UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARAMENTO DE APOIO Á PESQUISA
PROGRAMA INSTITUCIONAL DE INICIAÇÃO CIENTÍFICA

VERIFICAÇÃO DE PROGRAMAS C++ QUE USAM O FRAMEWORK MULTI-PLATAFORMA QT

Bolsista: Felipe Rodrigues Monteiro Sousa, CNPq

MANAUS – AM
2013

RELATÓRIO FINAL
PIB-E/0015/2012
VERIFICAÇÃO DE PROGRAMAS C++
QUE USAM O FRAMEWORK MULTI-PLATAFORMA QT

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARAMENTO DE APOIO Á PESQUISA
PROGRAMA INSTITUCIONAL DE INICIAÇÃO CIENTÍFICA

RELATÓRIO FINAL
PIB-E/0015/2012
VERIFICAÇÃO DE PROGRAMAS C++ QUE USAM O FRAMEWORK MULTI-PLATAFORMA QT

Bolsista: Felipe Rodrigues Monteiro Sousa, CNPq
Orientador: Prof. Dr. Lucas Carvalho Cordeiro

MANAUS – AM
2013

Todos os direitos deste relatório são reservados à Universidade Federal do Amazonas, ao Núcleo de Estudo e Pesquisa em Ciência da Informação e aos seus autores. Parte deste relatório só poderá ser reproduzida para fins acadêmicos ou científicos.

Esta pesquisa, financiada pelo Conselho Nacional de Pesquisa – CNPq, através do Programa Institucional de Bolsas de Iniciação Científica da Universidade Federal do Amazonas, foi desenvolvida pelo Núcleo de Estudo e Pesquisa em Ciência da Informação e se caracteriza como subprojeto do projeto de pesquisa Bibliotecas Digitais.

"Os cientistas estudam o mundo como ele é,
os engenheiros criam um mundo como ele nunca havia sido".

Theodore von Karman.

RESUMO

O avanço no desenvolvimento de sistemas embarcados vem ocorrendo de forma cada vez mais acelerada, e consequentemente, a complexidade de tais sistemas aumenta igualmente. Além disso, esses sistemas estão cada vez mais presentes em diversos setores do nosso dia-a-dia. Um bom exemplo de tais sistemas é encontrado nas tecnologias móveis, como os telefones celulares e *tablets*, que a cada dia apresentam um poder gráfico cada vez maior. Para alcançar tamanho poder e desempenho, os desenvolvedores utilizam de vários tipos de *frameworks* gráficos durante o processo de desenvolvimento. Essa ampla difusão unida com tamanha complexidade, estão fazendo com que as empresas invistam cada vez mais na verificação rápida e automática de seus produtos, visto que seria algo extremamente custoso lançar no mercado um produto defeituoso. Nesse cenário, surgem os verificadores de software, contudo, nem todos esses verificadores suportam a verificação dos softwares que utilizam os robustos *frameworks*, um bom exemplo disso é o *framework* multi-plataforma QT. Por esta razão, este trabalho tem como objetivo desenvolver um conjunto de bibliotecas simplificadas, similares ao *framework* QT, e integrá-las no verificador de software ESBMC (*Efficient SMT-Based Bounded Model Checking*) para que a partir dessas implementações simplificadas, o verificador consiga analisar aplicações reais que utilizam o *framework* em questão. Como ponto de partida no desenvolvimento deste conjunto de bibliotecas simplificadas, foi feita uma análise de aplicações reais que utilizam o *framework* QT, e então foram definidas quais eram as principais funcionalidades do mesmo que seriam trabalhadas e verificadas. Após isso, uma suíte de teste automatizada, contendo todos os códigos que englobassem as funcionalidades destacadas, foi criada e uma estrutura das bibliotecas desenvolvidas.

