

Computação Reconfigurável em Sistemas Operacionais Baseados em Contexto¹

Mauro M.Mattos², Roberto Pacheco³

²Depto.Sistemas e Computação – Universidade Regional de Blumenau (FURB)
Caixa Postal 1507 – 89.010-971 – Blumenau – SC – Brasil

³Depto.Eng.de Produção e Sistemas – Universidade Federal de Santa Catarina (UFSC)
Campus Universitário, Trindade, Florianópolis – SC - Brasil

mattos@furb.br, pacheco@eps.ufsc.br

Abstract. *The main purpose of any operating system has being until now to multiplex shared resources between applications by making them virtual to the applications in such a way that each application seems to have all resources available. Support for reconfiguration by general purpose computing platforms has been the subject of considerable research. Much of this work is based on an evolutionary strategy in which small changes to existing systems are made. This work analyses some important characteristics that should be present at the O.S. level in order to provide better serviceability and presents the concept of a context-based operating system architecture that can conduce towards a new generation operating systems more user-friendly and that provides better QoS.*

Keywords. *QoS, Context-based O.S., Machine Learning.*

Resumo. *Uma das principais características dos sistemas operacionais é a de multiplexar recursos compartilhados de forma a virtualizá-los às aplicações de tal forma que as mesmas pareçam possuir todos os recursos disponíveis. Neste contexto, o suporte a reconfiguração tem sido objeto de considerável esforço de pesquisa. O presente trabalho analisa algumas características importantes que devem ser suportadas a nível de sistema operacional de forma a permitir um melhor nível de atendimento e, apresenta o conceito de uma arquitetura de sistema operacional baseada em contexto, a qual deve conduzir a uma nova geração de sistemas operacionais mais amigáveis e que apresentem melhor qualidade de serviço.*

Palavras Chave. *Qualidade de Serviço, Sistemas Operacionais Baseados em Contexto, Aprendizado de Máquina.*

1. Introdução

Os sistemas operacionais tradicionais suportam a noção de um nível de abstração de hardware sobre o qual cada aplicação supõe possuir seu próprio processador (e outros recursos). Assim é possível adotar uma política de compartilhamento dos recursos físicos

¹ Trabalho parcialmente suportado pela FURB e CNPQ.

de forma transparente. O que no passado constituía-se numa solução aceitável, hoje não mais atende aos requisitos das aplicações porque cada vez mais há necessidade de desenvolvimento de produtos de software maiores, mais complexos, e que executem em diversas plataformas e sistemas operacionais diferentes.

Recentes avanços na tecnologia de componentes permite a construção de sistemas complexos através da composição dos mesmos. No entanto, ainda é difícil desenvolver sistemas eficientes, confiáveis e dinamicamente configuráveis visto que os componentes frequentemente são desenvolvidos por equipes diferentes que utilizam metodologias diferentes o que naturalmente conduz a falhas inesperadas o que compromete a confiabilidade do produto final.

[Kon 2000] propõe uma infraestrutura para suporte ao gerenciamento de dependências para permitir as seguintes funções: configuração automática, distribuição de código, suporte a desenvolvedores, tolerância a falhas, reconfiguração consistente e adaptação. Basicamente a solução proposta executa as seguintes funções: recupera o código do componente e os pré-requisitos do mesmo a partir de um repositório de componentes e dinamicamente insere-o no espaço de endereços da aplicação. Com base nos pré-requisitos, repete o processo para os demais componentes (se houverem). Embora seja uma proposta interessante, ela recai no lugar comum dos projetos de sistemas operacionais existentes – trata o problema pontualmente.

O presente trabalho está organizado da seguinte forma: a seção 2 apresenta a problemática constatada nos atuais projetos de sistemas operacionais (comerciais e acadêmicos), a seção 3 analisa os projetos de sistemas operacionais sob a ótica de usabilidade e robótica, a seção 5 descreve o modelo proposto e finalmente são apresentadas as conclusões do trabalho.

2. A Situação Atual

A possibilidade de acesso a informações sob demanda em qualquer local caracteriza uma tendência de pesquisa multidisciplinar cada vez maior, principalmente a partir da disseminação de redes sem fio (*wireless*) e tecnologia de interconexão cada vez mais disponível (hardware e protocolos). Neste contexto deve-se destacar dois projetos: o projeto Aura [Wang 2000] e projeto GaiaOS [Román 2000].

