Testando Sistemas Incorporados — Você Tem Coragem?

Vincent Encontre

Rational Software White Paper

TP 317, 11/01

Rational®
the software development company

Índice Analítico

Introdução

Este documento fornece uma introdução geral para testar sistemas incorporados seguida por uma discussão sobre como os problemas dos sistemas incorporados afetam o teste de processos e tecnologias e como o Rational Test RealTime fornece soluções para esses problemas.

Definições para um Conjunto Comum de Conceitos

Vamos começar com algumas definições para estabelecer um conjunto comum de conceitos.

O que é um teste? O teste é um processo disciplinado que consiste em verificar se o comportamento, o desempenho e a robustez de seu aplicativo (incluindo seus componentes) correspondem aos critérios esperados. Um dos padrões principais, embora normalmente implícito, é que seus aplicativos fiquem o máximo possível sem defeitos. Portanto, o comportamento, o desempenho e a robustez esperados devem ser descritos e medidos formalmente. A depuração significa literalmente remover defeitos ("erros") e é considerada apenas parte do processo de teste.

O que é exatamente um "sistema incorporado"? É difícil e altamente controverso fornecer uma definição precisa, portanto, aqui estão alguns exemplos. Sistemas incorporados estão em cada dispositivo "inteligente" que se infiltra em nossas vidas diárias, como o telefone celular em seu bolso e toda a infra-estrutura wireless por trás dele; o Palm Pilot em sua mesa; o roteador de Internet através do qual seus e-mails são transmitidos; seu sistema de home theater de tela grande; a estação de controle de tráfego aéreo, bem como o avião atrasado que ele está monitorando! O software agora compõe 90% do valor desses dispositivos.

Figura 1 O Mundo Gira em Torno de Software Incorporado

A maioria dos sistemas incorporados, se não forem todos, são em "tempo real". Os termos "tempo real" e "incorporado" são freqüentemente utilizados de forma intercambiável. Um sistema em tempo real é aquele que a precisão das computações não depende apenas de sua precisão lógica, mas também do tempo em que o resultado é produzido. Se as restrições de tempo do sistema não forem atendidas, diz-se que ocorreu uma falha no sistema. Para alguns sistemas identificados como críticos à segurança, a falha não é uma opção. Portanto, ***o teste de restrições de tempo é tão importante quanto testar o comportamento funcional para um sistema incorporado***.

Embora o processo de teste do sistema incorporado assemelhe-se ao utilizado em vários outros tipos de aplicativos, alguns problemas importantes do mundo incorporado têm impacto sobre ele:

- separação entre as plataformas de desenvolvimento e execução do aplicativo;

- uma grande variedade de plataformas de execução e ambientes de desenvolvimento cruzado;
- uma ampla faixa de arquiteturas implantadas;
- coexistência de vários paradigmas de implementação
- recursos escassos e restrições de tempo na plataforma de execução;
- falta de modelos de design claros;
- padrões de qualidade e certificação emergentes.

Uma discussão mais detalhada desses problemas está localizada próxima ao final deste documento.

Esses problemas afetam em muito a capacidade de testar e medir um sistema incorporado. Isso também explica por que testar tais sistemas é tão difícil e, conseqüentemente, é um dos pontos mais fracos nas práticas de desenvolvimento atuais. Portanto, não é de se admirar que, de acordo com um estudo recente (consulte [R1]), mais de 50% dos projetos de desenvolvimento de sistemas incorporados estão meses atrás do planejamento e apenas 44% dos designs estão dentro de 20% das expectativas de recursos e desempenho—mesmo quando + de 50% do esforço de desenvolvimento total é gasto em teste.

No restante deste documento, iremos:

- percorrer uma iteração de teste genérica a partir da qual iremos derivar um conjunto mínimo de tecnologia de teste desejada;
- instanciar essa iteração para enfocar sistemas de teste complexos, conforme encontrados no mundo incorporado e, com base nessas considerações, incluiremos capacidades em nossa tecnologia de teste ideal;
- examinar o que torna os sistemas incorporados tão difíceis de desenvolver e testar;
- avaliar como esses problemas são incluídos na lista de recursos preenchida pela tecnologia utilizada para testá-los.

O Rational Test RealTime será utilizado como um exemplo de um produto que implementa uma grande parte dessa tecnologia ideal.

Uma Iteração de Teste Genérica

Preparando o Grânulo sob Teste

A primeira etapa obrigatória em qualquer teste é identificar o grânulo a ser testado. A palavra “grânulo” é utilizada para evitar o uso de outras palavras, como componente, unidade ou sistema, todas com menos definições genéricas. Por exemplo, em UML um componente é uma parte de um aplicativo que tem seu próprio encadeamento de controle (ou seja, uma tarefa ou um processo UNIX). Também gosto de utilizar a palavra “grânulo” como a raiz para granularidade, que traduz bem um amplo intervalo de elementos que podem ser testados—desde uma única linha de código até um grande sistema distribuído.

