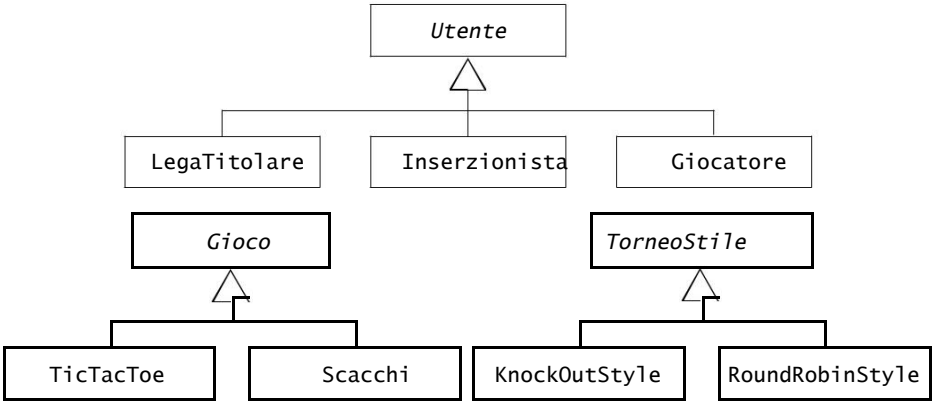


Figura 5-30 Gerarchia ereditaria tra gli oggetti entità del caso d'uso `AnnounceTournament`.

relationship. First, the UML symbols used to denote each type are similar and easily confused. Second, analysts usually focus on inheritance and associations at different times. We will see later, in Chapters 6 through 10, that this is not the case during system design and object design, where it is often necessary to consider both relationships to understand how different classes are related.

Figure 5-30 shows three inheritance hierarchies. First, we identified an abstract class *`User`* through generalization. This enables us to treat common attributes of various users in a more general fashion, including contact information and registration procedures. Note that in the problem statement and in the use cases, we already used the term "user," so we are simply formalizing a concept that was already in use. We identified two other inheritance hierarchies, *`Game`* and *`TournamentStyle`*, identified through specialization to provide examples for both concepts and to provide traceability to the problem statement. The `TicTacToe` and the `Chess` classes are concrete specializations of *`Game`* that embody rules for the games called "tic tac toe" and "chess," respectively. The `KnockOutStyle` and the `RoundRobinStyle` classes are concrete specializations of the *`TournamentStyle`* that provide algorithms for assigning `Players` to knock-out tournaments (in which players need to win to remain in the tournament) and round robin tournaments (in which each player plays all other players exactly once), respectively.

Finally, we draw a class diagram that depicts the associations among the boundary, control, and selected entity objects associated with the use case (Figure 5-31). To generate this diagram from the sequence diagrams, we draw the equivalent communication diagram, with the control object to the left, the boundary objects in the center, and the entity objects on the right. We then replace the iterations with associations, where necessary, so that the objects in the workflow can carry send messages to objects depicted in the sequence diagrams. We then add navigation to the associations to denote the direction of the dependencies: control and boundary

objects usually know about each other, but entity objects do not depend on any control or boundary objects.

Whereas the class diagram in Figure 5-29 focused primarily on the relationships among application domain concepts, the class diagram of Figure 5-31 focuses on the concepts associated with workflow of the use case at a coarse level. The control object acts as the glue among boundary and entity objects, since it represents the coordination and the sequencing among the forms and notices. As indicated in the sequence diagrams in Figures 5-26 through 5-28, the control object also creates several of the boundary objects. The class diagram in Figure 5-31 provides a summary of the objects participating in the use case and the associations traversed during the use case. However, the sequence diagrams provide the complete sequencing information of the workflow.