A idéia é transformar os espaços físicos em sistemas interativos ou *active spaces*. A proposta é estender os sistemas computacionais atuais transformando dispositivos e as máquinas contidas em determinado espaço físico em entidades físicas e virtuais permitindo uma forma de interação com os equipamentos de forma simples e transparente. Neste contexto, um usuário pode dispor de todos os recursos audiovisuais disponíveis em uma sala de apresentação simplesmente solicitando que a apresentação armazenada em um *notebook* seja projetada por um projetor de parede da mesma forma que solicita a execução em seu próprio *notebook*.

Contudo, a heterogeneidade, mobilidade e diversidade de dispositivos tornam o sistema muito mais complexo. A proposta dos *active spaces* abre uma nova dimensão em termos computacionais comparada aquela aberta com a introdução da tecnologia de redes. Deixou-se de ter equipamentos independentes e passou-se a contemplar o compartilhamento de recursos. Agora quebra-se o paradigma da interligação de computadores e abre-se uma nova porta, permitindo a integração ao ambiente computacional de outros equipamentos que hoje requerem configuração, coordenação e tratamento diferenciados. Considerando-se que ainda não foram solucionados os

problemas relacionados a suporte adequado a reconfiguração em ambientes tradicionais, no contexto de um computador pessoal (por exemplo), restrito às interfaces conhecidas, é de se imaginar a extensão do problema quando se passa a considerar um contexto de *active spaces*.

Em função das afirmações acima e que [Fleish 1983] afirmava (no início da década de 80) que a inteligência deveria ser um elemento chave no projeto dos futuros sistemas operacionais, o que se verifica de fato é que ainda há muito a ser feito para que os sistemas atinjam o nível de qualidade acima proposto. Isto porque, a medida em que cada vez mais recursos são disponibilizados aos usuários, ocorre uma demanda cada vez mais crescente por qualidade de serviço e capacidade de reconfiguração automática o que vem superando a capacidade dos projetistas em prover soluções adequadas.

3. Novos Requisitos

Segundo projeções baseadas em vários estudos realizados por [Patki 1996] o enfoque convencional de arquitetura de hardware bem como dos sistemas operacionais e softwares aplicativos não serão suficientes para atender as necessidades de tecnologia de informações no contexto de prover informações para as grandes massas. Isto porque, ao contrário da visão atualmente utilizada, os futuros sistemas a ser construídos necessitarão apresentar, entre outras, as seguintes características:

- capacidade de tratar informações parciais e/ou imprecisas e de extração de conceitos;
- raciocínio aproximado e aprendido;
- facilidades de uso para usuários sem background em computadores;
- facilidades para usuários especializados aumentar as bibliotecas do S.O. sem comprometer os aspectos de confiabilidade e segurança;
- capacidade de auto-diagnóstico visando facilitar atividades de administração e interação;
- sistema de arquivos com características não limitantes (como atualmente);
- reconhecimento da frequência de uso de comandos durante o ciclo de vida de uma aplicação;
- identificação de perfil de usuário e adequação do ambiente de acordo com cada perfil.

[Wang 2000] acrescenta alguns aspectos relacionados a chamada computação pervasiva onde os usuários podem acessar e manipular informações de qualquer lugar em qualquer tempo. Naturalmente a infraestrutura computacional não suporta este modelo de computação de forma adequada, principalmente porque os computadores interagem com usuários em termos de abstrações de baixo nível: aplicações e dispositivos individuais.

O grande desafio consiste portanto em adequar os requisitos identificados nos chamados *active spaces* com aqueles necessários à geração de uma tecnologia que ao mesmo tempo de suporte ao projeto tecnológico e consiga tornar as tecnologias computacionais mais eficientes, amigáveis e adaptáveis, ou seja, possibilitem a reconfiguração do sistema operacional em função de alterações no ambiente de tal forma a minimizar a necessidade de intervenção do usuário.

