Verifica dei sistemi incorporati — Sono pronti i GuT?

Vincent Encontre

White paper del software Rational

TP 317, 11/01

Rational®
the software development company

Indice

Introduzione

Questo documento è un'introduzione generale ai test sui sistemi incorporati, cui segue una discussione su come le problematiche inerenti a tali sistemi influenzano processi e tecnologie di test e sulle relative soluzioni proposte da Test RealTime di Rational.

Definizione di una serie di concetti comuni

Iniziamo con alcune definizioni da cui ricaveremo una serie di concetti comuni.

In cosa consiste il test? Il test è un processo disciplinato che accerta se il comportamento,
le prestazioni e la solidità dell'applicazione (e dei suoi componenti) soddisfano i criteri prestabiliti. Uno dei principali standard, seppure impliciti, consiste nell'avere applicazioni quanto più possibile prive di difetti. Quindi, il comportamento, le prestazioni e la solidità desiderati dovrebbero essere descritti formalmente e misurabili. Eseguire il debug significa letteralmente rimuovere i difetti (bug) e viene considerato soltanto parte del processo di test.

Cosa è esattamente un "sistema incorporato"? È difficile darne una definizione precisa (oltre al fatto che potrebbero nascere delle polemiche), quindi ci limiteremo ad alcuni esempi. I sistemi incorporati si trovano in ogni dispositivo "intelligente" che riguarda la nostra vita quotidiana, come il cellulare che abbiamo in tasca (e tutte le infrastrutture radio che stanno dietro), il mouse sulla scrivania, il router Internet attraverso il quale passano le e-mail, l'home-cinema, la stazione di controllo del traffico aereo e l'aereo ritardato che controlla! Il software costituisce oggi il 90% del valore di questi dispositivi.

Figura 1: il mondo gira con software incorporati

Gran parte dei sistemi incorporati, se non tutti, sono "real-time". I termini "real-time" ed "incorporato" sono spesso intercambiabili. Un sistema real-time è un sistema la cui correttezza dei calcoli dipende non solo dalla correttezza logica, ma anche dal tempo impiegato. Se i vincoli temporali del sistema non vengono rispettati, si verifica un errore all'interno del sistema. Per sistemi a rischio di sicurezza, l'errore non è un optional. Pertanto, il test dei vincoli temporali è importante almeno quanto quello del comportamento funzionale di un sistema incorporato.

Anche se il processo di test del sistema assomiglia a quello utilizzato su molti altri tipi di applicazioni, può essere influenzato da alcune problematiche inerenti ai sistemi incorporati:

- la divisione tra sviluppo dell'applicazione e piattaforme di esecuzione
- un'ampia varietà di piattaforme di esecuzione e ambienti di sviluppo incrociato

- numerose architetture di distribuzione
- coesistenza di vari paradigmi di implementazione
- risorse limitate e vincoli temporali sulle piattaforme di esecuzione
- mancanza di modelli di progettazione chiari
- standard emergenti di qualità e certificazione

Una descrizione più dettagliata di tali problematiche si trova nelle ultime pagine di questo documento.

Le problematiche accennate influenzano notevolmente la possibilità di misurare e testare un sistema incorporato. Ciò spiega inoltre perché sia così difficile testare simili sistemi e rappresenta uno dei punti più deboli nelle attuali pratiche di sviluppo. Non ci si meraviglia dunque se si pensa che secondo uno studio recente (vedere [R1]), il 50% e oltre dei progetti di sviluppo di sistemi incorporati sono indietro di mesi rispetto alla pianificazione, mentre soltanto il 44% delle pianificazioni soddisfano il 20% delle aspettative relative a funzioni e prestazioni, anche quando più del 50% dell'impegno di sviluppo globale si concentra sul test.

Nella parte restante del documento troveremo:

- l'esame di un'iterazione generica di test a partire dalla quale ricaveremo una serie limitata e ideale di tecnologie di test
- la suddivisione in istanze di questa iterazione per la verifica di sistemi complessi, così come avviene nei sistemi incorporati e, in base a tali considerazioni, saranno aggiunte delle capacità alla nostra tecnologia di test ideale
- l'esame dei fattori che rendono così difficili lo sviluppo e la verifica di sistemi incorporati
- una considerazione di come queste problematiche si aggiungono alle funzioni della tecnologia che serve a testarle.

Test RealTime di Rational sarà utilizzato come esempio di un prodotto che implementa una grande fetta di questa tecnologia ideale.

Un'iterazione di test generica

Preparazione del granulo da testare

La prima tappa obbligatoria in un test è l'identificazione del granulo da testare. La parola “granulo” viene impiegata per non utilizzarne altre come "componente", "unità" o "sistema", le quali hanno definizioni meno generiche. Ad esempio, in UML un componente è una parte dell'applicazione con uno specifico thread di controllo (ossia, un compito o un processo UNIX). Mi piace anche utilizzare il termine “granulo” per introdurre il concetto di granularità, che indica la grande varietà di elementi da testare, da una singola riga di codice ad un ampio sistema distribuito.