Identifique o grânulo a ser testado, em seguida, transforme-o em um grânulo testável ou em um GuT (Grânulo sob Teste). Esta etapa consiste em isolar o grânulo de seu ambiente tornando-o independente com a ajuda de *stubs* e, às vezes, *adaptadores*. Um stub é uma parte de código que simula acesso bidirecional entre o grânulo e o restante do aplicativo.

Em seguida, construa um driver de teste. Ele simula o GuT com a entrada adequada, em seguida, mede as informações de saída e as compara com a resposta esperada. Um adaptador é utilizado para permitir que o driver de teste estimule o GuT. O estímulo e a medida seguem caminhos específicos através de portas no GuT: nós os chamaremos de PCOs (Pontos de Controle e Observação), um termo que vem diretamente do segmento de mercado de telecomunicações. Um PCO pode estar localizado no limite ou dentro do GuT.

Exemplos de PCOs para um grânulo de função C são:

- Ponto de Observação dentro do grânulo: cobertura de uma linha de código específica na função;
- Ponto de Observação no limite do grânulo: valores de parâmetro retornados pela função;
- Ponto de Controle dentro do grânulo: alteração de uma variável local;

- Ponto de Controle no limite do grânulo: a chamada de função com parâmetros reais.

Stubs e até mesmo drivers de teste podem ser outras partes do aplicativo (se estiverem disponíveis) e não precisam necessariamente estar desenvolvidos para testar um função C ou uma classe C++. O GuT pode ser acessado ou simulado através de outra parte do aplicativo, que então executa a função de stub ou driver de teste. Stubs e drivers de teste constituem o ambiente de *trabalho de teste* do GuT.

Descrevendo o Caso de Teste

A descrição de um caso de teste é uma questão de imaginá-lo:

- os PCOs adequados — isso depende do tipo de teste a ser executado, como funcional, estrutural, carregamento e assim por diante;
- e como explorá-los — quais informações enviar e esperar receber através deles e, em que ordem, se houver alguma.

O tipo de teste, bem como as informações enviadas ou esperadas, é conduzido pelo conjunto de requisitos para o GuT. No caso de sistemas críticos à segurança, os requisitos formais e precisos são uma parte essencial do processo de desenvolvimento. Enquanto requisitos formais são uma fonte de motivação importante para testes, eles nem sempre identificam explicitamente os testes que descobrirão falhas importantes no sistema. Utilizando requisitos formais, e para aplicativos menos críticos ao especificar os requisitos específicos que estão faltando, o testador deve considerar a condução de um conjunto adequado de testes (também referido como "idéias de teste") para extrair alguns requisitos para testar o GuT. Esses requisitos devem ser traduzidos em casos de teste formais que obtêm vantagem de PCOs disponíveis. Para "formal", para simplificar, queremos dizer *executável*.

Normalmente, os próprios requisitos não são formais e não são traduzidos naturalmente em um caso de teste formal. Esse processo de tradução freqüentemente introduz erros nos quais os casos de teste não refletem precisamente os requisitos. As linguagens de especificação tornaram-se mais formais desde a introdução do UML e agora tornou-se possível expressar casos de teste formais baseados em requisitos para evitar ciladas de tradução. O Rational QualityArchitect e uma parte do Rational Test RealTime fornecem bons exemplos de utilização de tais técnicas de teste baseadas em modelos.

Infelizmente, nem todos os requisitos são descritos utilizando o UML, especialmente no mundo incorporado: a técnica de descrição formal mais comum para um caso de teste é simplesmente utilizar uma linguagem de programação como C ou C++. Enquanto o C e o C++ são universalmente conhecidos (portanto, há uma curva de aprendizado reduzida), eles não são bons para levar em conta necessidades de casos de teste, como definição de PCO ou comportamento esperado do GuT. Então, eles não permitem que você escreva casos de teste abrangentes. Esse problema foi enfocado pelo design de linguagens de teste de nível alto específicas, que são bem adaptadas a domínios de teste específicos, como teste intensivo de dados ou baseado em transações. O Rational Test RealTime propõe uma mistura de 3GL nativo e linguagens de script de nível alto dedicadas, apresentando o melhor dos dois mundos—curva de aprendizado reduzida e eficiência na escrita de casos de teste.

Outra forma extremamente valiosa e produtiva de escrever casos de teste é utilizar gravadores de sessão. Quando o GuT é estimulado (manualmente ou por seu ambiente futuro), Pontos de Observação específicos registram informações dentro e fora do GuT, que são automaticamente traduzidas em um caso de teste adequado para serem reproduzidas posteriormente. Um exemplo de tal gravador de sessão está localizado no Rational Rose RealTime no qual a execução do modelo leva à geração de um diagrama de seqüência de UML que reflete a execução de rastreio, que é então utilizada como um modelo de caso de teste pelo Rational QualityArchitect RealTime.