Palavras-chaves: Métodos Formais; Verificação de Software; *Framework* QT; C++;

LISTA DE ABREVIAÇÕES

ESBMC	*Efficient SMT-based Context-Bounded Model Checker*
SVCOMP	*Competition on Software Verification*
TACAS	*International Conference on Tools and Algorithms for the Construction and Analysis of Systems*
SMT	*Satisfiability Modulo Theories*
PIM	Pólo Industrial de Manaus
QF_AUFBV	*Closed quantifier-free formulas over the theory of bitvectors and bitvector arrays extended with free sort and function symbols.*
QF_AUFLIRA	*Closed quantifier-free formulas with free sort and function symbols over one- and two-dimentional arrays of integer index and real value.*
LTL	Lógica Temporal Linear
CTL	*Computer Tree Logic*
ASA	Árvore de Sintaxe Abstrata

LISTA DE FIGURAS

SUMÁRIO

INTRODUÇÃO

Nossa dependência no funcionamento correto de sistemas embarcados está aumentando rapidamente. A grande difusão de dispositivos móveis e da evolução do software e hardware que os compõe, é um bom exemplo da importância desses sistemas. Os mesmos estão se tornando cada vez mais complexos, e requerem processadores com vários núcleos de processamento usando memória compartilhada escalável, com o intuito de atender a crescente demanda do poder computacional. Desta maneira, a confiabilidade dos sistemas (distribuídos) embarcados é um assunto chave no processo de desenvolvimento de tais sistemas [7]. As empresas, cada vez mais, procuram formas mais rápidas e baratas para verificar a confiabilidade dos seus sistemas, evitando assim grandes prejuízos. Uma das formas mais eficazes e de mais baixo custo é a verificação de modelos [20]. Apesar da eficácia desse método de verificação, existem muitos sistemas que não podem ser verificados de forma automática, devido à indisponibilidade no mercado de verificadores que suportem determinadas linguagens e *frameworks* (isto é, conjuntos reutilizáveis de bibliotecas ou classes). Um bom exemplo disso são as aplicações que utilizam o *framework* multi-plataforma QT [1]. Desta forma, este projeto de pesquisa visa mapear as principais funcionalidades do *framework* QT, e a partir disto, desenvolver um modelo operacional capaz de verificar as propriedades relacionadas com tais funcionalidades que estão integradas em programas C++. Os algoritmos desenvolvidos neste projeto de pesquisa estão integrados no verificador de código ESBMC (*Efficient SMT-based Context-Bounded Model Checker*) [4, 6, 7, 12], o qual é reconhecido internacionalmente pela sua robustez e eficácia na verificação de programas ANSI-C/C++, ganhando assim destaque pela sua atuação nas duas primeiras edições da Competição Internacional em Verificação de Software (SVCOMP'12 e SVCOMP'13), realizada pela conferência TACAS [10].

REVISÃO BIBLIOGRÁFICA

Atualmente, a quantidade de software em produtos embarcados tem aumentado significantemente de tal modo que sua verificação desempenha um papel importantíssimo para assegurar a qualidade do produto. As empresas instaladas no Pólo Industrial de Manaus (PIM) têm focado, grande parte dos seus investimentos da lei de informática, no desenvolvimento de software para dispositivos móveis. Diversos *frameworks* de software têm sido utilizados para acelerar o desenvolvimento das aplicações. Dentro deste contexto, o *framework* QT representa um bom exemplo deste conjunto reutilizável de classes, onde o engenheiro de software pode utilizá-lo para acelerar o desenvolvimento de aplicações gráficas para dispositivos móveis.

É neste cenário que, a chamada verificação formal surge como uma técnica eficiente no que tange a validação desses tipos de sistemas, oferecendo assim garantia de corretude. Esta técnica tem por objetivo provar, matematicamente, a conformidade de um determinado algoritmo a cerca de uma determinada propriedade utilizando métodos formais. Ela se divide em dois tipos de abordagens: a verificação dedutiva e a verificação de modelos. No caso da verificação dedutiva, que funciona para sistemas de estado infinito, o sistema e as propriedades de corretude são descritos em conjuntos de fórmulas, de tal forma que através da utilização de axiomas e regras de prova, possa ser provada a corretude das mesmas, contudo a tecnologia de provador de teorema é difícil e não é completamente automática [23]. Por outro lado, no caso da verificação de modelos, do inglês *Model Checking*, que tipicamente funciona em modelos finitos, o sistema é representado por um determinado modelo, de maneira que, através da exploração exaustiva de todos os possíveis comportamentos, possam ser checadas todas as propriedades do mesmo de forma completamente automática [24]. Para tratar a segunda abordagem de forma algorítmica, é representado matematicamente o sistema e suas especificações utilizando lógica, como por

exemplo, lógica proposicional e lógica temporal linear, de tal forma que se possa verificar se uma dada fórmula é satisfeita dada uma determinada estrutura.