Identificare il granulo da testare, quindi trasformarlo in un granulo da testare o in un granulo sotto esame. Questa fase consiste nell'isolare il granulo dall'ambiente circostante, rendendolo indipendente con l'aiuto di stub e, in altri casi, di adattatori. Uno stub è una parte del codice che riproduce il percorso bilaterale tra il granulo e il resto dell'applicazione.

Quindi, costruire un driver di test. Permette di sollecitare il GuT (Granule under Test) con l'input adeguato, valuta poi le informazioni output e le confronta con la risposta attesa. Un adattatore viene utilizzato per consentire al driver di test di sollecitare il GuT. La sollecitazione e il test seguono percorsi specifici all'interno del GuT: li chiameremo PCO (Points of Control and Observation), termine improntato direttamente al settore delle telecomunicazioni. Un PCO può essere situato all'estremità o all'interno del GuT.

Esempi di PCO relativi a un granulo di funzione C:

- Point of Observation all'interno del granulo: copertura di una riga di codice specifica nella funzione
- Point of Observation all'estremità del granulo: valori di parametro restituiti dalla funzione
- Point of Control all'interno del granulo: modifica di una variabile locale
- Point of Control all'estremità del granulo: la funzione richiama parametri reali

Gli stub e i driver di test possono essere altre parti dell'applicazione (se disponibili) e non devono necessariamente essere sviluppati per testare una funzione C o una classe C++. Il GuT può essere raggiunto o sollecitato attraverso un'altra parte dell'applicazione, che svolge il ruolo di stub o di driver di test. Gli stub e i driver di test costituiscono l'ambiente di test della sicurezza del GuT.

Descrizione dello scenario di test

Una buona descrizione dello scenario di test dipende da:

- i PCO appropriati - a seconda del tipo di test da svolgere (di funzioni, struttura, carico e via dicendo)
- e dal modo in cui servirsene - che tipo di informazioni inviare, quali ricevere in ritorno e in quale ordine, se ne esiste uno

Il tipo di test, come le informazioni inviate o attese, si basa sulla serie di requisiti relativi al GuT. Nel caso di sistemi a rischio di sicurezza, requisiti formali e precisi sono essenziali per il processo di sviluppo. Sebbene i requisiti formali costituiscono un'importante fonte di motivazione per i test, non identificano sempre in maniera esplicita i test che scopriranno difetti significativi nel sistema. Utilizzando i requisiti formali, in applicazioni meno critiche in cui mancano requisiti specifici, il tester deve realizzare una serie di test adeguati (chiamati anche "idee di test") per definire alcuni requisiti di verifica relativi al GuT. Questi requisiti devono tradursi in scenari di test formali che traggono vantaggio dai PCO disponibili. Per "formale" intendiamo semplicemente eseguibile.

Solitamente, i requisiti non sono formali, né si traducono naturalmente in uno scenario di test formale. Questo processo di "traduzione" introduce spesso errori, i quali denotano che gli scenari di test non riflettono in modo preciso i requisiti. I linguaggi di specifica sono diventati più formali dopo l'introduzione di UML ed è ora possibile realizzare scenari di test basati sui requisiti per evitare l'insidia della traduzione. QualityArchitect e parte di Test RealTime, entrambi di Rational, forniscono buoni esempi su come utilizzare tecniche di test basate sui modelli.

Sfortunatamente, non tutti i requisiti sono descritti con UML, specialmente nei sistemi incorporati: la tecnica di descrizione formale più comune per uno scenario di test consiste semplicemente nell'utilizzo di un linguaggio programmatico come C o C++. Mentre C e C++ sono universalmente conosciuti (la curva di apprendimento è quindi ridotta), non tengono debitamente conto delle esigenze degli scenari di test come la definizione di PCO o il comportamento atteso dal GuT. Quindi, non consentono di redigere scenari di test comprensibili. Tale problema è stato affrontato con la progettazione di linguaggi di test high-level specifici, adatti a particolari domini di test come quelli data-intensive o basati sulle transazioni. Test RealTime di Rational propone un mix di linguaggi 3GL e linguaggi high-level di script dedicati che offre il meglio in termini di riduzione della curva di apprendimento e di scrittura efficace di scenari di test.