Cada caso de teste deve levar o GuT para um estado inicial específico que permite que o teste seja executado. Esta parte do caso de teste é conhecida como introdução. No final do teste efetivo, qualquer que seja o resultado, o script do caso de teste deve levar o GuT para um estado final estável que permita a execução do próximo caso de teste. Esta parte do caso de teste é chamada de pós-introdução.

Implantando e Executando o Teste

Depois de descrito, o caso de teste é então transformado e integrado como a parte de informações (versus operativa) do driver de teste e stubs. É importante observar que a descrição do stub é uma parte integral do caso de teste. Os casos de teste são executados durante a execução do trabalho de teste.

Observando os Resultados de Teste

Os resultados da execução do teste são monitorados através de Pontos de Observação.

No limite do grânulo, tipos típicos de Pontos de Observação incluem:

- *parâmetros retornados* por funções ou mensagens recebidas;
- *valor de variáveis globais;*
- *ordem e tempo das informações.*

Dentro do grânulo, tipos típicos de Pontos de Observação incluem:

- *cobertura do código fonte* para fornecer detalhes sobre qual parte do software o GuT cobre;
- *gráfico de controle* para seguir as várias ramificações lógicas que foram executadas;
- *fluxo de informações* para visualizar a troca de informações relacionadas à hora entre as partes diferentes do GuT. Normalmente, esse tipo de fluxo é representado como um diagrama de seqüência UML como no Rational Test RealTime.
- *uso de recursos* mostrando informações não funcionais, como gasto de tempo nas várias partes do GuT, gerenciamento do conjunto de memória ou desempenhos de manipulação de eventos.

Todas essas observações podem ser coletadas para um único caso de teste e/ou agregadas para um conjunto de casos de teste.

Decidindo sobre as Próximas Etapas

Depois de todos os dados de teste terem sido reunidos e sintetizados, poderá haver dois resultados: um ou mais casos de teste com falha ou todos os testes aprovados.

Os casos de teste podem falhar por uma série de razões:

- *Falta de conformidade com os requisitos* (incluindo requisito de falha-zero implícito) — Você terá de voltar para a implementação, ou pior, para o design para corrigir o problema no GuT.
- *O caso de teste está errado* — Isso acontece muito mais freqüentemente do que você pode pensar. O teste, como o software, nem sempre funciona conforme o esperado na primeira vez que é utilizado. Modifique o caso de teste para corrigir o problema.
- *O caso de teste não pode ser executado* — Novamente, como no software, tudo parece correto, mas você não pode exibir, iniciar ou conectar o trabalho de teste ao GuT.

Se todos os testes tiverem sido aprovados, você poderá desejar considerar esses cursos de ação:

- *Reavaliar seu teste* — se estiver no início do processo de teste, você poderá questionar o valor e o objetivo do teste. Os testes devem localizar problemas, especialmente no início do esforço de desenvolvimento, e se não aparecer nenhum, bem…
- *Aumentar o número de casos de teste* — isso deve aumentar a confiabilidade do GuT. A objeção pode ser feita para que a confiabilidade seja parte dos requisitos e que será corrigida. No entanto, o nível de confiabilidade está com freqüência diretamente correlacionado ao nível de cobertura do GuT pelo conjunto global de casos de teste.
- *A cobertura de código é o tipo de cobertura mais amplamente utilizado* — quando implementada no Rational Test RealTime. O teste baseado na cobertura de código ajuda a definir casos de teste adicionais que aumentarão o nível de cobertura até o nível aceito para os requisitos. Essa parte do teste é freqüentemente referida como um teste estrutural—os casos de teste baseiam-se no conteúdo do GuT, não diretamente em seus requisitos.
- *Aumentar o escopo do teste agregando grânulo*s — você aplicará então esse processo genérico a uma parte maior de seu sistema, conforme descrito no próximo parágrafo.

Quando o Teste Pára?

Esta é uma pergunta perene para os profissionais de software e não é o objetivo deste documento resolvê-la. No entanto, um heurístico que pode ser utilizado é considerar o quão crítica é a segurança do sistema sob teste. O sistema pode ser considerado crítico à segurança ou não? Para sistemas não-críticos à segurança, o teste pode ser baseado em mais ou menos critérios subjetivos, como prazo de lançamento no mercado, orçamento e "suficientemente bom". No entanto, para sistemas críticos à segurança, nos quais a falha não é uma opção, a decisão de parar o teste não pode ser feita sob tais critérios. Na seção intitulada *Emergindo Padrões de Qualidade e Certificação* próxima ao final deste documento, há algumas recomendações para manipular esse problema.