De forma a realizar este processo automaticamente, são utilizados softwares, conhecidos como verificadores, que são desenvolvidos baseados nas teorias de verificação de modelo, tendo como principal objetivo verificar de forma confiável, as propriedades de um determinado tipo de código. Um exemplo de tais softwares, o qual será utilizado nesta pesquisa, é o verificador de modelos ESBMC.

Este verificador, baseado em SMT (*Satisfiability Modulo Theories*), verifica execuções limitadas de um programa onde, usando um limite de chaveamento de contexto, faz o desdobramento de loops e *inlining* de funções, para então gerar um grafo de controle acíclico contendo todas as informações sobre o programa. Este grafo é convertido em uma fórmula SMT, que então é solucionada utilizando as lógicas QF_AUFBV e QF_AUFLIRA [21], [22] dos solucionadores SMT. A partir disso, se existir um *bug* dentro do código, o ESBMC retornará um contraexemplo mostrando todo o caminho que deve ser percorrido para se reproduzir o *bug,* informando assim o tipo/localização do *bug*. Com este processo, o ESBMC pode verificar várias propriedades em um determinado código, como por exemplo, *overflow* aritmético, segurança de ponteiros, vazamento de memória, limites do vetor, violações de atomicidade e ordem, *deadlock*, corrida de dados e assertivas especificadas pelo usuário. É importante ressaltar que, a partir dessas assertivas, é que serão checadas as propriedades dos códigos que utilizam o *framework* QT.

METODOLOGIA

Esta pesquisa tem como principal objetivo implementar um modelo operacional, com a finalidade de verificar a utilização de cada estrutura do *framework* QT e integrar o respectivo modelo no verificador ESBMC. De forma a alcançar tal objetivo, esta pesquisa adota uma metodologia que pode ser dividia em três etapas principais: estudo das tecnologias utilizadas por meio de uma revisão bibliográfica, o desenvolvimento da estrutura base do modelo operacional e a implementação do modelo focando na verificação das propriedades relacionadas ao *framework*.

Primeiramente, foi realizada uma revisão da literatura a respeito da teoria de verificação de modelos. Nesta etapa, a maioria dos conceitos importantes acerca de Lógica Proposicional, Lógica Temporal Linear (LTL), Lógica de Árvore de Computação (CTL), foram estudadas com o intuito de entender o funcionamento do verificador ESBMC [11]. Esta parte contempla também um estudo aprofundado sobre o *framework* QT, onde através de análises sistemáticas de programas que utilizam este *framework*, foi possível identificar suas principais funcionalidades.

A partir da revisão de literatura, foi desenvolvida a estrutura base do modelo operacional, abordando assim os principais módulos do *framework*, suas respectivas bibliotecas e estruturas (i.e., classes, métodos e funções). Além disso, de forma a validar tais implementações, foi construída uma suíte de teste automatizada contendo programas em C++/QT, que exploram as propriedades a serem verificadas pelo modelo operacional.

Por fim, após a conclusão das etapas anteriores, foram modeladas as estruturas definidas no modelo operacional, de tal forma que, através destas implementações, o verificador ESBMC possa checar todas as propriedades das funcionalidades do *framework,* que são utilizadas nos códigos da suíte de teste e, posteriormente, verificar aplicações reais.

