28th Brazilian Symposium on Computer Networks and Distributed Systems
24-28 May 2010
Gramado, RS, Brazil

12th Brazilian Workshop on Real-Time and Embedded Systems (WTR)

Published by
Sociedade Brasileira de Computação (SBC)

Organizing Committee
Carlos Eduardo Pereira (UFRGS)
Leandro Buss Becker (UFSC)
Antônio Jorge Gomes Abelém (UFPA)
Luciano Paschoal Gaspary (UFRGS)
Marinho Pilla Barcellos (UFRGS)

Organization
Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS)

Promoted by
Sociedade Brasileira de Computação (SBC)
Laboratório Nacional de Redes de Computadores (LARC)

Cover: Josué Klafke Sperb
Editorial Production: Flávio Roberto Santos, Roben Castagna Lunardi, Matheus Lehmann, Rafael Santos Bezerra, Luciano Paschoal Gaspary, Marinho Pilla Barcellos.

Additional Symposium Record:
Sociedade Brasileira de Computação (SBC)
Av. Bento Gonçalves, 9500 - Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia - CEP 91.509-900 - Porto Alegre - RS
Phone: +55 51 33086835
E-mail: sbc@sbc.org.br

International Data for Cataloguing in Publication (CIP)

Workshop de Tempo Real e Sistemas Embarcados (12. : 2010 : Gramado, RS).

Anais / XII Workshop de Tempo Real e Sistemas Embarcados; organizadores Carlos Eduardo Pereira... et al. − Porto Alegre : SBC, c2010.
193 p.

ISSN 2177-496X

1. Redes de computadores. 2. Sistemas distribuídos. I. Pereira, Carlos Eduardo. II. Título.

Promotion

Sociedade Brasileira de Computação (SBC)

Board of Directors

President
José Carlos Maldonado (USP)

Vice-President
Marcelo Walter (UFRGS)

Director of Administration
Luciano Paschoal Gaspary (UFRGS)

Director of Finance
Paulo Cesar Masiero (USP)

Director of Conferences and Special Interest Groups
Lisandro Zambenedetti Granville (UFRGS)

Director of Education
Mirella Moura Moro (UFMG)

Director of Publications
Karin Breitman (PUC-Rio)

Director of Special Programs
Ana Carolina Salgado (UFPE)

Director of Regional Chapters
Thais Vasconcelos Batista (UFRN)

Director of Promotions
Altigran Soares da Silva (UFAM)

Director of Professional Activities
Ricardo de Oliveira Anido (UNICAMP)

Director of Special Events
Carlos Eduardo Ferreira (USP)

Director of Cooperation with Scientific Societies
Marcelo Walter (UFRGS)

Promotion

Council

2009-2013
Virgílio Almeida (UFMG)
Flávio Rech Wagner (UFRGS)
Silvio Romero de Lemos Meira (UFPE)
Itana Maria de Souza Gimenes (UEM)
Jacques Wainer (UNICAMP)

2007-2011
Cláudia Maria Bauzer Medeiros (UNICAMP)
Roberto da Silva Bigonha (UFMG)
Cláudio Leonardo Lucchesi (UNICAMP)
Daltro José Nunes (UFRGS)
André Ponce de Leon F. de Carvalho (USP)

Additional Members - 2009-2011
Geraldo B. Xexeo (UFRJ)
Taisy Silva Weber (UFRGS)
Marta Lima de Queiroz Mattoso (UFRJ)
Raul Sidnei Wazlawick (UFSC)
Renata Vieira (PUCRS)

Laboratório Nacional de Redes de Computadores (LARC)

Board of Directors

Director of the Scientific Technical Council
Artur Ziviani (LNCC)

Executive Director
Célio Vinicius Neves de Albuquerque (UFF)

Vice-Director of the Scientific Technical Council
Flávia Coimbra Delicato (UFRN)

Executive Vice-Director
Luciano Paschoal Gaspary (UFRGS)

Institutional Members
CEFET-CE, CEFET-PR, IME, INPE/MCT, LNCC, PUCPR, PUC-RIO, SESU/MEC, UECE, UERJ, UFAM, UFBA, UFC, UFCG, UFES, UFF, UFMG, UFPA, UFPB, UFPE, UFPR, UFRGS, UFRJ, UFRN, UFSC, UFSCAR, UNICAMP, UNIFACS, USP.

Organization

Organizing Committee

General Chairs
Luciano Paschoal Gaspary (UFRGS)
Marinho Pilla Barcellos (UFRGS)

Technical Program Committee Chairs
Luci Pirmez (UFRJ)
Thaís Vasconcelos Batista (UFRN)

International Talks and Tutorials Chair
Lisandro Zambenedetti Granville (UFRGS)

Panels Chair
José Marcos Silva Nogueira (UFMG)

National Tutorials Chair
Carlos Alberto Kamienski (UFABC)

Workshops Chair
Antônio Jorge Gomes Abelém (UFPA)

Tools Session Chair
Nazareno Andrade (UFCG)

Steering Committee
Artur Ziviani (LNCC)
Carlos André Guimarães Ferraz (UFPE)
Célio Vinicius Neves de Albuquerque (UFF)
Francisco Vilar Brasileiro (UFCG)
Lisandro Zambenedetti Granville (UFRGS)
Luís Henrique Maciel Kosmalski Costa (UFRJ)
Marcelo Gonçalves Rubinstein (UERJ)
Nelson Luis Saldanha da Fonseca (UNICAMP)
Paulo André da Silva Gonçalves (UFPE)

Organization

Local Organizing Committee
Adler Hoff Schmidt (UFRGS)
Alan Mezzomo (UFRGS)
Alessandro Huber dos Santos (UFRGS)
Bruno Lopes Dalmazo (UFRGS)
Carlos Alberto da Silveira Junior (UFRGS)
Carlos Raniery Paula dos Santos (UFRGS)
Cristiano Bonato Both (UFRGS)
Flávio Roberto Santos (UFRGS)
Jair Santanna (UFRGS)
Jéferson Campos Nobre (UFRGS)
Juliano Wickboldt (UFRGS)
Leonardo Richter Bays (UFRGS)
Lourdes Tassinari (UFRGS)
Luís Armando Bianchin (UFRGS)
Luis Otávio Luz Soares (UFRGS)
Marcos Ennes Barreto (UFRGS)
Matheus Brenner Lehmann (UFRGS)
Pedro Arthur Pinheiro Rosa Duarte (UFRGS)
Pietro Biasuz (UFRGS)
Rafael Pereira Esteves (UFRGS)
Rafael Kunst (UFRGS)
Rafael Santos Bezerra (UFRGS)
Ricardo Luis dos Santos (UFRGS)
Roben Castagna Lunardi (UFRGS)
Rodolfo Stoffel Antunes (UFRGS)
Rodrigo Mansilha (UFRGS)
Weverton Luis da Costa Cordeiro (UFRGS)

Message from the SBRC General Chairs

Bem-vindo(a) ao XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2010)! Esta edição do simpósio está sendo realizada de 24 a 28 de maio de 2010 na pitoresca cidade de Gramado, RS. Promovido pela Sociedade Brasileira de Computação (SBC) e pelo Laboratório Nacional de Redes de Computadores (LARC) desde 1983, o SBRC 2010 almeja não menos que honrar com uma tradição de quase 30 anos: ser reconhecido como o mais importante evento científico em redes de computadores e sistemas distribuídos do país, e um dos mais concorridos em Informática. Mais do que isso, pretende estimular intercâmbio de idéias e discussões qualificadas, aproximá-lo(a) de temas de pesquisa efervescentes e fomentar saudável aproximação entre estudantes, pesquisadores, professores e profissionais.

Para atingir os objetivos supracitados, reunimos um grupo muito especial de professores atuantes em nossa comunidade que, com o nosso apoio, executou com êxito a tarefa de construir um **Programa Técnico** de altíssima qualidade. O SBRC 2010 abrange as seguintes atividades: 20 sessões técnicas de artigos completos, cobrindo uma grande gama de problemas em redes de computadores e sistemas distribuídos; 2 sessões técnicas para apresentações de ferramentas; 5 minicursos ministrados de forma didática, por professores da área, sobre temas atuais; 3 palestras e 3 tutoriais sobre tópicos de pesquisa avançados, apresentados por especialistas nacionais e estrangeiros; e 3 painéis versando sobre assuntos de relevância no momento. Completa a programação técnica a realização de 8 *workshops* satélites em temas específicos: WRNP, WGRS, WTR, WSE, WTF, WCGA, WP2P e WPEIF. Não podemos deixar de ressaltar o **Programa Social**, organizado em torno da temática "*vinho*", simbolizando uma comunidade de pesquisa madura e que, com o passar dos anos, se aprimora e refina cada vez mais.

Além da ênfase na qualidade do programa técnico e social, o SBRC 2010 ambiciona deixar, como marca registrada, seu esforço na busca por excelência organizacional. Tal tem sido perseguido há mais de dois anos e exigido muita determinação, dedicação e esforço de uma equipe afinada de organização local, composta por estudantes, técnicos administrativos e professores. O efeito desse esforço pode ser percebido em elementos simples, mas diferenciais, tais como uniformização de datas de submissão de trabalhos, portal *sempre* atualizado com as últimas informações, comunicação sistemática com potenciais participantes e pronto atendimento a qualquer dúvida. O nosso principal objetivo com essa iniciativa foi e continua sendo oferecer uma elevada *qualidade de experiência* a você, colega participante!

Gostaríamos de agradecer aos membros do Comitê de Organização Geral e Local que, por conta de seu trabalho voluntário e incansável, ajudaram a construir um evento que julgamos de ótimo nível. Gostaríamos de agradecer, também, à SBC, pelo apoio prestado ao longo das muitas etapas da organização, e aos patrocinadores, pelo incentivo à divulgação de atividades de pesquisa conduzidas no País e pela confiança depositada neste fórum. Por fim, nossos agradecimentos ao Instituto de Informática da UFRGS, por viabilizar a realização, pela quarta vez, de um evento do porte do SBRC.

Sejam bem-vindos à Serra Gaúcha para o "SBRC do Vinho"! Desejamos que desfrutem de uma semana agradável e proveitosa!

Luciano Paschoal Gaspary
Marinho Pilla Barcellos
SBRC 2010 General Chairs

Message from the WTR Chairs

It is our great pleasure to welcome all participants of the 12nd Brazilian Workshop on Real-Time and Embedded System (WTR 2010). For the fifth consecutive year, the workshop is held in the first day of the Brazilian Networks Symposium (SBRC). WTR 2010 is very special because this year it is held in conjunction with the First South-American Embedded and Real-Time Systems Week, which is an event of 5 days also composed by the 1st Embedded Systems Workshop (WSE) and by the 4th ARTIST2 South-American School for Embedded Systems. The program of WTR 2010 is composed by 10 regular papers, which reflect the state-of-the art of our research, and by 11 Work-In-Progress papers, representing new research ideas and directions. We would like thank to all people and organizations that helped in making the WTR 2010. This list included but it is not restrict to: our universities UFRGS and UFSC, the SBRC organizing committee for all support, mainly in the conference logistics, all reviewers for their great work, the authors for submitting very interesting papers that we hope will be a basis for interesting discussion and all participants. We wish you all an excellent workshop and a pleasant stay in Gramado.

Carlos Eduardo Pereira
WTR 2010 General Chair

Leandro Buss Becker
WTR 2010 Program Committee Chair

WTR Program Committee

Antônio Augusto Fröhlich, Federal University of Santa Catarina, Brazil
Carlos Montez, Federal University of Santa Catarina, Brazil
Daniel Mossé, Univ. of Pittsburgh, USA
Eduardo Tovar, ISEP, Portugal
Francisco Vasques, Univ. of Porto, Portugal
George Lima, Federal University of Bahia, Brazil
Gerhard Fohler, UNI-KL, Germany
Jörg Kaiser, OvG Univ. of Magdeburg, Germany
Jean-Marie Farines, Federal University of Santa Catarina, Brazil
João Netto, Federal University of Rio Grande do Sul, Brazil
Julius Leite, Fluminense Federal University, Brazil
Keiko Fonseca, Federal Technological University of Paraná, Brazil
Luciano Porto Barreto, Federal University of Bahia, Brazil
Mamoun Filali Amine, IRIT, France
Orlando Loques, Fluminense Federal University, Brazil
Raimundo Barreto, Federal University of Amazonas, Brazil
Raimundo Macêdo, Federal University of Bahia, Brazil
Rodrigo Santos, Univ. Nacional del Sur, Argentina
Rômulo Silva de Oliveira, Federal University of Santa Catarina, Brazil
Thomas Nolte, Malardalen Univ., Sweeden

WTR Reviewers

Antônio Augusto Fröhlich, Federal University of Santa Catarina
Carlos Montez, Federal University of Santa Catarina
Carlos Pereira, Federal University of Rio Grande do Sul
Daniel Mossé, University of Pittsburgh
Filipe Pacheco, Polytechnic Institute of Porto
Francisco Vasques, University of Porto
George Lima, Federal University of Bahia
Gerhard Fohler, TU Kaiserslautern
Giovani Gracioli, Federal University of Santa Catarina
Gustavo Medeiros, Federal University of Santa Catarina
Jörg Kaiser, Otto-von-Guericke-University of Magdeburg
Jean-Marie Farines, Federal University of Santa Catarina
João Netto, Federal University of Rio Grande do Sul
Julius Leite, Fluminense Federal University
Keiko Fonseca, Federal Technological University of Paraná
Leandro Buss Becker, Federal University of Santa Catarina
Luciano Porto Barreto, Federal University of Bahia
Luis Ferreira, Polytechnic Institute of Porto
Luis Nogueira, Polytechnic Institute of Porto
Mamoun Filali Amine, Institut de Recherche en Informatique de Toulouse
Marco Wehrmeister, Federal University of Rio Grande do Sul
Orlando Loques, Fluminense Federal University
Rafael Cancian, Federal University of Santa Catarina
Raimundo Barreto, Federal University of Amazonas
Raimundo José de Araújo Macêdo, Federal University of Bahia
Rodrigo Santos, Universidad Nacional del Sur
Rômulo Silva de Oliveira, Federal University of Santa Catarina
Thomas Nolte, Malardalen University
Valter Roesler, Federal University of Rio Grande do Sul

Index

Technical Session 4 – Sensor Networks and Wireless Communication

Work in Progress Papers

12th Brazilian Workshop on
Real-Time and Embedded Systems

♦

Technical Session 1

Software Development and Architectures

Model-Driven Development of Embedded Systems

Tino Brade, Michael Schulze, Sebastian Zug, Jörg Kaiser

[1]Department for Distributed Systems
Universität Magdeburg

Universitätsplatz 2, 39106 Magdeburg, Germany

tino.brade@student.uni-magdeburg.de
{mschulze, zug, kaiser}@ivs.cs.uni-magdeburg.de

Abstract. *Distributed mechatronic systems integrate sensors, processing units, communication networks, and actuators. In order to achieve a rapid development process and an improved maintainability it is necessary to combine and replace such modular components in a flexible way. For seamless composability we developed communication middleware and a programming abstraction for distributed sensors and actuators. In this paper we describe a comprehensive development toolchain based on these abstractions. Sensors and actuators are specified by an extended electronic datasheet for smart embedded devices. Following this approach, the user defines the capabilities of a device on a high system level in a declarative way. From that description, the functionality is generated using domain-specific tools like Matlab/Simulink. Finally, we improved the back-end tools that provide the code for the target system. Thus this code is derived with minimal user-intervention.*

1. Introduction

The development of distributed mechatronic applications requires an interdisciplinary effort of electrical and mechanical engineering as well as computer science. Such applications integrate heterogeneous systems like different sensor types, computers/microcontrollers and multiple diverse communication networks. To simplify and support the development process [Schulze and Zug 2008] describe a modular system structure. This allows combining components of different development stages, e.g. components in a preliminary stage, simulated by Simulink blocks with already tested hardware components (hardware-/software-in-the-loop systems). The modularity also supports testing and improves maintainability of the final product.

Precondition for this functional composability is a common abstraction for the components. We distinguish between three main aspects. Firstly, we need a generic communication interface, which decouples the application development from underlying networks and their specific characteristics. The second aspect is concerned with the internal structure of a component to provide a high degree of freedom when combining sensors, actuators and network interfaces. Thirdly, we suggest a method to integrate components that are defined and configured outside the development process (e.g. legacy components). We strive for an easy adaptation and integration of such components. A mismatch of configurations should be avoided by respective compatibility checks [Kaiser et al. 2008].

General communication mechanisms and interfaces are important for distributed applications. Usually, a broad spectrum of networks and lower level protocols has to be integrated. At a certain communication level, however, all elements of the distributed application have to agree on a common structured communication object. This requires the encapsulation of the underlying heterogeneous network structure. Therefore, we developed our communication middleware FAMOUSO (Family of Adaptive Middleware for autonomOUs Sentient Objects [Herms et al. 2008, Schulze 2009]) that provides event-based communication over different network types according to the publish/subscribe paradigm (CAN [Robert Bosch GmbH 1991], 802.15.4 [ZigBee Alliance 2003], Ethernet communication, etc.). FAMOUSO allows communication between components specified in different programming languages (C/C++, Python, Java, .NET) or by domain-specific engineering tools (LabVIEW, Matlab/Simulink). In contrast to other communication middleware, FAMOUSO supports different qualities of communication and is particularly developed for resource constraint devices as 8-bit controllers often used in smart devices.

A additional abstraction level addresses the internal structure of a sensor/actuator node. It describes typical internal modules required in smart devices. These include modules for data acquisition, signal conditioning, filtering and fault detection. Such a programming abstraction forces the developer to structure application specific code into modular and replaceable subsystems that e.g. can be represented and specified as blocks in Matlab/Simulink [MathWorks 2010a]. In [Zug and Kaiser 2009] we examined typical faults in sensor applications and suggested an architecture capable to cope with such situations. These concepts are also used in our proposed development chain.

Finally, we provide an abstract description of hardware and software configuration sets for a network node. This substantially simplifies and accelerates the development process. A standardized description also supports the use of code generation tools. Hence, changing parameters in some system component or even the integration of completely new hardware may be performed automatically or with minimal intervention only. As a result, the descriptions of the hardware components are capable to detect compatibility problems and may avoid faulty combinations. This is described in [Kaiser et al. 2008].

The aspects addressed above, i.e. communication, internal structures, and component description, represent the structure and the interfaces of a sensor/actuator node but do not reflect the component's behavior so far. The functionality of a component, e.g. the signal processing and the algorithms for filtering, has to be specified for each type of sensor and actuator specifically. This is the realm of domain specific programming languages and tools. Therefore, we combined the proposed abstractions with the Mathworks Simulink toolchain. This offers a large library of packages for control design and signal and state processing. Additionally, it includes tools (e.g. Real-Time-Workshop) to generate code from such a set of blocks for a specific hardware target. Our work includes an enhancement of such back-end tools for the AVR micro-controller.

The paper is structured as follows: In Section 2 we introduce the framework and illustrate the main concepts. Subsequently we use an example scenario to present the system descriptions of an appropriate node in Section 3 and the behavior and code generation for this application Section 4. Section 5 illustrates the state of the art and lists a survey of related approaches. Section 6 summarizes the paper and specifies current and future work.