Requisito para uma Tecnologia de Teste Genérica

A partir da iteração de teste genérica descrita anteriormente, podemos deduzir um conjunto mínimo de recursos que devem ser preenchidos pelas ferramentas de teste. Eles devem:

- ajudar a definir e a isolar o GuT;
- fornecer uma notação de caso de teste, 3GL ou script visual ou de nível alto, suportando definição para PCOs, informações enviadas e esperadas do GuT e introdução/pós-introdução;
- ajudar a derivar precisamente casos de teste de requisitos ou idéias de teste;
- fornecer formas alternativas de implementar casos de teste utilizando gravadores de sessão;
- suportar a implantação e a execução de casos de teste;
- relatar observações;
- avaliar o sucesso ou falha.

O Rational Test RealTime suporta esses recursos e vai além desses requisitos para enfocar o teste de sistemas complexos localizados no domínio de sistemas incorporados.

Seis Etapas Incrementais para Testar Sistemas Complexos

Arquitetura e Implementação Genérica para Sistemas Complexos

Sistemas incorporados são sistemas complexos que podem ser compostos de arquiteturas extremamente diversas, que vão de minúsculos microcontroladores de 8 bits até grandes sistemas distribuídos compostos de plataformas de multiprocessadores. No entanto, dois-terços desses sistemas são executados em um RTOS (Sistema Operacional em Tempo Real), disponíveis comercialmente off-the-shelf ou in-house, e implementam o conceito de encadeamentos que se estendem para a tarefa ou processo do RTOS. (Um encadeamento é um grânulo com um fluxo de controle independente.) No UML, esse conceito é referido como um Componente, enquanto um nó refere-se a uma unidade de processamento independente executando um conjunto de tarefas gerenciadas por um RTOS. Qualquer comunicação entre nós é normalmente executada utilizando protocolos de transmissão de mensagem, como o TCP/IP.

A grande maioria dos desenvolvedores de sistemas incorporados utiliza C, C++, Ada ou Java como linguagens de programação (70% estará utilizando o C em 2002, 60% C++, 20% Java, 5% Ada, conforme observado em [R1]). Não é incomum ver mais de uma linguagem em um sistema incorporado, em particular C e C++ juntas ou C e Java. C deve ser mais eficiente e próxima dos detalhes da plataforma, enquanto Java ou C++ devem ser mais produtivas, graças aos conceitos de objeto orientado. No entanto, deve ser observado que os programadores de sistemas incorporados não são devotos de objetos!

No contexto de sistemas incorporados, um grânulo pode ser um dos seguintes. A lista é classificada pela complexidade incremental.

- função C ou procedimento Ada;
- classe C++ ou Java;
- módulos C ou Ada (conjunto de);
- cluster de classes C++ ou Java;
- uma tarefa do RTOS;

- um nó;
- o sistema completo.

Para sistemas incorporados menores, o sistema completo é composto apenas de um conjunto de módulos C e não integra nenhum código relacionado ao RTOS. Para os maiores (sistemas distribuídos), os protocolos de rede incluem o nível de complexidade.

A próxima seção mostra como essa arquitetura genérica influencia as várias etapas de teste.

As Seis Etapas Incrementais de Teste

Dependendo do tipo de grânulo e de acordo com o *uso* comum no segmento de mercado, discutido posteriormente nesta seção, seis etapas de teste são necessárias para verificar se o comportamento, o desempenho e a robustez correspondem aos critérios esperados. Estas etapas são:

- teste da unidade de software;
- teste de integração do software;
- teste de validação do software;
- teste da unidade de sistema;
- teste de integração do sistema;
- teste de validação do sistema.

Teste da Unidade de Software

O GuT é uma função isolada do C ou uma classe do C++. Dependendo do objetivo do GuT, o caso de teste consiste em:

- *Teste intensivo de dados* — aplicando um intervalo de variação de dados para valores de parâmetro de função ou
- *Teste baseado em cenário* — treinando seqüências diferentes de chamada de método do C++ para executar todos os casos de uso possíveis, conforme localizados nos requisitos.

Os Pontos de Observação são parâmetros de valores retornados, avaliações de propriedades de objetos e cobertura do código fonte. O teste da caixa branca é utilizado para testar unidades, o que significa que o testador deve estar familiarizado com o conteúdo do GuT. O teste de unidade é de responsabilidade do desenvolvedor.

Como não é fácil rastrear erros triviais em um sistema incorporado complexo, cada esforço deve ser feito para localizá-los e removê-los no nível de teste da unidade.

Teste de Integração do Software

O GuT é agora um conjunto de funções ou um cluster de classes. A validação da interface é a essência do teste de integração. O mesmo tipo de Pontos de Controle aplica-se para o teste de unidade (chamada de função principal intensiva de dados ou seqüências de chamadas de métodos), enquanto os Pontos de Observação focalizam interações entre grânulos de nível mais baixo utilizando fluxogramas de informações.