DISCUSSÃO DOS RESULTADOS

Durante o processo de verificação com o EBSMC, o primeiro estágio que o código a ser verificado enfrenta, é o *PARSER*. Nesse estágio, o verificador transcreve o código em uma estrutura de dados, denominada Árvore de Sintaxe Abstrata (ASA), contendo todas as informações necessárias para a verificação do algoritmo. Entretanto, para realizar este estágio com sucesso, o ESBMC precisa identificar corretamente todas as estruturas do código que será verificado. Porém, até então o verificador não era capaz de identificar as estruturas de um código que utiliza o *framework* QT, visto que o ESBMC somente suporta as linguagens ANSI-C e C++. Além disso, existe o fato das bibliotecas padrões do QT, possuírem estruturas complexas, de baixo nível, que tornam a utilização das mesmas, durante o processo de verificação, uma técnica inviável.

Devido a estas circunstâncias, que se faz necessário o desenvolvimento de um modelo operacional, escrito em C++, para representar da forma mais simplificada possível, o *framework* QT. A partir da utilização de modelos mais simples, é possível diminuir a complexidade da ASA, e consequentemente, diminuindo o custo computacional, de tal forma que o ESBMC possa se basear neste modelo para, durante o *PARSER*, construir uma ASA que consiga englobar todas as propriedades necessárias para a verificação do código de forma rápida.

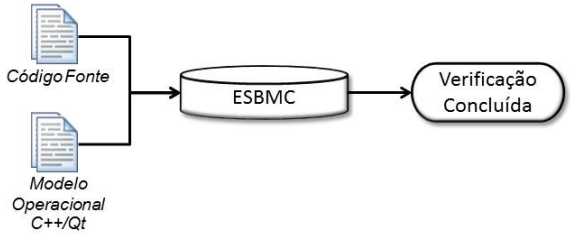

Figura 1 - Diagrama ilustrando o processo de verificação utilizando o modelo operacional.

O modelo operacional é desenvolvido separadamente do núcleo do ESBMC, e é integrado somente no processo de verificação. O diagrama da Figura 1 ilustra como decorre o processo de

verificação utilizando o modelo operacional. Como se trata de um processo automático, deve-se passar para o verificador o código que será verificado, juntamente com o modelo operacional e, a partir disso, o verificador realizará todo o processo de verificação, retornará como resultado a existência ou a inexistência de algum *bug* e, além disso, no caso da detecção do respectivo *bug*, também retornará um contraexemplo indicando o caminho que deve ser percorrido, durante a execução do programa, para expor o determinado erro. Na Figura 2, pode ser observada uma imagem referente à aplicação denominada "*Animated Tiles*" [15], um dos exemplos de códigos disponíveis na documentação do QT [16], que foi desenvolvido pelo Instituto Nokia de Tecnologia [17]. Nesta aplicação, existem alguns ícones no canto inferior direito, que possibilitam ao usuário escolher a disposição das imagens presentes no centro da aplicação. Este código contém 215 linhas e exemplifica a utilização de várias bibliotecas, contidas no módulo do *framework* denominado *QtGui,* relacionadas ao desenvolvimento de animações e interfaces gráficas.

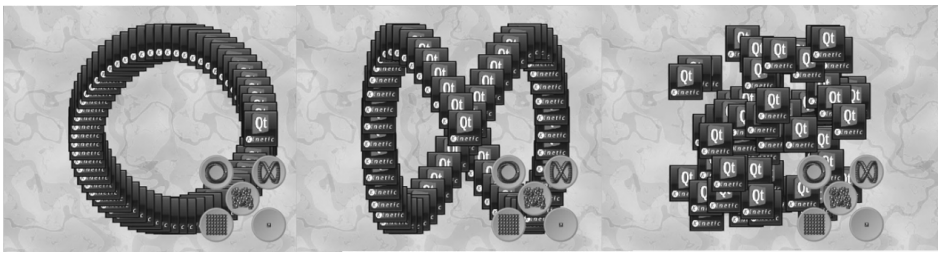

Figura 2 - Imagens dos diferentes tipos de disposições das imagens contidas na aplicação *Animated Tiles*.