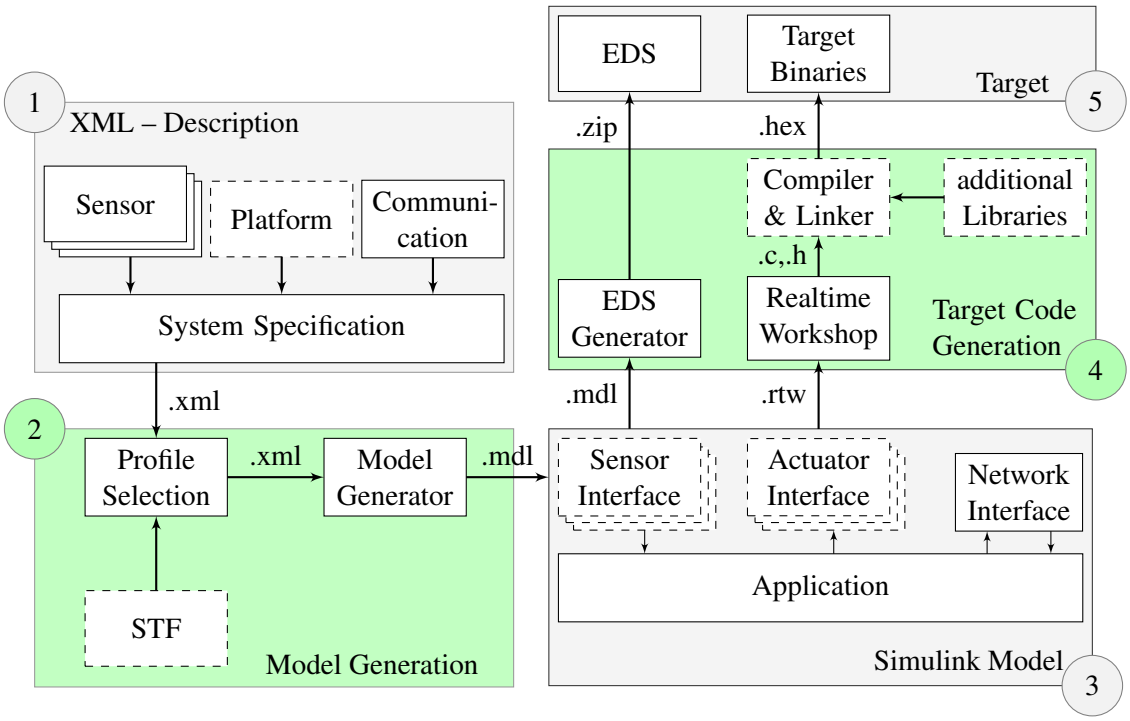


Figure 1. Framework development chain from XML descriptions to target code

2. Development Tool Chain for Distributed Applications

In this section we describe the main concepts of our development framework. Figure 1 illustrates the main workflow (1 - 5) and shows the three steps – *XML Description, Simulink Model* and *Target* – connected by two transformation/generation tools – *Model Generation* and *Target Code Generation*.

The box marked by (1) on the left site, *XML Description*, is the starting point of our development chain. Three different XML files include all information of the connected sensors. A platform specification defines the processor type, the board properties and the communication settings. Section 3 describes the structure and contents of those files. The clear separation of the descriptions in different files enables a flexible combination in varying scenarios. The *System Specification* determines the connection between sensor and board interfaces (an example is given in Section 3) .

Based on this abstract system specification we derive an appropriate Simulink model during the first generation step, marked by (2) in the workflow illustration. The *Profile Selection* module provides a flexible definition of the run-time environment of the Simulink model. This can be a simulation in the Simulink environment or an implementation for an embedded target. The *Profile Selection* algorithms check the availability on a Simulink *System Target File* (STF) necessary for the target code generation process and part of the Simulink model file. In a second step the users have to decide about the sensor inputs which can be implemented as simulation module or by an interface block to a real transducer. The sensor data sheets contain all information required for a basic sensor signal simulation. Hence, it is possible to coordinate simulated and real sensors on the target. The *Profile Selection* provides a graphical user interface and calls the *Model Generator*, that transforms the collected information into a Simulink file (.mdl).

We obtain the basic structure of a Simulink model ready for an integration of the *Application* methods as depicted in the right box labeled with (3). The *Application* processes input signals from simulated or real sensors as from the networks and calculates output values transmitted to *Actuator Interfaces*. At this stage the benefit of the framework is obvious. The developer does not need to cope with network or hardware interfaces. This is done automatically. It should be noted, that the user is able to control the *Profile Selection* from the *Model Generation* out of Simulink. The special block calls the generation process that derives a new Simulink model. Hence, simulated sensors can be replaced by real ones without any error-prone copy and paste actions between different model variants.

Simulink offers a comprehensive code generation tool chain, called *Realtime Workshop* [MathWorks 2010b], which was enhanced by a target configuration for a small 8-bit controller during a previous work [Brade 2009]. The necessary information for the target code generation process is stored in the Simulink model during the model generation process. Suitable compiler, linker and flash tools etc. are defined as well as additional libraries, run-time specification etc. so that it needs only a mouse click to bring the Simulink model down to the embedded target platform. Additionally, the Simulink model is used for the derivation of an Electronic Datasheet (*EDS*) of node's output done by our *EDS Generator*. It contains all information that is helpful for a correct interpretation, validation, and processing of the results transmitted by this node. A compressed version is stored on the target. This allows the use of service discovery methods for a dynamic integration and interaction.

In the following subsection we illustrate two parts of our framework more in detail, the XML description files for sensors and platforms and the profile switch mechanisms controlled out of the Simulink environment. For a comprehensible presentation we introduce an example scenario and assume, that we want to develop a smart temperature sensor. Our sensor node is equipped with an Atmel AT90CAN128 processor combines two types of temperature transducers, a AD592 and a CON-THEMOD with a higher precision but a smaller range. The AD592 offers only a voltage output while the CON-THEMOD module provides an additional digital I2C interface. The temperature signals should be jointed and the result published via FAMOUSO.

3. XML Description Files

The electronic data sheets were stored in an XML structure. XML offers a simple, standard way to exchange structured textual data. The advantage of this technique lies in the availability of machine processing using the Document Object Model (DOM) [Apparao et al. 1998] and the human readability in contrast to the binary representation of IEEE-1451. As depicted in Figure 1 we divide the entire sensor description into three types for sensor, platform and communication.

3.1. Sensor/Actuator Description

In a sensor data sheet we store general information of the transducer, interface description and context information. Due to the similarity, we handle both component types, sensors and actuators, with the same file type and structure. We are talking about sensors and sensor description in the following for the sake of simplicity and readability. However, similar statements are also applicable to actuators.

The first part of a sensor description file contains general information like sensor type, the vendor, layout pins and its supply properties similar to IEEE 1451.2. The interface description informs about all available interfaces and their configuration parameters that deliver sensor data or receive actuator commands. The third section is used to store context sensor properties. These properties describe signal behavior and parameters that are necessary for fusion and weighting mechanisms, simulation purposes and fault detection techniques concerning a reference output [Dietrich et al. 2010].

Considering our example scenario we have to write two sensor description files using an appropriate editor. This tool allows a convenient handling of the XML Files by a GUI that provides form structure with input boxes, buttons and drop down frames. The interface description list contains one entry for the AD592 sensor and two specifications for the CON-THEMOD, for each sensor statistical signal parameters and a linearization function are integrated.

3.2. Platform Description

The platform description combines basic information and the available interfaces of the platform. The first part has a similar background as the general information section from sensor description. Here we store general information like board type, board revision and processor type similar to CODES described in [Kaiser and Piontek 2006]. The second part encounters the interfaces and corresponding device drivers of the platform.

Our sensor board used in the example scenario provides the periphery of the Atmel processor and supplies interfaces to the analog digital conversion, I2C, and CAN buses, as well as two UART.

3.3. Communication Description

The communication description is tailored for the integration of FAMOUSO. It contains necessary information like subjects of the events for publishing and subscription as well as parameters like periodicity, omission degrees, etc.

In the example scenario the temperature values and their validity are published periodically.

3.4. System Specification

The three XML files mentioned above are composed in a system specification file. The specification file selects the interfaces between sensor components and the used target platform. Storing the connection data in a separate file results in a high degree of flexibility because it opens the ability to compose and replace different sensors, actuators and platforms. Consequently, it is possible to develop the application on different boards and select yet another one for series production. The system specification file is also generated by a graphical tool that helps to hide the error-prone task of manual XML editing and checks the compatibility of the interfaces. E.g. if a developer tries to combine a sensor with a LIN bus with our scenario board, this results in an error message.

Listing 1 summarizes parts of a system specification file. Line 4 and 5 define references to the connected sensors and their descriptions. As noted above, the first sensor (AD592) is assigned in line 10 to the third channel of the analog digital converter of our platform. The second transducer (CON-TEMOD-I2C) is connected to the I2C bus and uses the fifth address.

Listing 1. Example of a System Specification XML file

```
<?xml version="1.0" encoding="utf-8"?>
<container>
    <context>
        <Sensor1>AD592-BN</Sensor1>
        <Sensor2>CON_TEMOD_I2C</Sensor2>
    </context>
    <connections>
        <AVR-Processor>
            <Sensor1>
                <plattform>analog.channel3</plattform>
                <sensor>analog</sensor>
            </Sensor1>
            <Sensor2>
                <plattform>i2c.address5</plattform>
                <sensor>i2c</sensor>
            </Sensor2>
        </AVR-Processor>
    </connections>
</container>
```

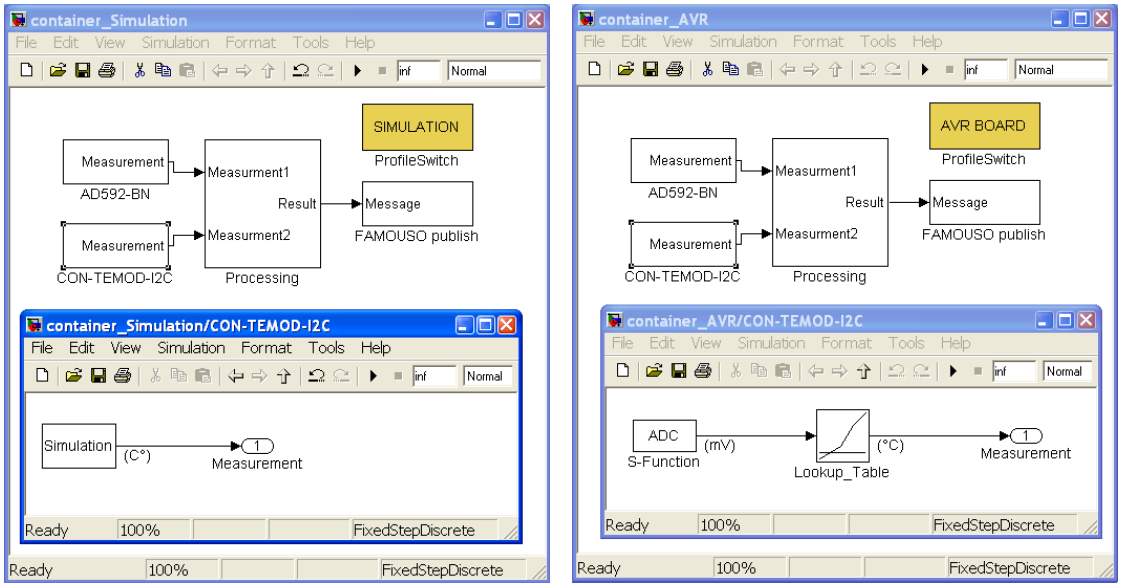
4. Simulink Model Generation

The model generator transforms the information of the specification file into a Simulink model. Beside the visible model structure of a node, the process generates the context information, device drivers for sensor components and the scripts necessary for profile switches. For this purpose a configuration set for target code generation is selected and the user can customize the sensor interfaces with the respective graphical tool. The configuration set, called profile, contains device drivers and the Simulink parameterization for the associated platform. In our example project we use two general profiles: the simulation in the Simulink environment and the execution of node behavior on an Atmel AVR processor. Within real target profiles the developer can replace physical sensors interfaces by simulation blocks.

A Simulink model combines different blocks connected by arrows that represents the data flow as shown in Figure 2. A block is presented by a rectangle and hides its functionality behind the respective graphical representation. The developer may describe each block by an individual S-Function, by a model reference or including a common library block.

Figure 2 shows the output of the generation process. We obtain a basic structure that offers suitable interfaces to the sensors, actuators and to FAMOUSO. The developer implements the behavior of the *Processing* block only and integrate the desired application here.

The Simulink models in Figure 2 of our example scenario look similar on the highest level, but for the different profiles two various sub-models implement the interfaces of the AD592 and CON-TEMOD-I2C sensor (visible in the sub window in each figure). In Figure 2(a) we depicted the simulation profile. The actual state of profile switch is



(a) Simulation Environment (b) Code Generation Configuration

Figure 2. Profile switch between a simulation environment (a), and a code generation toolchain (b)

documented in the headline of the *ProfileSwitch* block. For simulation practice, there is only an emulation driver with the output unit "Degrees Celsius". On the right side, we use the profile *AVR BOARD* connected to the real sensors. Hence, the first block named S-Function represents the analog device driver for the Atmel processor. The device driver delivers a signal with a voltage, which have to be transformed in "Degrees Celsius" and linearized by a "Lookup_Table" block.

After the implementation of the behavior and validation by simulation, the target code generation process translates a Simulink model into the target languages. The process of code generation is controlled by a System Target File (STF). One STF represents a range of processors, typically a processor family. The Embedded Real-time Workshop [MathWorks 2010c] provides a number of these STFs. However, in most case, especially for 8-bit devices, there are basic structures only. Hence, we created our own STF for the AVR that is now available for future projects [Brade 2009].

5. Related Work

Our approach covers a broad spectrum of methods used in system development and a number of ongoing research topics. Hence, it is difficult to compare our work to other projects under a single perspective. Therefore, we evaluated several tools, standards etc. that include ideas related to our approach. In Tab. 1 we summarize the results according to the three abstraction categories of our approach. An empty field in the table means that this feature is not supported, a "+" marks some basic support while a "++" denotes the comprehensive integration of an aspect. The "✓" symbol validates the existence of the main feature categories.

In the first group of categories we determine the integration of a communication abstraction that offers a common interface and helps to hide specific properties of the

		CODES	SensorML	OMG STIS	IEEE-1451.2	LabView	Simulink	CANopen
Communication Abstraction		✓		✓	✓			
Middleware Integration		++		++	+			
Modular Concept		✓			✓	✓	✓	✓
Module	sensor interface	+			++	++		
	communication	++				+	+	+
	functionality				+	+	++	
Electronic Data Sheets		✓	✓	✓	✓	✓		✓
EDS	sensor		++		++	++		+
	platform		++					+
	communication	++	+					+
Description Languarge		XML	XML	XML	binary	binary		text
Functionality			✓			✓	✓	
Programming Interfaces		C	Math ML, Java			C, g, m, etc.	m, mex, etc.	
Code Generation		✓				✓	✓	
Code Generation Tool		XSLT					RTW	
Target Languarge		C				C	C	
Error prevention		+				+		

Table 1. Comparison of different development environments for distributed applications considering high level sensor descriptions

underlying network. Some of the referenced approaches provide or integrate an existing communication middleware for this purpose. The second group categorizes the existence of a modular structure, which combines predefined functions like sensor and communication interfaces, error detection modules, as well as application specific modules. The next category examines the usage of an abstract component description in electronic data sheets for different categories of sensor node components. Category four divides the references considering the implementation interfaces for behavior related functionality. The utilization of this information is depicted in category five. Here we mark the capabilities of a code generation and combination in the development process. The following tools and approaches were classified according to the four categories:

CODES (COsmic embedded DEvice Specifications) described in [Kaiser and Piontek 2006] represents a predecessor of some parts of our framework. The approach focuses on a XML based description language for the specification of sensor features and communication parameters for smart autonomous components. The communication abstractions of the underlying middleware COSMIC [Kaiser et al. 2005] are mapped to the electronic data sheet and allow a dynamic setup of the communi-

cation. The sensor descriptions follow some ideas of the Transducer Electronic Data Sheets (TEDs) according to unit coding, data types, and boundaries. The CODES data sheet is compressed available on each node. CODES supports an underlying middleware extensively, but it does not consider realistic sensors and whose parameters in the data sheet design.

Sensor Model Language (SensorML) provides a framework for describing sensor systems, as well as the associated data processing. In contrast to the following standards the user can define filter or fusion functions in the Mathematical Markup Language (MathML). The comprehensive concept is used to describe sensor, platform, and functionality. The description of process properties is based on Sensor Web Enablement Common namespace and provides the entire Sensor Web Enablement functionality. The communication interface would be described by an extension of OSI. The concatenation of separate processes enables processing, analysis, and visual fusion of multiple sensors. SensorML does not include appropriate tools for behavior development beside the MathML interface. For complex fusion applications an abstract description of algorithms is not possible in this way.

The OMG Smart Transducer Interface Specification (STIS) [Object Managment Group (OMG) 2003] provides an access via the CORBA real-time service (RS) interface, the diagnostic and management (DM) interface, and the configuration and planning (CP) interface of small, smart transducers in a distributed control system. The standardization of the different interfaces is mapped on an interface file system (IFS) typically in the memory of each Smart Transducer. For an interpretation of the data in the IFS additional meta data about the particular IFS are stored on a central node with higher performance. The authors of [Elmenreich et al. 2004] enhance the Standard Transducer Interface (STI) concept and developed a XML description of the functionality for simple fusion tasks. As mentioned one section before, OMG STIS divides interface programming and application development. The descriptions are used for message identification but not in the development process.

IEEE 1451 Smart Transducer is a family of standards for connecting smart devices [IEEE Standards Association 1997]. IEEE 1451.2 defines an electronic data sheet and a digital sensor interface to access sensor measurements, set actuators, control maintenance functions, or to obtain the data sheet of the sensor/actuator system. Hence, the standard establishes the communication between a Network Capable Application Processor (NCAP) and an actual sensor node called Smart Transducer Interface Modules (STIM). Those structure represents a mechanism that enables a flexible network interface via special NCAPs. The standards 1451.3 to 1451.5 enhance the interaction between STIMs and NCAPs to various protocols and interfaces. The description of the sensors, stored at each node contains a detailed specification of the sensor's vendor, firmware, and physics in a compressed TEDs [Char 1997]. Tools for an additional use of the electronic data sheets beside message identification and interpretation are not known yet.