Logo que o GuT começa a ficar significativo—que ocorre quando um cenário de ponta a ponta pode ser aplicado ao GuT—os testes de desempenho podem ser executados, o que deve fornecer uma boa indicação sobre a validade da arquitetura. Como ocorre no teste funcional, quanto mais cedo melhor. 1Cada etapa seguinte incluirá então o teste de desempenho. O teste da caixa branca é também o método utilizado durante essa etapa. O teste de integração de software é de responsabilidade do desenvolvedor.

Teste de Validação do Software

O GuT é todo o código do usuário dentro de um componente. Isso pode ser considerado a etapa final na integração de software. Os casos de uso estão agora abordando cenários do usuário final e distanciando-se dos detalhes de implementação. Os Pontos de Observação incluem avaliação de uso de recursos porque o GuT é uma parte significativa do sistema global. Novamente (e finalmente), consideramos esta etapa o teste da caixa branca. O teste de validação do software ainda é de responsabilidade do desenvolvedor.

Teste da Unidade de Sistema

O GuT é agora um componente do sistema completo; ou seja, o código do usuário, conforme testado durante o teste de validação de software, mais partes relacionadas ao RTOS e à plataforma: mecanismos de tarefas, comunicações, interrupções, etc. O protocolo de Ponto de Controle não é mais uma chamada para uma função ou chamada de método, mas uma mensagem enviada ou recebida utilizando as filas de mensagens do RTOS, por exemplo.

A simetria localizada no paradigma de transmissão de mensagem implica que a distinção entre os drivers de teste e os stubs pode ser considerada irrelevante neste estágio. Pode chamá-los de *testadores virtuais,* porque cada um pode substituir ou agir como outro componente do sistema face a face com o GuT. O "simulador" e o "testador" são sinônimos para "testador virtual". A tecnologia do testador virtual deve ser versátil o suficiente para adaptar-se a um grande número de RTOS e protocolos de rede. De agora em diante, os scripts de teste levam o GuT para o estado inicial desejado, então, geram as seqüências ordenadas de amostras de mensagens e validam as mensagens recebidas comparando o conteúdo da mensagem com as mensagens esperadas e a data da recepção com as restrições de tempo. O script de teste é distribuído e implantado através de vários testadores virtuais. Os recursos do sistema são monitorados para avaliar a capacidade do sistema em sustentar a execução do sistema incorporado. A partir dessa etapa, o teste da caixa cinza é executado durante o método de teste. Tudo que é requerido é o conhecimento da interface para o GuT. Dependendo da organização, o teste da unidade de sistema é de responsabilidade do desenvolvedor ou de uma equipe dedicada de integração do sistema.

Teste de Integração do Sistema

O GuT inicia a partir de um conjunto de componentes dentro de um único nó e, eventualmente, incluindo todos os nós do sistema até um conjunto de nós distribuídos. Os PCOs são uma mistura de protocolos de comunicação relacionados ao RTOS e à rede, como eventos do RTOS e mensagens de rede. Além de um componente, um testador virtual também pode executar a função em um nó. Como ocorre na integração do software, o foco está na validação de várias interfaces. O teste da caixa cinza é o método de teste preferido. O teste de integração do sistema é de responsabilidade da equipe de integração do sistema.

Teste de Validação do Sistema

O GuT é finalmente o sistema incorporado completo global. Os objetivos dessa etapa final são vários:

- *Atender aos requisitos funcionais do usuário final*. Observe que um usuário final pode ser um dispositivo em uma rede de telecomunicações (se nosso sistema incorporado for um roteador de Internet), uma pessoa (se o sistema for um dispositivo do consumidor) ou ambos (um roteador de Internet que pode ser administrado por um usuário final).
- *Executar o teste final não funcional, como teste de carga e robustez.* Os testadores virtuais podem ser duplicados para simular a carga e programados para gerar falhas no sistema.
- *Assegurar a interoperabilidade com outro equipamento conectado*. Verifique a conformidade com os padrões de interconexão aplicáveis.

Está além do escopo deste documento entrar em detalhes para esses objetivos. O teste da caixa preta é o método preferido—o testador concentra-se nos casos de uso freqüentes e perigosos.

Decidindo Como Seqüenciar Estas Etapas

Agora que temos seis etapas incrementais, como as utilizamos? Existem vários critérios que são utilizados para a decisão:

- se todas essas etapas aplicam-se aos seus sistemas;
- se todas as etapas selecionadas devem ser executadas por todas as partes dos sistemas ou apenas por algumas;
- a ordem na qual as etapas selecionadas devem ser aplicadas às partes selecionadas.

A decisão, baseada na reunião desses critérios, depende muito do tipo de sistema incorporado que você está desenvolvendo: crítico à segurança ou não, prazo de lançamento no mercado, alguns implantados ou milhões e assim por diante. Um documento que fornece diretrizes para ajudá-lo com esse processo de seleção será escrito no futuro. Você poderá ter acesso através do Rational Developer Network quando ele estiver disponível.