4. Considerações Importantes

[Nielsen 1994] apresenta um conjunto de regras heurísticas que foram usadas para examinar uma base de dados contendo 249 relatos de problemas de usabilidade de computadores. Aplicando-as para avaliar o sistema operacional como um todo, pode-se

chegar a algumas conclusões interessantes e identificar-se alguns problemas ainda sem solução imediata, os quais são discutidos a seguir:

- Visibilidade do estado do sistema: O sistema deve sempre manter os usuários informados sobre o que está acontecendo, através de um feedback apropriado e dentro de um tempo razoável; Como é que um S.O. vai informar ao usuário o que está acontecendo, se nem mesmo ele tem conhecimento para isto? O nível de domínio do S.O. restringe-se principalmente ao tratamento de exceções, e estas são de muito baixo nível e não agregam informação útil ao usuário (ex: famosa tela azul do Windows). Se os programas são bem comportados, não há muito feedback a ser fornecido além dos tradicionais: quantidade de memória e disco disponíveis, percentagem de uso de processador, etc.

- Sincronismo (*match*) entre sistema e o mundo real: O sistema deve falar a língua do usuário, com frases e conceitos familiares ao invés de usar termos do domínio do sistema; seguir convenções do mundo real fazendo as informações parecerem de forma lógica e natural; Esta questão ainda está longe de ser solucionada a nível de S.O. visto que, como afirmado acima, o sistema possui pouco conhecimento sobre seu real estado, e por outro lado, desconhece o perfil do usuário e das aplicações.

- Controle do usuário e liberdade: Eventualmente os usuários escolhem alguma função por engano e necessitam de uma saída de emergência para sair deste estado sem ter que navegar por vários menus; Como o sistema não tem controle sobre as aplicações, isto fica a cargo de cada aplicação individualmente o que faz com que o usuário necessite treinamento específico a cada tipo de aplicação, e o comportamento geral do sistema sob a ótica do usuário seja orientada por aplicativos. É interessante observar que, por conta deste fato, quando uma aplicação produz uma situação de erro, a "culpa" recai sobre o S.O. .

- Consistência e padronização: Os usuários não deveriam ter que se preocupar se diferentes palavras, situações ou ações significam a mesma coisa; deve-se seguir os padrões da plataforma; Esta questão é interessante tendo em vista que, um S.O. sendo um software genérico, tem que manter compatibilidade com versões antigas de aplicativos ao mesmo tempo em que insere novos conceitos de interface e recursos adicionais. Esta mesma situação pode conduzir a ambigüidades na medida em que tarefas podem ser executadas de várias formas possíveis dificultando o aprendizado de usuários menos experientes.

- Identificar ao invés de relembrar: Tornar objetos, ações e opções visíveis. O usuário não deveria ter que lembrar informações de um diálogo para outro. Instruções para uso do sistema deveriam ser visíveis ou facilmente recuperáveis sempre que apropriado; Este é um dos aspectos mais críticos constatados na interface dos atuais S.O.'s. Quando um problema acontece, freqüentemente há a necessidade de percorrer vários menus e aplicações para relembrar configurações atuais para tentar solucionar o problema. Por exemplo: configurações de rede.

- Projeto harmonioso e simples: Os menus não devem conter informações que são irrelevantes ou raramente utilizadas; cada unidade extra de informação em um diálogo compete com unidades relevantes e diminui a visibilidade relativa das mesmas; Esta facilidade já pode ser encontrada em sistemas mais recentes (Windows 2000, ME) e aplicativos (ex: pacote *office* da Microsoft); e apresentam como característica principal a limpeza das listas de opções dos menus; além disso, pode-se encontrar no mesmo pacote *office* a figura do agente que procura antecipar ações do usuário, auxiliando assim a usuários leigos.

- Auxiliar o usuário a reconhecer, diagnosticar e recuperar-se de erros: Mensagens de erros não devem ser expressas através de códigos e devem indicar precisamente o problema e

construtivamente sugerir soluções; Este aspecto está longe de ser solucionado, visto que a própria estrutura de API's dos sistemas é orientada por códigos de erro. Há algumas iniciativas de minimizar o efeito dos códigos de erro através do uso de *wizards* (windows), no entanto, sua abrangência ainda é limitada .

- Ajuda e documentação: Mesmo se o sistema possa ser utilizado sem documentação, ela pode ser necessária para prover auxílio em determinado momento; qualquer fonte de informação deve ser de fácil consulta, orientada para tarefas, listar passos concretos a serem seguidos e não ser muito extensa; Este aspecto apresenta características conflitantes se analisado sob o contexto de um sistema operacional, dada a sua abrangência e generalidade. No entanto, um sistema com maior nível de conhecimento interno, contribuiria para a redução no número de possibilidades de diferentes soluções de problemas.