Na Figura 3, pode ser observado um trecho do código da aplicação *Animated Tiles*. Neste trecho de código, é definido um objeto da classe *QTimer* e, em seguida, este objeto chama o método *start()* que inicia ou reinicia uma contagem em um determinado intervalo de tempo (em milissegundos), que é então passado como parâmetro [18]. Baseando-se nas especificações da documentação do *framework*, é construída uma estrutura da respectiva classe, contendo o corpo da mesma e a assinatura dos seus respectivos métodos. Após a estrutura ser definida, é necessário

que cada método seja modelado. A utilização de assertivas na composição desta modelagem garante a verificação das propriedades relacionadas com o respectivo método. Existem várias propriedades que devem ser checadas, tais como, acesso inválido de memória, definição de tempo e tamanho com números negativos, acesso de arquivos inexistentes, ponteiros nulos e assim por diante. Entre as propriedades que devem ser checadas, existem as pré-condições, que determinam as condições mínimas para que uma determinada funcionalidade seja utilizada corretamente.

No trecho de código em questão, por exemplo, o método *start()* possui uma pré-condição: o valor que é passado como parâmetro deve ser, obrigatoriamente, maior ou igual a zero, visto que se trata da especificação de um determinado tempo. Deste modo, dentro do método é adicionada uma assertiva checando se o valor segue a especificação e, caso essa propriedade seja violada, o ESBMC indica que o código contém um erro e envia então uma mensagem especificando o erro em questão. Pode ser observado no lado esquerdo da Figura 3, o trecho de código retirado da aplicação *Animated Tiles*, e no lado direito, o modelo operacional criado para a classe *QTimer*.

Além disso, existem métodos que, do ponto de vista da verificação, não apresentam nenhuma propriedade a ser verificada. Por exemplo, métodos cuja única funcionalidade é imprimir um determinado valor. Deste modo, como o ESBMC testa o software e não o hardware, verificar se um valor foi impresso corretamente no hardware não é analisado por este trabalho. Tais métodos apresentam no modelo uma estrutura (assinatura), pois é preciso que o verificador os reconheça, mas não apresentam nenhuma modelagem (corpo), pois não existem propriedades a serem verificadas.

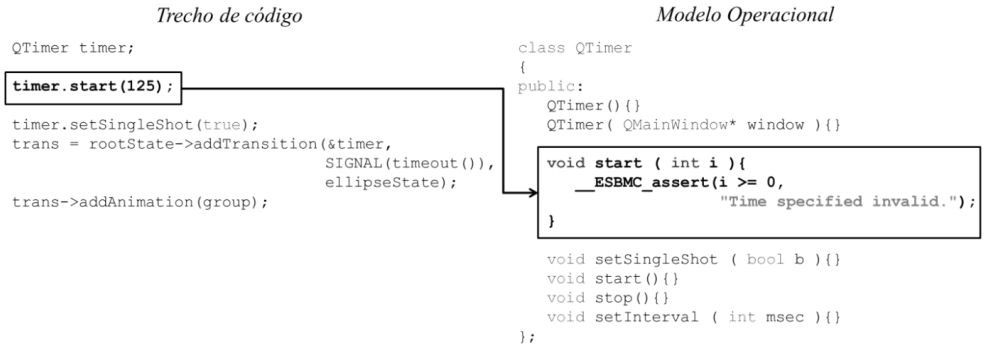

Figura 3 - A esquerda se encontra um trecho de código retirado da aplicação *Animated Tiles*, e a direita se encontra o modelo operacional da classe *QTimer*.