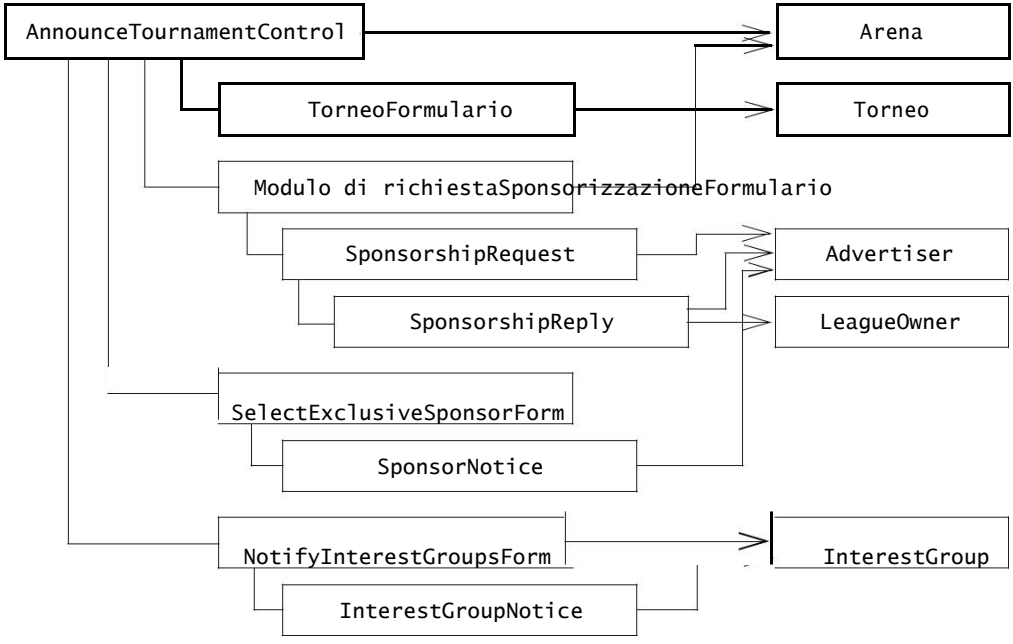


Figure 5-31 Associations among boundary, control, and selected entity objects participating in the `AnnounceTournament` use case.

5.6.6 Lessons Learned

In this section, we developed the part of the analysis object model relevant to the `AnnounceTournament` use case of `ARENA`. We started by identifying entity objects using Abbott's heuristics, then identified boundary and control objects, and used sequence diagrams to find additional associations, attributes, and objects. Finally, we consolidated the object model and depicted it with a series of class diagrams. We learned that:

- Identifying objects, their attributes and associations, takes many iterations, often with the client.
- Object identification uses many sources, including the problem statement, use case model, the glossary, and the event flows of the use cases.
- A nontrivial use case can require many sequence diagrams and several class diagrams. It is unrealistic to represent all discovered objects in a single diagram. Instead, each diagram serves a specific purpose—for example, depicting associations among entity objects, or depicting associations among participating objects in one use case.
- Key deliverables, such as the glossary, should be kept up to date as the analysis model is revised. Others, such as sequence diagrams, can be redone later if necessary. Maintaining consistency at all times, however, is unrealistic.
- There are many different ways to model the same application domain or the same system, based on the personal style and experience of the analyst. This calls for developing style guides and conventions within a project, so that all analysts can communicate effectively.

5.7 Further Readings

The classification of analysis objects into entity, boundary, and control objects has been made popular by the Objectory method [Jacobson et al., 1992]. These concepts originated from the model/view/controller (MVC) paradigm used in the Smalltalk-80 environment and also found their way into the Java Swing user interface framework [JFC, 2009].

CRC cards were introduced by Beck and Cunningham for teaching object-oriented thinking to novices and experienced developers in an OOPSLA paper entitled *A Laboratory For Teaching Object-Oriented Thinking* [Beck & Cunningham, 1989]. CRC cards are used extensively in the responsibility-driven design method from Wirfs-Brock [Wirfs-Brock et al., 1990].

Object-oriented analysis and design has evolved from many different sets of heuristics and terminologies. Modeling, like programming, is a craft, and requires much experience and willingness to make mistakes (hence the importance of client and user feedback). *Object-Oriented Modeling and Design* [Rumbaugh et al., 1991] provides an excellent guide to novices for class modeling. A more recent book, *Applying UML and Patterns* [Larman, 2005], provides a comprehensive treatment of object-oriented analysis and design, including use case modeling and reusing design patterns. For dynamic modeling with state machines, *Doing Hard Time: Using Object Oriented Programming and Software Patterns in Real Time Applications* [Douglass, 1999] provides detailed information and modeling heuristics on the topic.

5.8 Exercises

5-1 Consider a file system with a graphical user interface, such as Macintosh's Finder, Microsoft's Windows Explorer, or Linux's KDE. The following objects were identified from a use case describing how to copy a file from a floppy disk to a hard disk: `File`, `Icon`, `TrashCan`, `Folder`, `Disk`, `Pointer`. Specify which are entity objects, which are boundary objects, and which are control objects.