Outra maneira de enfocar essas etapas de teste é fundi-las em uma! O teste de validação pode ser considerado como uma unidade testando um GuT maior. A integração também pode ser verificada durante o teste de validação. É apenas uma questão de como você pode acessar o GuT, qual tipo de PCOs pode inserir e onde e como pode utilizar como destino uma parte específica do GuT (possivelmente muito remota) a partir do caso de teste. Mas vamos deixar isso para outra discussão.…

Requisitos Adicionais para uma Tecnologia de Teste de Sistemas Complexos

Para enfocar o desafio de testar sistemas incorporados complexos, a tecnologia de teste deve incluir os seguintes recursos:

- *Gerenciar vários tipos de Pontos de Controle* — Para estimular o GuT, faça-o utilizando chamadas de função, uma chamada de método e uma transmissão de mensagem ou RPCs (Chamadas de Procedimentos Remotos) através de ligações de linguagens diferentes, como Ada, C, C++ ou Java.
- *Oferecer uma grande variedade de Pontos de Observação* como parâmetros e inspeções de variáveis globais; rastreio de asserções, de informações e de caminho de controle; e registro de cobertura de código ou monitoramento do uso de recursos. Cada um desses Pontos de Observação deve fornecer os recursos de avaliação esperados versus reais.

Como os Sistemas Incorporados Afetam o Processo e a Tecnologia de Teste

Nesta seção, destacaremos os problemas específicos de sistemas incorporados e avaliaremos como eles afetam a tecnologia utilizada para testá-los com o Rational Test RealTime como nossa ferramenta de teste.

Separação entre as Plataformas de Desenvolvimento e Execução de Aplicativos

Uma das várias definições para um sistema incorporado é:

> *Um sistema incorporado é qualquer sistema de software que deve ser projetado em uma plataforma diferente da plataforma na qual o aplicativo deve ser implantado e utilizado como destino.*

No lado do desenvolvimento, plataforma normalmente significa um sistema operacional, como Windows, Solaris ou HP-UX. Deve ser observado que a porcentagem de usuários UNIX e Linux é muito maior (40% conforme observado em [R1]) no domínio de sistemas incorporados quando comparado a outros domínios de sistemas de TI. Para o destino, as plataformas incluem qualquer um dos dispositivos mencionados anteriormente.

Por que a restrição? Porque a plataforma de destino é otimizada e adaptada exclusivamente para o usuário final (pode ser uma pessoa real ou outro conjunto de dispositivos) e não terá os componentes necessários (como teclados, rede, discos, etc.) para a execução do desenvolvimento.

Para poder arcar com esse problema de plataforma dupla, a ferramenta de teste deve fornecer acesso à plataforma de execução a partir da plataforma de desenvolvimento da forma mais transparente e eficiente possível. De fato, a complexidade de tal acesso deve ser escondida do usuário. O acesso inclui download de informações de caso de teste, monitoramento remoto da execução do teste (iniciar, sincronizar, parar) e upload dos resultados de teste e observação. No Rational Test RealTime, todos os acessos da plataforma de destino são controlados pela tecnologia de Implantação do Destino.

Além disso, o Rational Test RealTime está disponível no Windows, Solaris, HP e Linux; as plataformas de desenvolvimento que as empresas líderes do segmento de mercado de dispositivos, sistema incorporado e infra-estrutura estão utilizando.

Uma Variedade Grande e Crescente de Plataformas de Execução e Ambientes de Desenvolvimento Cruzado

A plataforma de execução pode variar de uma minúscula placa de microcontrolador de 8 bits a um grande sistema distribuído e de rede. Todas as plataformas precisarão de ferramentas diferentes para desenvolvimento de aplicativo devido à profusão dos fornecedores de chips e sistemas. É cada vez mais comum que várias plataformas sejam utilizadas dentro do mesmo sistema incorporado.

Em geral, o desenvolvimento desse tipo de ambiente é referido como "ambiente de desenvolvimento cruzado". A grande variedade de plataformas de execução implica a disponibilidade de um grande conjunto de ferramentas correspondente, como compiladores, vinculadores, carregadores e depuradores.

A primeira conseqüência disso é que a tecnologia de Pontos de Observação pode ser baseada apenas no código fonte. Ao contrário da tecnologia de Inserção de Código de Objeto utilizada pela família do Rational PurifyPlus e disponível para um pequeno conjunto de compiladores nativos, o Rational Test RealTime utiliza instrumentação de código fonte para poder arcar com o fator numérico. Dez anos de experiência nessa tecnologia resultaram em uma instrumentação de código altamente eficiente.