Por outro lado, existem métodos que, além de apresentarem propriedades que precisam ser analisadas como pré-condições, também apresentam propriedades que devem ser analisadas como pós-condições. Um exemplo disso está presente no código do lado esquerdo da Figura 4, onde são inseridos elementos no início de uma determinada lista (*mylist*), posteriormente é retirado o elemento no início da lista e então é checado, através de uma assertiva, o elemento inicial da respectiva lista. Neste caso, se forem feitas apenas as checagens das pré-condições, não seria possível validar de forma correta as assertivas no código, portanto é necessário que no modelo operacional dos respectivos métodos, seja implementada a simulação do comportamento dos mesmos, desta forma, conseguindo armazenar as informações necessárias sobre a estrutura em questão (a lista), e então fazer a verificação de forma correta. Um exemplo dos modelos de métodos, onde é necessária a simulação de suas funcionalidades, pode ser observado no lado direito da Figura 4. Na primeira fase desta pesquisa, a partir de análise de alguns *benchmarks* fornecidos pelo Instituto Nokia de Tecnologia, foi observado o constante uso das bibliotecas pertencentes aos módulos *QtGui*, que contém as definições gráficas, e *QtCore*, que contém as classes base para outros módulos. A partir disso, foi construída uma suíte de teste, denominada

esbmc-Qt, que verifica de forma automática todos os códigos nela adicionados, e que contém códigos que englobam as bibliotecas dos módulos citados anteriormente.

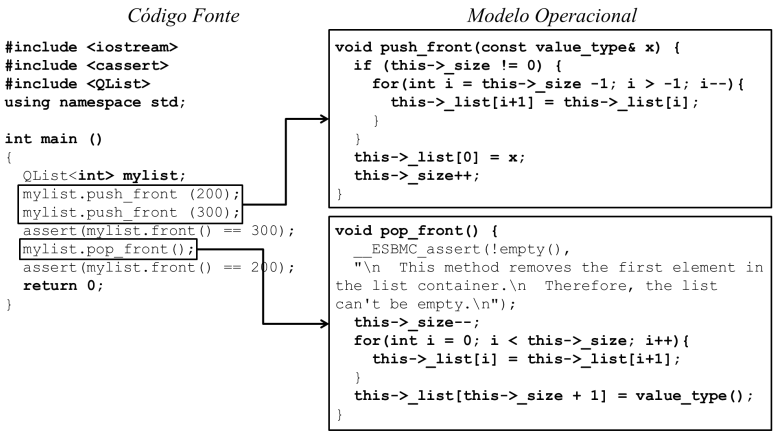

Figura 4. Exemplo de modelos de métodos cujas funcionalidades são simuladas no processo de verificação.

À medida que foram desenvolvidos os modelos operacionais, foram também adicionados casos de teste na respectiva suíte para validar as implementações. Todos os testes foram realizados em um computador Intel Core i7-2600, 3,40 GHz com 24 GB de RAM executando o sistema operacional *Ubuntu* de 64 bits. Para todos os conjuntos de testes, o prazo individual e limite de memória para cada caso de teste foi definido para 900 segundos e 24 GB (22 GB de RAM e 2 GB de memória virtual), respectivamente. Os tempos indicados foram medidos usando o comando *time*. Atualmente, a suíte de teste contém 52 casos de teste, totalizando 1671 linhas de código puro, levando 414,355 segundos para serem verificados, sendo que 91,67% são verificados corretamente e 8,33% apresentam um resultado "falso negativo", que ocorre quando um caso não contém erro e o verificador indica o contrário. Tais erros ocorrem devido a problemas com a manipulação de ponteiros internamente no ESBMC. Além disso, foram desenvolvidos modelos operacionais referentes a 128 bibliotecas do *framework* QT, totalizando 3483 linhas de código puro.

CONCLUSÃO

Esta pesquisa visa desenvolver um modelo operacional capaz de representar, de forma simplificada, o *framework* multi-plataforma QT e então integrá-lo no processo de verificação, de códigos em C++/QT, com o verificador de modelos ESBMC.

Neste relatório foram abordados todos os conceitos fundamentais a cerca da funcionalidade do ESBMC, como por exemplo, a Verificação Formal, a Verificação de Modelos, SMT, Lógica Proposicional, Lógica Temporal Linear, Lógica de Árvore de Computação, dentre outros, sendo que, é importante ressaltar que esta ferramenta se encontra no estado da arte da área de Verificação de Software.

