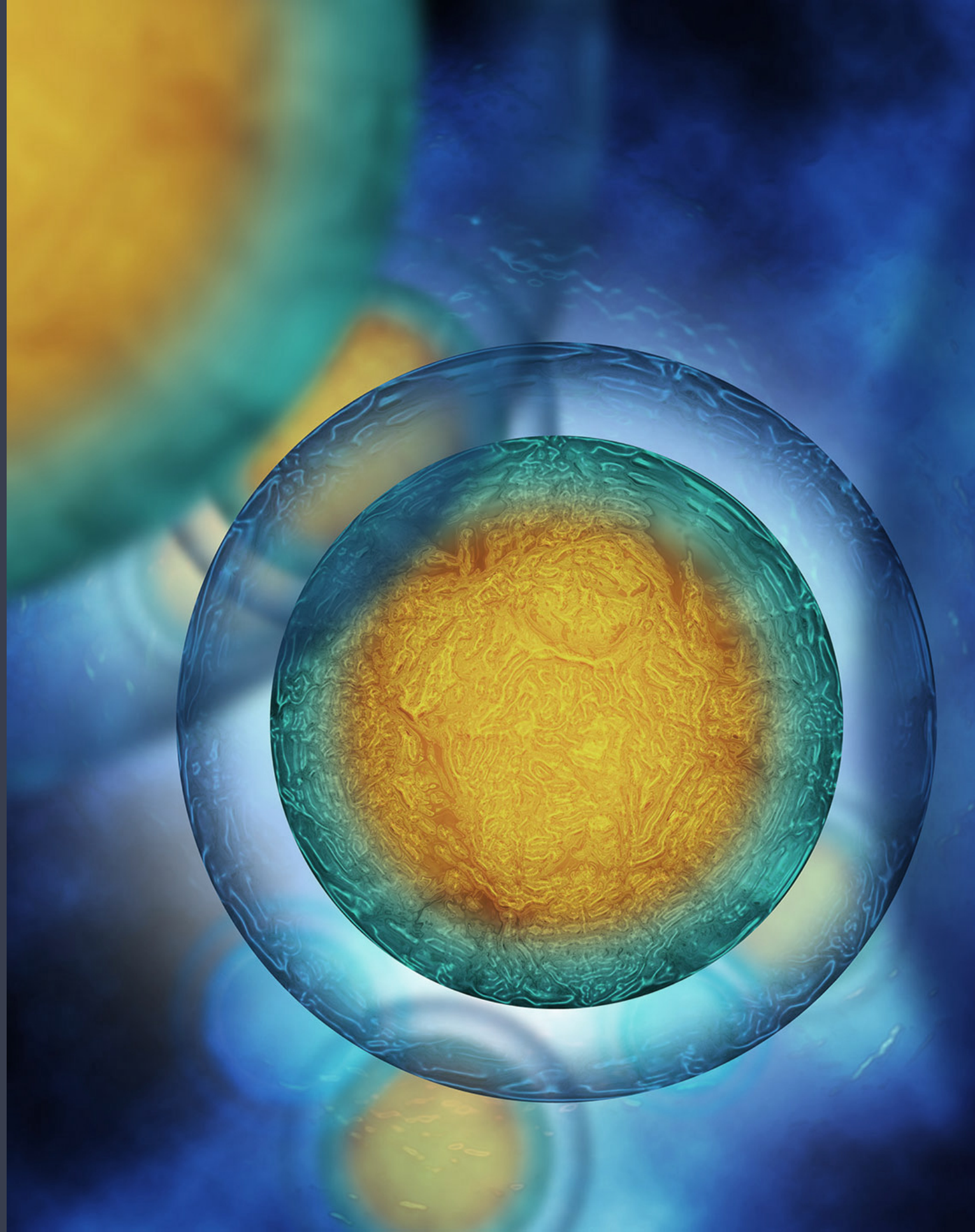
FE3CWS

MATERIAIS ESCOLARES DE INVERNO OS TRÊS “CO”

Produção intelectual 1 do projeto ERASMUS + 2017-1-SK01-KA203-035402



Algumas palavras sobre o

CONTEÚDO

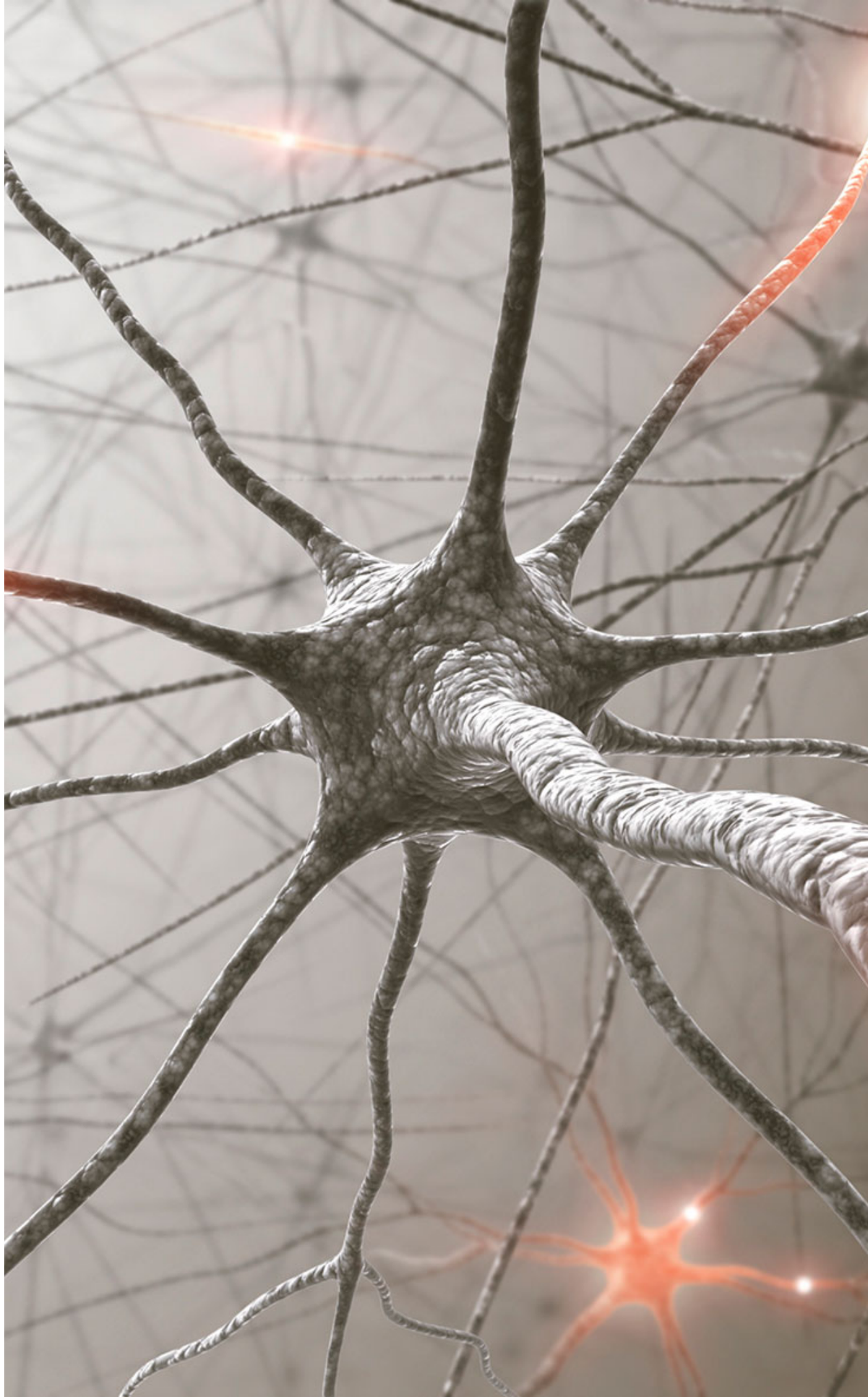
- **9 tópicos relacionados à composição, compreensão e correção de software**
- **10 autores de 7 universidades europeias da Croácia, Hungria, Holanda, Portugal e Eslováquia**
- **Disponível em 7 idiomas: inglês, húngaro, eslovaco, croata, romeno, búlgaro e português**

Co-funded by the Erasmus+ Programme of the European Union

As três escolas de inverno “CO” (compostabilidade, compreensibilidade e correção) (3COWS) são o primeiro programa intensivo para alunos do ensino superior e professores que estendem a comunidade da escola de verão da Programação Central da Europa (CEFP), no marco do projeto ERASMUS + 2017-1-SK01-KA203-035402 “Focusing Education on Composability, Comprehensibility and Correctness of Working Software”, realizada entre 22 e 26 de janeiro de 2018.

O material incluído foi criado e apresentado no quadro do projeto mencionado acima. Esta publicação é a versão impressa da saída intelectual O1 do projeto.