Mathworks Simulink [MathWorks 2010a] and National Instruments LabVIEW [National Instruments 2009] are widely used toolchains for simulation, Hardware-in-the-Loop (HiL) scenarios and code generation. Therefore a broad variety of tool-boxes (e.g. control engineering, data acquisition, image processing, etc.) are available and helpful for rapid developments. Both tools offer interfaces for different programming languages

beside the standard graphical oriented development systems. Simulink does not support the utilization of meta information about data sources like sensors. The code generation process checks only data types of the used variables. LabVIEW integrates the concept of TEDs from IEEE 1451 and identifies connected sensor based on this information. The user has the possibility to scan the network during the development process and to call calibration functions. Data sheets can be located on the node or in an extended version as a virtual TED on a server application. Simulink and LabVIEW offer a large amount of development tools in particular for code generation. However, they do not allow an external description of the used sensors, processors etc. All parameter have to be defined directly in the model.

CANopen [CAN in Automation 2005] is a high-level protocol for CAN-bus [Robert Bosch GmbH 1991]. Every CANopen device is delivered with a vendor electronic data sheet. This electronic data sheet specifies communication, error and application profiles. The developer can define profiles to customize a CANopen device. These modifications are deposited by a device configuration file. The exchange of information occurs by a process data object and a service data object. Process data objects carry the real-time data with a look-up mechanism to encode units. In contrast, service data objects were used to configure the CANopen device. All configurations of a device were placed in the device object dictionary. Thus, the device object dictionary is the abstraction between application and communication.

From Tab. 1 we can conclude that none of the related approaches meets all requirements completely. Each of the presented tools, standards etc., covers an individual focus only and shows excellence in just this point. While engineering tools like Simulink and LabVIEW do not consider (or in very limited extent only) external knowledge about sensors and communication specification, they are very suitable for developing the respective control and filter algorithms. Additionally, incoming data structures have to be correctly interpreted by the developer. The standards for smart transducer interfaces define the communication, services, data types, etc., but they do not care about the relation between input and output values. Hence, we have to combine several approaches to meet our requirements addressed in Section 1.

6. Conclusions and Outlook

The intention of our framework is to enable a flexible combination of different software and hardware components during a development process. Furthermore we aim at the integration of domain specific tools in distributed control applications composed from networks of smart sensors and actuators. Therefore, we introduce abstractions of processors, sensors and actuators defined in XML descriptions and connect them with widely used development tools. This enables the developer to configure sensor and network interfaces on a high, declarative level. As a result applications can be developed independently. The developer may also choose the favourite domain specific language.

In the future we will test and refine the framework in a distributed robotic scenario. Additionally, we want to develop an appropriate way to derive the *System Target File* automatically for code generation using a XML definition file too.

Acknowledgement

This work has partly been supported by the Ministry of Education and Science (BMBF) within the project "Virtual and Augmented Reality for Highly Safety and Reliable Embedded Systems" (VierForES).

References

Apparao, V., Byrne, S., Champion, M., and Isaacs, S. (1998). Document object model (DOM) technical reports : Level 1. Candidate recommendation, W3C.

Brade, T. (2009). Codegeneration aus Simulink / Embedded Real Time Workshop Modellen am Beispiel eines AVR Targets, Student Research Project, Otto-von-Guericke Universität Magdeburg.

CAN in Automation, editor (2005). *CiA 306 DS V1.3: Electronic Data Sheet Specification for CANopen*. CiA, CANopen.

Char (1997). *IEEE Std 1451.2-1997, IEEE Standard for a Smart Transducer Interface for Sensors and Actuators*.

Dietrich, A., Zug, S., and Kaiser, J. (2010). Detecting external measurement disturbances based on statistical analysis for smart sensors. In *Procedings of the IEEE International Symposium on Industrial Electronics (ISIE)*.

Elmenreich, W., Pitzek, S., and Schlager, M. (2004). Modeling distributed embedded applications on an interface file system. In *Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 175–182.

Herms, A., Schulze, M., Kaiser, J., and Nett, E. (2008). Exploiting publish/subscribe communication in wireless mesh networks for industrial scenarios. In *Proceedings of Emerging Technologies in Factory Automation (ETFA '08)*, pages 648–655, Hamburg, Germany.

IEEE Standards Association (1997). *IEEE Standard for a Smart Transducer Interface for Sensors and Actuators (IEEE 1451.2)*.

Kaiser, J., Brudna, C., and Mitidieri, C. (2005). COSMIC: A real-time event-based middleware for the CAN-bus. *Journal of Systems and Software*, 77(1):27–36. Special issue: Parallel and distributed real-time systems.

Kaiser, J. and Piontek, H. (2006). CODES: Supporting the development process in a publish/subscribe system. In *Proceedings of the fourth Workshop on Intelligent Solutions in Embedded Systems WISES 06*, pages 1–12, Vienna. ISBN: 3-902463-06-6.

Kaiser, J., Zug, S., Schulze, M., and Piontek, H. (2008). Exploiting self-descriptions for checking interoperations between embedded components. In *International Workshop on Dependable Network Computing and Mobile Systems (DNCMS 08)*, pages 41–45, Napoli, Italy.

MathWorks, T. (2010a). Matlab/Simulink - Website.

MathWorks, T. (2010b). Real Time Workshop - User's Guide.

MathWorks, T. (2010c). Real Time Workshop Embedded - User's Guide.

National Instruments (2009). LabVIEW 2009 - Herstellerseite.

Object Managment Group (OMG) (2003). *Smart Transducer INterface Specification*.

Robert Bosch GmbH (1991). *CAN Specification Version 2.0*. Robert Bosch GmbH.

Schulze, M. (2009). FAMOUSO – Eine adaptierbare Publish/ Subscribe Middleware für ressourcenbeschränkte Systeme. *Electronic Communications of the EASST (ISSN: 1863-2122)*, 17.

Schulze, M. and Zug, S. (2008). Exploiting the FAMOUSO Middleware in Multi-Robot Application Development with Matlab/Simulink. In *Proceedings of the 9th International Middleware Conference (Middleware2008) ACM/IFIP/USENIX*, Leuven, Belgium.

ZigBee Alliance (2003). *ZigBee Specification - IEEE 802.15.4*.

Zug, S. and Kaiser, J. (2009). An approach towards smart fault-tolerant sensors. In *Procedings of the International Workshop on Robotics and Sensors Environments (ROSE 2009)*, Lecco, Italy.

Software Architecture for Mobile Interaction in Intelligent Environments

Reiner F. Perozzo[1], Carlos E. Pereira[1]

[1]Departament of Electrical Engeneering – Federal University of Rio Grande do Sul (UFRGS)
Av. Osvaldo Aranha, 103 – 90035-190 – Porto Alegre – RS – Brazil

reiner.perozzo@ufrgs.br, cpereira@ece.ufrgs.br

Abstract. *Since the latest years, Intelligent Environments (IE) has been becoming a very discussed and recurrent subject. Part of such success is given by the meaningful diffusion and offer of portable electronic devices, with great power of computation, low energy consumption and, mainly, high level of connectivity. This paper proposes a software architecture for mobile interaction in IE. The proposed architecture has three main characteristics: (i) discovery and remote composition of available services in the IE; (ii) adaptation of services and functionalities according to the user's profile; (iii) flexibility in the insertion of new home automation devices, adding, dynamically, services to the IE.*

1. Introduction

With the advance in home automation and the technologies that are more present in the Intelligent Environments (IE), the ubiquitous computation starts becoming reality, defined by Mark Weiser [Weiser 1991] and that includes a technological level where computational systems provide information and services to people, anywhere and anytime. Ubiquitous and pervasive computation can be seen as a group of characteristics and functionalities that composes an IE [Anastasopoulos 2005].

In the IE there is the vision of a world surrounded by a big amount of devices that offer intelligent assistance on users daily activities. The IE consists of a technological information paradigm in which the computerized objects are introduced in a specific physical environment that adapts itself to the users' different necessities and situations [Kirste 2005], [Arts 2002] having autonomy to act [Lindwer 2003] and with possibilities to be programmed to recognize and learn the user's behavior who lives inside that environment [Yang 2004], [Hagras 2004].

The home automation is inserted in the context of IE, with projects that need automated physical spaces, including sensors, actuators and, mainly, intelligent systems for tasks management and optimization [Nazari 2007], [Edwards 2006]. A large variety of services in fields like security (access control, users' identification), comfort (temperature and humidity control, illumination) and entertainment are emerging daily, indicating problems to be solved, as the electric energy consumption in these environments, that can be minimized through the utilization of intelligent computational systems that concern energy optimization and management. Other problems are related

to mobility, adaptability and heterogeneity aspects in these environments, due to the large number of solutions, both in hardware and in software.

A great variety of intelligent devices networks may be found inside an IE, that admit the integration of electronic devices and people, providing information, communication, services and entertainment [Arts 2004]. The utilization of communication networks in these environments is one of the primordial points for projects execution, because they can be related to both the information exchange among electrical appliances – through power line communication (PLC) – and with wireless access to this same information, through mobile computation devices – such as Personal Digital Assistants (PDAs) and smartphones. This way, there is a necessity of creating strategies that consider the most different points, like: the users' mobility, the discovery and dynamic composition of new services and the flexibility with the automation devices that are inserted in the IE.

2. Related Works

2.1. Architectures and Middlewares

The μJini architecture [Lee 2006] was proposed with the objective of offering a discovery of services that is aware of the context for mobile devices with limited resources. The main point of this architecture is defined by a proxy in charge of controlling discovery processes and services delivery. Inside the μJini proxy system are the service discovery components that supply the aware discovery to the context for μJini proxy, which utilizes three approaches: (i) having a service code executed completely in the client computational platform, with MIDLets Java 2 Micro Edition (J2ME) or with Java 2 Standard Edition (J2SE) services, through a network that sends up-to-date screens to the client. The decision about which way to utilize is clearly made to the users. μJini proxy is able to select the best adaptation, according to the service context.

Basically, the μJini architecture was projected to overcome limitations related to mobile devices, communication networks and services. The proposal offers a solution that comprehends the request of mobile services and their distribution to the clients' devices. Although, some challenges still proceed, such as the dynamic composition of new services and the limitation about the applications requiring a Java virtual machine (JVM) available in the client mobile device.

The Mobile Context Explorer proposal – or, simply, MoCE [Kang 2006] – as it is known the architecture of a middleware that is aware about the context, was projected to find, gather and provide context to services in a mobile environment. Besides, a context discovery protocol was developed in order to discover contexts efficiently in unknown contexts providers, both for mobile devices and for embedded devices based on Wi-Fi networks. The MoCE architecture was specifically defined in order to offer support to services based on context, and this one is shared by the data communication network. Making reference to context sharing, there is the necessity of a mediating architecture to interface and control the context request and offer among different devices. Thus, some components in the architecture were defined in order to identify and support context consumers.

In respect to data access in the IE, an architecture designated by Ubiquitous Data Access (UbiData) [Helal 2004] was specified with the purpose of concerning some challenges originated by the proliferation of devices and mobile applications, as well as additional requirements of clear access to ubiquitous data. Among the challenges discussed by UbiData architecture, are in relief: (i) access to data anywhere and anytime; (ii) access to data independently on the device, which is possible for the user to alternate among different mobile devices with different access capabilities; (iii) data access and update independently on the application, enabling the users to modify documents and files.

Another important project is Gator Tech [Helal 2005] that considers the existence of execution environments and software libraries for the development of programmable IE. Services discovery and gateways protocols automatically integrate systems components, using a generic middleware that supports a service definition to each sensor and actuator in the environment. There are mechanisms in the Gator Tech which consist from intelligent e-mails treatment – with perception and notification to users – up to points like the use of Radio-Frequency Identification (RFID) to access and identify users inside the environment, intelligent bathrooms, residential security and monitoring.

2.2. Services Discovery and Composition

The Open Services Gateway initiative/Business Process Execution Language (OSGi/BPEL) is the proposal of a framework [Redondo 2007] that suggests to increase the services OSGi composition support present in IE. Due to the fact that, usually, there are lots of services in these environments, such as energy control and optimization, security, illumination control, alarm system for fire and others, composed services are the ones that activate a group of OSGi services, where each one is led to a specific activity.

In the OSGI/BPEL, the idea is to encapsulate logical information of services composition inside an OSGi application, registering it in the framework. When the composed service is required, an execution engine interprets the composition description to require and manage the processes under execution. The proposal tries to be clear both for required composition services and for OSGi registration service. That is why virtual bundles are defined, in charge of specifying composed services, which use the BPEL for composition in the OSGi framework.

2.3. Mobile Interaction

The possibility of controlling a smart home using the TV appears from a proposal [Cabrer 2006] which integrates IE, digital TV and domestic networks. The adopted technologies in the proposal are the Multimedia Home Platform (MHP), for Digital TV and the OSGi as a platform for residential gateways settings. According to the proposal, the main point is to permit the user to control the whole IE assisted by the TV. This way, MHP and OSGi are utilized – the later is oriented to services and the former is oriented to functions. As both technologies have quite different architectures, there is integration between them, and the problem is solved by creating an Xbundlet – an application that permits the natural interaction between MHP and OSGi. A Xbundlet not only defines a communication link between these platforms, but also constitutes a hybrid software element that can be executed in both architectures.

As regards to the use of mobile devices in IE, there is a proposal [Helal and Mann 2002] intending to increase life quality and reduce old people dependence or limitation, somehow, at home. One of these research activities concerns, precisely, the utilization of smartphones as a tool to offer support to elderly in the IE. Smartphones become something like a "magic wand" that provides, among other things, functionalities of a universal remote control for all kinds of automation devices present in the intelligent environment.

The idea of using smartphones appears as an offer to several services for elderly, by not moving in the ambient, or asking favors to another person in the same place. According to researchers, the proposal greatest challenge is to have an open integration of the different involved technologies, neither endangering nor limiting an architecture.

2.4. Users Profile

From the attractions offered by ubiquitous computation – in which are available the information mobility and the networked devices, some efforts come to sight with the objective of concerning security mechanisms and techniques in IE. One of the choices for this kind of security is the profile management, proposed by a middleware [Loeser 2005] which is able to integrate data bases of several profiles, with generic mechanisms of authentication that permit the management in the OSGi framework. Users' profiles are utilized to have information about their basic preferences and abilities, whereas devices profiles provide the main characteristics and describe the present situations of each device. There are, also, the definitions of the safe access kinds in each environment, having confidential information of identification that must be preserved. Because of this reason, there are several levels of security for different services and users, defining a distributed architecture, in which the communication component may utilize TCP/IP on technologies with Wireless Local Area Network (WLAN) and Bluetooth. On the other hand, the security guarantee in distributed systems is given by a communication based on Secure Sockets Layer (SSL) / Transport Layer Security (TLS) with certificates X.509.

Likewise, another proposal concerns a framework [Groppe and Mueller 2005] for profile management in IE, customizing this environment according to each situation, with the users' preferences and the devices capability. Profiles customization methods are used based on evaluation and processing methods to expand, automatically, the users' preferences.

Profiles management applied to IE is motivated because these ones integrate a great variety of embedded computation devices, mobile, and the capability of communication, which offers agility and comfort to the users. That is why there is the proposal of a profile processing method that intends to adapt, automatically, the environment according to users' necessities and preferences. Possible conflicts and decision strategies in profile processing are investigated, as well as mechanisms for profile evaluation that assimilate modification in an environment. It is also analyzed a methodology by identifying the project, main points and involved requirements.

3. Proposed Architecture

In this paper, it is proposed an architecture that offers mobility in the remote interaction between the user and the IE, as well as the services remote management.

The architecture has three main characteristics: (i) discovery and remote composition of services available in the IE; (ii) functionalities and services adaptation according to the user's profile, utilizing security policies with different levels of accessing the system; (iii) flexibility in the insertion of home automation new devices, in which services are dynamically added to the IE.

3.1. Conceptual Model

The proposed architecture is presented in Figure 1 and is divided among five layers: (i) physical devices, (ii) logic devices, (iii) services layer, (iv) services composition layer and (v) management layer.

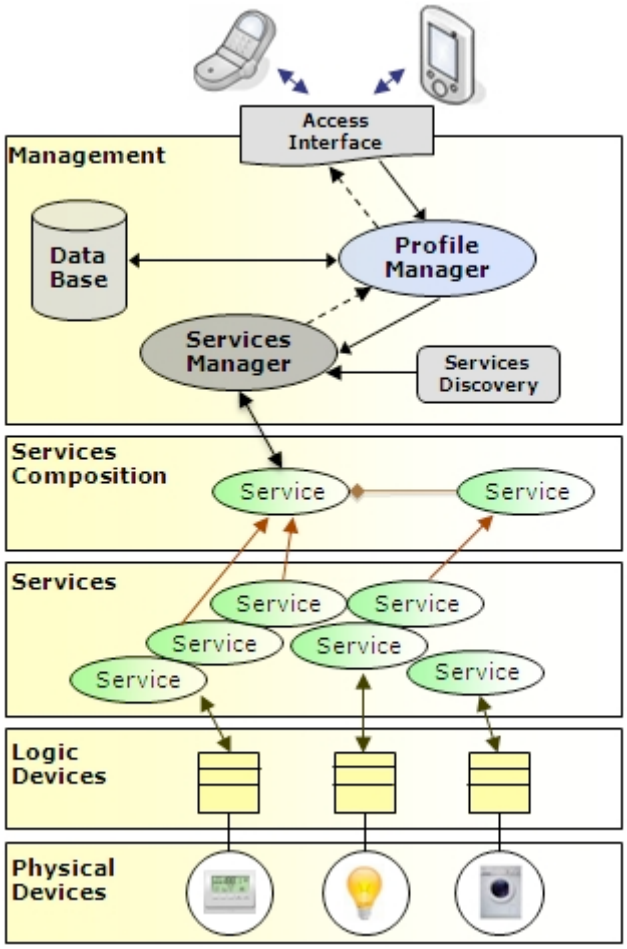


Figure 1. Proposed Architecture

The layers in the architecture are defined with the purpose of offering modularity to the system, in which new components may be inserted in the lower layers and available to the upper ones. This way, they are specified below:

Physical Devices Layer: It is composed by electro-electronic devices in home sceneries. Inserted in this layer are devices found in the real world, such as: lamps, televisions, set-top-boxes (STP), refrigerators, air-conditioners, thermostats, sensors, actuators and controllers, in general terms.

Logic Devices Layer: It is composed by the computational representation of the devices that are inserted in the physical layer, which are mapped by the logical layer

through devices classes that implement a known communication interface. An advantage of this layer is the possibility of defining services in the upper layer, even if the physical layer is not aware of that.

Services Layer: It is composed by a group of services available in the IE, which may be requested and executed by the users. In this layer, there are the functionalities that an IE might offer to its users, like: conditioning the ambient, turning off the lights, verifying the state of any sensor, shutting the door, and so forth. Each service is associated to a logic device and, once implemented, it might be reused or adapted in the interaction with other devices.

Services Composition Layer: In this layer, it is possible to create composed systems, which integrate other services with the objective of executing a specific task. An example of composition could be applied to a situation in which the user wants to set the illumination level of the living room, according to a source of natural light got at the moment. In that case, the composed service would be the association of a sensor reading service with an illumination adjustment service. One of the composition advantages is the possibility of reusing services dynamically, offering flexibility in the execution of services that depend on others.

Management Layer: This layer is in charge of managing the IE, comprehending components that concern form the discovery and availability of services to users, up to the definition of what kinds of services a user may access. The five components present in this layer are specified as follows below:

Access Interface: Permits users to interact with the IE utilizing their mobile devices. The main point is that, facing an IE, the user is able to get information about what kinds of services that environment offers and which of them are available to him.