[Hugh 1997] finaliza suas considerações afirmando que a complexidade da questão é enorme e efetivamente não há um padrão de fato estabelecido. Este problema é potencializado a medida em que se incluem novas aplicações no contexto de *active spaces* onde a diversidade e heterogeneidade de equipamentos interligados é muito maior e os requisitos de QoS e capacidade de reconfiguração aumentam.

A pesquisa em robótica já há muito tempo vem trabalhando no desenvolvimento de arquiteturas tanto de hardware como de software objetivando a construção de robôs móveis autônomos capazes de operar em ambientes complexos, parcialmente conhecidos e que oferecem riscos, usando limitados recursos físicos e computacionais. Segundo [Medeiros 1998] os principais requisitos a serem atingidos neste tipo de projeto são: Reatividade ao ambiente, comportamento inteligente, integração de múltiplos sensores, resolução de múltiplos objetivos, robustez, confiabilidade, modularidade, flexibilidade, facilidade de expansão, adaptabilidade e raciocínio global.

Como em toda área de pesquisa, há algumas decisões de compromisso que devem ser consideradas em relação a autonomia e a complexidade dos projetos. Por exemplo, reduzindo-se o nível de autonomia através da atribuição de tarefas complexas a agentes externos pode simplificar o projeto em detrimento da capacidade de decisão do robô; reduzindo-se a complexidade do ambiente através da introdução de pontos de referência também diminuem a complexidade das atividades de planejamento de rota, mas limitam a capacidade de aplicação dos robôs; e finalmente, alguma solução de compromisso deve ser adotada para compatibilizar a necessidade de respostas rápidas em detrimento de assimilação de informação necessária ao aprendizado do robô.

5. O Modelo Proposto

Um aspecto de fundamental importância nos projetos de robótica e que é negligenciado pelos projetistas de sistemas operacionais refere-se ao fato que, em projetos de robótica (considerando-se robôs autônomos), do mais simples ao mais complexo, há uma função de mapeamento entre a realidade e uma representação interna da mesma denominada: "modelo de mundo" - ou seja, há alguma forma de representação do ambiente onde o robô vai operar. E é baseado neste modelo de mundo que as decisões são tomadas.

Embora os sistemas operacionais também utilizem registros (na forma de diretórios especiais *-/proc* no *linux*; *registry* do *windows*, etc) para descrever seu estado em determinado momento, eles não apresentam um comportamento externo tão inteligente quanto àquele apresentados pelos robôs. Naturalmente, o escopo e complexidade das

aplicações são diferentes, no entanto, o aspecto filosófico é que deve ser destacado. E é justamente este o ponto de partida do projeto que está sendo desenvolvido.

A grande questão por trás de uma gama enorme de problemas relacionados a informática de uma forma geral são decorrentes do modelo utilizado onde tem-se uma camada de sistema operacional que provê uma abstração dos detalhes de mais baixo nível (periféricos de um modo geral) e sobre esta abstração, são programadas seqüências de comandos que devem ser executadas. Quando estas seqüências não são completas, ou não prevêm todas as possibilidades possíveis incorre-se em situações de erro – onde algumas são tratáveis e outras não. Assim sendo, o modelo atual segue o paradigma de fazer-pela-máquina (ou de fora-para-dentro), ou seja, o programador não “ensina” o sistema como proceder, mas faz por ele (na forma de programas utilitários, scripts e programas). Assim sendo, abre-se toda uma gama de possibilidades de acertos e erros cuja ocorrência e detecção tornam-se cada vez maiores e mais complexos. Complexos para entender, para desenvolver, para implementar, para testar e finalmente para usar.

O modelo proposto baseia-se no paradigma de ensinar-a-fazer, ou seja, o programador passa a treinar o sistema (de forma semelhante ao treinamento de uma rede neuronal). E o sistema tem o controle de toda a situação. Assim sendo o sistema nunca ‘entrega’ a cpu para um código externo (tal como um programa .exe ou .com, ou um Shell script etc) mas sim recebe uma especificação de comportamento a ser executada. A medida em que o tempo passa, o sistema vai “amadurecendo”. O que não significa que não possa ocorrer erros. No entanto, tais situações são objeto de estudo de tal forma que, uma vez que o sistema não tenha conhecimento suficiente para resolver a situação, ele solicita intervenção do usuário e esta intervenção passa a incorporar a base de conhecimento.