Outra conseqüência direta dessa variedade é que os fornecedores de ferramentas de desenvolvimento cruzado tendem a oferecer IDEs (Integrated Development Environments) para ocultar a complexidade e deixar o desenvolvedor confortável. É um requisito forte que qualquer ferramenta adicional esteja bem integrada ao IDE correspondente. Por exemplo, o Rational Test RealTime está bem integrado nos IDEs WindRiver's Tornado ou GreenHills' Multi.

Outra característica é que, conduzidas pelo segmento de mercado de chip de computador dinâmico, novas plataformas e ferramentas de desenvolvimento associadas são liberadas com extrema freqüência. Isso impõe um requisito que é a tecnologia de teste ser altamente flexível para adaptar-se a essas novas arquiteturas em tempo recorde. Uma implantação de destino do Rational Test RealTime para uma nova plataforma de destino é normalmente obtida em menos de uma semana, em geral, dois dias.

Recursos Escassos e Restrições de Tempo na Plataforma de Execução

Por definição, um sistema incorporado tem recursos limitados além do aplicativo que deve executar. Isso é especialmente verdadeiro sobre plataformas minúsculas nas quais a RAM disponível é menos que 1 KB; a conexão ao ambiente de desenvolvimento pode ser estabelecida apenas utilizando probes, emuladores ou links seriais JTAG ou quando a velocidade do microprocessador é boa o suficiente para manipular a tarefa. A ferramenta de teste enfrenta um comércio difícil—ter os dados de teste na plataforma de desenvolvimento e enviar dados sobre o link de conectividade à custa (normalmente insuportável) do desempenho ou ter os dados de teste interpretados pelo driver de teste na plataforma de destino.

A tecnologia utilizada pelo Rational Test RealTime é incorporar o trabalho de teste no sistema de destino. Isso é feito compilando os dados de teste anteriormente traduzidos para a linguagem de programação de aplicativo (C, C++ ou Ada) dentro do trabalho de teste, utilizando o compilador cruzado disponível e, em seguida, vinculando esse arquivo de objeto do trabalho de teste ao restante do aplicativo. Essa cadeia de construção fica transparente para o usuário utilizando a interface de linha de comandos do Rational Test RealTime nos makefiles. A geração de códigos otimizada, a alocação inteligente de mapas de links e a área de cobertura pouca memória, tudo age para otimizar os recursos requeridos pelo trabalho de teste na plataforma de destino enquanto fornece os seguintes benefícios:

- A exatidão da hora é aprimorada e utilizar o local no destino reduz o impacto em tempo real no desempenho.
- Evitar que quaisquer informações de dados circulem no link de conectividade durante a execução do caso de teste minimiza a comunicação no host de destino. Se necessário, os dados de observação são armazenados na RAM e transferidos por upload com a ajuda de um depurador ou emulador quando o caso de teste estiver estritamente finalizado.

Embora ambientes de desenvolvimento cruzado estejam se tornando mais amigáveis, uma grande parte dos sistemas incorporados atualmente fornecidos para desenvolvimento e teste ainda são complicados o suficiente que dão a maioria de nossas dores de cabeça. O objetivo do Rational Test RealTime é simplificar a dura rotina do desenvolvedor de sistemas incorporados.

Falta de Modelos Visuais Claramente Projetados

Desenvolvedores incorporados gostam de códigos! Infelizmente, graduados no pós-secundário não são bem treinados em modelagem visual e tendem a acreditar que código é a "coisa real". Embora a modelagem visual, através de linguagens como o UML, tenham tido muito progresso em direção à incorporação de conhecimento incorporado em um design, a maioria dos programadores de sistemas incorporados ainda adoram fazer a maior parte de seu trabalho utilizando uma linguagem de programação antiga pura. E Java não está introduzindo muitas pessoas ao design!

Para ajudar os desenvolvedores a trabalhar da maneira que preferem, o Rational Test RealTime concentra-se em ajudá-los a projetar casos de teste com base no código fonte do aplicativo fornecendo assistentes, como geradores de gabaritos de teste e wrappers de API. Certificar-se de que o caso de teste pode ser aplicado ao aplicativo é o principal benefício desse processo de

construção de teste baseado em código. A desvantagem é que não há garantia de que o caso de teste reflita o requisito que ele deve verificar. Claramente, a adoção mais ampla da modelagem visual reduzirá o intervalo entre o requisito e o caso de teste. O Rational Test RealTime também está preparando o caminho para essa técnica oferecendo um diagrama de seqüência UML do Rational Rose RealTime para o compilador de caso de teste.