Data Base: It is responsible for storing the users' data, registering details reports, levels of safe access and profiles that do not really need to be stored at the moment.

Profile Manager: It is responsible for receiving the users' requests and deciding which will be the access level of each user according to his/her profile and a detail report that, occasionally, has been registered. A profile may be given by inserting a private or common password, under the domain of the desirable service.

Services Discovery: This component, in system execution time, searches for all services that are available at that moment in the services layer and, hence, to the IE.

Services Manager: Perceiving all the available services through the received information in the Services Discovery, this component provides users, by means of the Profile Manager, all the services that are available for execution. The Services Manager is in charge of executing single or composed services, according to each request and the user's profile. After passing through the Profile Manager, the list of available services in the IE is provided in the Access Interface to that particular user.

3.2. Architecture Implementation

With the definition of a conceptual model, the architecture is implemented and validated inside an IE scenery, which consists of a seminaries room that has a home automation system developed built by a company [Homesystems 2009] which develops researches

projects applied in cooperation with the Control, Automation and Robotics Group of the Federal University of Rio Grande do Sul. In this automation system, there is a central controller unit called Systembox, that is, basically, a computer with Linux operational system, responsible for controlling Homesystems Network (HSNET – proprietor protocol that works in the physical layer RS-485) and executing commands for a group of devices that can be utilized in illumination systems, air conditioners and security systems.

This way, in the physical layer of the proposed architecture, are inserted a group of lamps, the air conditioner, a luminosity sensor and the central controller (Systembox). In the logic layer, the classes are implemented, in Java language, that know the Systembox communication interface. The exchange of command messages between the logic devices (classes) and the physical devices (Systembox, lamps and air conditioner) is carried out through HyperText Transfer Protocol (HTTP), implemented on an Ethernet network and known both for the classes and for the Systembox.

On the services layer, the OSGi framework and the Knopflerfish implementation [Knopflerfish 2009] are used. OSGi has an architecture oriented to services and based on Java, providing the standardization of primitives that permit building applications with small components that are reusable. In this layer, execution components (bundles) are created, and these ones utilize and register services in the OSGi Service Registry. Thus, it is implemented three bundles that interact with the logic devices classes: the first bundle implements two services – turning the environment lights on and off; the second one verifies the light intensity level in this same environment; and the third one sets the air conditioner temperature.

On the management layer, it is used the MySQL as data base for storing the user's profiles and the system utilization detailed report. The components profile manager, services manager, services discovery and access interface are implemented into Java language, and the later is developed in Java Server Page (JSP). Java technology was chosen because of the platform portability that it offers. This layer still counts on Tomcat, which is a Java applications server for the web that provides the access interface between the mobile devices and the services offered by the IE.

In the considered scenery to the case study, by entering the seminaries room with his/her mobile device, the user solicits the manager layer what services are offered by this environment and which of them are available to him/her. Thus, the profile manager verifies, in the data base, the access level that will be allowed. After that, the Services Discovery component researches inside the OSGi framework and finds the registered services (turning the lights on/off, verifying the illumination intensity and setting the air conditioner temperature). At this moment, the Services Manager creates a list of available services and sends it to the Profile Manager, in order to provide it to the user. In Figure 2, it is shown the communication scenery utilized in the case study.

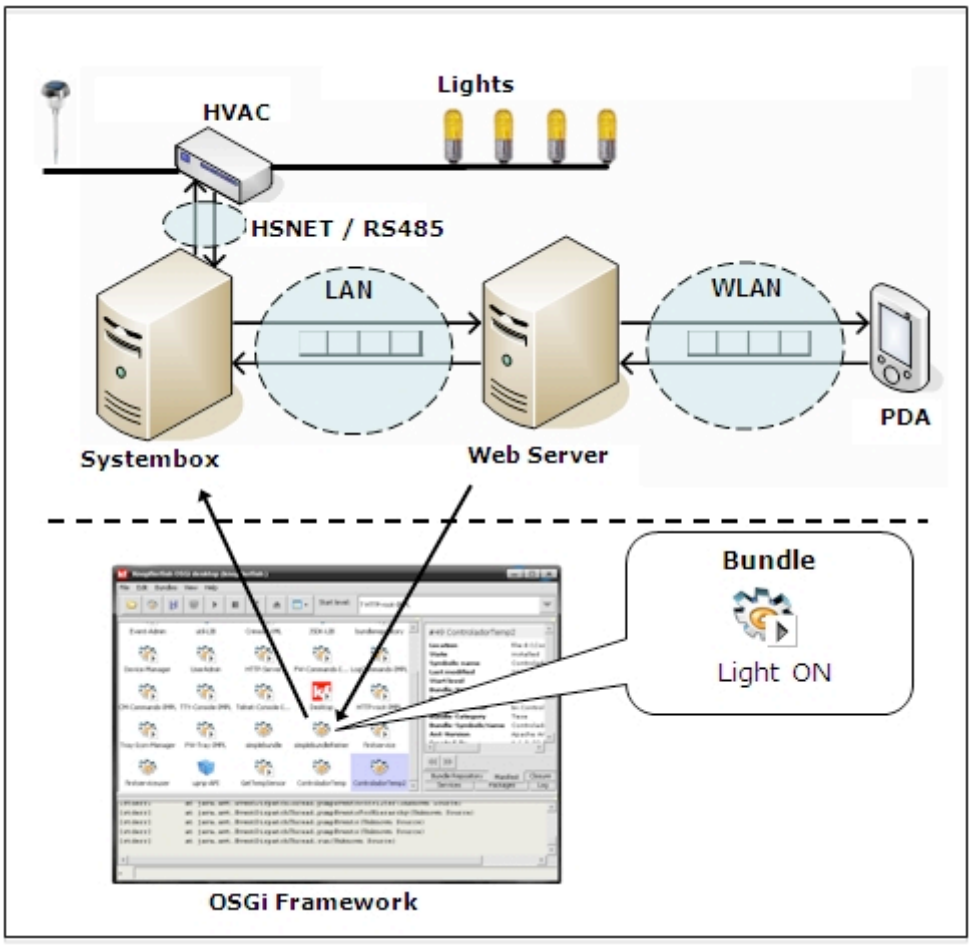


Figure 2. IE Scenery

4. Conclusion

In this paper, it has been proposed a software architecture for mobile interaction in IE. In the architecture, it is presented components which concern form the discovery and composition of services up to the user's profiles management in his/her interaction with the IE. One of the main advantages presented is the possibility offered to the user to create different sceneries in environments with residential automation devices, without the necessity of being acquainted with them previously.

Another advantage is that, even being indispensable an JVM for performing most of the components in the architecture, it is not necessary to have it in the mobile devices, because the interface with the user happens according to JSP pages which are interpreted in the server. This way, the user's mobile device must have a web navigator to access the available services in the IE, making the architecture flexible according to the several computational resources found in mobile devices. At last, it is opportune to mention that all the architecture components were implemented and are perfectly working in the automated seminaries room. However, new strategies and technologies for services discovery and composition continue being investigated.

References

Anastasopoulos, M., et al. (2005) “Towards a Reference Middleware Architecture for Ambient Intelligent Systems”, In: Proceedings of the Workshop for Building Software for Pervasive Computing..

Arts, E. (2004) “Ambient intelligence: a multimedia perspective”. In: IEEE. Vol. 11. Issue 1, Jan-Mar, pp. 12–19.

Arts, E., et al. (2002) “Ambient intelligence”. In: McGraw-Hill, Inc., New York, NY, USA, pp. 235–250.

Cabrer, M. R., et al. (2006) “Controlling the Smart Home form TV” In: Proceedings of the International Conference on Consumer Electronics, IEEE, Las Vegas p. 421-429.

Edwards, W. K. (2006) “Discovery systems in ubiquitous computing”. In: Proceedings of the Pervasive Computing, IEEE. Vol. 5, Issue 2, April-June, pp. 70-77.

Groppe, J. and Mueller, W. (2005) “Profile Management Technology for Smart Customizations in Private Home Applications”. In: Proceedings of the International Workshop on Database and Expert Systems Applications, IEEE, Copenhagen, p. 226-230.

Hagras H., et al. (2004) “Creating an ambient-intelligence environment using embedded agents”. In: Proceedings of the Intelligent Systems, IEEE. Vol. 19, Issue 6, Nov-Dec, pp. 12-20.

Helal, A. and Hammer, J. (2004) “UbiData: Requirements and Architecture for Ubiquitous Data Access”. In: SIGMOD Rec Journal… New York: ACM, p. 71–76.

Helal, S. et al. (2005) “The Gator Tech Smart House: A Programmable Pervasive Space”. In: IEEE Pervasive Computing, p 64-74.

Helal, S., Mann, W. (2002) “Smart Phones for the Elders: Boosting the Intelligence of Smart Homes”. In: Proceedings of the Workshop Automation as Caregiver: The Role of Intelligent Technology in Elder Care, Edmonton, AAAI Press, p. 74-79.

Homesystems (2009) Intelligent Environments. <http://www.homesystems.com.br>.

Kang, B. S., et al. (2006) “Context-aware Middleware Architecture for Intelligent Service in Mobile Environment”, Proceedings of the International conference on Computer and Information Technology, IEEE, p. 240–244.

Kirste, T. (2005) “Smart environments and self-organizing appliance ensembles”. In: Aarts E, Encarnação J. L. (eds): True Visions, Springer.

Knopflerfish (2009) “Open Source OSGi”. <http://www.knopflerfish.org>.

Lee, C., et al. (2006) “The μJini Proxy Architecture for Impromptu Mobile Services”. In: Proceedings of the International Symposium on Applications and The Internet Workshops, IEEE, 2006, Phoenix, p. 23–27.

Lindwer, M. et al. (2003) “Ambient intelligence visions and achievements: linking abstract ideas to real-world concepts”. In: Design, Automation and Test in Europe Conf and Exhib., pp. 10–15.

Loeser, C., et al. (2005) “Secure Profile Management in Smart Home Networks”. In: Proceedings of the International Workshop on Database and Expert Systems Applications, IEEE, Copenhagen, p. 209-213.

Nazari, A. S., et al. (2007) ”3DSim: Rapid Prototyping Ambient Intelligence”. <http://www.igd.fhg.de/igd-a1/projects/amilab/index.html>.

Redondo, R. P. D. et al (2007) “Enhancing Residential Gateways: OSGi Services Composition”. In: Proceedings of the International Conference on Consumer Electronics, IEEE, Las Vegas, p. 1-2.

Weiser, M. (1991) “The Computer for the 21st Century”. In: Scientific American 265 n.3, p. 94-104.

Yang, H., et al. (2004) “An evolutionary system development approach in a pervasive computing environment”. In: Proceedings of the International conference on Cyberworlds, pp. 194-199, 2004.

12th Brazilian Workshop on Real-Time and Embedded Systems

♦

Technical Session 2

Scheduling

smartenum: A Branch-and-Bound Algorithm for Optimum Frequency Set Establishment in Real-Time DVFS

E. B. Valentin[1], **R. S. Barreto**[1]

[1]Department of Computer Science
Federal University of Amazonas
Manaus-AM, Brazil

{ebv,rbarreto}@dcc.ufam.edu.br

Abstract. *This paper describes an offline branch-and-bound algorithm to establish the optimum frequency set to execute real-time tasks taking into account worst-case scenarios on the DVFS technique. The real-time tasks are scheduled by a dynamic fixed priority scheduler, such as Rate-Monotonic. The task model also considers mutual exclusion relations adopting the Priority Ceiling Protocol. The schedulability analysis is carried out by response time technique, which had to be adjusted to consider several frequencies. Two case studies are detailed. Results have shown a reduction of* 91% *and* 78% *in the number of evaluated configurations. In addition, experimental results pointed out energy reductions of* 38.79%*, and* 30.46%*.*

1. Introduction

Modern processors design provides the possibility to change operating frequency dynamically at runtime. Therefore, clock speed and corresponding voltage may be dynamically controlled to the lowest available level while meeting task's timing constraints. This is the key idea behind a technique known as *Dynamic Voltage and Frequency Scaling* - DVFS. When considering processor circuitry, there is a quadratic relationship between energy consumption and voltage. This relation is described on Equation (1):

$$E = C_l \times N_{cycle} \times V_{dd}^2 \tag{1}$$

where E is energy, C_l is circuitry capacitance, N_{cycle} is number of cycles and V_{dd} is the voltage [Gutnik 1996].

In some situations, you can lower the supply voltage in such a way to take advantage from this quadratic relation between voltage and energy consumption. However, lowering the supply voltage also reduces linearly the clock speed. For instance, consider a task with 25ms deadline executing in a 50MHz processor under 5V. If this task requires 5×10^5 execution cycles, the processor finishes its execution within 10ms and stays idle for the remaining 15ms. However, if user scales processor speed and voltage down to 20MHz and 2V, the processor finishes this task exactly at its deadline, 25ms, resulting in a 84% energy consumption reduction [Shin et al. 2001].

Although this technique can yield meaningful energy consumption reduction, its usage requires care, especially when timing constraints are considered. This problem may be trivial if each task is taken into account isolated without any concern of other tasks interference. Once it is known the task's deadline and the required worst-case execution cycles, the best execution frequency can be selected in order to achieve lowest energy consumption and still reach the deadline. On the other hand, when a system with more than one task is considered, this problem becomes more complicated.

An interesting problem arises when executing such systems in DVFS-enabled processors: *"In which frequency each task must be executed, so that the whole system reaches minimum energy consumption from processor circuitry and all tasks meet their deadlines?"* In this case, for each frequency combination, a new schedulability analysis is required, because a single change into one task execution time will reflect the whole system, because interference between every tasks will suffer modification. If the system has N real-time tasks and the processor has Γ possible frequencies, then there are Γ^N possible frequency combinations.

The proposed method is an offline branch-and-bound algorithm to establish the optimum frequency set to execute real-time tasks taking into account worst-case scenarios on the DVFS technique. This is the main contribution of this paper. So far, the algorithm only considers pruning unschedulable nodes. Real-time tasks are scheduled by a dynamic fixed priority scheduler, such as Rate-Monotonic or Deadline-Monotonic. The task model also considers mutual exclusion relations adopting the priority ceiling protocol (PCP). Thus, each task can be interfered by factors like high priority tasks and shared resources being held by low priority tasks. A schedulability analysis is performed in order to check if these interferences can lead the system to an undesirable state in which a real-time task would miss its deadline.

This work is part of a larger project that aims to propose a framework to help designers to develop embedded multimedia applications with low energy consumption for wireless portable devices. Although this work considers worst-case execution scenarios, it is worth mentioning that this is only the starting point since there is ongoing projects that takes into account the actual execution scenarios that exploits slack time.

This text is structured as follows: related works are reviewed in Section 2, the problem is modeled in Section 3, the proposed algorithm is presented in Section 4, experimental results are then analyzed in Section 6 and conclusions and future works are discussed in Section 7.

2. Related Works

Problems related to energy consumption have been solved by using several known techniques. Havinga [Havinga 1997] has produced a survey about most of them. Dynamic voltage and frequency scaling, idle and sleep operating modes, dynamic power management (DPM), clock regions, co-processors for specific application type, and operating system tuning are some of discussed techniques. It is worth mentioning that

while implementing those techniques, usually, application's timing constraints are not taken into account.

Zhao and Aydin [Zhao and Aydin 2009] proposed a combined approach between DVFS and DPM techniques to reduce energy consumption on real-time systems. Although a worthy work, they propose deal with only one task leading to a non-preemptive solution. An important contribution in the power-aware scheduling of real-time tasks was done by [Mejia-alvarez et al. 2004]. They propose a solution based on knapsack problem considering system utilization factor. Nevine in [AbouGhazaleh et al. 2003] proposes a colaborative solution which involves scheduler and compiler utilizing intra-task DVFS. However, both works do not take shared resource into account. Choi in [Choi and Pedram 2005] presents an intra-process approach to making use of runtime information about the external memory access statistics in order to perform CPU voltage and frequency scaling with the goal of minimizing the energy consumption. But in this case, it does not consider real-time tasks. An inter-task DVFS approach is done by Yao in [Yao et al. 1995]. This work is based on time slices. It is proposed a mechanism to determine the optimum execution frequency for N tasks in each time slice. It is also proposed a modification on the Earliest Deadline First (EDF) algorithm in order to cover the proposed method. Another strategy to explore DVFS usage is to insert frequency scaling instructions in specific points of tasks code. Azevedo in [Azevedo et al. 2002] proposes an intra-task DVFS based to determine these points and which frequencies are required. The points are chosen based on code profiling and simulations. Its simulation results have shown an energy reduction of about 60%. However, the code profiling is hard to produce and usually application dependent. Shin in [Shin et al. 2001] presents an intra-task DVFS solution based on static analysis. Task code is evaluated and analyzed in order to produce a control flow graph. Each node represents a basic block and contains information about the number of work case execution cycles. This graph is then utilized to determine which points in the code are eligible to change frequency, based on the number of not executed cycles. That solution does not consider preemption in the system. Pillai and Shin [Pillai and Shin 2001] present a class of novel algorithms called real-time DVS (RT-DVS) that modify the OS's real-time scheduler and task management service to provide significant energy savings while maintaining real-time deadline guarantees. Shared resources are not considered in that work.

3. Problem Definition and Modeling

Consider a processor $\mathcal{P}$ with DVFS feature available for Γ frequencies. The set of all possible frequencies in $\mathcal{P}$ is $F = \{f_i \mid f_i \text{ is an available frequency in Hz for } \mathcal{P} \text{ and } 1 \leq i \leq \Gamma\}$. Consider also a model $\mathcal{M}$ with N tasks. In $\mathcal{M}$, the system is executed under a dynamic fixed priority scheduling.

Consider now the set β consisting of N-tuples. Let $K \in \beta$, $K = \langle k_1, k_2, \ldots, k_N \rangle$, where $k_i \in F$. K represents a possible configuration of frequencies assignment to T_i tasks in $\mathcal{M}$, where T_i is executed with frequency k_i. Each ele-

ment $K \in \beta$ must be interpreted as an arrangement of the possible assignments that can be created using all available frequencies in $\mathcal{P}$. Thus, elements in β are possible configurations for execution of $\mathcal{M}$. There are Γ^N possible configurations K's in β.