Além disso, foram descritas as implementações da estrutura base dos modelos operacionais, que se concretizaram após uma análise dos *benchmarks* fornecidos pelo Instituto Nokia de Tecnologia e de uma revisão da literatura. Também foi abordado sobre os dois módulos do *framework,* que estão sendo trabalhados na pesquisa, *QtGui* e *QtCore*, e a construção da suíte de teste automatizada, criada para validar as implementações realizadas, que contém códigos englobando diversas bibliotecas dos respectivos módulos. Além disso, foi apresentada a abordagem utilizada durante o processo de desenvolvimento do modelo operacional e os resultados obtidos através das verificações realizadas na suíte de teste automatizada *esbmc-Qt*, totalizando uma cobertura de 91,67% da mesma.

Por fim, os trabalhados futuros a serem desenvolvidos nessa pesquisa, visam aumentar a variedade de bibliotecas representadas no modelo operacional, possibilitando assim uma maior cobertura do *framework* pelo verificador ESBMC. De modo similar, identificar mais propriedades a serem checadas e implementá-las nos modelos, e por fim ampliar o número de casos de teste, incluindo na respectiva suíte de teste, *benchmarks* e aplicações do mundo real.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] *Qt Project Hosting*. Documentação Qt. *Qt Project*. 2012. Disponível em: <http://Qt-project.org/doc/>. Acesso em: Setembro, 2012.

[2] CIMATTI, A.; MICHELI A.; NARASAMDYA, I.; e ROVERI, M. *Verifying SystemC: A Software Model Checking Approach*. In: FORMAL METHODS IN COMPUTER-AIDED DESIGN, 2010. Lugano, Suíça. p. 121-128.

[3] BARRETO, R., CORDEIRO, L. e FISCHER, B. *Verifying Embedded C Software with Timing Constraints using in Untimed Model Checker*. In: *13th* BRAZILLIAN WORKSHOP ON REAL-TIME SYSTEMS, 2011. Florianopolis, Brasil, p.89-100.

[4] CORDEIRO, L., FISCHER, B. e MARQUES-SILVA, J. *SMT-Based Bounded Model Checking for Embedded ANSI-C Software*. In: IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 2009. Auckland, Nova Zelândia, p. 137-148.

[5] CORDEIRO, L., FISCHER, B. e MARQUES-SILVA, J. *Continuous Verification of Large Embedded Software Using SMT-Based Bounded Model Checking*. In: 17TH IEEE INTERNATIONAL CONFERENCE AND WORKSHOPS ON ENGINEERING OF COMPUTER-BASED SYSTEMS, 2010. Universidade de Oxford, Reino Unido, p. 160-169.

[6] CORDEIRO, L. e FISCHER, B. *Verifying Multi-threaded Software using SMT-based Context-Bounded Model Checking*. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2011. Honoulu, Hawaii, p. 331-340.

[7] CORDEIRO, L. *SMT-Based Bounded Model Checking of Multi-Threaded Software in Embedded Systems*. 2011. Tese, Universidade de Southampton. Southampton, UK. 2011.

[8] CORDEIRO, L., FISCHER, B., e MARQUES-SILVA, J. *SMT-Based Bounded Model Checking for Embedded ANSI-C Software*. In: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2011.

[9] CORDEIRO, L., FISCHER, B., CHEN, H. e MARQUES-SILVA, J. *Semiformal Verification of Embedded Software in Medical Devices Considering Stringent Hardware Constraints*. In: 6TH IEEE INTERNATIONAL CONFERENCE ON EMBEDDED SOFTWARE AND SYSTEMS, 2009. Hangzhou, China, p. 396-403.

[10] CORDEIRO, L., MORSE, J., NICOLE, D. and FISCHER, B. *Context-Bounded Model Checking with ESBMC 1.17*, 2012. In: 18TH INTERNATIONAL CONFERENCE ON TOOLS AND ALGORITHMS FOR THE CONSTRUCTION AND ANALYSIS OF SYSTEMS. Tallinn, Estonia, LNCS 7214, p. 533-536.

[11] MORSE, J., CORDEIRO, L., NICOLE, D. and FISCHER, B. *Context-Bounded Model Checking of LTL Properties for ANSI-C Software*. In: 9TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND FORMAL METHODS, 2011, LNCS 7041, p. 302-317.