ÍNDICE



AQUISIÇÃO DE LINGUAGEM NATURAL POR MÁQUINAS

A construção da máquina pensante é um grande desafio. O relacionamento da linguagem e da mente é interessante para cientistas da computação é discutido em um contexto mais amplo também por historiadores, psicólogos, linguistas e filósofos.

Por exemplo, Renfrew [31] caracteriza humano como um ser simbólico considerando que o desenvolvimento da linguagem natural é o resultado de troca de símbolos na comunicação. Gardenfors [14] afirma que o conceitos de linguagem têm uma estrutura hierárquica. Evolução gramatical [24] e a evolução genética das línguas [15] impactam o problema de expansão estrutural, portanto, são aplicáveis apenas a simples idiomas de maneira determinística. Acontece que as cadeias genéticas no papel dos valores das conclusões semânticas é mais promissor abordagem à evolução da linguagem do que a simples aplicação da estrutura genética do cérebro humano por um analogia. Também reconhecemos que o processo de evolução ocorre em diferentes níveis meta-idioma

Hipótese de Chomskys sobre a existência de uma gramática universal para todos línguas naturais [7] é muito interessante e motivador. Mas talvez a gramática universal não deve ser entendida como um texto fixo e estrutura gramatical parametrizada, mas como um instrumento universal algoritmo, ou seja, progresso determinístico no nível da meta-linguagem parametrizado pelos parâmetros que representam significativos, ou seja, valores semânticos. Edelman [12] afirma que a mente da máquina tem substância da linguagem e é simbólico e ao mesmo tempo computacional. Segue-se que a mente não é apenas simbólica, mas também um processo dinâmico, que expressa uma mudança incessante de linguagem.

E finalmente, na gramática universal aplicada de Shaumyans [34], a linguagem é também expresso em termos de processos dinâmicos e até aplicativos.

O benefício da teoria da linguagem semiótica é que a sintaxe e semântica são mutuamente ligadas a cada vez e são não separável. Obviamente, isso invoca a necessidade de tomar em conta categorias semânticas para expressar a mudança de um idioma.

Em nossa opinião, a teoria semiótica das línguas fornece uma oportunidade para evolução determinística da máquina mente para o caso de subconjuntos de idiomas naturais e linguagens formais em diferentes formas usadas em comunicação. Nesse sentido, Human-Computer e Comunicação computador-computador baseada em idiomas pode ser unificado. Algum ponto de partida dessa idéia, restrito para idiomas regulares que apresentamos em [17]. Atualmente sabemos o algoritmo para a transformação de conceitos de linguagem para a linguagem interna da mente da máquina e estamos também capaz de derivar conceitos dessa linguagem interna. Essa linguagem interna é uma analogia à linguagem interna de pensamento humano - é uma linguagem calma por trás do barulho linguagem natural. No momento, não sabemos nem regras de conceituação nem regras de raciocínio sobre conceitos representados na mente da máquina. No entanto, nós ter uma metodologia clara da evolução da máquina representada por processos dinâmicos aplicativos com alto grau de paralelismo que representam informações de substância da linguagem de forma não redundante. Além disso, aumentar a abstração dos conceitos de linguagem diminui o número de meta-operações e aumenta o número de ligações aplicativas.

Discutimos o algoritmo eficiente do pensamento infantil e estimar a velocidade do fluxo de informações no cérebro humano para 300 trilhões de sinais por segundo. Então apresentamos o eslovaco gramática textual e fonética, bem como o método para tradução de texto para voz. Além disso, ilustramos o processo de aquisição de elementos linguísticos de abstração diferente lendo uma pequena amostra de texto e usando hash hierárquico tabelas como modelo. Também avaliamos o processo de aquisição para texto massivo do livro selecionado e concluir que o idioma aquisição com abstração hierárquica produz não gráfico redundante com o mesmo número de seqüências e ligações paralelas sem a necessidade de armazenamento físico de dados.

Em seguida, usamos a abordagem gramatical para comunicar reconhecimento de objetos visuais. Primeiro, precisamos descobrir como para descrever os objetos e então podemos aplicar o método de abstração para esses dados. Nós nos concentramos principalmente em 3D descrição do objeto usando gramáticas. Na teoria gramatical, essa etapa é chamada simbolização.

A simbolização garante uma descrição do objeto e fornece a camada abstrata de dados fundamentais. Como nós podemos veja, abstração de dados usando linguagem funcional nos permite abstrair e processar objetos facilmente. A abordagem aplicável pode ser usada no processamento de idiomas mesmo quando usamos gramáticas sem contexto. Mostramos uma algoritmo capaz de transformar qualquer contexto livre gramática em forma de supercombinador. O formulário resultante depende da forma de uma gramática de entrada, portanto, uma nova surge um problema: encontrar a gramática apropriada para a tarefa em mão.

Mostramos brevemente as formas supercombinadoras resultantes de vários tipos de gramática e compare suas propriedades. Nós também comparam a eficiência do algoritmo casos gramaticais e mostram que nosso algoritmo pode ser melhorado caso usemos gramáticas sem ciclos. Nós discutir também as formas supercombinadoras resultantes em termos de compressão gramatical e reutilização de elementos que são o resultado final do processamento de textos em maior escala como entrada amostras.

Referências

[1] Renfrew, C.: Prehistory: The Making of the Human Mind. Weidenfeld & Nicholson, (2009)

[2] Gardenfors, P.: Symbolic, conceptual and subconceptual representations, Human and Machine Perception, pp. 255-270, (1997)

[3] ONeill, M., and Ryan, C.: Grammatical evolution. IEEE Transactions on Evolutionary Computation, Vol. 5, No. 4, pp. 349–358, (2001)

[4] Hugosson J., Hemberg E., Brabazon A., ONeill M.: Genotype Representations in Grammatical Evolution. Applied Soft Computing, Vol.10, No.1, pp.36-43, (2010)

[5] Chomsky, N.: Syntactic Structures, Walter De Gruyter: Mouton classic, (1957)

[6] Edelman, Sh.: Computing the Mind: How the Mind Really Works, (2008)

[7] Shaumyan, S: A Semiotic Theory of Language. Bloomington: Indiana University Press, (1987)

[8] Kollar, J.: Formal Processing of Informal Meaning by Abstract Interpretation, Smart Digital Futures 2014, June 18-20, Chania, Greece, pp. 122–131, (2014)

ANÁLISE DE CÓDIGO ESTÁTICO COM CODECHECKER

VISÃO GLOBAL

A execução simbólica [4] é uma análise estática popular técnica usada tanto na verificação do programa quanto no bug ferramentas de detecção. Ele funciona interpretando o código, introdução de um símbolo para cada valor desconhecido na compilação tempo (por exemplo, dados fornecidos pelo usuário) e realizar cálculos simbolicamente. O mecanismo de análise se esforça para explorar vários caminhos de execução simultaneamente, embora verifique todas caminhos é um problema intratável, devido ao grande número de possibilidades.

Embora exista uma rica literatura sobre ferramentas de verificação de programas, as ferramentas de busca de erros normalmente precisam se contentar com a pesquisa trabalhos sobre técnicas individuais [1]. Neste artigo, não discutir apenas métodos individuais, mas também como esses decisões interagem e se reforçam, criando um sistema que é maior que a soma de suas partes. Nós nos concentramos em uma ferramenta de localização de erros chamada Clang Static Analyzer [2] (doravante referido como Analisador) e um infraestrutura construída em torno dele chamada CodeChecker [3]. o A ênfase está no alcance da escalabilidade de ponta a ponta.

Isso inclui o tempo de execução e o consumo de memória do análise, apresentação de erros aos usuários, falsificação automática supressão positiva, análise incremental, padrão descoberta nos resultados e uso em contínuo loops de integração. Também delineamos orientações futuras e problemas em aberto relacionados a essas ferramentas.

Embora o analisador possa lidar apenas com C/C++/ Objective-C código, as técnicas introduzidas neste artigo são independente do idioma e aplicável a outros ferramentas de análise estática.

ANALISADOR ESTÁTICO CLANG

Resumimos o mecanismo de trabalho simbólico execução e sua implementação no Analyzer. Nós discutir sua representação de memória [6], seu manuseio do ligações entre valores e localizações da memória e seus representação de estados específicos de cheques (onde, por cheque, significa um módulo do Analyzer escrito para encontrar um tipo específico de erro). Também introduzimos o conceito de cálculos simbólicos. As escolhas das representações usadas analisador desempenham um papel crucial na criação de análise de software viável.

Como a verificação de todos os caminhos de execução possíveis não é possível dentro de um prazo razoável, precisamos introduzir o conceito do orçamento de análise: uma estimativa do tempo período, podemos nos dar ao luxo de analisar um determinado pedaço de código. o O objetivo é encontrar o maior número possível de erros com um valor baixo de falso. taxa positiva. Mostramos como o analisador prioriza mais caminhos interessantes para análise e como elimina caminhos inviáveis de maneira eficiente, usando restrições em camadas resolução [5].