Due to shared resource there is the need to consider mutual exclusion relations in tasks of $\mathcal{M}$. $T_i \in \mathcal{M}$ has the following properties: D_i is its deadline; P_i is its period of execution; $WCEC_i$ is its worst-case execution cycles; $C_i(f)$ is T_i's execution time which is function of frequency f. $C_i(f) = \frac{WCEC_i}{f}$; p_i is T_i's priority; J_i is the *release jitter* for task T_i, which indicates the worst release case for T_i; $I_i(K)$ is the interference suffered by T_i, which is function of frequency set K. $I_i(K)$ corresponds a time window in which there is continuous execution of tasks with greater or equal priorities to T_i's; $R_i(K)$ is T_i's response time as function of frequency set K. $R_i(K) = I_i(K) + J_i$. $I_i(K)$ is result of a schedulability test. The schedulability test is based on response time. It can be calculated using Equation (2).

$$I_i^{n+1}(K) = C_i(k_i) + B_i + \sum_{j \in hp(i)} \left\{ \left\lceil \frac{I_i^n(K) + J_j}{P_j} \right\rceil \times [C_j(k_j) + \sigma] \right\} \tag{2}$$

In Eq.(2), $hp(i)$ is the set of tasks which has higher priorities than T_i. $\frac{I_i^n(K)+J_j}{P_j}$ represents the number of occurences of T_j over I_i. σ is the overhead caused by the frequency and voltage switch process. B_i represents the time spent in shared resources locks. In order to deal with priority inversion problem, B_i is calculated considering the Priority Ceiling Protocol (PCP) [Sha et al. 1990]. Sha *et al.* [Sha et al. 1990] does not treat with different available frequencies. It is worth noting that Eq.(2) has been rewritten considering the selected frequency for execution of each task T_i. From this definition, it is noticeable the relevance of properly selecting a frequency to a task. Because selecting an ordinary frequency f to a specific task T_i, it will distinguish T_i's execution time $C_i(k_i)$ and consequently its response time $R_i(K)$. In this sense, a task is always completelly defined as function of its selected frequency. Another property derived from the frequency choice is the task idle time. T_i's idle time is $D_i - C_i(K)$. When the target is to maximize processor utilization time, the selected frequency is considered optimum when T_i's idle time is the minimum. However, there is an intrinsic condition on this problem, because all tasks must meet their timing constraints, in this case D_i. Hence,

$$\forall\, i < N,\ R_i(K) \leq D_i \tag{3}$$

Elements in β must be evaluated by means of response time schedulability test. In this case, $K \in \beta$ is considered a feasible configuration if, and only if, it does not violate the condition (3). The problem consists in finding the element K in β which leads the system to a configuration where constraints (3) are satisfied and the lowest possible consumption in $\mathcal{P}$ is achieved.

The objective is then to determine $M \in \beta$ so that:

$$Minimize\{idle(M) = \sum_{i=1}^{N} (D_i - C_i(k_i))\} \quad (4)$$

in such a way that, using M, $R_i(M)$ respects condition (3). $idle(M)$ is referred in this paper as *idle time* of frequency configuration M. Thus, $idle(M)$ represents the maximum utilization of idle times in each task.

4. *Branch-and-Bound* Algorithm to Establish Optimum Frequency Set

The proposed algorithm, here called *smartenum*, generates a search tree for all possible arrangements of tasks in $\mathcal{M}$ and frequencies in F. The idea behind constructing the search tree is to put each task in a specific level of the search tree and, associated with this task, to determine its best frequency. This way, the search tree will be limited up to N levels. In the proposed algorithm the search tree is traversed using a depth first search method. The stop condition is the impossibility of satisfying condition (3). The algorithm performs two initial prunings, throwing away frequencies that alone produces computation time which violates (3) and establishing a starting point for the search. These prunings correspond to a upper local limit and a lower limit for the combinatorial space. Figure 1 shows its diagram.

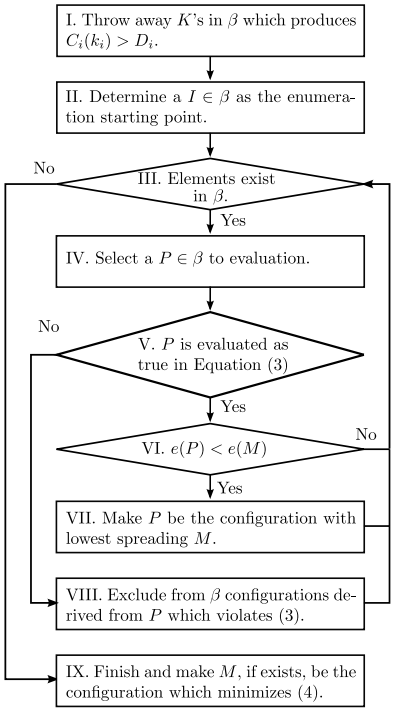


Figure 1. *smartenum* Algorithm Diagram

The proposed algorithm in Figure 1 has two operation types: enumeration and pruning. Enumeration operation consists of listing and evaluating elements $K \in \beta$. The evaluation is the schedulability test in order to check constraint (3) for that configuration K. Pruning operation consists of eliminating elements in order to reduce the number of elements to verify during the enumeration operation.

All available frequencies are evaluated for each task in a decreasing order, so that the greater computation times are considered only as a last resource. This premise allows to throw away lower frequencies, in case an ordinary frequency has been found as useless to one specific task. This evaluation method compounds the search tree illustrated in Figure 2. Thus, all frequencies related to task T_i are at level i of search tree. And every path starting from level 1 and ending at level N represents an element

$K \in \beta$. It is worth emphasizing that, even though this algorithm uses the idea of search tree, it is not required to instantiate all elements in memory. It is possible to generate every element K in an iterative fashion considering all frequencies in each level.

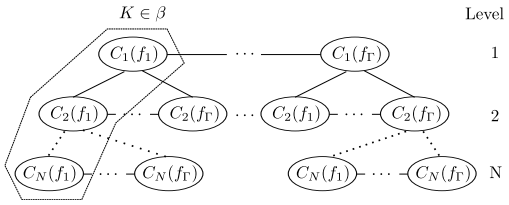


Figure 2. Search tree used by *smartenum*

In Figure 2, $f_1 > f_2 > \ldots > f_\Gamma$ and $C_i(f_1) < C_i(f_2) < \ldots < C_i(f_\Gamma)$. While evaluating frequencies in a decreasing order, it is possible to perform steps (I) and (II) of initial pruning, from Figure 1. The pruning step (I) comprises evaluation of all frequencies in F for each task, individually. While evaluating a single task, it is possible to take into account the local constraint (5).

$$C_i(k_i) \leq D_i \tag{5}$$

If an ordinary K has a k_i which violates the local condition (5), hence K also violates the restriction (3). The pruning in step (I) corresponds to perform this local analysis and to eliminate K's which do not respect the condition (5). This means to establish an upper local limit for each task. This limit determines what is the maximum computation time each task can assume and also the lowest frequency that can be applied on it, taking into account only properties from that task. It is worth mentioning that this pruning does not throw away all K's in β which violate (3). There are K's that meet local restriction (5) for all tasks but fail to satisfy (3) due to the existence of interference between them. This pruning step does not perform any kind of direct evaluation of the whole configuration.

The pruning step (II) corresponds to finding a starting point for the enumeration operation. It aims to determine a lower limit I. This pruning procedure considers the set $\beta' \subseteq \beta$, where $\beta' = \{K_i \mid K_i = \langle k_i, k_i, \ldots, k_i \rangle\}$. β' is a set of configurations where all tasks executes at same frequency. Elements in β' hold the property (6).

$$idle(K) \leq idle(K') \Leftrightarrow k_i \leq k'_i,\ K_i = \langle k_i, k_i, \ldots, k_i \rangle,\ K'_i = \langle k'_i, k'_i, \ldots, k'_i \rangle \tag{6}$$

As the frequency k_i of configuration K is lower than k'_i of configuration K', then computation times $C_i(k_i)$ of all tasks T_i in configuration K will be greater than equivalents computation times $C_i(k'_i)$. As consequence, idle times in configuration K are lower than in configuration K', thus $idle(K) \leq idle(K')$. Taking into account the property from Equation (6), this pruning step performs a binary search in β'. Thus, the aim of this pruning is to find $I \in \beta'$ which does not violate the condition (3) and produces the lowest idle time of all configurations in β'. This pruning step evaluates configurations in order to perform the search. The following algorithm's steps consist of enumeration itself. In the worst case, even if all pruning steps are applied, all elements $K \in \beta$ are evaluated. The stop condition (III) of this enumeration is while there exist elements

$K \in \beta$ not evaluated. Step (IV) selects an element P for evaluation. It traverses the tree present in Figure 2 from left to right and from bottom to up. Frequencies are assigned and fixed for each higher levels, while frequencies are decreasingly variated in lower levels. At beginning of the search, the highest available frequency in each level is fixed as starting point. Next P's elements are determined by decreasingly variating only the frequency on last level of the tree, in this case level N. Once all frequencies on level N are already evaluated, the algorithm returns it to the highest frequency available and on level $N-1$ it selects the next available frequency, in decreasing order. This process is repeated for all N levels of the search tree. When all frequencies on level 1 are evaluated, or all elements $K \in \beta$ have been evaluated, this means that the whole search tree has been transversed. This situation represents the stop condition of this enumerative search.

In the enumeration process, each selected configuration P carries out the response time test. This test is represented by (3), which corresponds to step (V). If a selected configuration P is feasible, then the algorithm checks if it produces the lowest configuration idle time so far. This is performed at step (VI), that is, if $idle(P) < idle(M)$. If yes, P becomes the new configuration M with lowest idle time. This part is in the step (VII).

There is the possibility to apply some prunings when a selected configuration P violates condition (3). This pruning is performed on step (VIII) of the algorithm. This pruning relies on the fact that on all levels the computation time is evaluated in an increasing manner. So, having fixed frequencies at levels 1 to $N-1$, if a frequency f_i on level N is considered to produce a not feasible configuration P, then, a frequency $f_{i+1} < f_i$ will produce another configuration P', also not feasible. Thus, there is no need to evaluate P'. This pruning is performed in a similar way for all levels. This pruning operation adopts the result from evaluation done by the enumeration operation in order to execute prunings. The step (VIII) is the *bound* process on this algorithm. Step (IX) is the end point in this algorithm. If it exists one, this step reports a feasible configuration M that produces the lowest possible idle time.

5. Exemplification

This section details how to apply the proposed algorithm in a small case study. Let a model $\mathcal{M}$ with two tasks executed in a processor with four frequencies available and sharing three resources. The available frequencies and voltages are listed in Table 1(a). The part of the task model with percentage of resource usage are shown in Table 1(b).

Table 1. Case study for exemplification.

(a) Freq. and Voltages

i	Frequency	Voltage
1	$600MHz$	$1.19V$
2	$466MHz$	$1.05V$
3	$333MHz$	$0.91V$
4	$80MHz$	$0.72V$

(b) Task model

				Resource usage		
Task	p_i	Cycles	D_i	i	ii	iii
1	0	1500	30	0%	40%	40%
2	1	900	10	20%	20%	0%

Table 2. Complete enumeration

									Can be pruned by		
f_1	f_2	K#	$C_1(s)$	$C_2(s)$	R_1	R_2	Schedulable?	$idle(K)$	(I)	(II)	(VIII)
600	600	01	2.50	1.50	3.80	5.00	YES	31.20		Y	
	466	02	2.50	1.93	3.89	5.43	YES	30.68		Y	
	333	03	2.50	2.70	4.04	6.20	YES	29.76		Y	
	80	04	2.50	11.25	5.75	∅	NO	∅	Y	Y	
466	600	05	3.22	1.50	4.52	5.72	YES	29.76		Y	
	466	06	3.22	1.93	4.61	6.15	YES	29.24			
	333	07	3.22	2.70	4.76	6.92	YES	28.32		Y	
	80	08	3.22	11.25	6.47	∅	NO	∅	Y	Y	
333	600	09	4.50	1.50	5.80	7.00	YES	27.19		Y	
	466	10	4.50	1.93	5.89	7.44	YES	26.67		Y	
	333	11	4.50	2.70	6.05	8.21	YES	25.75			
	80	12	4.50	11.25	7.75	∅	NO	∅	Y		
80	600	13	18.75	1.50	20.05	∅	NO	∅			
	466	14	18.75	1.93	20.14	∅	NO	∅			Y
	333	15	18.75	2.70	20.29	∅	NO	∅			Y
	80	16	18.75	11.25	22.00	∅	NO	∅	Y		Y

Initially, consider the complete enumeration of all possible configurations $K \in \beta$ for this example. In this case, there are 4^2 possible configurations for this model. All these 16 configurations are listed in Table 2. If pruning step (I) is applied, frequency $80MHz$ is discarded for task $T2$, as its computational time would be $C_2(80) = 11.25$ and $D_2 = 10$, then $C_2(80) > D_2$. Hence, K's 04, 08, 12 and 16 are not evaluated if pruning step (I) is used. If pruning step (II) is executed, first $\beta' = \{01, 06, 11, 16\}$ is taken into consideration. Configuration 11 is the one in β' which is schedulable and provides the minimum idle time. So, 11 would be taken as the start point to evaluate remaining configurations in β. If pruning step (VIII) is considered during the enumeration of this task model, configurations 14, 15 and 16 would be not evaluated. At the point the enumeration reaches configuration 12, it will discover that 12 is not schedulable, and then, it will prune configurations 14, 15 and 16, because lowering frequency for task T_2 in this case will not give any other schedulable model. The resulting enumeration of β, if all pruning steps are used, contains only configurations 06, 11 and 13. For this task model, configuration 11 is the one which is schedulable and provides the minimum idle time. Configuration 01, which is the configuration with maximum frequency and voltage, would consume about $3398.64 \times C_l$, accordingly with Equation 1, considering C_l (circuitry capacitance) as a constant over time. Configuration 11, on the other hand, would consume about $1987.44 \times C_l$. Therefore, the optimum frequency set determined by the algorithm reduces about 41.52% of energy consumption, if compared to configuration 01.

6. Experimental Results

This is a *branch-and-bound* algorithm. Therefore, in the worst-case, even though all pruning steps are applied, all possible configurations $K \in \beta$ will be enumerated. The proposed algorithm performance evaluation is based on the number of enumerated configurations and on the time spent.

Table 3. Execution types

Type	Execution of
-	No type of prunings.
A	Step (I).
B	Step (II).
C	Step (VIII).
A+B	Steps (I) and (II).
A+C	Steps (I) and (VIII).
B+C	Steps (II) and (VIII).
A+B+C	Steps (I), (II) and (VIII).

All possible combination of pruning operations has been evaluated. In this case, there are eight

types of execution that are listed on Table 3.

For simplification purposes, the frequency and voltage switching overhead, σ from Equation 2, is considered to be zero in these experiments.

6.1. Case Study I

In this case study, there are six tasks in this model, they are executed in a system with four frequencies available and they share two resources. The set of available frequencies and voltages is listed in Table 4(a). The part of the task model with percentage of resource usage are shown in Table 4(b). The computational time spent during execution for each pruning combination is illustrated in Figure 3(a). The maximum time was 18 ms (no pruning) and the minimum time was 2 ms (pruning A+B+C). The number of evaluated configurations done during the enumeration for each pruning combination is illustrated in Figure 3(b).

Table 4. Case study I.

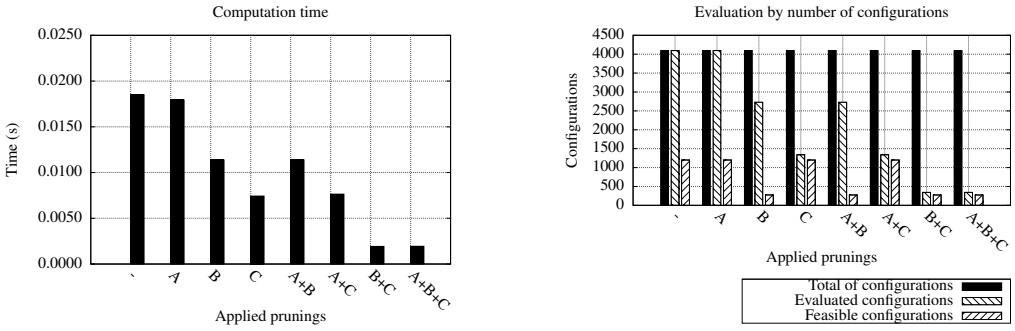
(a) Freq. and Voltages.

i	Frequency	Voltage
1	$440MHz$	$1.6V$
2	$120MHz$	$1.2V$
3	$60MHz$	$1.0V$
4	$30MHz$	$0.9V$

(b) Task model.

			Resource usage	
Task	Cycles	D_i	i	ii
1	2400	200	25%	0%
2	1200	50	13%	0%
3	1200	150	0%	30%
4	900	100	50%	0%
5	600	100	20%	0%
6	600	100	20%	10%

Results show that type A did not produce any pruning. Whilst types B and C have produced a considerable amount of prunings. This can be seen in the computation



(a) Computation time

(b) Number of evaluated configurations

Figure 3. Results for case study I.

time of Figure 3(a) and in the number of evaluated configurations of Figure 3(b), where type C has produced the lowest amount of time for execution and the lowest number of evaluated configurations. As presented in Figure 3(b), in order to determine the optimum frequency set, it was required to evaluate 361 of 4096 configurations, which

represents a reduction of 91.2% in the number of evaluated configurations. Considering the system executing under highest operating frequency, $440MHz$ under $1.6V$, from Equation (1), this model would consume $17664.00 \times C_l$. For this case study, the optimum frequency set is $\beta = \{60MHz, 440MHz, 30MHz, 120MHz, 440MHz, 440MHz\}$. Therefore, if the model uses the optimum configuration, it would consume $10812.00 \times C_l$, which represents a 38.79% of energy consumption reduction.

6.2. Case Study II

In this case study, there are twelve tasks in this model, they are executed in a system with four frequencies available and they share two resources. The set of available frequencies and voltages is listed in Table 5(a). The part of the task model with percentage of resource usage are shown in Table 5(b).

Table 5. Case study II.

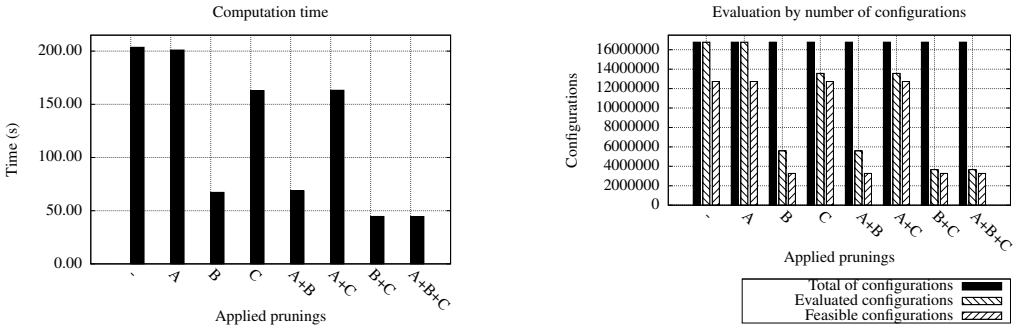
(a) Freq. and Voltages.

i	Frequency	Voltage
1	$300MHz$	$1.5V$
2	$250MHz$	$1.38V$
3	$220MHz$	$1.32V$
4	$150MHz$	$0.90V$

(b) Task model.

			Resource usage	
Task	Cycles	D_i	i	ii
1	180	10	13.4%	0%
2	180	20	44%	10%
3	180	30	0%	70%
4	180	30	0%	12%
5	180	30	12%	15%
6	180	30	0%	0%
7	180	30	13%	45%
8	120	10	3%	14%
9	120	20	4%	1%
10	120	30	0%	0%
11	120	30	0%	0%
12	100	10	0%	0%

The computational time spent during execution for each pruning combination of the proposed algorithm is illustrated in Figure 4(a). The maximum time was 210 s (no pruning) and the minimum time was 46 s (pruning A+B+C). The number of evaluated configurations done during the enumeration for each pruning combination is illustrated in Figure 4(b).