[12] CORDEIRO, L., FISCHER, B., and MARQUES-SILVA, J. *SMT-Based Bounded Model Checking for Embedded ANSI-C Software*. In: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2011.

[13] L. CORDEIRO, B. FISCHER, e J. MARQUES-SILVA. *SMT-based bounded model checking for embedded ANSI-C software*. In: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2012, v. 38, n. 4, pp. 957–974.

[14] MORSE, J., CORDEIRO, L., NICOLE, D. and FISCHER, B. *Handling Unbounded Loops with ESBMC 1.20*. In INTL. CONF. ON TOOLS AND ALGORITHMS FOR THE CONSTRUCTION AND ANALYSIS OF SYSTEMS (TACAS), Springer-Verlag, 2013 (ganhou medalha de bronze no ranking geral da *Second Intl. Competition on Software Verification*).

[15] *Qt Project Hosting*. *Animated Tiles Example*. Qt Digia. 2012. Disponível em: < http://doc.Qt.digia.com/Qt/animation-animatedtiles.html>. Acesso em: Setembro, 2012.

[16] *Qt Project Hosting*. *Animated Tiles Example*. Qt Project. 2012. Disponível em: <https://Qt-project.org/doc/Qt-4.8/animation-animatedtiles.html>. Acesso em: Setembro, 2012.

[17] NOKIA CORPORATION. NOKIA Global. 2012. Disponível em: <http://www.nokia.com/global/>. Acesso em: Setembro, 2012.

[18] *Qt Project Hosting*. *QTimer Class Reference*. Qt Project. 2013. Disponível em: <http://Qt-project.org/doc/Qt-4.7/Qtimer.html>. Acesso em: Setembro, 2012.

[19] ROCHA, H., BARRETO, R., CORDEIRO, L. and DIAS-NETO, A. *Understanding Programming Bugs in ANSI-C Software Using Bounded Model Checking Counter-Examples*. In INTL. CONF. ON INTEGRATED FORMAL METHODS, LNCS 7321, 2012. Springer-Verlag, p. 128-142.

[20] B. BERARD, M. BIDOIT, A. FINKEL. *Systems and Software Verification: Model-Checking Techniques and Tool*, 2010. Ed. 1, ISBN: 3642074782. Springer Publishing Company.

[21] BARRETT, C.; STUMP, A. e TINELLI, C. *The SMT-LIB Standard - Version 2.0*. In: PROCEEDINGS OF THE 8TH INTERNATIONAL WORKSHOP ON SATISFIABILITY MODULO THEORIES, 2010. Edinburgh, England.

[22] STUMP, A. e DETERS, M. *Satisfiability Modulo Theories Competition* (SMT-COMP), 2010. Disponível em: <http://smtcomp.sourceforge.net/2012/index.shtml>. Acesso em: Agosto, 2012.

[23] CLARKE, E. M.; GRUMBERG, O. and PELED, D. *Model Checking*. MIT Press, Cambridge, MA, 1999.

[24] CLARKE, E. M. and SCHLINGLOFF, H. *Model checking*. In A. Voronkov, *Handbook of Automated Deduction*. Elsevier, 2000.

Cronograma de Atividades

Nº	Descrição	Ago 2012	Set	Out	Nov	Dez	Jan 2013	Fev	Mar	Abr	Mai	Jun
1	Estudar a técnica de Verificação Formal de Modelos.	**X**	**X**									
2	Estudar a estrutura e a aplicação do framework QT		**X**									
3	Análise de programas e construção de uma suíte de teste			**X**								
4	Análise de benchmarks e definição de uma estrutura simplificada do framework QT				**X**							
5	Implementar as assinaturas das bibliotecas QT definidas na estrutura simplificada					**X**	**X**	**X**				
6	Modelagem dos métodos QT								**X**	**X**	**X**	
7	- Elaboração do Resumo e Relatório Final (atividade obrigatória) - Preparação da Apresentação Final para o Congresso (atividade obrigatória)											**X**