O analisador também emprega várias heurísticas para suprimir automaticamente os relatórios que provavelmente serão falsos positivos.

Quando um erro é encontrado, o caminho e o conjunto correspondentes de restrições são úteis para entender o problema. Isto é, no entanto, impraticável apresentar todas essas informações ao do utilizador. Mostramos como o analisador se esforça para apresentar o usuário com um relatório de erro conciso e perspicaz que minimiza a hora de corrigir o erro.

CODECHECKER

Definimos a escalabilidade da análise estática não apenas em termos uso eficiente dos recursos de computação, mas também em termos de uso eficiente de recursos humanos, como tempo do desenvolvedor.

CodeChecker é uma ferramenta projetada para facilitar a integração de o Analyzer e outras ferramentas de análise estática semelhantes construir sistemas e loops de integração contínua. É também um sistema completo de gerenciamento de bugs que monitora erros encontrados por essas ferramentas.

Dado um orçamento finito de tempo do desenvolvedor e milhares de relatórios em software grande, é importante avaliar relatórios com o melhor retorno do investimento em primeiro lugar.

O CodeChecker também suporta análises diferenciais que impede que os desenvolvedores introduzam novos bugs sem exigindo que eles corrijam todos os relatórios herdados com antecedência.

SUMÁRIO

Neste artigo, resumimos nossas experiências coletadas enquanto contribui para o estado da arte de Clang Static Analisador e CodeChecker. Nossa esperança é que provará ser um recurso útil para quem decide para trabalhar em ferramentas de análise estática.

Referências

[1] Baldoni, R., Coppa, E., Delia, D.C., Demetrescu, C. and Finocchi, I., 2018. A survey of symbolic execution techniques. ACM Computing Surveys (CSUR), 51(3), p.50.

[2] Clang Static Analyzer, https://clang-analyzer.llvm.org/. Last accessed 4 Nov 2018

[3] CodeChecker, https://github.com/Ericsson/codechecker. Last accessed 4 Nov 2018

[4] King, J.C., 1976. Symbolic execution and program testing. Communications of the ACM, 19(7), pp.385-394.

[5] Kovacs, R., Horvath, G., 2018. An Initial Prototype of Tiered Constraint Solving in the Clang Static Analyzer. Studia Universitatis Babes-Bolyai: Series Informatica, 63:(2) pp. 88-101.

[6] Xu, Z., Kremenek, T. and Zhang, J., 2010, October. A memory model for static analysis of C programs. In International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (pp. 535-548). Springer, Berlin, Heidelberg.

PROGRAMAÇÃO NA GESTÃO E ORQUESTRAÇÃO DE REDE VIRTUALIZADA DE RECURSOS

O Network Functions Virtualization (NFV) é um novo paradigma para mudar a maneira como as redes são construídas e operadas. Desacoplar software implementação a partir de recursos de rede através de uma camada de virtualização introduz uma necessidade de desenvolver conjuntos de gerenciamento e funções de orquestração (MANO). Focamos na coordenação da gestão funções implementadas em diferentes blocos funcionais para realizar operação confiável para as funções MANO operando em ambientes. Os desafios são ilustrados em um exemplo prático de Tecnologia virtual Open Stack e sobre os problemas inspirados pela indústria de telecomunicações.

INTRODUÇÃO

The purpose of these lecture notes is to introduce the new concepts, such as Network Functions Virtualization (NFV), that are currently implemented within complex software systems and networks, explain the new challenges arising and show students how to deal with them using the programming techniques for coordination of management and orchestration functions of virtualized network resources operating in distributed environments.

SISTEMAS COMPLEXOS

O cenário dessas palestras está dentro da teoria de sistemas complexos, em particular, os complexos sistemas de software e complexos redes. Assim, as palestras começam com uma introdução suave ao teoria, posicionando cuidadosamente os problemas e desafios considerados dentro da evolução atual de redes e software sistemas. A virtualização é um paradigma frequentemente usado no gerenciamento de sistemas de software complexos. Implica a introdução de um novo resumo layer, uma edição virtual da camada do sistema e suas funções, que evita a introdução de dependência entre as camadas do sistema.

FUNÇÕES DE GESTÃO E ORQUESTRAÇÃO

Nas redes de telecomunicações, um novo paradigma é introduzido, chamado Virtualização de Funções de Rede (NFV), que desacopla a rede função de recursos físicos de rede através de uma nova virtualização camada [2]. No entanto, isso introduz a necessidade de desenvolver conjuntos de NFV funções de gerenciamento e orquestração (MANO). Para esse fim, um grupo de trabalho especial é definido dentro da União Européia Instituto de Normas de Telecomunicações (ETSI). A função de rede A estrutura arquitetônica de gerenciamento e orquestração de virtualização é definido em [1]. Nestas notas de aula, focalizamos o gerenciamento e funções de orquestração implementadas em diferentes blocos funcionais, para realizar operação confiável para gerenciamento e funções de orquestração operando em ambientes distribuídos.

EXEMPLOS

Essas notas fornecem uma introdução ao assunto, com o objetivo de explicando os problemas e os princípios, métodos e técnicas usado para sua solução. Os exemplos e exercícios trabalhados servem como material didático, a partir do qual eles podem aprender a usar programação funcional para coordenar eficaz e eficientemente funções de gerenciamento e orquestração em sistemas complexos distribuídos usando NFV.

Os métodos e técnicas explicados nestas notas de aula e aplicada aos problemas de gestão e orquestração de referências de virtualização de rede, já existem e não reivindicamos originalidade nesse sentido. O objetivo destas notas é servir como um material didático para esses métodos.

Os problemas e desafios da coordenação da administração e funções de orquestração são tratadas usando a plataforma OpenStack [3] É um código aberto sistema operativo em nuvem que integra uma coleção de software módulos necessários para fornecer a computação em nuvem em camadas modelo. Essa tecnologia é necessária para lidar com os problemas que surgem do paradigma da virtualização nas redes atuais e os alunos soluções de entendimento no OpenStack poderão transferir seus conhecimento de outras tecnologias existentes com o mesmo ou similar objetivo. Os desafios decorrentes dos novos paradigmas de rede, como bem como suas soluções, são ilustradas através de exemplos práticos usando a tecnologia virtual OpenStack e inspirada nos problemas de o setor de telecomunicações. Todos os exemplos e exercícios são trabalhados na tecnologia virtual OpenStack.

Referências

[1] ETSI Industry Specification Group (ISG) NFV: ETSI GS NFV- MAN 001 v1.1.1: Network Functions Virtualisation (NFV); Management and Orchestration European Telecommunications Standards Institute (ETSI), 2014, https://www.etsi.org/deliver/etsi_gs/NFV- MAN/001_099/001/01.01.01 60/gs_NFV-MAN001v010101p.pdf, accessed July 1, 2018

[2] Han, B., Gopalakrishnan, V., Ji, L., Lee, S.: Network function virtualization: Challenges and opportunities for innovations. IEEE Communications Magazine 53(2), 90–97 (2015)

[3] OpenStack Cloud Software. OpenStack Foundation (2018), www.openstack.org, accessed July 1, 2018

COMPUTAÇÃO NA NUVEM E PROGRAMAÇÃO FUNCIONAL NA EDUCAÇÃO

A computação na nuvem se tornou uma tecnologia essencial e, portanto, parte de muitos currículos de ciências da computação. No design de aplicativos, os microsserviços são um conceito chave. A decomposição funcional intrínseca aos microsserviços pode ser bem servido por plataformas sem servidor, como o AWS Lambda.

MICROSERVIÇOS

Mais e mais profissionais recomendam a adoção de um microsserviço arquitetura ao projetar aplicativos em nuvem [1,2]. Vagamente definido, o estilo arquitetural do microsserviço representa um conjunto comum de características: implantação automatizada, endpoints inteligentes, tubos burros, controle descentralizado de dados [2]. Esforços recentes para migrar aplicativos em nuvem tradicionais para uma arquitetura de microsserviço alerta sobre a complexidade aumentada ao gerenciar os muitos, embora pequenos, serviços de composição [3]. Aqui, é importante desambiguar: vamos foco na orquestração como a composição da lógica de negócios de serviços de nuvem refinados. Se a implantação deles é orquestrada (componente central que rege a execução) ou coreografado (cada serviço age de forma independente) se resume a saber se o aplicativo precisa de controle síncrono ou pode tolerar assíncrono ao controle. No entanto, dado que agilidade e flexibilidade são altamente Para os atributos desejados, é preferível uma implantação coreografada [1].

No entanto, aspectos não funcionais dos microsserviços, como tempo de execução desempenho, desempenham um papel importante na implantação do aplicativo. Nós portanto, planeje familiarizar os alunos com o conceito de desempenho criação de perfil na nuvem no contexto do AWS Lambda. Além disso, nós complementam o paradigma funcional do AWS Lambda usando um Haskell estrutura para controlar os experimentos.