(a) Computation time

(b) Number of evaluated configurations

Figure 4. Results for case study II.

Results show that type B has produced best enumeration reduction. Type A has not produced any pruning. Reduction produced by type B has also appeared combined with other type of pruning. As presented in Figure 4(b), in order to determine the optimum frequency set, it was required to evaluate 3,662,613 of 16,777,216 configurations, which represents a reduction of $78,16\%$ in the number of evaluated configurations. Considering the system executing under highest operating frequency, $300MHz$ under $1.5V$, from Equation (1), this model would consume $4140.00 \times C_l$. In this case study, the optimum frequency set is $\beta = \{150MHz, 150MHz, 150MHz, 150MHz, 220MHz, 250MHz, 250MHz, 300MHz, 300MHz, 300MHz, 300MHz, 300MHz\}$. Therefore, if the model uses the optimum configuration, it would consume $2887.42 \times C_l$, which represent 30.26% in energy consumption reduction. This reduction is considerable and it would require only about tens of seconds during system design time in order to determine the optimum frequency set, as presented in Figure 4(a).

7. Conclusions and Future Works

This paper has shown a *branch-and-bound* algorithm to establish a set of optimum frequencies to be assigned to a real-time task model executed into a system with DVFS-enabled processor. The set of optimum frequencies is the one which produces lowest energy consumption and meets timing constraints from task model in such a way that provides the maximum utilization of idle times in each task. This work showed the response time schedulability analysis considering the specific frequency set.

The algorithm considers each frequency assignment combination as a configuration of a new model. A schedulability test is adopted in order to check each configuration feasibility. The algorithm itself consists of an enumerative search. Thus, in worst case it may reach unacceptable computation time. However, as presented in Section 6, the proposed pruning steps of this algorithm reduces considerably the amount of evaluated configurations and, therefore, the time spent for executing the algorithm. In the experiments, the number of evaluated configurations has been reduced to only 9% and 22% of total possible configurations, which is very a good result. Another important result is the amount of energy reduction when compared with executing tasks at high processor frequency. Our experimental results show that the proposed method may obtain energy reductions of 38.79%, and 30.46%.

For further research we intend: *(i)* to consider the energy needed to run a fraction of the tasks. If such amount of energy is already exceeding that needed to run a complete solution on a different branch, then the partial solution and its descendants can generally be pruned; *(ii)* to change the dynamic processor energy model to also include leakage current, memory access cost, I/O cost, and other energy overheads; and *(iii)* to develop a heuristic to produce a feasible configuration, maybe not optimum, but in a polynomial (non-exponential) computational time. *(iv)* to combine dynamic power management technique in our scheduling approach in order to reduce energy consumption by putting devices into sleep modes to optimize remaining idle time.

Acknowledgements

The authors would like to thank the partial financial support received from the Brazilian Council for Scientific and Technological Development (CNPq) through projects 554071/2006-1 and 575696/2008-7. And also for the partial financial support from the Nokia Corporation.

References

AbouGhazaleh, N., Mossé, D., Childers, B., Melhem, R., and Craven, M. (2003). Collaborative operating system and compiler power management for real-time applications. In *IEEE Real-Time Embedded Technology Applications Symposium*.

Azevedo, A., Issenin, I., Cornea, R., Gupta, R., Dutt, N., Veidenbaum, A., and Nicolau, A. (2002). Profile-based dynamic voltage scheduling using program checkpoints in the COPPER framework. In Design, Automation and Test in Europe Conference.

Choi, R. and Pedram, M. (2005). Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to onchip computation times. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

Gutnik, V. (1996). Variable supply voltage for low power dsp. Master's thesis, Massachusetts Institute of Technology.

Havinga, S. (1997). A survey of energy saving techniques for mobile computers. Internal Report, University of Twente.

Mejia-alvarez, P., Levner, E., and Mossé, D. (2004). Adaptive scheduling server for power-aware real-time tasks. *ACM Transactions on Embedded Computing Systems*, 3(2):284–306.

Pillai, P. and Shin, K. G. (2001). Real-time dynamic voltage scaling for low-power embedded operating systems. pages 89–102.

Sha, L., Rajkumar, R., and Lehoczky, J. P. (1990). Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. on Computers*, 39:1175–1185.

Shin, D., Lee, S., and Kim, J. (2001). Intra-task voltage scheduling for low-energy hard real-time applications. In *IEEE Design & Test of Computers*.

Yao, F., Demers, A., and Shenker, S. (1995). A scheduling model for reduced cpu energy. In *IEEE Symposium on Foundations of Computer Science (FOCS'95)*.

Zhao, B. and Aydin, H. (2009). Minimizing expected energy consumption through optimal integration of dvs and dpm. In *International Conference on Computer-Aided Design (ICCAD'09)*, pages 449–456, New York, NY, USA. ACM.

Impact of server dynamic allocation on the response time for energy-efficient virtualized web clusters

Carlos Oliveira, Vinicius Petrucci, Orlando Loques

Universidade Federal Fluminense (UFF), Niteroi, RJ, Brazil

{cjunior,vpetrucci,loques}@ic.uff.br

Abstract. *Virtualization has been widely adopted in data centers around the world for improving resource usage efficiency; particularly helping to make these computing environments more energy-efficient. Server virtualization allows for on-demand allocation (using either migration or replication) of virtual machines (VMs), which run the web applications and services, to physical servers. In this paper, we measure and analyze the disruptive impact on the QoS (quality-of-service) provided by the applications, in terms of server-side response time and throughput, during dynamic allocation of virtual machines in a server cluster. The response time of the web applications in the cluster is adopted as our main QoS metric since it is crucial for qualifying the end-user experience. In our experiments, we use Xen as the virtual machine manager and Apache servers for running the web applications. Our results show that VM replication with workload balancing may lead to reduced disruption impact on the QoS measures when compared to VM migration.*

1. Introduction

Virtualization has been widely adopted in data centers around the world for improving resource usage efficiency; particularly helping to make these computing environments more energy-efficient. Several virtual machine monitors or hypervisors have been developed to support virtualization, e.g., VMware [Sugerman et al. 2001] and Xen [Barham et al. 2003]. The key idea is that server virtualization allows for on-demand configuration (either by migration or replication) of virtual machines (VMs), which run the web applications and services, to physical servers. Concentrating applications and services in a smaller number of servers, using this capability, helps to increase resource utilization, allowing to reduce the use of computer resources and the associated power demands.

In previous works [Petrucci et al. 2009, Petrucci et al. 2010], we have proposed an optimization solution for power and performance management in virtualized server clusters. The optimization deals with the problem of selecting at runtime a power-efficient configuration and a corresponding mapping of the multiple applications running on top of virtual machines to physical servers. The optimization decision also includes selecting the best voltage/frequency combination for each physical server, which can be imposed using DVFS (Dynamic Voltage and Frequency Scaling) support available in current processors. We have experimented our optimization approach through simulations driven by real workload traces. However, in practice, a problem that arises in this context is that migration and replication activities in a virtualized environment may lead to disruption

on the quality of service provided by the applications. For example, live migration mechanisms allow to make workload movements with a relatively short service downtime. However, the quality-of-service of the running applications are likely to be negatively affected during the migration activities [Voorsluys et al. 2009].

In this work, we consider a real virtualized cluster platform aimed at supporting the deployment of web applications. We carry out a set of experiments with different test scenarios to evaluate the application behavior during the course of migration and replication actions. We measure and analyze the disruptive impact on the QoS (quality-of-service) provided by the applications, by means of server-side response time and throughput, during dynamic allocation operations of virtual machines in a server cluster. The response time of the web applications in the cluster is adopted as our main QoS metric since it is crucial for qualifying the end-user experience. In our experiments, we use Xen as the virtual machine manager and Apache servers for running the web applications. Our results show that VM replication with workload balancing may lead to reduced disruption impact on the QoS compared to VM migration.

The paper is organized as follows. The description of our virtualized cluster and testbed is presented in Section 2. In Section 3, we present some experiments for evaluating the response time impact during virtual machine migration and replication. Section 4 summarizes related works and Section 5 concludes the paper.

2. Virtualized cluster description

2.1. Architecture

Our target architecture (shown in Figure 1) consists of a cluster of replicated web servers. The cluster presents a single view to the clients through a *front-end* machine, which distributes incoming requests among the actual *servers* that process the requests (also known as *workers*). These servers run CentOS Linux 5.4 with Xen hypervisor enabled to support the execution of virtual machines.

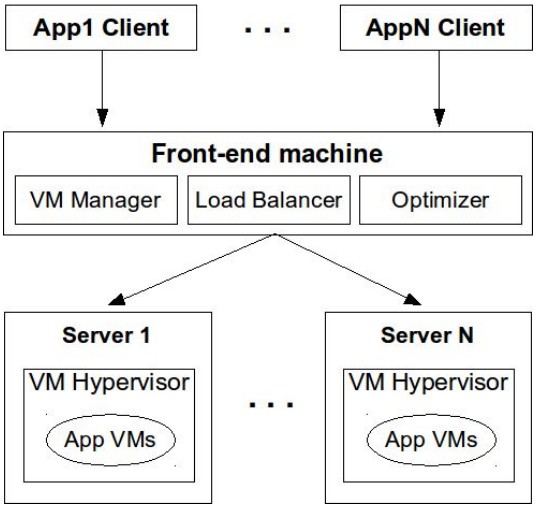


Figure 1. Server cluster architecture

The front-end machine is a key component in the architecture including three entities: (a) VM manager, (b) Load balancer, and (c) Optimizer. The `VM Manager` is imple-

mented using the OpenNebula toolkit [OpenNebula 2010] which enables the management of the VMs in the cluster, such as deployment and monitoring. The `Load Balancer` implements a weighted round-robin scheduler strategy provided by the Apache's `mod_proxy_balancer` module [The Apache Software Foundation 2010]. Finally, the `Optimizer` is designed to monitor and configure the virtualized cluster. It consists of an external module implemented in Python that relies on the primitives provided by the `VM Manager` and `Load Balancer` modules. The goal of the `Optimizer` is to dynamically configure the processors (using DVFS) and allocate the applications over the processor's cluster, in order to reduce power consumption, while meeting the application's performance requirements (see details of the overall optimization scheme in [Petrucci et al. 2010]).

2.2. Testbed

The testbed platform used to implement the proposed architecture is described in Figure 2. The web requests from the clients are redirected to the corresponding VMs that run the web servers on physical machines called *workers*. Each VM has a copy of a simple CPU-bound PHP script to characterize a web application. We define two different applications in the cluster, named App1 and App2. To generate the workload for each application, we use two machines with the httperf tool. The load generator machines `camburi` and `cumulus` (two Intel Pentium 4 2.80GHz, 1GB RAM, Ubuntu Linux 9.04) are physically connected via a gigabit switch. The worker machines `maxwell` (Intel Core i5 2.67GHz, 8GB RAM, CentOS 5.4) and `edison` (Intel Core i7 CPU 2.67GHz, 8GB RAM, CentOS 5.4) are connected via another gigabit switch. The front-end machine `henry` (AMD Athlon 64 3500+, 3GB RAM, CentOS 5.4) has two gigabit network interfaces, each one connected to one of the switches. All machines share an NFS (Network File System) storage mounted in the front-end to store the VM images.

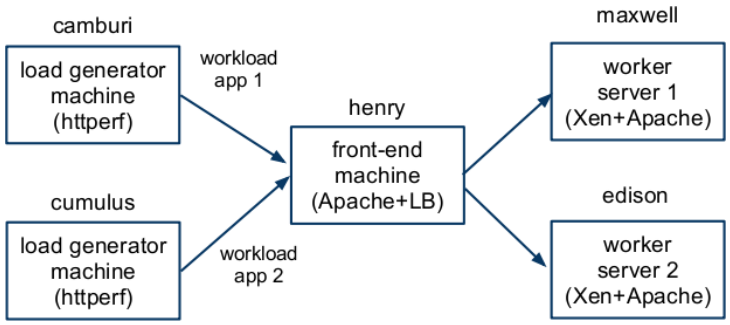


Figure 2. Cluster testbed setup

2.3. Response time measurement

The response time considered in this work is related to the server side. Thus, the response time is defined by the difference between the time a response is generated and the moment the server has accepted the associated request. To obtain the response time for the web applications we have implemented a new Apache module that collects the time information between these two moments using pre-defined hooks provided by the Apache Module API [Kew 2007]. The hooks used to measure the response time are: `post_read_request` and `log_transaction`. The `post_read_request` phase allows our module to store

the moment a request was accepted by Apache and the `log_transaction` phase allows it to store the moment a response was sent back to the client. The difference between these values gives the response time.

To smooth out high short-term fluctuations in measurements readings, we have integrated a filter procedure in our Apache module based on a single exponential moving average [Engineeting Statistics Handbook 2010]. Specifically, the filter computes the next value, S_t, by summing the product of the smoothing constant α ($0 < \alpha < 1$) with the new value (X_t), and the product of $(1 - \alpha)$ times the previous average, as follows: $S_t = \alpha * X_t + (1 - \alpha) * S_{t-1}$. Values of α close to 1.0 have less smoothing effect and give greater weight to recent changes in the data, while values of α close to 0.0 have a greater smoothing effect and are less responsive to recent changes. Some techniques may be used to optimize the value of α, such as using the Marquardt procedure to find the value of α that minimizes the mean of the squared errors (MSE) [Engineeting Statistics Handbook 2010]. In the filter module, we have used $\alpha = 0.5$ as the default smoothing factor; based on our experiments this value was found suitable.

2.4. Xen hypervisor

In a Xen system, the virtual machines are termed *domains*. The `Domain0` or `Dom0` is the first domain launched when the system is booted. It can be used to create and configure all other regular guest domains. A regular guest domain is called a `DomU` or unprivileged domain. `Dom0` is scheduled like `DomUs`. If a domain has only one VCPU, it can be executed in one processor or core at a time. Each domain may have one or more virtual CPUs (VCPUs) which run on physical CPUs. In our experiments, each VM has four VCPUs since we use a quad-core architecture. Xen has another feature called "cap" that can be used to control the maximum percentage of CPU a domain can use, even if there are free CPU cycles. This may be useful if one wants to control how the Xen schedules the domains to the physical CPUs.

The Xen hypervisor offers two kinds of migration: cold and live migration. The difference between them is that on cold migration the VM stops running during migration. Otherwise, on live migration the VM keeps running most of the time; actually, it stops only for a few milliseconds at Stage 3. The live migration stages are listed below [Clark et al. 2005]:

- *Stage 0 (Pre-Migration)*: Alternate physical host may be preselected for migration. Block devices mirrored and free resources maintained;
- *Stage 1 (Reservation)*: Initialize a container on the target host;
- *Stage 2 (Iterative Pre-copy)*: Enable shadow paging. Copy dirty pages in successive rounds;
- *Stage 3 (Stop and copy)*: Suspend VM on source host. Generate ARP to redirect traffic to target host. Synchronize all remaining VM state to target host;
- *Stage 4 (Commitment)*: VM state on target host is released;
- *Stage 5 (Activation)*: VM starts on target host. Connects to local devices. Resumes normal operation.

Notice that cold migration does not have the Stage 2 as in live migration. Notice also that caches in hardware are not migrated [Verma et al. 2008], which can lead to

cache misses in the target machine and impact application's performance when performing migrations. As pointed out by [Voorsluys et al. 2009], both migration activities need extra CPU cycles for the pre-copying process which are consumed on both source and destination servers. Moreover, an additional amount of network bandwidth is consumed as well, which may affect the quality-of-service in the cluster. A third available option is VM replication, which means creating a new VM (application instance) from a stored image, instead of migrating an already running one (see Section 3). It is worth mentioning that our optimization model includes the possibility of running replicated servers for fulfilling the resources required by an application at a given operational stage.

3. Experiments

We performed a set of experiments in our testbed (described in Section 2.2). In the first step, we used the Apache Benchmark (ab) [The Apache Software Foundation 2010] to measure the maximum number of requests per second that our physical machines can handle. We found that our worker machines (`maxwell` and `edison`) achieved a maximum of 1145 requests/sec for a typical PHP script web request with an average processing time of 6 milliseconds.

As shown in Figure 3, when the CPU utilization of an application is low, the average response time is also low. This is expected since no time is spent queuing due to the presence of other requests. On the other hand, when the utilization is high, the response time goes up abruptly as the CPU utilization gets close to 400% (the maximum value for utilization is 400% because we are using quad-core machines and each core represents 100%). In order to meet fair response time requirements, we shall perform VM migration or replication before the machine saturates, dimensioned for playing safe as 300% of CPU utilization. This leaves an amount of 100% CPU capacity available to be used by the VM management domain (`Dom0`) on the physical servers during the migration or replication activities.

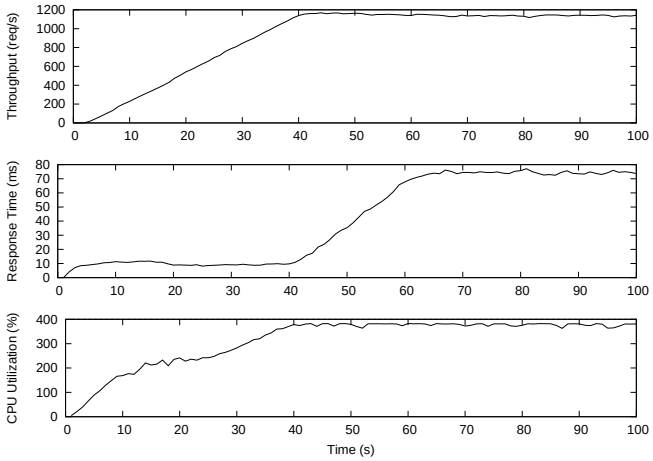


Figure 3. Relationship among throughput, response time, and CPU utilization

In the next step, we allocate two virtual machines (VMs) to run on the `maxwell` machine. Each VM has 256 MB of RAM, running an Apache 2.2 over Debian 4.0. Since our applications are CPU-bound, this memory capacity was found suitable for our experiments. Notice that the quantity of memory a VM is using may impact on how much time

is needed to complete a migration [Hermenier et al. 2009]. We plan to evaluate this issue in future experiments.

The first VM runs the application `App1` and it uses 120% of the total CPU resources (considering a quad-core machine). The second VM, which runs the application `App2`, starts using 40% of the CPU resources. Then, we consider increasing the `App2` workload demand until both VMs for `App1` and `App2` (along with `Dom0`) are using 300% of the physical CPU resources. After such a condition occurs we perform the actions described in the following experimental scenarios in order to maintain quality-of-service requirements. We ran experiments with three different scenarios: (a) cold migration, (b) live migration, and (c) replication. Each of the experiments showed similar results for repeated executions.