Un altro modo estremamente valido e produttivo di scrivere scenari di test è utilizzare registratori di sessione. Quando il GuT è sollecitato (manualmente o dal suo ambiente futuro), Points of Observation particolari registrano informazioni dentro e fuori il GuT, che si traducono poi in uno scenario di test da ripetere successivamente. In RealTime di Rational Rose, un esempio di registratore di sessione si trova nel modo in cui la realizzazione di modelli genera un diagramma di sequenza UML che riflette l'esecuzione di tracce, utilizzata poi come modello di scenario di test da QualityArchitect RealTime di Rational.
Ogni scenario di test deve portare il GuT ad uno stato di avvio particolare che consenta al test di essere eseguito. Questa parte dello scenario di test è conosciuta come preambolo. Al termine del test reale, e qualunque sia il risultato, lo script dello scenario di test deve portare il GuT ad uno stato finale stabile che permetta l'esecuzione dello scenario di test successivo. Questa parte dello scenario di test è chiamata conclusione.

Preparazione e svolgimento di un test

Una volta descritto, lo scenario di test è poi trasformato e integrato come parte informativa (opposta ad operativa) del driver di test e degli stub. È interessante notare come la descrizione di stub sia parte integrante dello scenario di test. Gli scenari di test sono realizzati durante la procedura di programmazione del test.

Osservazione dei risultati del test

I risultati dell'esecuzione del test sono monitorati attraverso i Points of Observation.

All'estremità del granulo, i Points of Observation caratteristici includono:

- parametri restituiti da funzioni o messaggi ricevuti
- valore delle variabili globali

- ordine e tempistica delle informazioni

All'interno del granulo, i Points of Observation caratteristici includono:

- copertura del codice sorgente per fornire dettagli circa la parte software del GUT che viene trattata
- grafico di controllo per monitorare i vari rami logici eseguiti
- flusso di informazioni per visualizzare lo scambio di informazioni tra le varie parti del GuT in termini di tempo. Solitamente, questo tipo di flusso è rappresentato sotto forma di diagramma di sequenza UML, come in Test RealTime di Rational.
- utilizzo di risorse che mostra le informazioni non funzionali, come il tempo impiegato nelle varie parti del GuT, gestione del pool di memoria o prestazioni nella gestione degli eventi

Tutte queste osservazioni possono essere raccolte per un singolo scenario di test e/o aggregate per una serie di scenari di test.

Definizione delle fasi successive

Una volta raccolti e sintetizzati tutti i dati del test, i risultati possono essere due: uno o più casi di test non riusciti o tutti i test riusciti.

Gli scenari di test possono non riuscire per svariate ragioni:

- Non conformità dei requisiti (incluso il requisito implicito zero-crash) - Occorrerà tornare all'implementazione o, peggio, alla progettazione per individuare il problema nel GuT.
- Lo scenario di test è sbagliato — Ciò avviene molto più spesso di quanto si pensi. Il test, come il software, non funziona sempre come si vorrebbe fin dalla prima volta. Modificare lo scenario di test per individuare il problema.
- Lo scenario di test non può essere eseguito — Di nuovo, come nel software, tutto sembra corretto, ma non si riesce a distribuire, avviare o connettere il programma di test al GuT.

Se tutti i test sono riusciti, si può procedere nel modo seguente:

- Riesaminare il test — se è solo l'inizio del test, è possibile definire il valore e l'obiettivo del test. Si suppone che i test individuino problemi, specie nell'impegno di sviluppo all'inizio, e se non se ne identifica nessuno,…
- Aumentare il numero di scenari di test — per ottenere più affidabilità da parte del GuT. Si può obiettare che l'affidabilità fa parte dei requisiti, e sarebbe giusto farlo. Tuttavia, il livello di affidabilità è spesso direttamente correlato al livello di copertura del GuT garantito dall'insieme degli scenari di test.
- La copertura del codice è il tipo di copertura più comunemente utilizzato— implementato in Test RealTime di Rational.

  La verifica basata sulla copertura del codice aiuta a definire ulteriori scenari di test che porteranno al livello di copertura desiderato per i requisiti. Questa parte della verifica è spesso chiamata verifica strutturale; gli scenari di test si basano sul contenuto del GuT e non direttamente sui relativi requisiti.
- Ampliare l'ambito del test aggregando i granuli — si applicherà poi questo processo generico ad un'ampia porzione del sistema, come descritto nel paragrafo successivo.

Quando termina il test?

Si tratta di una domanda che si pongono spesso i professionisti di software e a cui non desideriamo rispondere in modo esaustivo in questo documento. Tuttavia, possiamo prendere in considerazione la criticità della sicurezza del sistema da testare. Il sistema può essere considerato sicuro o no? Nei sistemi la cui sicurezza non è a rischio, è possibile interrompere i test in funzione di criteri più o meno soggettivi come time-to-market, budget e soddisfazione personale relativa ai risultati. In sistemi a rischio di sicurezza, invece, l'errore non è raro e non si può decidere di interrompere i test in funzione di simili criteri. Nella sezione intitolata Principali standard di qualità e certificazione verso la fine del documento, si fanno alcune raccomandazioni su come gestire questo problema.