COMPUTAÇÃO NA NUVEM CENTRADA NO USUÁRIO

Na área da computação na nuvem, um aspecto importante é auxiliando os usuários em suas decisões. Tais decisões falam com as seguintes questões:

A. Como o aplicativo se comporta num sistema virtualizado de Recursos?

B. Quantos recursos virtuais de que tipo de qual provedor de nuvem deve ser adquirido para aplicação desdobramento, desenvolvimento?

C. por quanto tempo? Quanto vai custar?

Essas perguntas são tipicamente modeladas como um problema de agendamento, funcionando sob a suposição de que não há conhecimento a priori sobre o inscrição. Um conjunto simples de requisitos é que o aplicativo seja implantado com sucesso e os custos são minimizados.

A arquitetura de um planejador para um aplicativo em nuvem

O planejador BaTS [4] foi desenvolvido para auxiliar os usuários ao implantar seus aplicativos na nuvem. É preciso um abordagem de auto-agendamento para conseguir isso e regularmente verifica o progresso da implantação. Numa primeira fase, o BaTS coleta estatísticas de amostragem com substituição. Aqui, é necessária apenas uma pequena amostra (30 a 50 tarefas) para calcular a média e desvio padrão do tempo de execução das tarefas em várias ofertas na nuvem. O módulo de estimativa de orçamento em seguida, executa regressão linear para otimizar esta fase.

Usando a metodologia BaTS em virtualização leve

Apresentamos aos alunos o problema de ajudar os proprietários de aplicativos procurando selecionar as melhores opções em termos de peso leve virtualização ao implantar seu aplicativo como um conjunto de microsserviços.

AWS Lambda

O AWS Lambda é um recurso de computação virtualizado altamente leve oferecido pela Amazon. A granularidade é o nível da função e o O tempo de execução recomendado por chamada de função é no máximo na ordem de segundos. Ele também assume um comportamento não bloqueador. Existem 46 tipos da AWS Lambda com um modelo de precificação de euro por GB * s.

AWS API Haskell implementation

Com base em uma implementação abrangente da Haskell do API da AWS [5], pretendemos replicar a estimativa do BaTS metodologia em recursos virtualizados leves.

Trabalho prático

Instruímos os alunos a relacionarem o desempenho da execução do fatorial em vários tipos de lambda usando amostragem e linear regressão para traçar a taxa de transferência versus as curvas de eficiência de preços. Para Por exemplo, eles devem considerar qual tipo tem a melhor taxa de transferência para o preço mais barato. Em seguida, eles devem identificar como encontre a combinação mais lucrativa: o menor custo para o melhor desempenho

Benchmarking AWS Lambda

As funções do AWS Lambda são executadas em um ambiente virtualizado tipo container meio Ambiente.

A quantidade de recursos de computação alocados para o sabe-se que as funções são proporcionais à DRAM solicitada pelo usuário memória. Para determinar como as funções do Lambda podem processar vários cargas de trabalho, podemos comparar cada um de seus recursos computacionais independentemente: CPU, largura de banda de memória, largura de banda de I/O e rede largura de banda e latência. Tais marcas de micropigmentação podem ser realizadas por lançar, dentro das funções, computadores, memória, I/O, e cargas de trabalho intensivas em rede, como a computação do primeiro N prime números, executando o benchmark de fluxo, executando o iozone benchmark, ou lendo ou gravando dados no AWS S3.

A transferência de metodologias estabelecidas é essencial para a educação. Como futuro trabalho, gostaríamos de oferecer suporte às funções do Haskell AWS Lambda implantadas por meio da implementação da API da Haskell AWS.

Referências

[1] Newman, Sam. Building Microservices. O'Reilly Media, Inc., 2015.

[2] Fowler, M., Lewis, J.: Microservices. http://martinfowler.com/articles/microservices.html (March 2014), Last accessed: 15-08-2018

[3] Balalaie, Armin, Abbas Heydarnoori, and Pooyan Jamshidi. "Migrating to cloud- native architectures using microservices: An experience report." arXiv preprint arXiv: 1507.08217 (2015).

[4] AM Oprescu. Stochastic Approaches to Self-Adaptive Application Execution on Clouds. PhD Thesis, Amsterdam, Vrije Universiteit, 2013.

[5] https://hackage.haskell.org/package/amazonka-lambda-1.5.0, Last accessed: 15-08-2018.

TYPE-SAFE EMBEDDING DE GRAMÁTICAS DE ATRIBUTOS

Gramáticas de atributos são um formalismo declarativo poderoso para implementar e razão sobre os programas que, por design, são convenientemente modular. Embora um compilador de gramática de atributo completo possa ser adaptado para necessidades específicas, sua implementação é altamente não trivial e seus manutenção a longo prazo é um grande empreendimento. De fato, manter um sistema tradicional de gramática de atributos é um esforço tão grande que a maioria sistemas que foram propostos no passado não estão mais ativos. Nosso A abordagem para implementar gramáticas de atributos é escrevê-las como cidadãos de primeira classe de uma linguagem de programação funcional moderna. Nós melhorar a incorporação gramatical de atributos baseada em zíper anterior, tornando-o não invasivo (ou seja, nenhuma alteração nos tipos de dados definidos pelo usuário é obrigatório) e com segurança de tipo. Além disso, alcançamos uma sintaxe mais clara ao usando extensões Haskell modernas. Acreditamos que nossa incorporação pode ser empregado na prática para implementar módulos elegantes, eficientes e modulares soluções para os desafios da programação da vida real.

INTRODUÇAO

Gramáticas de Atributos (AGs) são um formalismo declarativo que foi proposto por Knuth [7] no final dos anos 60 e permite implementar e raciocinar sobre os programas de maneira modular e conveniente. Um AG concreto depende de uma gramática livre de contexto para definir a sintaxe de um idioma, e nos atributos associados às produções da gramática para defina a semântica dessa linguagem.

AGs foram utilizados na prática para especificar linguagens de programação reais, como por exemplo Haskell [2], bem como poderosos algoritmos de impressão bonita [16], desmatamento técnicas [4] e sistemas de tipos poderosos [11]. Ao programar com AGs, a modularidade é alcançada devido à possibilidade de definir e usando diferentes aspectos de cálculos como atributos separados.

Os atributos são unidades de computação distintas, geralmente bastante simples e modular, que pode ser combinado em soluções elaboradas para complexos problemas de programação. Eles também podem ser analisados, depurados e independente, o que facilita o desenvolvimento do programa e evolução.

As AGs provaram ser particularmente úteis para especificar cálculos em árvores: dada uma árvore, vários sistemas AG, como [14,3,8,17] levam especificações de quais valores ou atributos precisam ser computados e realize esses cálculos. Os esforços de design e codificação colocados no criação, aprimoramento e manutenção desses sistemas de AG, no entanto, é tremendo, muitas vezes um obstáculo para alcançar o sucesso que eles merecer.

Uma abordagem alternativa cada vez mais popular ao uso de AGs se baseiam em incorporá-los como cidadãos de primeira classe de linguagens de programação de uso geral [12, 9, 13, 15, 18, 1]. Isso evita o ônus de implementar um sistema totalmente novo idioma e sistema associado, hospedando-o em linguagens de programação de última geração. Seguindo isto abordagem um explora as construções modernas e infraestrutura que já é fornecida por esses idiomas e foca nas particularidades do domínio específico linguagem em desenvolvimento.

O zipper funcional [6] é uma poderosa abstração que muito simplifica a implementação de algoritmos transversais executando muitas atualizações locais. Os zipperes funcionais têm aplicado com sucesso para construir uma incorporação AG em Haskell [9,10]. Apesar de sua elegância, esta solução teve um grande desvantagem que impedia seu uso no mundo real aplicações: os atributos não foram armazenados em cache, mas sim recalculado repetidamente, o que prejudicou gravemente o desempenho. Recentemente, essa falha foi eliminada [5] e substituída com outro: a abordagem se tornou intrusiva, ou seja, para se beneficiar da incorporação de dados definidos pelo usuário estruturas precisam ser ajustadas

Neste artigo, apresentamos um mecanismo alternativo para armazenar em cache atributos baseados em uma grade infinita auto-organizada. este gráfico é colocado sobre o tipo de dados algébrico definido pelo usuário (ADT) e espelha sua estrutura. O tipo de dados definido usado em si permanece intocado. A incorporação é então baseada em dois (em vez de um) zíperes coerentes atravessando os dados estruturas em paralelo. Além de não ser intrusivo, nossa solução é completamente segura para o tipo. Modern Haskell extensões como ConstraintKinds nos permitem propagar restrições no ADT, eliminando completamente o tipo de tempo de execução lança nas versões anteriores.

Outro benefício colateral do uso de recursos modernos de Haskell é a sintaxe mais limpa, com menos código sendo gerado por meio de Modelo Haskell.