5-2 Assuming the same file system as before, consider a scenario consisting of selecting a `File` on a floppy, dragging it to `Folder` and releasing the mouse. Identify and define at least one control object associated with this scenario.

5-3 Arrange the objects listed in Exercises 5-1 and 5-2 horizontally on a sequence diagram, the boundary objects to the left, then the control object you identified, and finally, the entity objects. Draw the sequence of interactions resulting from dropping the file into a folder. For now, ignore the exceptional cases.

5-4 Examining the sequence diagram you produced in Exercise 5-3, identify the associations between these objects.

5-5 Identify the attributes of each object that are relevant to this scenario (copying a file from a floppy disk to a hard disk). Also consider the exception cases "There is already a file with that name in the folder" and "There is no more space on disk."

5-6 Consider the object model in Figure 5-32 (adapted from [Jackson, 1995]):

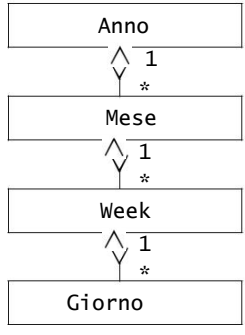


Figure 5-32 A naive model of the Gregorian calendar (UML class diagram).

Given your knowledge of the Gregorian calendar, list all the problems with this model. Modify it to correct each of them.

5-7 Consider the object model of Figure 5-32. Using association multiplicity only, can you modify the model such that a developer unfamiliar with the Gregorian calendar could deduce the number of days in each month? Identify additional classes if necessary.

5-8 Consider a traffic light system at a four-way crossroads (two roads intersecting at right angles). Assume the simplest algorithm for cycling through the lights (e.g., all traffic on one road is allowed to go through the crossroad, while the other traffic is stopped). Identify the states of this system and draw a state machine describing them. Remember that each individual traffic light has three states (green, yellow, and red).

5-9 From the sequence diagram Figure 2-34, draw the corresponding class diagram. Hint: Start with the participating objects in the sequence diagram.

5-10 Consider the addition of a nonfunctional requirement stipulating that the effort needed by `Advertisers` to obtain exclusive sponsorships should be minimized. Change the `AnnounceTournament` (Figure 5-23) and the `ManageAdvertisements` use case (solution of Exercise 4-12) so that the `Advertiser` can specify preferences in her profile so that exclusive sponsorships can be decided automatically by the system.

5-11 Identify and write definitions for any additional entity, boundary, and control objects participating in the `AnnounceTournament` use case that were introduced by realizing the change specified in Exercise 5-10.

5-12 Update the class diagrams of Figure 5-29 and Figure 5-31 to include the new objects you identified in Exercise 5-11.

5-13 Draw a state machine describing the behavior of the `AnnounceTournamentControl` object based on the sequence diagrams of Figures 5-26 through 5-28. Treat the sending and receiving of each notice as an event that triggers a change of state.

Riferimenti

[Abbott, 1983] R. Abbott, “Program design by informal English descriptions,” *Communications of the ACM*, Vol. 26, No. 11, 1983.

[Beck & Cunningham, 1989] K. Beck & W. Cunningham, “A laboratory for teaching object-oriented thinking,” *OOPSLA’89 Conference Proceedings*, New Orleans, LA, Oct. 1–6, 1989.

[De Marco, 1978] T. De Marco, *Structured Analysis and System Specification*, Yourdon, New York, 1978.

[Douglass, 1999] B. P. Douglass, *Doing Hard Time: Using Object Oriented Programming and Software Patterns in Real Time Applications*, Addison-Wesley, Reading, MA, 1999.

[Jackson, 1995] M. Jackson, *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices*, Addison-Wesley, Reading, MA, 1995.

[Jacobson et al., 1992] I. Jacobson, M. Christerson, P. Jonsson, & G. Overgaard, *Object-Oriented Software Engineering—A Use Case Driven Approach*, Addison-Wesley, Reading, MA, 1992.

[Jacobson et al., 1999] I. Jacobson, G. Booch, & J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, Reading, MA, 1999.

[JFC, 2009] *Java Foundation Classes*, JDK Documentation, Javasoft, 2009.

[Larman, 2005] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, 3rd ed., Prentice Hall, Upper Saddle River, NJ, 2005.

[Rumbaugh et al., 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, & W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[Wirfs-Brock et al., 1990] R. Wirfs-Brock, B. Wilkerson, & L. Wiener, *Designing Object-Oriented Software*, Prentice Hall, Englewood Cliffs, NJ, 1990.