Requisiti di una tecnologia generica di test

A partire dall'iterazione generica di test descritta in precedenza, è possibile ricavare una piccola serie di funzioni che devono essere svolte dai tool di test. I tool di test devono:

- aiutare a definire e isolare il GuT
- fornire una notazione dello scenario di test, scripting 3GL, visivo o high-level che supporta la definizione di PCO, per informazioni inviate al GuT o attese dal GuT e per l'aiuto fornito da preambolo e conclusione per ricavare
- con precisione scenari di test da requisiti o idee di test
- fornire alternative per implementare scenari di test mediante registratori di sessione
- supportare la distribuzione e l'esecuzione di scenari di test,
- il report delle osservazioni e la
- valutazione di successo o fallimento

Test RealTime di Rational supporta tali funzioni e va oltre questi requisiti per affrontare la verifica di sistemi complessi nell'ambito dei sistemi incorporati.

Sei passaggi incrementali per testare sistemi complessi

Architettura ed implementazione generiche per sistemi complessi

I sistemi incorporati sono sistemi complessi che possono essere composti da architetture estremamente diverse, dai minuscoli microcontroller a 8 bit agli ampi sistemi distribuiti con piattaforme
multiprocessore. Tuttavia, due terzi di tali sistemi vengono eseguiti su un RTOS (Real-Time Operating System), disponibile nei negozi o all'interno della società, e implementano il concetto di thread che si estendono ad attività o processi RTOS. (Un thread è un granulo con un flusso di controllo indipendente). In UML, questo concetto è riassunto dal termine Componente, mentre un nodo si riferisce a un'unità indipendente di elaborazione che esegue una serie di attività gestite da un RTOS. Tutte le comunicazioni tra i nodi avvengono solitamente mediante protocolli di trasmissione dei messaggi come TCP/IP.

La grande maggioranza degli sviluppatori di sistemi incorporati utilizza C, C++, Ada o Java come linguaggi programmatici (il 70% C nel 2002, il 60% C++, il 20% Java e il 5% Ada, come descritto in [R1]). Non è raro che vi sia più di un linguaggio in un sistema incorporato, in particolare C e C++ insieme o C e Java. Si crede che C sia più efficace e rispondente ai dettagli della piattaforma, mentre Java o C++ sono in teoria più produttivi, grazie ai concetti object-oriented. Bisogna però tenere bene in mente che i programmatori di sistemi incorporati non sono fissati con gli oggetti!

Nel contesto dei sistemi incorporati, un granulo può essere uno dei seguenti: Questo elenco è ordinato in funzione della complessità incrementale.

- funzione C o procedura Ada
- classe C++ o Java
- (serie di) moduli C o Ada
- cluster di classi C++ o Java
- un'attività RTOS
- un nodo
- l'intero sistema

Nei sistemi incorporati più piccoli, l'intero sistema è composto solo da una serie di moduli C e non include nessun codice correlato a RTOS. Nei più grandi (sistemi distribuiti), i protocolli di rete aggiungono un ulteriore livello di complessità. La prossima sezione mostra gli impatti di questa architettura generica sulle varie fasi della verifica.

I sei passaggi incrementali del test

A seconda del tipo di granulo, e in base all'utilizzo comune nell'industria, che discuteremo più in là in questa sezione, sono necessari alcuni passaggi per controllare che il comportamento dell'applicazione, le prestazioni e la solidità soddisfino i criteri attesi. Questi passaggi sono:

- test delle unità software
- test dell'integrazione software
- test di validazione del software
- test delle unità di sistema
- test di integrazione del sistema
- test di validazione del sistema

test delle unità software

Il GuT è una funzione C lisolata oppure una classe C++. A seconda dello scopo del GuT, lo scenario di test consiste in:

- Test data-intensive — che applicano numerose variazioni di dati in base ai valori del parametro di funzione, oppure
- Test basati sullo scenario — che svolgono sequenze di chiamate del metodo C++ per realizzare tutti i possibili scenari di test individuati nei requisiti

I Points of Observation sono i parametri di valore restituiti, le valutazioni di proprietà degli oggetti e la copertura del codice sorgente. La verifica scatola bianca viene impiegata per testare le unità, quindi il tester deve conoscere bene il contenuto del GuT. Il test delle unità è sotto la responsabilità dello sviluppatore.

Poiché non è facile trovare errori insignificanti in un sistema incorporato complesso, ci si deve concentrare sulla loro individuazione e rimozione al momento del test delle unità.

Test dell'integrazione software

Il GuT è ora una serie di funzioni o un cluster di classi. La validazione dell'interfaccia è l'essenza del test di integrazione. Entra in gioco lo stesso tipo di Points of Control del test delle unità (principale chiamata di funzione data-intensive o sequenze di chiamata del metodo), mentre i Points of Observation si focalizzano sulle interazioni tra i granuli di livello più basso che utilizzano diagrammi di flusso delle informazioni. Quando il GuT comincia ad essere significativo (ossia quando un intero scenario di test può essere applicato al GuT) possono iniziare i test delle prestazioni, che dovrebbero indicare correttamente la validità dell'architettura. In pratica vale lo stesso discorso dei test funzionali, prima si fanno, meglio è. Ogni passaggio successivo includerà test delle prestazioni. Anche il test scatola bianca è un metodo utilizzato in questo passaggio. Il test dell'integrazione software è sotto la responsabilità dello sviluppatore.