Referências

[1] Balestrieri, F.: The Productivity of Polymorphic Stream Equations and The Com- position of Circular Traversals. Ph.D. thesis, University of Nottingham (2015)

[2] Dijkstra, A., Fokker, J., Swierstra, S.D.: The architecture of the Utrecht Haskell compiler. In: Haskell Symposium. pp. 93–104 (2009)

[3] Dijkstra, A., Swierstra, D.: Typing Haskell with an Attribute Grammar (Part I). Tech. Rep. UU-CS-2004-037, Institute of Information and Computing Sciences, Utrecht University (2004)

[4] Fernandes, J.P., Saraiva, J.: Tools and Libraries to Model and Manipulate Circular Programs. In: Symposium on Partial Evaluation and Program Manipulation. pp. 102–111. ACM (2007)

[5] Fernandes, J.P., Martins, P., Pardo, A., Saraiva, J., Viera, M.: Memoized zipper- based attribute grammars and their higher order extension. Science of Computer Programming (2018)

[6] Huet, G.: The zipper. Journal of functional programming 7(5), 549–554 (1997)

[7] Knuth, D.: Semantics of Context-free Languages. Mathematical Systems Theory 2(2) (June 1968), Correction: Mathematical Systems Theory 5 (1), March 1971.

[8] Kuiper, M., Saraiva, J.: Lrc - A Generator for Incremental Language-Oriented Tools. In: International Conference on Compiler Construction. pp. 298–301. Springer-Verlag (1998)

[9] Martins, P., Fernandes, J.P., Saraiva, J.: Zipper-based attribute grammars and their extensions. In: Programming Languages - 17th Brazilian Symposium, SBLP 2013, Brasılia, Brazil, October 3 - 4, 2013. Proceedings. pp. 135–149 (2013)

[10] Martins, P., Fernandes, J.P., Saraiva, J., Wyk, E.V., Sloane, A.: Embedding at- tribute grammars and their extensions using functional zippers. Science of Computer Programming 132, 2 – 28 (2016), selected and extended papers from SBLP 2013

[11] Middelkoop, A., Dijkstra, A., Swierstra, S.D.: Iterative type inference with attribute grammars. In: International Conference on Generative Programming. pp. 43–52. ACM (2010)

[12] de Moor, O., Backhouse, K., Swierstra, D.: First-Class Attribute Grammars. In: 3rd. Workshop on Attribute Grammars and their Applications. pp. 1–20. Ponte de Lima, Portugal (2000)

[13] Norell, U., Gerdes, A.: Attribute Grammars in Erlang. In: Workshop on Erlang. pp. 1–12. 2015, ACM (2015)

[14] Reps, T., Teitelbaum, T.: The synthesizer generator. SIGPLAN Not. 19(5), 42–48 (Apr 1984)

[15] Sloane, A.M., Kats, L.C.L., Visser, E.: A pure object-oriented embedding of attribute grammars. Electronic Notes in Theoretical Computer Science 253(7), 205–219 (2010)

[16] Swierstra,D.,Azero,P.,Saraiva,J.:Designing and Implementing Combinator Languages. In: Third Summer School on Advanced Functional Programming. LNCS Tutorial, vol. 1608, pp. 150–206. Springer Verlag (1999)

[17] Van Wyk, E., Bodin, D., Gao, J., Krishnan, L.: Silver: an Extensible Attribute Grammar System. Electronic Notes in Theoretical Computer Science 203(2), 103–116 (2008)

[18] Viera, M., Swierstra, D., Swierstra, W.: Attribute Grammars Fly First-class: how to do Aspect Oriented Programming in Haskell. In: International Conference on Functional Programming. pp. 245–256. ACM (2009)

QUÃO VERDE É O SEU PROCESSO?

Como nos produtos biológicos, o mundo está se desenvolvendo para se tornar um ecossistema consciente da natureza. A iniciativa verde define dois objetivos principais: reduzir o consumo de energia e usar fontes naturais básicas em produção de energia.

Um dos desafios para os fabricantes de baterias é quanto tempo o a bateria pode funcionar sem ser continuamente carregada. Há muitos outros desafios também, como o tamanho que afeta bastante o forma e peso do dispositivo. A bateria é considerada um pouco mais leve em comparação com o dispositivo que precisa de uma bateria para operar. O desafio aqui é como diminuir o tamanho e peso mais leve e certamente uma alta eficiência em termos de operação hora do dispositivo móvel sem ser cobrado.

No topo desse desafio de hardware, existe um desafio de software: o próprio software deve suportar economia de energia. Fazendo isso sem limitar a experiência do usuário é considerado atualmente como um objetivo silencioso, mas importante, de cada desenvolvimento de software que seja visando qualquer tipo de dispositivo portátil [1]. Esse objetivo geralmente surge quando os requisitos relacionados mudarem de incalculável para uma questão importante.

Lembrando que o consumo de energia de qualquer dispositivo móvel é influenciado pelos aplicativos em execução através da frequência de uso de serviços até o humor real do usuário, para desenvolver software para esses dispositivos já são um desafio [4]. Pode-se dizer, o software desafio de desenvolvimento é sempre o mesmo, mas temos que ressaltar “Mobilidade” como propriedade chave do sistema aqui. O status da bateria também determina o desempenho do sistema devido ao nível do sistema operacional configuração - conhecida como “preferências de economia de energia”.

É ainda mais difícil se um (no nosso caso, o professor) tiver que se preparar estudantes para tais desafios [6].

Todos os já conhecidos “melhores práticas "e" dicas de economia de energia "devem ser apresentadas em um contexto, o que é facilmente compreensível para os alunos. Isso pode ser feito posicionando os conceitos em um ambiente conhecido como teste de software e automação de teste [2]. É isso que pretendemos com este tutorial, este é o conteúdo das próximas seções, começando com a proposta, seguido de exemplos e fechando com mais dicas sobre melhoria. O foco principal está no consumo de energia do software em funcionamento e processos de desenvolvimento, onde cada fase de desenvolvimento desempenha papel importante. Considerando qualquer processo de desenvolvimento de software, a energia está sendo consumidos durante a análise do problema, construindo e avaliando o código também [3]. Ferramentas de software ou hardware devem ser usadas para implementar monitoramento do consumo de energia do software executado na parte superior sistemas operacionais e para avaliação do consumo de energia. Usual cenários de uso são monitorar o uso de energia do software selecionado [5], mas também daremos uma olhada na possibilidade de usar essas ferramentas para medir quão verde é o processo que produz a (s) versão (ões) final (is) software de trabalho. Os exemplos abrangem uma variedade de situações. Começando com o caso de perfil de energia de software de trabalho de terceiros em uso específico cenários, destacamos as principais propriedades (vantagens e desvantagens) de ferramentas existentes. Durante o teste unitário do código do software que está sendo desenvolvido, apresentamos o uso típico de software de perfil de energia.

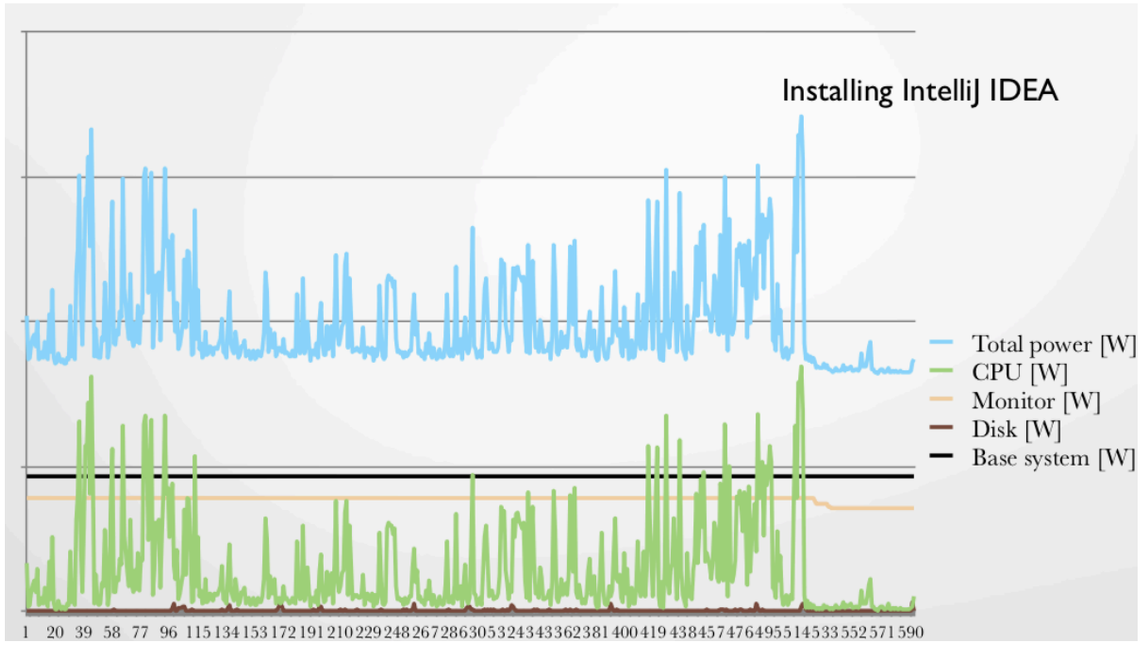
A capacidade de escalar a abordagem de medição a partir do perfil de um código trecho ou aplicação única à análise de consumo de energia de cadeias de ferramentas é o último exemplo que apresentamos. Este está apresentando um abordagem genérica para criação de perfil de energia e visa substituir o ponto de interrogação do título do tutorial por um período que representa o avaliação de resultados para cada caso específico coberto pelo tutorial.