3.1. Scenario 1: Cold migration

In this scenario, the cold migration mechanism is applied to move the `App2` VM to a physical machine with spare capacity (that is, from `maxwell` machine to `edison` machine). As expected, we observe that in the cold migration, the VM stops during the migration. Figure 4 shows the throughput, response time and CPU utilization for both VMs during the course of the experiment. The experiments have approximately 10 minutes in duration.

We show that this kind of migration cannot be used in a soft real-time system because the VM being migrated stops during the course of migration. This is explicitly shown at the throughput curve of the application `App2`, which was migrated and then stopped for approximately 10 seconds. During the course of this kind of migration, the slowdown in the service was 100% because the execution of `App2` had been completely suspended and both response time and throughput measurements dropped to zero. In all scenarios, the drop in the throughput shows the instant in which the VM movement was performed. After the migration phase, the response time varied considerably reaching up to 8 seconds.

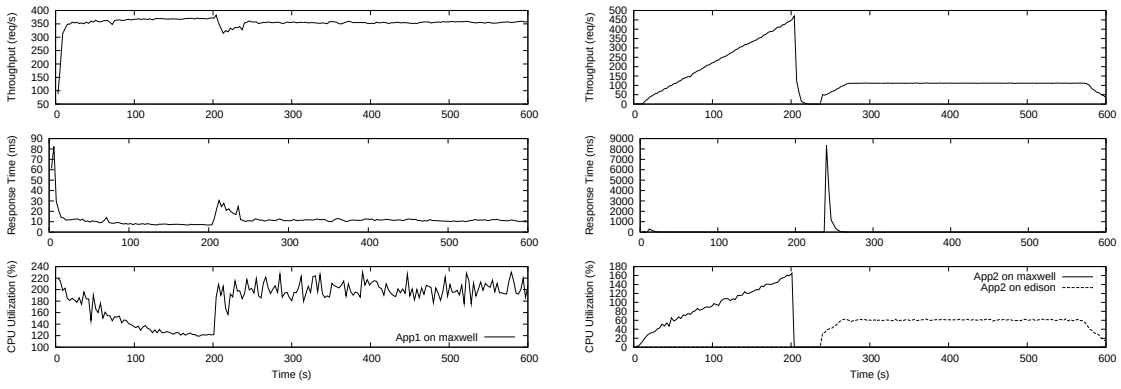


Figure 4. Execution of the *cold migration* scenario: App1 (left) and App2 (right)

3.2. Scenario 2: Live migration

In this scenario, we use live migration to move a VM to a new server machine without service interruption [Clark et al. 2005]. Besides not stopping the service during migration, we still need to maintain an acceptable quality-of-service in terms of application's

response time. The goal of the experiment (shown in Figure 5) was to evaluate the impact of applying the live migration mechanism.

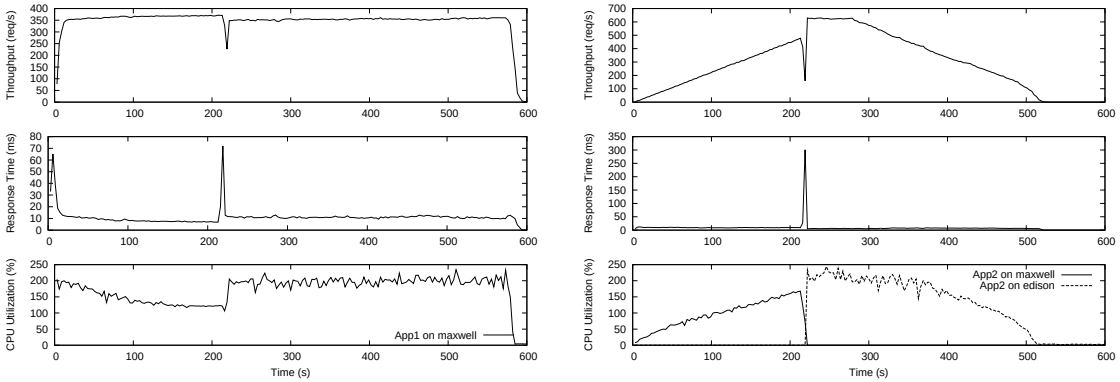


Figure 5. Execution of the *live migration* scenario: App1 (left) and App2 (right)

As would be expected, unlike cold migration, we observe that in live migration the VM is not paused during the migration. In Figure 5, the application `App2` was migrated with no interruption to the service. However, we noticed that the response time for `App2` increased substantially during the course of migration. For instance, the response time measured for `App2` raised from 11 milliseconds to 300 milliseconds on average for a period of 3 seconds. The throughput measurement was also affected by the migration. We also notice that even `App1` was slightly affected when migration was performed. The slowdown in the throughput was 61.5% (from 414.1 req/s to 159.5 req/s). We can note that the disruptions observed when performing dynamic changes through live migration last a short time and are basically unavoidable.

3.3. Scenario 3: Replication

We also have investigated an alternative approach using replication to help minimize these disruptive impacts in the QoS of the applications. In this scenario, we consider creating and deploying a new VM replica for the application `App2` on the destination server. At the moment the new replica is ready for processing the client requests, we start redirecting the requests to this new replica. We may then either shutdown or keep running in the origin physical server the old VM replica. The first option was adopted in the experiment. The last option may be adopted to provide a high capacity server, summing the replicas resources, to support an application with high resource demands.

The goal of the experiment for this scenario (see Figure 6) is to evaluate the response time impact compared to the live migration scenario presented in Section 3.2. Specifically, we have measured the response time (and throughput) during the replication process to identify potential practical issues such as the time delay to boot a new VM and the stabilization time of the load balancer when transferring requests to the new replica.

The use of replication shows improvements compared to the live migration, for example, by analyzing the decrease observed in the throughput in Figure 6, contrasting it to Figure 5. Specifically, the execution behavior of both applications `App1` and `App2` was found more stable when using replication in comparison to live migration. For example, the response time observed for `App2`, which was replicated in another server machine,

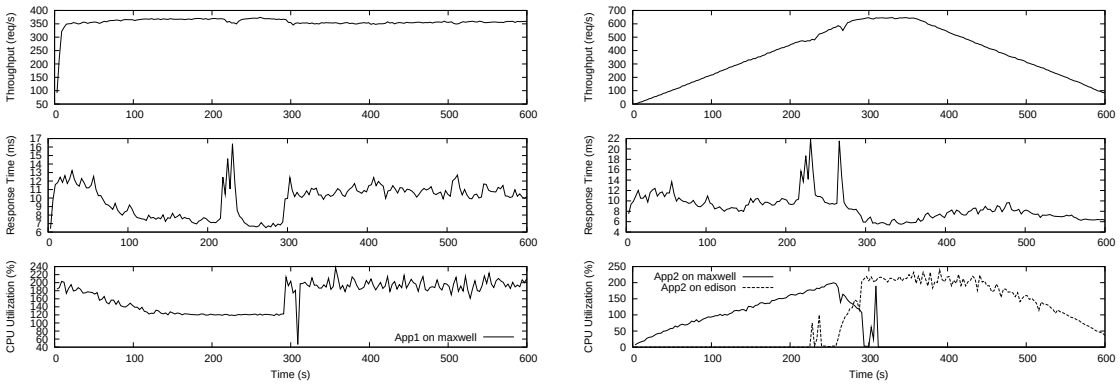


Figure 6. Execution of the *replication* scenario: App1 (left) and App2 (right)

increased from 10 milliseconds to 22 milliseconds. In addition, the throughput observed had a very slightly drop from 473 req/s to 467 req/s. We can also emphasize that `App1` was less affected when replication was performed instead of migration.

The basic steps for replication consists of (1) booting a new VM replica and (2) redirecting the requests to the new replica. The time needed to boot a VM is in between 25 and 40 seconds, which may be a bit longer than 10 seconds, on average, observed in the live migration in Scenario 2 (see Section 3.2). We argue that if the replication scheme is able to take advantage of prediction techniques to anticipate the booting process considering typical load patterns, such as proposed in [Dinda and O'Hallaron 2000], the time delay for booting a new VM may be minimized. We may also boot the new replica on the target machine a few seconds earlier to have it running and ready at the moment necessary for using the replicated VM.

The phase of redirecting requests for the new replica raised an implementation issue that needs to be addressed. We have observed that if all the current requests were abruptly redirected to the new VM replica it would take a long time to get both throughput and response time stable. The sticking point is that Apache has a single control process responsible for launching child processes (daemons) which listen for connections and serve their requests when they arrive. To tackle this redirecting bottleneck, we used a configurable mechanism termed "spare servers" [The Apache Software Foundation 2010]; setting the Apache configuration to maintain a suitable set of idle server daemons, which standby ready to serve incoming requests. In this way, clients do not need to wait a long time for a new child processes to be forked before their requests can be served. Moreover, redirecting the requests at a slower rate we achieved further reduction of the server settling time. In this experiment, we redirected 10% of the requests each time, until 100% of the requests were redirected to the VM replica.

4. Related Work

A heuristic algorithm for server consolidation is available [Khanna et al. 2006], but it does not take into account the cost of migrating virtual machines from one physical machine to another. Another approach [Wang et al. 2008] presents a two-layer control architecture aimed at providing power-efficient real-time guarantees for virtualized computing environments. The work relies on a control theory based framework, but does

not addresses live migration in a multiple server context. In [Kusic et al. 2009], a dynamic resource provisioning framework is developed based on lookahead control. A power-aware migration framework for virtualized HPC (High-performance computing) applications, which accounts for migration costs during virtual machine reconfigurations, is presented in [Verma et al. 2008]. As in our approach, it relies on virtualization techniques used for dynamic consolidation, although the application domains are different. In [Voorsluys et al. 2009], the authors quantify the effect of VM live migrations in the performance of social networking websites. The overall objective of their experiments is to quantify slowdown and downtime experienced by the application when VM migrations are performed in the middle of a run.

5. Conclusion and future work

We have presented a virtualized server environment targeted for dynamic deployment and allocation of VMs to physical machines. Our goal was to carry out experiments to evaluate the performance impact in terms of response time and throughput of applications during the course of VM migration and replication.

The replication steps involved starting a VM replica in the target host and redirecting requests to the new VM replica. Our results showed that by using replication we can minimize some performance disruption incurred during migration. Finally, the evaluation described in this work will help us to implement our dynamic optimization model and strategy for power and performance management of virtualized web clusters [Petrucci et al. 2010].

As for future work, we plan to develop additional experiments with state-aware applications considering another layer as database. To address this, we intend to use Rubis [RUBiS 2010] multi-tier benchmark. In the replication process, the VM in the source server was turned off. We would like to investigate if it would be valuable while maintaining this application on the source server too for load balancing proposes. And, if so, we would like to identify what part of the application workload would be allocated both in the source and target physical machines.

References

Barham, P. et al. (2003). Xen and the art of virtualization. In *SOSP'03*, pages 164–177. ACM.

Clark, C. et al. (2005). Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation*, pages 273–286.

Dinda, P. A. and O'Hallaron, D. R. (2000). Host load prediction using linear models. *Cluster Computing*, 3(4):265–280.

Engineeting Statistics Handbook (2010). Single exponential smoothing. http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm.

Hermenier, F. et al. (2009). Cluster-wide context switch of virtualized jobs. Technical Report 6929, INRIA.

Kew, N. (2007). *The Apache Modules Book: Application Development with Apache*. Prentice Hall.

Khanna, G. et al. (2006). Application performance management in virtualized server environments. *10th IEEE/IFIP Network Operations and Management Symposium*, pages 373–381.

Kusic, D. et al. (2009). Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15.

OpenNebula (2010). The open source toolkit for cloud computing. http://opennebula.org/.

Petrucci, V., Loques, O., and Mossé, D. (2009). A dynamic configuration model for power-efficient virtualized server clusters. In *11th Brazillian Workshop on Real-Time and Embedded Systems (WTR)*.

Petrucci, V., Loques, O., and Mossé, D. (2010). A dynamic optimization model for power and performance management of virtualized clusters. In *1st Int'l Conf. on Energy-Efficient Computing and Networking. In cooperation with SIGCOMM*. ACM (to appear).

RUBiS (2010). Rubis: Rice university bidding system. http://rubis.ow2.org/.

Sugerman, J. et al. (2001). Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. *USENIX Annual Technical Conference*.

The Apache Software Foundation (2010). Apache HTTP server version 2.2. http://httpd.apache.org/docs/2.2/.

Verma, A. et al. (2008). pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware'08*, pages 243–264.

Voorsluys, W., Broberg, J., Venugopal, S., and Buyya, R. (2009). Cost of virtual machine live migration in clouds: A performance evaluation. In *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, pages 254–265, Berlin, Heidelberg. Springer-Verlag.

Wang, Y. et al. (2008). Power-efficient response time guarantees for virtualized enterprise servers. In *RTSS'08*, pages 303–312.

12th Brazilian Workshop on
Real-Time and Embedded Systems

♦

Technical Session 3

Device Drivers and Operating Systems

Exploiting Template-Metaprogramming for Highly Adaptable Device Drivers – a Case Study on CANARY an AVR CAN-Driver

Christoph Steup, Michael Schulze, Jörg Kaiser

[1]Department for Distributed Systems
Universität Magdeburg

Universitätsplatz 2, 39106 Magdeburg, Germany

Christoph.Steup@st.ovgu.de, {mschulze, kaiser}@ivs.cs.uni-magdeburg.de

Abstract. *Providing applications with a perfectly tailored device driver is essential to avoid the waste of resources. This is even necessary for the broad field of embedded systems development. However, the development of device drivers is in general a difficult task, and supporting a portable, configurable as well as adaptable device driver is even harder. To achieve such a device driver architecture, we propose declarative configuration specifications, exploit template-metaprogramming, and introduce the new concept of RegisterMaps. We evaluate the device driver architecture, showing that the device driver's resource usage scales with different configurations. We compare our device driver architecture against a device driver implementation of a hardware vendor, proving the competitiveness of our solution.*

1. Introduction

The development of device drivers is a difficult task. Allowing the use of a driver in different products, on different platforms, and with different application demands, requests development mechanisms that handle the challenge of these variability. In the embedded systems field, an important requirement – resource consumption – has to be treated additionally. Thus, a minimal RAM/ROM footprint device driver is desirable that gives applications only the functionality they need but not more. Providing a device driver that is portable, configurable as well as adaptable and uses resources very efficiently makes the development even harder, and in development for embedded devices, all these aspects are of major interest.

Portability, configurability and adaptability are properties device drivers should have, however to understand the differences between them, it needs further explanation. In general, a portable device driver allows for using on different platforms. From the application point of view, a need is a stable driver interface. Against such an interface applications can be implemented. If the interface stays unchanged even in case of different hardware platforms, application migration without code changes in the best case will be possible, leading to decreasing costs when new platforms have to be used. Hence, the development of such application is simplified, since no platform dependencies are propagated from the driver level up to the application level.

The configuration of a device driver is the process of setting parameters for tailoring the functionality to application demands, thus the device driver behaves as intended.

For example, if a device driver supports a polling and an interrupt feature, and an application decides to use polling only, the interrupt functionality will be switched off. This transition or change in the functionality is part of the adaptation process of the device driver. Thus, an adaptable device driver has the ability to serve different demands on the one hand arising from applications as configuration requests and on the other hand due to features given by the used hardware platform. The latter adaptation is done automatically without an explicit trigger by an application or user.

Developing a device driver having all preferred characteristics is as mentioned a difficult task. However, we are not the first one that try to realize adaptable software at all. Different development techniques like conditional compilation, object-orientation, feature-orientation, aspect-orientation, or component-orientation approaches try to tackle the problem, but all have their own strength and weaknesses. Because device drivers are usually written in C or C++ we focus on techniques that are only available for these languages.

From the developer's point of view, conditional compilation with the help of a preprocessor tool, mainly the C-preprocessor *cpp*, is easy to learn. Statements like *#define*, *#ifdef*, *#else*, etc. are used for framing areas that should be included or excluded in dependence of the given condition. However, due to the annotation of driver code with preprocessor directives the whole code is cluttered and obfuscated, leading to hard maintainability. Furthermore, the annotations are not type safe, because it is usual text processing only, allowing for every transformation, even those creating incorrect sources. This is error-prone, because the developer has a different view on the sources as the compiler. In the literature, the use of the preprocessor is often criticized. For example, [Spencer and Collyer 1992] demand "*#ifdef* considered harmful" and [Lohmann et al. 2006] speak from the "*#ifdef*-hell".

An alternative development technique is object-orientation, which is also supported by C++, due to its nature as multi-paradigm language. Implementing the preferable features with the object-oriented approach can be done in two directions. Firstly, one can implement the device driver as a class library, that defers all possible configuration steps to the runtime, having a dynamic, parameterizable driver, but leading to an immense overhead concerning resource usage. Secondly, the opposite is programming all the functionality by fine-modular, structured sub-classing. While this achieves minimal code overhead, it is a maintenance nightmare due to the exponential class explosion with every new upcoming feature [Gamma et al. 1995].

Device driver development with the component-oriented approach like with UML [Object Management Group 2001] generates a resource overhead, which is at least in the same magnitude as the dynamic, parameterizable device driver of the object-oriented approach. A disadvantage in this context is the loose coupling of components that lead to always have function calls. Furthermore, the compiler is not able to optimize code above component boundaries. Thus, reasoned by the black-box concept of components fine-granular adaptations are not designated, and applications are encumbered with unneeded functionality.

Components as well as object-orientation try to go the way of "separation of concern". Techniques like aspect-orientation (AOP) and feature-orientation (FOP) aim in

the same direction, but consider other mechanisms. Usually a programming language is extended. On the one side, AOP [Kiczales et al. 1997, Tarr et al. 1999] tackles the problem of cross-cutting concerns with aspects as modularisation concept, encapsulating the cross-cutting issue. Aspects are often related with (global) system policies like synchronisations, tracing, and so on. However, in the context of developing a device driver, cross-cutting issues arise seldomly. To allow aspects getting involved in C++ developments, a language extension like AspectC++ [Spinczyk et al. 2002] is needed, which works as source-to-source preprocessor having type-safety in place.

Feature-oriented programming is a technique that tackles the class explosion problem of the object-oriented approach by adding new keywords to the programming language, allowing for describing the relation of base classes and their extensions without directly inheriting. This enables defining multiple extensions to a given base. The configuration is done with the help of an additional grammar and a FOP compiler that transforms the sources according to the configuration. FOP, first introduced by [Prehofer 1997], aims at supporting a way to have variability in the design, and allowing the system tailoring to applications needs in a separate configuration step [Batory et al. 2004].

All the mentioned mechanisms have either drawbacks like error-proneness, heavy resource-consumption or need additional tools that are often in an early development state due to their research character. Another way to reach variability and adaptability is using techniques like generic programming [Czarnecki and Eisenecker 2000], and one form is template-metaprogramming. Template-metaprogramming is a turing-complete functional language that is processed by the compiler during the template instantiation phase. It allows for code generation, constants calculation, type selection, etc. and enables producing only the needed functionality having optimal fitting code at the end. However, if something goes wrong during the compilation, the compiler is highly verbose and generates a lot of messages that can be difficult to interpret. This is a drawback for the development in general, but there are mechanisms to address this problem: on one side by using special programming constructs that enables customized error message generation and on the other side by better compiler support.