Test della validazione software

Il GuT è rappresentato dal codice utente di un componente. Questa può essere considerata la fase finale dell'integrazione software. I casi d'uso affrontano ora scenari per l'utente finale, che si discostano dai dettagli di implementazione. I Points of Observation includono la valutazione dell'impiego di risorse, dato che il GuT è parte significante del sistema nel suo complesso. Infine, consideriamo nuovamente questa fase come test scatola bianca. Il test di validazione del software è anch'esso sotto la responsabilità dello sviluppatore.

Test delle unità di sistema

Il GuT è ora un componente del sistema a tutti gli effetti, è cioè composto dal codice utente così come testato durante la verifica di validazione del software più tutte le parti RTOS e relative alla piattaforma: assegnazione di meccanismi, comunicazioni, interruzioni e così via. Il protocollo Point of Control non è tanto una chiamata di funzione o di metodo, quanto piuttosto un messaggio inviato o ricevuto mediante code di messaggio RTOS, ad esempio.

La simmetria nel paradigma di trasmissione dei messaggi fa sì che a questo stadio sia irrilevante la distinzione tra i driver di test e gli stub. Li chiameremo tester virtuali, poiché ciascuno di essi può agire come (o rimpiazzare) un altro componente del sistema rispetto al GuT. "Simulatore" e "tester" sono sinonimi di "tester virtuali". La tecnologia del tester virtuale dovrebbe essere così versatile da adattarsi a molti RTOS e protocolli di rete. Già da adesso, gli script di test portano abitualmente il GuT allo stato iniziale desiderato, poi generano sequenze ordinate di campioni di messaggi e convalidano i messaggi

ricevuti, confrontando il contenuto dei messaggi con i messaggi attesi e la data di arrivo con i vincoli temporali. Lo script di test è distribuito ai vari tester virtuali. Le risorse di sistema sono tenute sotto controllo per valutare l'abilità del sistema a supportare l'esecuzione di sistemi incorporati. A partire da questo passaggio, si predilige il test scatola grigia. Tutto ciò che è richiesto è la conoscenza dell'interfaccia al GuT. A seconda dell'organizzazione, il test delle unità di sistema sono sotto la responsabilità dello sviluppatore o di un team preposto all'integrazione del sistema.

Test di integrazione del sistema

Il GuT parte da una serie di componenti in un singolo nodo fino a coinvolgere tutti i nodi del sistema o addirittura una serie di nodi distribuiti. I PCO sono un mix di protocolli di comunicazione legati alla rete o agli RTOS, come gli eventi RTOS e i messaggi di rete. Il tester virtuale può anche svolgere il ruolo di nodo, oltre che di componente. Come nell'integrazione software, ci si concentra sulla convalida delle varie interfacce. Il test scatola grigia è il favorito. Il test dell'integrità di sistema è sotto la responsabilità del team di integrazione del sistema.

Test di validazione del sistema

Il GuT, alla fine, è il sistema incorporato completo. Gli obiettivi di questa fase finale sono:

- Soddisfare i requisiti funzionali per gli utenti finali. Notare che un utente finale può essere un dispositivo in una rete di telecomunicazioni (specificare se il sistema incorporato è un router Internet) o una persona (se il sistema è un dispositivo utente) o entrambi (un router Internet che può essere amministrato da un utente finale).

- Infine, eseguire un test non funzionale come il controllo di carico e di solidità. I tester virtuali possono essere duplicati per simulare il carico e programmati per produrre errori nel sistema.

- Assicurare l'interoperabilità con altre apparecchiature connesse. Controllare la conformità agli standard di interconnessione applicabili.

Scendere nei dettagli di questi obiettivi non rientra nell'ambito di questo documento. Il test scatola nera è il metodo favorito (il tester si concentra sia sui casi d'uso frequenti sia su quelli pericolosi).

Definizione della sequenza di questi passaggi

Ora che abbiamo i sei passeggi incrementali, vediamo come utilizzarli. Sono molti i criteri che aiutano a prendere una decisione:

- se tutte queste fasi sono applicabili ai sistemi
- se le fasi selezionate devono essere eseguite per tutte le parti dei sistemi o solo per alcune
- l'ordine in cui le fasi scelte devono essere applicate alle parti interessate

La decisione, basata sull'insieme di tali criteri, dipende principalmente dal tipo di sistema incorporato che si sta sviluppando: a rischio di sicurezza o meno, time-to-market, di piccola o grande distribuzione ecc. Verrà creato in futuro un documento con linee guida che facilitino questo processo di selezione. Vi si potrà accedere tramite Rational Developer Network, non appena sarà disponibile.