Referências

[1] D. Li, W. G. J. Halfond, An investigation into energy-saving programming practices for android smartphone app development, in Proc. of the 3rd International Workshop on Green and Sustainable Software, GREENS 2014, ACM, 2014, pp. 46–53.

The energy-measured development game

1. Setup the environment
2. Start the energy monitor
3. Develop (think, code, test, fix) for 15 minutes
4. Have a 5 minutes break (stop energy usage monitoring, set up the next one, get a coffee)
5. Finish (for this time) if there is no further idea
6. Repeat (jump to label 2)
7. Analyse collected data (energy efficiency of your development process) inside the team



[2] D. Li, Y. Jin, C. Sahin, J. Clause, W. G. J. Halfond: Integrated energy-directed test suite optimization, in Proc. of the 2014 International Symposium on Software Testing and Analysis, ISSTA 2014, ACM, 2014, pp. 339–350.

[3] M. Santos, J. Saraiva, Z. Porkolab, D. Krupp: Energy Consumption Measurement of C/C++ Programs Using Clang Tooling, in Proceedings of the SQAMIA 2017: 6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Paper No. 15, 8 pages, CEUR Workshop Proceedings No. 1938 ISSN 1613-0073.

[4] J.Saraiva,M.Couto,Cs.Szabo,D.Novak:TowardsEnergy-AwareCodingPractices for Android, Acta Electrotechnica et Informatica, Vol. 18, No. 1, 2018, pp. 19–25. https://doi.org/10.15546/aeei-2018-0003

[5] Cs. Szabo, E.M.M. Alzeyani: Measuring Energy Efficiency of Selected Working Software, Studia Universitatis Babe¸s-Bolyai Informatica, Vol. 63, No. 1, 2018, pp. 5–16. https://doi.org/10.24193/subbi.2018.1.01

[6] Cs. Szabo, J. Saraiva: Focusing software engineering education on green application development, in Conference of Information Technology and Development of Education – ITRO 2017, Novi Sad, Serbia, pp. 165–169, ISBN 978-86-7672-302-7.

PROGRAMAÇÃO FUNCIONAL PARA DISPOSITIVOS SIMPLES

NOTA

Este artigo será uma versão ampliada da nossa contribuição RWDSL18 [2] No artigo atual, usaremos o mesmo DSL, apenas alguns pequenos extensões e melhorias são feitas. O artigo atual se concentrará em como o DSL mTask pode ser usado para programar a IoT e discutirá uma simulador de alto nível para programas mTask como um programa iTask. para simular.

INTRODUÇÃO

Atualmente, muitos dispositivos estão equipados com um microprocessador simples para controlar seu comportamento. Exemplos típicos são termostatos, luz lâmpadas, tomadas elétricas, alarmes de incêndio, abridores de portas e assim por diante. Quando esses dispositivos podem se comunicar uns com os outros ou com alguns Dizem que eles fazem parte da Internet das Coisas, IoT. o microcomputadores nesses dispositivos são muito acessíveis e estão se tornando onipresente. Dispositivos caros, como carros e aparelhos com muito tarefas complexas são equipadas com um computador incorporado completo e software apropriado. Para a maioria dos dispositivos IoT pequenos e relativamente baratos esse computador incorporado é muito caro ou consome muito energia; um microprocessador simples e barato é usado para executar o Programas. Esses sistemas têm poder e memória de computação muito limitados, normalmente 30 KB a 4 MB de memória flash para armazenar o programa. A vida dessa memória é restrita a 1000 ciclos de gravação. Para armazenar variáveis, a pilha e a pilha dos sistemas têm 2 a 40 KB de RAM.

As limitações de velocidade e memória do processador excluem o uso de um sistema operacional. O aparelho apenas executa o programa controlando o dispositivo. Mesmo os programas de controle nesses dispositivos de IoT consistem em várias tarefas. Por exemplo, para verificar o estado de um botão dez vezes um segundo, para atualizar uma exibição a cada segundo, para medir a temperatura duas vezes por minuto e para alternar o aquecimento após pelo menos cinco minutos, a menos que o botão seja pressionado anteriormente. Devido às diferentes prazos e as dependências dessas tarefas, o programa de controle tende a se tornar um pouco confuso, independente da linguagem de programação usava. Além disso, os dispositivos IoT executam programas separados para o resto do aplicativo na IoT e da comunicação usando uma infinidade de protocolos. Isso torna o desenvolvimento e a manutenção da IoT aplicativos complexos e propensos a erros.

A Programação Orientada a Tarefas, TOP, oferece threads leves que podem facilmente ser composto por tarefas mais complexas. As tarefas são avaliadas passo a passo e pode inspecionar o valor atual de outras tarefas após tal passo. O TOP é implementado pela primeira vez no sistema iTask [4,5] incorporado no Clean [6]. No sistema iTask, tarefas primitivas são coletando informações via formulário da web gerado automaticamente ou coletando dados de outros programas e repositórios de dados. Um poderoso conjunto de combinadores é usado para compor tarefas para tarefas mais complexas. Nisso Neste artigo, mostramos que o TOP é muito adequado para a programação da IoT dispositivos. As tarefas primitivas entregam o valor atual das entradas e sensores. Construtores muito semelhantes ao sistema iTask são usados para combine tarefas a tarefas mais complexas.

Os dispositivos de IoT normalmente têm tarefas pouco dependentes que controlar os sensores, atuadores e comunicação do dispositivos. Programar isso em um estilo TOP oferece concisão programas. A execução dessas tarefas dentro das restrições de microcontroladores pequenos com poder de processamento muito limitado e alguns KBs de memória RAM merecem uma reflexão. Devido às severas limitações dos microcontroladores utilizados não podemos portar o sistema iTask para os dispositivos IoT, pois um Um programa típico do iTask requer cerca de 100 MB de heap espaço. Definimos um domínio específico incorporado Idioma, eDSL, chamado mTask para os dispositivos IoT. este O eDSL está incorporado no sistema iTask, pois planejamos torne esses

idiomas TOP totalmente interoperáveis. As contribuições deste artigo serão:

- Este artigo apresenta uma linguagem de programação funcional baseada em tarefas para dispositivos IoT. Comparado com o nosso idioma anterior para programação por microprocessador Referências [3] do periférico imperativo o controle é substituído por construções transparentes referenciais.

- Demonstramos como criar um extensível funcional, várias visualizações, DSL incorporado seguro para o tipo. Este é um eDSL sem etiqueta [1].

- O código gerado é executado em dispositivos pequenos e lentos, bem como em máquinas maiores e uma máquina simulada.

- Devido ao uso do Arduino C++ como linguagem intermediária, este O eDSL funcional roda em muitos microcontroladores diferentes.

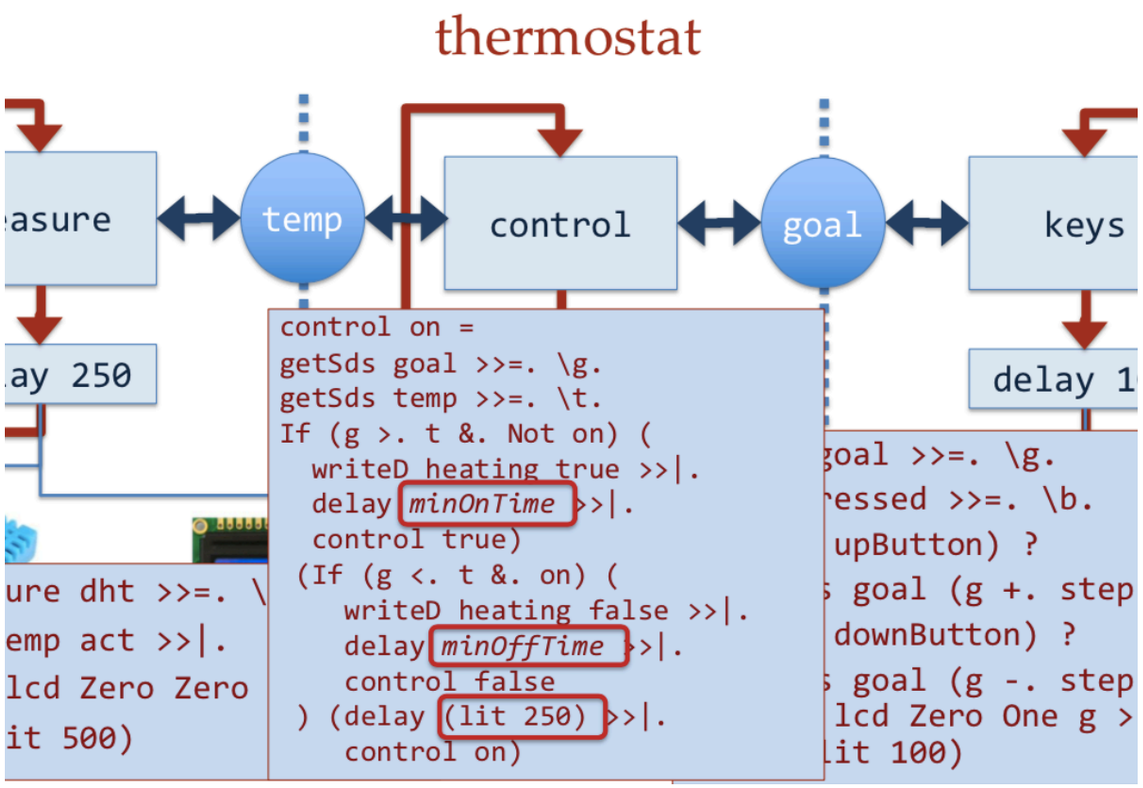
- A simulação de alto nível dos programas mTask em um programa iTask oferece a possibilidade de visualizar o efeito do programa eDSL e de manipular o ambiente simulado para experimentar com o especificado comportamento. Nesse simulador, é muito mais fácil manipular o tempo e sensores do que em uma configuração da vida real, por exemplo, podemos mudar a temperatura relatada por um sensor pressionando um botão expor fisicamente o dispositivo IoT a essas temperaturas.