Portanto, para exteriorizar um comportamento inteligente, faz-se necessária uma base de conhecimento que seja expansível e verificável. Esta base de conhecimento inicialmente contém a descrição do sistema físico e de suas capacidades de comunicação com o mundo exterior (tais como : vídeo, teclado, mouse, modem, rede, etc). O próprio conceito de driver como utilizado hoje acaba sendo substituído por especificações de comportamento. O código é gerado e administrado pelo sistema. O programador informa ao sistema o comportamento do periférico e sua utilidade. A medida da necessidade o periférico é utilizado ou não.

O processo de “instalação” de uma aplicação apresenta o mesmo comportamento. Ao invés de transferir toneladas de código para o sistema (como ocorre atualmente), apresenta-se a descrição do comportamento da aplicação. O sistema transforma este comportamento em código executável e insere-o na base de conhecimento. A descrição do comportamento não restringe-se somente a especificação de passos de programa, mas da descrição conceitual do mesmo. Esta descrição conceitual é aquela utilizada pelo desenvolvedor quando utiliza uma ferramenta case para desenvolver o modelo lógico da aplicação. É opinião dos autores que, uma grande parcela dos erros de programação e das decorrentes anomalias de comportamento das aplicações deve-se ao fato de que informações adquiridas através do processo de modelagem das aplicações são perdidas na transição entre a especificação e a efetiva implementação do código. Uma vez que estas informações são transferidas para a base de conhecimento do sistema, elas contribuem para tornar o sistema mais robusto.

A aquisição de conhecimento ocorre também durante a transferência para o sistema de algum documento (quer via disquete, via rede, ou via teclado, ou via voz , etc). No momento da introdução da informação, o sistema classifica o conteúdo do documento de tal forma a ter conhecimento sobre o que ele está armazenando. Quando de uma busca, o

sistema é capaz de responder sobre o que ele possui sob custódia, não sendo necessário ao usuário instalar indexadores e construir consultas proprietárias. O sistema adapta-se ao perfil de cada usuário a medida do seu uso.

Uma decorrência natural disto é que deixa de existir o conceito de *file-system* (e portanto de programas tipo windows explorer) como utilizado atualmente. Isto porque o sistema sabe que tipo de informação o arquivo contém, e onde ele o colocou. Se o usuário deseja utilizar um determinado arquivo, solicita ao sistema e este o disponibiliza através da aplicação adequada. Não é mais necessário ao usuário navegar por diretórios e pastas e sub-pastas. O sistema tem o controle da organização da informação armazenada. Assim, conceitos de permissão sobre arquivos passam a ser desnecessárias pois as aplicações desconhecem onde os arquivos estão armazenados e portanto não há como fazer mau uso dos mesmos.

Outro aspecto importante refere-se ao fato de que como o sistema não entrega o controle para código externo, diminuem-se as possibilidades de ocorrências de invasão por código não autorizado (tais como os famosos vírus de hoje). Um vírus neste contexto seria uma especificação de comportamento anôma-la, mas jamais um código executável não autorizado. Assim, especificações de comportamento que possuam informações conflitantes ou são resolvidos pelo sistema através de soluções anteriores, ou são apresentados ao usuário para que “informe ao sistema como proceder”. Aplicações de anti-vírus neste contexto passam a constituir-se em um conjunto de regras de comportamento espúrio a ser evitado e não mais de bases de código a ser testada contra todos os arquivos armazenados como ocorre hoje. Associando-se isto ao fato de que não há mais acesso direto aos arquivos, reduz-se consideravelmente a amplitude da ação de um vírus como os que existem atualmente. Naturalmente não se pode afirmar que eles não possam (ou que não venham) a ser desenvolvidos para este tipo de arquitetura. No entanto, é opinião dos autores que tanto a amplitude, como a eliminação dos mesmos deve ser uma tarefa mais amena e de conseqüências menores que àquelas experimentadas atualmente.