Un altro modo per affrontare queste fasi di test è fonderle tutte in una! Il test di validazione può essere considerato come test di unità di un GuT più grande. L'integrazione può anche essere controllata durante il test di validazione. È solo questione di come accedere al GuT, di quale tipo di PCO inserire e dove, nonché di come mirare a una porzione specifica del GuT (possibilmente molto remota) presa dallo scenario di test. Rimandiamo comunque la discussione su questo argomento.…

Ulteriori requisiti di una tecnologia di test per sistemi complessi

Per accettare la sfida della verifica di sistemi incorporati complessi, la tecnologia di test deve anche avere le seguenti capacità:

- Gestire molti tipi di Points of Control — Sollecitare il GuT attraverso chiamate di funzione o di metodo e trasmissione di messaggi o RPC (Remote Procedure Calls) mediante collegamenti di linguaggio diversi, come Ada, C, C++ o Java.

- Offrire una grande varietà di Points of Observation come parametri e ispezioni complessive variabili; definizione di affermazioni, informazioni e percorsi di controllo; registrazione della copertura di codice o monitoraggio dell'utilizzo di risorse.
  Ciascuno di questi Points of Observation dovrebbe fornire capacità di valutazione sia attese che reali.

Come le problematiche dei sistemi incorporati influenzano il processo e la tecnologia di test

In questa sezione, evidenzieremo le problematiche specifiche dei sistemi incorporati e valuteremo come influenzano la tecnologia utilizzata per testarli, prendendo Test RealTime di Rational come tool di test.

Divisione tra sviluppo dell'applicazione e piattaforme di esecuzione

Una delle tante definizioni di sistema incorporato è la seguente:

> Un sistema incorporato è un sistema software da progettare su una piattaforma diversa da quella a cui l'applicazione deve essere distribuita e attribuita.

Dal punto di vista dello sviluppo, la piattaforma consiste solitamente in un sistema operativo come Windows, Solaris o HP-UX. È da notare che la percentuale di utenti di UNIX e Linux è molto più alta (40%, come mostra [R1]) nel dominio dei sistemi incorporati che in altri domini di sistemi IT. Quanto agli obiettivi, le piattaforme includono tutte quelle descritte in precedenza.

Perché si parla di vincoli? Poiché la piattaforma target è ottimizzata e personalizzata esclusivamente per l'utente finale (una persona reale o una serie di dispositivi) non avrà i componenti necessari allo sviluppo (tastiere, rete, dischi ecc.).

Per affrontare questa duplice problematica, il tool di test deve fornire l'accesso alla piattaforma di esecuzione a partire dalla piattaforma di sviluppo nel modo più efficace e trasparente possibile. Nei fatti, la complessità di questo tipo di accesso deve essere nascosta all'utente.
L'accesso include il download di informazioni sullo scenario di test, il monitoraggio remoto dell'esecuzione del test (avvio, sincronizzazione, fine) e un upload delle osservazioni e dei risultati. In Test RealTime di Rational, tutti gli accessi delle piattaforme target sono controllati dalla tecnologia Target Deployment.

Inoltre, Test RealTime di Rational è disponibile su Windows, Solaris, HP e Linux, piattaforme di sviluppo utilizzate dalle società leader nella produzione di dispositivi e sistemi incorporati e nell'industria dell'infrastruttura.

Un'ampia e sempre crescente varietà di piattaforme di esecuzione e ambienti di sviluppo incrociato

La piattaforma di esecuzione può variare da uno schermo basato su microcontroller a 8 bit ad un ampio sistema distribuito e in rete. Tutte le piattaforme avranno bisogno di vari tool per lo sviluppo dell'applicazione a causa della molteplicità di vendor di chip e sistemi. È sempre più comune utilizzare molte piattaforme all'interno dello stesso sistema incrociato.

In generale, lo sviluppo di questo tipo di ambiente viene chiamato "ambiente di sviluppo incrociato". La grande varietà di piattaforme di esecuzione implica la disponibilità di una serie corrispondente di tool di sviluppo come i compilatori, i linker, i caricatori e i debugger.

La prima conseguenza è che la tecnologia dei Points of Observation può essere basata soltanto sul codice sorgente. Al contrario della tecnologia Object Code Insertion utilizzata dalla famiglia PurifyPlus di Rational e disponibile per una serie ridotta di compilatori locali, Test RealTime di Rational utilizza strumenti di codice sorgente adatta al fattore numero. Dieci anni di esperienza in questa tecnologia hanno permesso di ottenere strumenti di codice veramente efficaci.