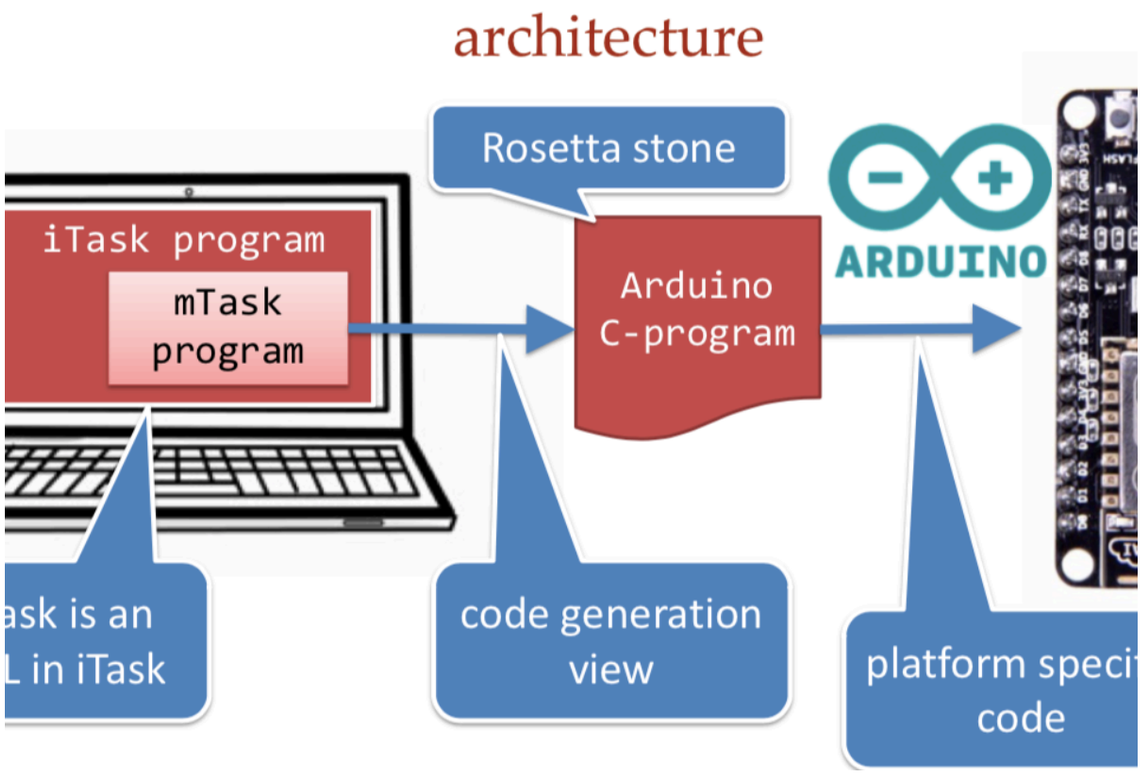
Referâncias

[1] Carette, J., Kiselyov, O., Shan, C.c.: Finally tagless, partially evaluated: Tagless staged interpreters for simpler typed languages. J. Funct. Program. 19(5) (Sep 2009). https://doi.org/10.1017/S0956796809007205

[2] Koopman, P., Lubbers, M., Plasmeijer, R.: A task-based dsl for microcomputers. In: Proceedings of the Real World Domain Specific Languages Workshop 2018. pp. 4:1–4:11. RWDSL2018, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3183895.3183902, http://doi.acm.org/10.1145/ 3183895.3183902

[3] Koopman, P., Plasmeijer, R.: A shallow embedded type safe extendable dsl for the arduino. In: Revised Selected Papers of the 16th International Symposium on Trends in Functional Programming - Volume 9547. pp. 104–123. TFP 2015, Springer-Verlag New York, Inc., New York, NY, USA (2016), http://dx.doi.org/ 10.1007/978-3-319-39110-6_6





[4] Plasmeijer, R., Achten, P., Koopman, P.: iTasks: executable specifications of inter- active work flow systems for the web. In: Hinze, R., Ramsey, N. (eds.) Proceedings of the ICFP'07. ACM, Freiburg, Germany (2007)

[5] Plasmeijer, R., Lijnse, B., Michels, S., Achten, P., Koopman, P.: Task-oriented programming in a pure functional language. In: Proceedings of the 14th Symposium on Principles and Practice of Declarative Programming. pp. 195–206. PPDP '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2370776.2370801, http://doi.acm.org/10.1145/2370776.2370801

[6] Plasmeijer, R., van Eekelen, M., van Groningen, J.: Clean language report (version 2.2) (2011), http://clean.cs.ru.nl/Documentation

COMPREENSÃO DE CÓDIGO COM CODECOMPASS

VISÃO GLOBAL

Desenvolvimento e manutenção são dois estágios separados, com diferentes portanto, requerem suporte de ferramentas diferente, como bem. Durante o desenvolvimento, estamos escrevendo principalmente um novo código que requer suporte a ferramentas, como conclusão de código, correspondência de chaves, etc. e geralmente apenas alguns arquivos envolveram mais ou menos no mesmo nível de abstração.

Durante a manutenção, estamos principalmente lendo e navegando por um base de código existente entre o grande número de módulos e arquivos em diferentes níveis de abstração [1]. No desenvolvimento, as intenções são claro, em oposição à compreensão do código, onde a tarefa é recuperar o objetivo original de certos fragmentos de código.

No ambiente industrial [2], um projeto pode consistir em milhões linhas de código. Para grandes sistemas existentes há muito tempo, em que código base foi desenvolvido e mantido para décadas por equipes flutuantes, as intenções originais são perdidas, a documentação não é confiável ou está faltando, a única informação confiável é o próprio código. Compreensão de sistemas de software tão grandes é essencial, mas geralmente muito Tarefa desafiante. Isso implica que o suporte de ferramentas é necessário [3].

Ao se familiarizar com um código fonte desconhecido, o O primeiro passo é encontrar as partes relevantes do sistema. este processo requer localização rápida de recursos, com base em entidades nomeadas adquiridas a partir de mensagens de log ou outras Recursos. O próximo passo é ampliar nosso conhecimento sobre sistema através de diagramas, cadeias de chamadas de funções etc. no final, gostaríamos de verificar o conhecimento adquirido através de mensagens de controle de versão,

de arquitetura informações e referências sobre módulos relacionados.

CODECOMPASS

CodeCompass [4, 5] é um quadro de compreensão de código-fonte aberto trabalhos. Ele fornece uma arquitetura plugável para habilitar o adição de várias ferramentas de análise que produzem diferentes visualizações, coletores de informações, métricas [6], etc. O objetivo importante do projeto era dimensionar o CodeCompass para projetos industriais

Na primeira etapa, o produto deve ser analisado: todas as As informações são coletadas e armazenadas em um banco de dados que permite que a camada de serviço forneça as informações necessárias visualizações. Para uma pesquisa rápida, o CodeCompass usa texto indexação que resulta em navegação independente do idioma em o código fonte. Como o objetivo principal é fornecer informações precisas informações sobre elementos de linguagem, identificação de símbolos pelo nome não é suficiente. Usamos o compilador LLVM infra-estrutura para identificar símbolos com precisão e resolver entidades nomeadas usando a árvore de sintaxe abstrata. O CodeCompass é estendido por analisadores de idiomas. o As linguagens mais suportadas são C/C++, mas Java e Python também são parcialmente manipulados.