Nesta arquitetura proposta, as atividades de reconfiguração do sistema são transparentes uma vez que, os pré-requisitos são assimilados pelo sistema e a integridade também. Assim, módulos interdependentes não podem ser apagados e a substituição dos mesmos é parcial até que todas as aplicações dependentes atestem o funcionamento correto de um novo módulo instalado no sistema. Durante o processo de substituição (o qual não é imediato), aplicações já validadas utilizam os módulos novos, enquanto aquelas que ainda não foram validadas continuam a executar com os módulos antigos. Não cabe mais ao usuário decidir por exemplo, se uma determinada dll pode ser substituída ou não (como ocorre atualmente!).

6. Conclusão

O presente documento relata o andamento de um projeto de pesquisa que visa propor uma arquitetura de sistema operacional que forneça suporte adequado ao desenvolvimento de aplicações mais inteligentes e que permita a interação entre o usuário e os recursos computacionais de forma mais transparente. É opinião dos autores que uma nova geração de sistemas operacionais depende de recursos que devem ser providos pelo núcleo do sistema operacional de tal forma que, o controle e o conhecimento sobre o que as aplicações estão realizando seja unificado. Assim, vários aspectos relacionados anteriormente podem ser enfocados de forma mais natural o que deve contribuir para o surgimento de aplicações mais eficientes e orientadas ao auxílio à solução dos problemas do usuário, ou seja, orientadas a tarefas e não a aplicações como ocorre hoje.

Além disso, a tendência indica uma utilização cada vez maior no sentido de incorporar-se conceitos da área de inteligência artificial de um modo geral, nas várias etapas que compõe a arquitetura de um sistema operacional, de forma a torná-lo mais amigável e fazer com que ele efetivamente assuma um papel de administrador de recursos no sentido mais amplo da palavra facilitando não só a utilização dos equipamentos através de interfaces mais amigáveis, mas também auxiliando na solução de problemas que o que hoje se constitui um desafio.

Finalizando, o aspecto mais importante a ser considerado nos projetos de sistemas operacionais visando torná-los mais inteligentes, eficientes e amigáveis é como torná-los mais sensíveis ao contexto em que estão operando. Como visto até agora, os sistemas operacionais (e toda a gama de software desenvolvido –por consequência) estão baseados no paradigma de Entrada-Processamento-Saída a qual é muito restritiva. A tecnologia que tornará os equipamentos do futuro mais inteligentes, deverá ser capaz de operar sobre dados obtidos por eles mesmos. Eles terão que "sentir" o ambiente, decidir que aspectos da situação são realmente importantes e inferir a intenção do usuário a partir de ações concretas. As ações do sistema serão dependentes do tempo, local, ou do histórico das interações com o usuário – ou seja, deverão ser mais dependentes de contexto.

7. Referências

- Ahmed,O, (1998) "The Application-specific Operating System", Computer Design, pp78.
- Fleish,B.D. (1983) "Operating Systems:A perspective on future trends", ACM SIGOPS Operating Systems Review. Vol 17, no.2, pp 14-17.
- Hugh,J. (1997) "User Interface Design – How Can It Be Improved?"
<http://www-dce.doc.ic.ac.uk/~nd/surprise97/journal/vol2/hafj/hafj.html>, 1997
- Kon,F, (2000) "Automatic Configuration of Component-Based Distributed Systems", PhD Thesis. Department of Computer Science, University of Illinois. Urbana-Champaign.
- Leslie,I et al.(1997) "The Design and Implementation of an Operating System to Support Distributed Multimedia Applications"
<http://www.cl.cam.ac.uk/Research/SRG/netos/nemesis/documentation.html>, 1997.
- Medeiros,A.A.D. (1998) "A Survey of Control Architectures for Autonomous Mobile Robots", Journal of the Brazilian Computer Society n3 v4, Rio de Janeiro.
- Nielsen,J. (1994) "Enhancing the explanatory power of usability heuristics", Proc.ACM CHI'94 Conf. Boston, MA.
- Patki,A.B. (1996) Fuzzy Logic Based Hardware: Some Experiences, Proceedings of First International Discourse on Fuzzy Logic And The Management of Complexity FLAMOC'96, January, 15-18, 1996, Sydney, Australia, Vol.3, pp 247-251.
- Román,M., Campbell ,R.H., (2000) "Gaia: An Operating System to Enable Active Spaces", Submitted to the 9th ACM SIGOPS European Workshop.
- Wang, Z, Garlan, D. (2000) "Task-Driven Computing", Technical Report, CMU-CS-00-154, School of Computer Science, Carnegie Mellon University.