Un'altra conseguenza diretta di questa varietà è che i vendor di tool di sviluppo incrociato tendono ad offrire IDE (Integrated Development Environment) che nascondono la complessità e mettono a proprio agio lo sviluppatore. Un altro requisito importante è che ogni ulteriore tool sia strettamente integrato all'IDE corrispondente. Ad esempio, Test RealTime di Rational è ben integrato in Tornado di WindRiver o Multi IDE di GreenHills.

Un'altra caratteristica è che, guidati dall'industria dinamica di chip e computer, le nuove piattaforme di esecuzione e i tool di sviluppo associati sono rilasciati con estrema frequenza. Il requisito che la tecnologia di test deve avere è l'alta flessibilità, per adattarsi a queste nuove architetture in tempo record. Una distribuzione mirata di Test RealTime per una nuova piattaforma target è spesso archiviata in meno di una settimana, spesso entro due giorni.

Risorse limitate e vincoli temporali delle piattaforme di esecuzione

Per definizione, un sistema incorporato ha risorse limitate sull'applicazione che deve eseguire. Ciò avviene soprattutto nelle piattaforme minuscole dove il RAM a disposizione è meno di 1 KB; la connessione all'ambiente di sviluppo può essere stabilita soltanto mediante sonde JTAG, emulatori o link in serie, oppure quando la velocità del microprocessore è adatta a gestire il lavoro. Il tool di test si trova di fronte a un difficile compromesso: avere dati di test sulla piattaforma di sviluppo e spedire dati tramite il link di connettività a scapito delle prestazioni (cosa spesso non tollerabile) oppure far interpretare i dati del test al driver di test sulla piattaforma target.

La tecnologia impiegata da Test RealTime di Rational ha lo scopo di incorporare il programma di test nel sistema target. Per fare ciò vengono compilati i dati di test, precedentemente tradotti nel linguaggio di programmazione dell'applicazione (C, C++ o Ada), all'interno del programma di test, mediante

compilatori incrociati disponibili, collegando poi questo file oggetto del programma di test al resto dell'applicazione. Questa catena di montaggio viene resa trasparente all'utente attraverso l'interfaccia linea di comando Test RealTime di Rational. La generazione di codice ottimizzata, la giusta assegnazione e associazione di link e l'occupazione di una piccola parte della memoria contribuiscono tutte a minimizzare le risorse richieste dal programma di test sulla piattaforma target pur mantenendo i seguenti vantaggi:

- La precisione temporale aumenta e l'utilizzo di un'ubicazione in-target riduce l'impatto real-time sulle prestazioni.
- Evitando la circolazione di informazioni sui dati nei link di connettività durante l'esecuzione di scenari di test si riduce la comunicazione "host-target". Se necessario, i dati di osservazione sono memorizzati in RAM o caricati con l'aiuto di un debugger o emulatore quando lo scenario di test è veramente giunto al termine.

Sebbene gli ambienti di sviluppo incrociato siano sempre più comuni, gran parte dei sistemi incorporati attuali sono ancora così complessi da sviluppare e testare che costituiscono per molti di noi un vero rompicapo. L'obiettivo di Test RealTime di Rational è semplificare la pesante routine degli sviluppatori di sistemi incorporati.

Mancanza di modelli visivi progettati in modo chiaro

Gli sviluppatori incorporati amano il codice! Sfortunatamente, i laureati non sono esperti di modellazione visiva e tendono a credere che sia il codice la "vera essenza". Anche se la modellazione visiva, con linguaggi come UML, ha fatto molti progressi nell'introduzione di conoscenze incorporate in un progetto, la maggioranza dei programmatori di sistemi incorporati ama ancora svolgere il proprio lavoro utilizzando vecchi linguaggi comuni di programmazione. E Java non riesce a fare in modo che la gente si interessi di progettazione! Affinché gli sviluppatori lavorino come meglio credono, Test RealTime di Rational si sforza di aiutarli a progettare scenari di test basati sul codice sorgente dell'applicazione offrendo procedure guidate, come generatori di modelli di test e wrapper API. La sicurezza che il test sia adatto all'applicazione rappresenta il principale vantaggio di questo processo di costruzione del test basato sul codice. L'inconveniente è che non sempre lo scenario di test corrisponde al requisito che deve controllare. Chiaramente, un maggior impiego della modellazione visiva ridurrebbe il divario tra requisito e scenario di test. Test RealTime di Rational spiana anche la strada a questa tecnica offrendo un diagramma di sequenza UML Rational Rose RealTime al compilatore dello scenario di test.

Principali standard di qualità e certificazione

In una certa categoria di sistemi incorporati (i sistemi a rischio di sicurezza) gli errori sono ricorrenti. Troviamo questi sistemi nell'industria nucleare,
medica e aerea. Per tornare all'obiettivo zero-errori di prima, l'industria aerea e gli enti governativi
(come la Federal Aviation Authority negli USA) si sono riuniti per descrivere le Considerazioni sul software in Airborne Systems and Equipment Certification, chiamato anche standard DO-178B di RTCA e descritto in [R2]. Si tratta del principale
standard per lo sviluppo di software a rischio di sicurezza nell'industria aerea di tutto il mondo. Riconosciuto come uno degli standard più rigidi per lo sviluppo di software, ha anche cominciato ad essere parzialmente adottato in altri settori per la manutenzione di sistemi a rischio, come quello automobilistico, medico (FDA ha realizzato uno standard vicino al DO-178B), la difesa o le aziende di trasporto.