Emergindo Padrões de Qualidade e Certificação

Para uma determinada categoria de sistemas incorporados—sistemas críticos à segurança—a falha não é uma opção. Encontramos esses sistemas nas indústrias nuclear, médica e aviônica. Caminhando no objetivo de falha zero de tempos atrás, o segmento de mercado de aviões e as agências governamentais (como a Federal Aviation Authority nos EUA) uniram-se para descrever o *Software Considerations in Airborne Systems and Equipment Certification* referido como o padrão DO-178B do RTCA, descrito em [R2]. Esse é o padrão predominante para desenvolvimento de software crítico à segurança no segmento de mercado aviônico no mundo inteiro. Como um dos padrões de desenvolvimento de software mais rigorosos, ele também está sendo parcialmente adotado em outros setores nos quais sistemas críticos à vida são manufaturados, como empresas automotivas, médicas (o FDA apenas liberou um padrão próximo ao DO-178B), de defesa ou de transporte.

O DO-178B classifica o software em cinco níveis de criticidade relacionados ao comportamento de software anônimo que causaria ou contribuiria para uma falha de função do sistema. O mais crítico é o equipamento de nível A, no qual a falha resulta em uma condição de falha catastrófica para o sistema global. O DO-178B inclui etapas muito precisas para certificar-se de que o equipamento de nível A esteja suficientemente seguro, em particular, na área de teste. O Rational Test RealTime atende a todos os requisitos de teste obrigatórios do DO-178B, para todos os níveis, até e inclusive o equipamento de nível A.

Sumário

Neste documento, apresentamos uma visão geral do processo de seis etapas incrementais utilizadas para testar sistemas incorporados. Considerando esse processo (levando em conta todo o espectro de sistemas muito pequenos a sistemas muito grandes) e as características e restrições específicas de sistemas incorporados, deduzimos um conjunto de requisitos que uma tecnologia ideal deve possuir para enfocar o teste de sistemas incorporados. O Rational Test RealTime, a nova oferta da Rational para o domínio de sistemas incorporados, exemplifica grandes partes dessa tecnologia ideal.

Este documento forneceu uma introdução geral para testar sistemas incorporados e, durante o ano, será seguido de outros artigos que focalizam os vários tópicos discutidos. ☒

Terminologia

Paul Szymkowiak escreveu esta seção.

Infelizmente, muitos termos utilizados no mundo da engenharia de software têm definições diferentes. Os termos utilizados neste documento são provavelmente os mais familiares àqueles com os quais você trabalha no domínio de sistemas incorporados em tempo real. No entanto, existem outros termos equivalentes e relacionados utilizados no teste de software. Fornecemos um mapeamento na seguinte tabela.

Termos Utilizados Neste Documento	Termos Equivalentes/Relacionados
Grânulo sob Teste (GuT): Um elemento do sistema que foi isolado de seu ambiente com a finalidade de teste.	*termos relacionados*: Item de teste, Destino, Destino de Teste
Ponto de Controle e Observação: Um ponto específico em um teste no qual uma observação do ambiente de teste é registrada ou uma decisão é tomada com relação ao fluxo de controle do teste.	termo mais equivalente: **Ponto de Verificação**
Pós-introdução: As ações executadas para levar o sistema a um estado de finalização estável específico, requerido para executar o próximo teste.	Pós-condição, Reconfigurar
Introdução: As ações executadas para levar o sistema a um estado de início estável específico, requerido para executar o teste.	Condição prévia, Configurar
Trabalho de teste: Uma disposição de um ou mais drivers de teste, stubs específicos do teste e instrumentação combinada com a finalidade de executar uma série de testes relacionados.	*termos relacionados*: Conjunto de Teste ou Driver de Teste, Stub de Teste
Testador Virtual: (1) Algo externo ao GuT que interage com o grânulo durante um teste. (2) Uma instância de um teste em execução, normalmente representando as ações de uma pessoa.	*termos relacionados*: Script de Teste ou Driver de Teste, Stub de Teste, Simulador, Testador

Referências

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

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

Sobre o Autor

Vincent Encontre é o Diretor da Embedded and RealTime, Automated Testing Business Unit, localizada no novo centro de engenharia da Rational em Toulouse, França. Quando não está viajando, participando de reuniões ou respondendo a zilhões de e-mails, Vincent vê o destino do Rational Test RealTime como um produto da Rational e como tecnologias reutilizáveis para as próximas gerações de produtos da Rational. Vincent tem ampla experiência em tecnologias incorporadas de modelagem e teste e em boas práticas. Antes da Rational e da ATTOL, Vincent passou 13 anos na Philips, em seguida, na Verilog, projetando, comercializando e fornecendo suporte a ferramentas de engenharia de software, acreditando que essas ferramentas poderiam ajudar a construir melhor software de maneira mais rápida.

Nas horas livres, Vincent joga futebol com seus três filhos e colegas e aproveita a maravilhosa arte de viver do sul da França (antes que seja muito tarde…).

Rational®
the software development company

Duas Sedes:

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

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

Sem custo: (800) 728-1212
E-mail: info@rational.com
Web: www.rational.com
Localização Internacional:www.rational.com/worldwide