Além dos símbolos nomeados, algumas informações adicionais são também armazenados no banco de dados, como relações entre AST nós (chamadas de função, herança) e arquivos (interface relação de provedor, inclusão etc.). Estes são usados para exibindo uma imagem em nível de arquitetura sobre o sistema com base no uso dos símbolos [7].

A base de código não é a única fonte de documentação. O commit As mensagens de um sistema de controle de versão também contêm informações que são É importante entender por que certas mudanças ocorreram no momento módulo. O CodeCompass também lê o repositório Git, se houver. O CodeCompass também está equipado com funcionalidades avançadas. Pode exibir as funções geradas pelo compilador que estão faltando no fonte. A análise de ponteiro ajuda a entender quais variáveis são referindo o mesmo objeto. Podemos inspecionar as relações de chamada de função mesmo se aqueles invocados por meio de uma função virtual ou um ponteiro de função.

RESUMO

A compreensão do código requer suporte de ferramenta específica para compreensão de software em larga escala. Temos uma visão geral e categorizar as ferramentas de compreensão de código por arquitetura e funcionalidades para examinar suas capacidades.

Apresentamos o CodeCompass, que apresenta uma ampla variedade de funcionalidades sobre visualizações, informações fornecimento, controle de versão e coleta de documentação, métricas etc.

Referências

[1] Jonathan Sillito, Gail C. Murphy, Kris De Volder. (2008). Asking and Answering Questions during a Programming Change Task. IEEE Transactions on Software Engineering, VOL. 34, NO. 4, July/August 2008.

[2] Porkolab, Zoltan & Brunner, Tibor & Krupp, Daniel & Csordas, Marton. (2018). Codecompass: an open software comprehension framework for industrial usage. 361- 369. 10.1145/3196321.3197546.

[3] Nathan Hawes, Stuart Marshall, Craig Anslow. (2015). CodeSurveyor: Mapping LargeScale Software to Aid in Code Comprehension. 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT) , 27-28 Sept. 2015.

[4] Porkolab,Zoltan & Brunner,Tibor (2018). The codecompass comprehension framework. 393-396. 10.1145/3196321.3196352

[5] CodeCompass, https://github.com/Ericsson/CodeCompass. Last accessed 5 Nov 2018.

[6] Brunner, Tibor & Porkolab, Zoltan. (2017). Two Dimensional Visualization of Soft- ware Metrics. Proceedings of the Sixth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications.

[7] B. De Alwis and G.C. Murphy. (1998). Using Visual Momentum to Explain Dis- orientation in the Eclipse IDE. Proc. IEEE Symp. Visual Languages and Human Centric Computing, pp. 51-54, 2006.

PROGRAMAÇÃO FUNCIONAL PARA COMPUTAÇAO DE ALTO DESEMPENHO

Linguagens de programação funcional fornecem ferramentas e recursos para projetar e implementar distribuído inscrição. Como programas funcionais têm paralelo inerente recursos, ele pode ser explorado para obter informações simultâneas confiáveis processamento por distribuição e coordenação de alto nível.

O desenvolvimento de software paralelo de última geração uso extensivo de várias metodologias e abordagens para obter alta velocidade. No entanto, a simultaneidade permanece um dos domínios mais difíceis, especialmente no caso de abordagens de programação funcional.

O principal objetivo é explorar os esqueletos de cálculos paralelos em um novo ambiente, para ilustrar a adequação e aplicabilidade do FP em novas configurações de computação distribuída. Um conjunto de paralelos conhecidos esqueletos algorítmicos são testados como componentes HPC. Número significativo de exemplos fornecem alta velocidade. A quantidade de paralelismo sempre depende de muitos fatores, como: o padrão de computação aplicado, granularidade refinada, semântica de nós distribuídos, dados transmissão. Os exemplos inspecionam a coordenação bem modelada e a solidez semântica.

DOMÍNIOS DE APLICATIVO PARA SCHELETONS

Coordenação

O tópico específico do funcional distribuído e paralelo computação requer elementos da linguagem de coordenação. A pesquisa As questões abordadas anteriormente estavam preocupadas com o modo como comportamento e comunicação de programas funcionais podem ser obtidos em níveis mais altos. A linguagem de programação funcional introduzida elementos com maior nível de abstração provaram ser viáveis para cálculos paralelos, de alto nível e intensivos em dados [2].

As instruções seguidas resultaram em maior poder de expressão de elementos de linguagem para paralelismo em programas funcionais. Mais especificamente, conduziu ao design de uma extensão de idioma DClean para a programação distribuída e coordenação do programa Clean programas funcionais. A extensão consiste em linguagem de alto nível elementos coordenando nós computacionais funcionais puros no ambiente distribuído projetado em clusters.

As construções geram caixas de computação conectadas via buffer canais de comunicação. O uso de programadores DClean indica como o padrão de computação distribuído é organizado em um gráfico distribuído e controlam os fluxos de dados da rede de processo por canais digitados. A linguagem oferece a vantagem de escrever aplicativos distribuídos e funcionais sem estar familiarizado com detalhes do ambiente multicamada e serviços de middleware " Aspectos tecnicos. A distribuição do trabalho é feita de acordo com esquema computacional paralelo predefinido, scheleton algorítmico, parametrizado por funções, tipos e fluxos de entrada. O propósito principal introdução da linguagem de coordenação era definir funções funcionais de computação paralela. Em um grande número de exemplos, altos aceleração do paralelismo foi obtida. A quantidade real de paralelismo estava sujeito à ordem de criação do canal, a quantidade de trabalho neles, a velocidade de recuperação e armazenamento de dados e a complexidade dos nós [1].

A visualização gráfica da computação distribuída pelo O suporte da ferramenta de compreensão de código de semântica executável [4] foi indispensável em aplicações reais distribuídas. Isso representava o paralelismo esperado em caixas e canais gerados usando esqueletos de alto nível bem definidos. Visava modelar e formulação de propriedades da semântica operacional do DClean [3].

Sistemas físicos cibernéticos

A abordagem de modelagem funcional baseada em sceletons usado pelas línguas de coordenação é aplicado atualmente protótipos dos sistemas modernos de CPS também. Estudando relacionamentos entre CPS e sistemas distribuídos, ou CPS e sistemas instalados são importante para a tomada de decisões adequadas no projeto e modelagem etapas do protótipo dos sofisticados sistemas CPS.

Os estudos de caso do sistema CPS implementados [5] descrevem a colaboração unidades computacionais que controlam entidades físicas (sensores) e os relacionamentos com outros sistemas complexos. O sistema CPS inteligente estabelece novos aspectos, características e abordagens em geral prototipagem usando esqueletos. Nesse projeto de sistema do CPS, é importante questões semânticas são abordadas a partir de probabilística e comportamental pontos de vista, onde a interoperabilidade é a característica principal e específica para analisar e especificar.

Pesquisa sobre a extensão da aplicabilidade de esqueletos em ambiente computacional de alto desempenho é o ponto chave na abordagem FP paralela. Adaptação do know-how anterior sobre programação esquelética de múltiplos núcleos heterogêneos sistemas resulta em acelerações mais altas, onde o medições e comparações avaliam o novo processos de paralelização.

Os protótipos de scheletons são definidos em termos de funcionalidades e coordenações. Os estudos de caso ilustram as conexões com outro tipo de sistemas distribuídos, importantes devido à estrutura multicamada de eles. As propriedades do sistema distribuído fornecidas em formas executáveis são testadas por esqueletos de programação distribuída de clusters e grades.

Referências

[1] Zsok V.: D-Clean Semantics for Generating Distributed Computation Nodes, Work- shop on Generative Technologies, WGT 2010, Satellite workshop at ETAPS 2010, Paphos, Cyprus, March 27, 2010, pp. 77–84.

[2] Zsok V., Hernyak Z., and Horvath, Z.: Designing Distributed Computational Skeletons in D-Clean and D-Box. Central European Functional Programming School CEFP 2005, First Summer School, Budapest, Hungary, July 4-15, 2005, Revised Selected Lectures, LNCS Vol. 4164, Springer-Verlag, 2006, pp. 223–256.

[3] Zsok V., Koopman, P., Plasmeijer, R.: Generic Executable Semantics for D-Clean, Proceedings of the Third Workshop on Generative Technologies, WGT 2011, ETAPS 2011, Saarbrucken, Germany, March 27, 2011, ENTCS Vol. 279, Issue 3, Elsevier, December 2011, pp. 85–95.

[4] Zsok V., Porkolab Z.: Rapid Prototyping for Distributed D-Clean using C++ Tem- plates, Annales Universitatis Scientiarum Budapestinensis de Rolando Eotvos Nominatae, Sectio Computatorica, Eotvos Lorand University, Budapest, Hungary, 2012, Vol. 37, pp. 19–46.

[5] Zsok V. et al.: Modeling CPS Systems using Functional Programming, Proc. of IFL17, Uni. of Bristol, pp. 168–174.