Il DO-178B classifica il software secondo cinque livelli di criticità relativi a comportamenti software anomali che possono causare errori nelle funzioni del sistema. Il livello più critico è rappresentato dall'apparecchiatura di livello A, in cui l'errore risulta catastrofico per l'intero sistema. Il DO-178B include passaggi molto precisi per rendere abbastanza sicura l'apparecchiatura di livello A, specialmente nell'aera di test. Test RealTime di Rational soddisfa tutti i requisiti di test obbligatori del DO-178B, a tutti i livelli, fino all'apparecchiatura di livello A.

Riepilogo

Il documento propone una panoramica del processo composto da sei passaggi incrementali utilizzati per verificare sistemi incorporati. Considerando questo processo (e tenendo conto di tutti i sistemi, da quelli più piccoli a quelli più ampi), ma anche le caratteristiche e i vincoli specifici dei sistemi incorporati, abbiamo ricavato una serie di requisiti che una tecnologia ideale deve possedere per eseguire la verifica di sistemi incorporati. Test RealTime di Rational, il nuovo prodotto Rational per il dominio di sistemi incorporati, rappresenta un'ampia porzione di questa tecnologia ideale.

Questo documento ha fornito un'introduzione generale alla verifica di sistemi incorporati e sarà seguito negli anni a venire da altri articoli incentrati sui vari argomenti discussi.

Terminologia

Questa sezione è stata scritta da Paul Szymkowiak.

Sfortunatamente, molti termini impiegati nel mondo della progettazione di software hanno definizioni diverse. I termini utilizzati in questo documento sono probabilmente più familiari a quanti di voi lavorano nel dominio di sistemi incorporati real-time. Esistono comunque altri termini equivalenti e correlati utilizzati nella verifica di software. Ne forniamo una mappatura nella seguente tabella.

Termini utilizzati in questo documento	Termini equivalenti/correlati
Granule under Test (GuT): elemento sistema che è stato isolato dall'ambiente a scopo di test.	termini: voce di test, target, obiettivo di test
Point of Control and Observation: punto specifico di test in cui si registra una osservazione dell'ambiente di test o si prende una decisione sul flusso di controllo del test.	termine maggiormente equivalente: Verification Point
Conclusione: azioni intraprese per portare il sistema ad un particolare stato finale stabile, richiesto per eseguire il test successivo.	postcondizione, reimpostazione
Preambolo: azioni intraprese per portare il sistema ad uno stato iniziale stabile, richiesto per eseguire il test.	precondizione, impostazione
Programma di test: organizzazione di uno o più driver di test, stub specifici al test e strumenti combinati allo scopo di eseguire una serie di test correlati.	termini: suite di test o driver di test, stub di test
Tester virtuale: (1) qualcosa esterno al GuT che interagisce con il granulo durante un test. (2) istanza di un test in esecuzione, che rappresenta le azioni di una persona.	termini: script di test o driver di test, stub di test, simulatore, tester

Riferimenti

[R1] "Critical Issues Confronting Embedded Solutions Vendors", Electronics Market Forecast, http://www.electronic-forecast.com/, Aprile 2001.

[R2] DO-178B, Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission for Aeronautics — RTCA, http://www.rtca.org/, Gennaio 1992.

Informazioni sull'autore

Vincent Encontre è il direttore dell'Unità di business per test automatizzati, incorporati e real-time, situata nel nuovo centro di progettazione Rational di Tolosa, Francia. Quando non è in viaggio, in riunione o al computer per rispondere a una moltitudine di e-mail, Vincent è impegnato ad esaminare il futuro di Test RealTime considerato nelle sue qualità di prodotto Rational e di tecnologia riutilizzabile per i prodotti Rational della prossima generazione. Vincent ha una lunga esperienza in modellazione incorporata, tecnologie di test e pratiche ottimali. Prima di lavorare per Rational e ATTOL, Vincent ha trascorso 13 anni alla Philips, poi alla Verilog, progettando, commercializzando e supportando tool di progettazione software, con la convinzione che questi tool possano contribuire a creare più rapidamente software migliori.

Nel tempo libero, Vincent gioca a calcio con i suoi tre figli e altri amici e si gusta la magnifica art-de-vivre tipica della Francia meridionale (prima che sia troppo tardi...).

Rational®
the software development company

Sedi principali:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Numero verde: (800) 728-1212
E-mail: info@rational.com
Sito Web: www.rational.com
Sito internazionale: www.rational.com/worldwide