For our goal, obtaining a highly adaptable device driver, we exploit the template-metaprogramming concept. We enhance this concept, by providing declarative configuration descriptions that hold the application requirements on the driver, and using the descriptions as input to the metaprograms to tailor the driver's functionality. For driver adaptation on the hardware side, we propose a new concept – RegisterMaps.

The rest of the paper is structured as follows. We start with the explanation of the concept in Section 2, where we describe our basic device driver architecture first. Next, we consider the declarative configuration in Section 2.2, the new RegisterMaps concept in Section 2.3 and the used mechanisms of the template-metalanguage in Section 2.4. Section 3 discusses the resulting device driver and compares it with a device driver implementation of a hardware vendor. In Section 4 we conclude the paper and give a short outlook on future research questions.

2. Concept

2.1. Basic Architecture

The architecture of our highly adaptable device driver is depicted in Figure 1. There are three main aspects, which will be addressed in the fallowing. The definition of configuration parameters are the first aspect. This definition is declaratively done by the application developer, and in Section 2.2 we describe how the declarative specification is realized. The *Configuration* component visible in the Figure 1 uses this application-dependent parameter specification for adapting the driver.

Secondly being portable, we use our RegisterMaps (short *RegMap*) concept, which abstracts the access to the hardware. RegisterMaps acts as mediator between the hardware and the hardware abstraction layer, hiding the underlying bit structure and providing an access by name. In Section 2.3 we discuss the RegisterMaps concept in detail.

To adapt the driver to an application-dependent declarative configuration, we use template-metaprogramming. During the compilation of the driver, the compile-time interface of the driver is invoked by the template instantiation process, leading to a tailored device driver. In Section 2.4 we show which concepts of the template-metaprogramming approach we use and how we combine it with the RegisterMaps concept. The driver's resulting API is highly specific to the driver and in our case specific to a CAN driver. However to understand our proposed approach of an adaptable device driver, discussing the API is not needed.

In the middle of the Figure 1, the *interrupt handler* component is depicted. It is drawn dashed, because interrupts can be completely configured out of the driver, thus neither RAM nor code memory is needed, letting the application the option of having a polling- or an interrupt-driven device driver.

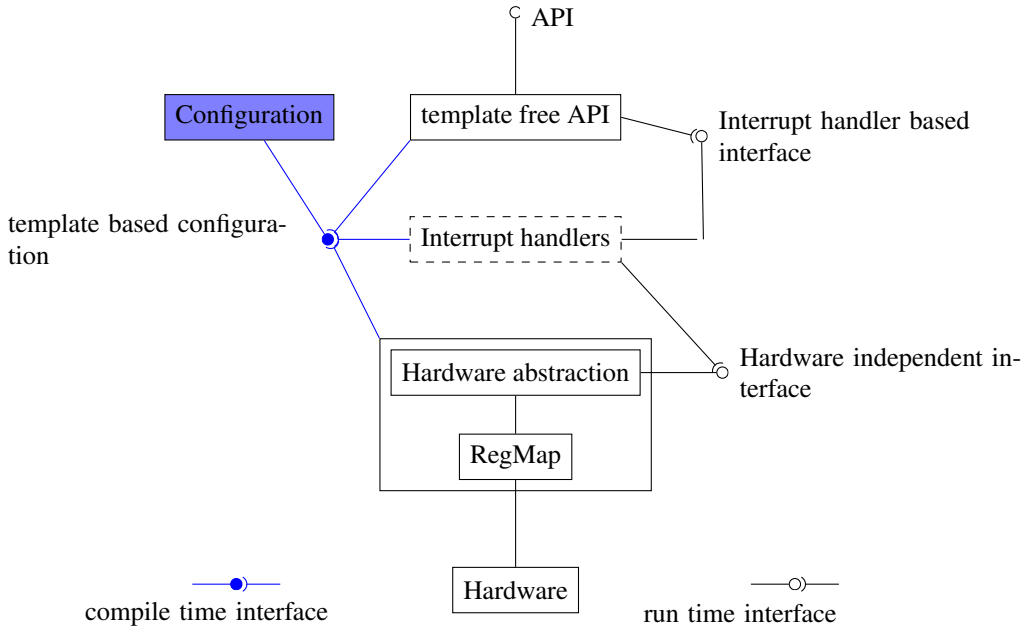


Figure 1. Components of an adaptive driver and their interfaces.

2.2. Declaration of configuration parameters

An adaptive device driver needs input, on which parameters it should adapt. The parameters are given by the user through a configuration structure. This configuration structure is the invocation parameter for the template-metaprogram at the compile-time interface, and it is used to adapt the functionality towards the user's requirements. However, there must also be certain default values in the case, that the user did not specify anything.

To have a flexible, yet resource efficient configuration, we use a C++-structure, which contains enumerations and typedefs. Both are compile-time constants, which will have no additional costs if not used.

```
1  struct CANConfig : defaultCANConfig{
2      typedef BaudRateConfig<
3                  F_CPU,
4                  SPEED_1M,
5                  SUBBITS_16
6              > baudRate;
7
8      enum Parameters{
9          version        = CAN_20B,
10         useReceiveInt = false
11     };
12 };
13
14 Canary<CANConfig>::type can_driver;
```

Listing 1. Declarative specification of configuration parameters for the CAN device driver

Getting a perfect user requirements fitting device driver, the user has to create such a C++-structure. The structure should be inherited from a default configuration to ensure that all configurable parameters are set at least to a default value. An example configuration is depicted in Listing 1.

After inheriting, the user can override the default values and adjust the parameters according to the desired behavior. To do so, he just creates an enumeration that contains the configuration parameters and the values he wants. In the case of the example the enumeration is called parameters. The example overrides the used CAN version to be CAN 2.0B and disables the receive interrupt. Also the baud rate is set to be 1 Mbit through the typedef of an additional helper structure BaudRateConfig. To apply this declarative parameter specification to the driver, the structure is given to the compile-time interface *Canary* as template parameter, resulting in an adapted device driver type *Canary<CANConfig>::type* in Line 14. Next the tailored device driver can be used through the created *can_driver* object.

The realisation of functionality inside the driver that correspond to the passed configuration parameters, depends on the used mechanism of the template-metalanguage, and we describe the used mechanisms in Section 2.4. With the help of the metaprogramming, additional features are possible like compile-time checks of configuration parameters. This enables us to notify the user of misconfiguration, e. g. if he tries to use functionality on the *can_driver* object, which is disabled in the current configuration.

As stated in the beginning of this paragraph the configuration is only one set of parameters, to which the driver has to adapt. The other set consists of the specification of the current hardware. Since these can be very tedious to handle, we propose a new concept to abstract the lowest level of hardware access in the next paragraph.

2.3. RegisterMaps – RegMaps

The goal of a RegMap is to provide an hardware independent, bit-wise access to memory mapped I/O-registers. This allows to reference single bits within a register by name. It also provides an abstraction mechanism that grants you hardware independence on I/O-register level. Thus means, that the position of registers in RAM and the order of bits inside a register are abstracted through the RegMap.

A RegMap first used in the avr-halib [Schulze et al. 2008] is realized as C++-structure with bit-field definitions, describing the abstracted registers and the contained bits. Bit fields are provided in the current C and C++ standards since years. Bit fields provide a way to access single bits of a register transparently without the need for bit mask and bit shift operations. Since the compiler does the burden of accessing the bits correctly, the usage of a RegMap reduces the amount of possible programming errors.

As an example of a RegMap we show Listing 2. This RegMap is used to abstract the access to the different interrupt enable flags of the used CAN hardware. An unused bit in a register cannot be left out of the RegMap, because it is still important for the order and position of the other bits in the considered underlying register. Declaring an unused bit, is done by creating an unnamed bit. Accessing such unnamed bit through the RegMap is impossible, which prevents programming errors. In Line four an example of an unnamed bit is shown. Unnamed bits are usually padding bits or bits that are reserved for future use e. g. in later processor revisions.

```
1 struct CanIntRegMap {
2     uint8_t timerOverrunInt   : 1;
3     uint8_t generalErrorsInt  : 1;
4     uint8_t                   : 1;
5     uint8_t timerOverrunInt   : 1;
6     uint8_t transmitInt       : 1;
7     uint8_t receiveInt        : 1;
8 };
```

Listing 2. An example RegMap, abstracting the interrupts of the CAN hardware.

Because a RegMap is designed to be an overlay for memory mapped I/O-registers, and therefore, it must be placed exactly at the position where the register resides inside the RAM. However, the compiler does not know about the special meaning of a RegMap's content, and if we would use the general construction process of C++ objects, it will be constructed somewhere in RAM. To tackle this problem, we use a special mechanism the *placement new operator* to overlay the RegMap exactly at the position of the respective I/O-registers. In our case, we use a macro *UseRegMap* to hide the syntactical burden of this mechanism.

Listing 3 shows the use of the defined RegMap. The example shows how the user specified configuration parameter *useReceiveInt* is used to set the receive interrupt flag

```
1  template<typename user_spec>
2  class Canary : ... {
3      public:
4          Canary(){
5              ...
6              UseRegMap(rm, user_spec::CanIntRegMap);
7              rm.receiveInt=user_spec::useReceiveInt;
8              SyncRegMap(rm);
9              ...
10         }
11         ...
12 };
```

Listing 3. Using the RegMap from Listing 2 to access the interrupt enable flag of the CAN hardware.

supported by the RegMap. At this point, no direct hardware access is needed. However in general, programming I/O-devices without much care leads to undefined behavior of devices. Since there is difference between the access to I/O-registers and memory. Because in case of I/O registers toggling bits multiple times in a sequence of commands is a valid programming sequence of a device. In contrast doing so on usual memory, only the last bit operation is important. Unfortunately, the compiler does not know about the difference between I/O-registers and memory. If we do not support the compiler with the right information, it will optimize aggressively by merging bit operations to create optimal code, but destroying the programming sequence of devices. This is in general a problem, and needs to be treated. We tackle the problem using a *SyncRegMap* that inserts an optimization boundary. Every operation concerning memory read or write must be finished up to this boundary until the next operation can be done by the compiler. This ensures correct behavior even with heavy optimization enabled. The *SyncRegMap* command is an alternative to make the whole RegMap volatile, which would have great negative influence on code memory size and performance.

To achieve hardware independence, for every used hardware a RegMap is needed. This is especially easy within similar families of micro-controllers, where many controllers have similar registers.

Having the abstract low-level access to the hardware available, the question how we select a specific RegMap arises. This problem is strongly coupled with the possibility to provide additional configuration parameters to the RegMap, which we provide by template parameters. The selection problem is solved using template-metaprogramming features of the C++-language, which we will look into in the next paragraph.

2.4. Template-metaprogramming

The concept of templates was already documented in 1990 in the “Annotated C++ Reference Manual” [Ellis and Stroustrup 1990]. Originally it was a technique to create parameterizable data structures. The parameterization is done by specifying types that are arguments for the data structure template instantiation. This allows the creation of generic data structures, because there is no need to write a data structure for every possible type. One popular example of this mechanism is the C++ Standard Template Library (STL)

vector class. Since object orientation is a way to reuse code, templates extended this concept to provide even more code reuse.

With the growth of the C++-language and the capabilities of templates, new ways to use templates were developed. Erwin Unruh showed the possibility to let the compiler calculate prime numbers as first [Unruh 1994]. However, Todd Veldhuizen was the one who recognized the potential behind this idea. He established a new programming approach known as template-metaprogramming [Veldhuizen 1995]. As said in Section 1 the template-metalanguage is a functional, turing-complete, side-effect-free language, that is processed during the compilation of an application. We mainly use certain aspects of the template-metalanguage, which are partial evaluation, template specialization and meta control structures. These aspects will be further described in the following.

The first used mechanism is partial evaluation. This gives us the ability to compile and include only the functions in the driver, which will be needed. For code, that is written in C or that is not using template-metaprogramming, the compiler will translate all defined function and put them in the resulting object file. In source code, that is parameterized by template-metaprogramming, however this is different. For this code, the compiler will only consider those functions that are at least called once. This saves code memory, because unused functions will not be included in the final driver.

The second used technique is template specialization. It is used to create special implementations of a class template for certain types like the CAN-ID class template in Listing 4. During the template instantiation process the compiler selects a specialized implementation that fits the given class template argument. We exploit this for obtaining the right CAN-ID class template specialization in Listing 4 in Line 34. The selection behavior is controlled by the *version* parameter of the *CANConfig* user configuration of Listing 1. The *version* parameter in the example configuration `CAN_20B` means the usage of CAN messages with extended IDs having 29 bits instead of 11 bits. A change of this parameter has an effect on the whole driver, because it leads to different used data structures. These structures differ in size, since the CAN-IDs need either 2 bytes for CAN 2.0A or 4 bytes in case of CAN 2.0B. Through the user-configured *version* parameter, the needed data structure will be automatically selected during compilation. The parameter is passed internally to all components of the driver to ensure consistent configuration. At this point only the driver is tailored to use the same CAN-ID in all components. However, the register content of the hardware also differs with the used CAN version. To cope with that we declared the RegMap to be a class template. This enables us to use the template specialization mechanism again to adapt to different I/O-register content dependent on the CAN *version* parameter. By using both times the same mechanism, the compiler ensures that transferring data between the driver data structures and the CAN hardware is correct by design.

Template specialization can also be exploited to create meta control structures like if-then-else for types. This opens the possibility to change whole block of functionality dependent on configuration parameters. For example, if a driver is configured only for sending messages, the receive functionality is not needed at all and is not existent in the final driver. In Listing 5 we show the selection of functionality for the receive interrupt behavior. The *Canary* template-metaprogam, here implemented as a class, inherits from the *if_then_else* class template. That meta control structure selects the second or the

```
1  // Supported CAN versions
2  enum Versions {
3      CAN_20A,
4      CAN_20B
5  };
6
7  template<Versions version>
8  class CAN_ID;
9
10 // Specialization for the CAN 2.0A CAN-ID specification
11 template<>
12 class CAN_ID<CAN_20A> {
13     public:
14         typedef uint16_t IdType;
15
16         enum Constants {
17                 idLength=11
18         };
19         ...
20 };
21
22 // Specialization for the CAN 2.0B CAN-ID specification
23 template<>
24 class CAN_ID<CAN_20B> {
25     public:
26         typedef uint32_t IdType;
27
28         enum Constants {
29                 idLength=29
30         };
31         ...
32 };
33
34 typedef typename CAN_ID<CANConfig::version>::IdType IdType;
```

Listing 4. Template specialisations of the CAN_ID type depending on the version

```
1  template<typename user_spec>
2  class Canary : public if_then_else <
3                              user_spec::useReceiveInt,
4                              ReceiveInterrupt,
5                              NoReceiveInterrupt
6                          > {
7      ...
8  };
```

Listing 5. Using a template meta control structure to (de)activate the use of the receive interrupt

third parameter dependent on the value of the first one. Thus, the *Canary* inherits from *ReceiveInterrupt* only if *useReceiveInt* is true in the configuration *user_spec*. This mechanism has the advantage, that features that are not present in the driver due to configuration does not cost anything (no code memory and no RAM).

The template-metaprogramming can be further used, to adapt a driver to hardware, where certain hardware features are not present. In such a case, a feature must be emulated by software. To only include the software emulation layer when it is needed, template specialization can be used to decide the usage. The compiler chooses the optimal fitting specialization for the platform. Since partial evaluation creates only the functions that are needed, and unused specializations are not considered and therefore not present in the final driver. An example for this would be a CAN hardware, which has no support for hardware ID filters. On such hardware, a device driver has to emulate the ID filter functionality in software if the application demands this. If the emulation is once implemented, it can be used on all hardware platforms that do not provide the feature. However platforms providing the feature do not need the emulation and the driver is created without it. With this mechanism used, creating a flexible, reusable and resource-optimal driver architecture is possible.

3. Evaluation

This paragraph deals with the results of the implementation. To show the advantages of our approach we compare the size of a set of example applications. On one side we use our highly adaptable device driver and on the other side an implementation from Atmel [at9], which is written in plain C.

We consider three different examples, describing usual use cases. All applications use CAN 2.0B, because the Atmel driver has no means to adapt itself to use CAN 2.0A only. The evaluated examples are: sending one message, receiving one message using polling and receiving one message using interrupts.

	Canary		Atmel	
example application	text	ram	text	ram
Sending	1248	17	4214	21
Receiving polling mode	1650	21	4168	21
Receiving interrupt mode	1950	29	n.a.	n.a.

Table 1. Program sizes and used RAM for different example applications

The compilation of all applications was done with gcc in version 4.3.4 and optimization switched on with -Os, leading to size optimized executables. For our driver we provided additional compilation flags concerning C++ code generation, which are: -fno-exceptions and -fno-rtti. These disable exception handling and runtime type informations, which are not used in our driver.

The result are contained in Table 1. It is clearly visible that our driver implementation is much more efficient in terms of code size. The RAM usage of our device driver is mostly better, but the interrupt driven version has an 8 byte overhead due to a delegate style interrupt handler. This delegate allows us to switch the interrupt callback at runtime, which is special feature that we offer. The RAM usage of the Atmel driver is constant, because it is not configurable and therefore uses the same data structures and functions for all examples. The interrupt example could not be measured, because, the driver from Atmel has no means to support interrupts natively.

To show the potential of our approach, we give additional measurements. As discussed, our driver can be configured in many ways. One of this is the minimal configuration of the third example supporting CAN 2.0A only. Doing so, saves additional 16.4% of code size and 19% RAM usage.

4. Conclusions and Outlook

In this paper we presented how template-metaprogamming is exploited for creating highly adaptable device driver. We applied our concepts for the development of a CAN device driver for an embedded platform, being adaptable, configurable and portable. The user has the possibility to configure the driver by using declarative descriptions. This descriptions are interpreted by template-metaprograms during the template instantiation phase. To abstract from low-level hardware issues, we introduced a new concept – RegisterMaps – allowing the development of portable driver architectures.

We evaluated the resulting driver architecture against a plain C implementation of a vendor's hardware driver. Our driver shows always better results in terms of code size and RAM usage. This is caused by its high configurability, because it contains only the needed functionality, but not more.

In the future we will apply our concepts on other types of devices to prove its generality. Furthermore, we will do measurements covering a broader range of configurations.

Acknowledgement

This work has partly been supported by the Ministry of Education and Science (BMBF) within the project "Virtual and Augmented Reality for Highly Safety and Reliable Embedded Systems" (VierForES).

References

AT90CAN32/64/128 software library. online, `http://www.atmel.com/dyn/resources/prod_documents/at90CANLIB_3_2.zip`. [(online), as at: 31.03.2010].

Batory, D., Sarvela, J. N., and Rauschmayer, A. (2004). Scaling Step-Wise refinement. *IEEE Transactions on Software Engineering*, 30:355–371.

Czarnecki, K. and Eisenecker, U. (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional. Published: Paperback.

Ellis, M. and Stroustrup, B. (1990). *The annotated C++ reference manual*. Addison-Wesley, Reading Mass.

Gamma, E., Helm, R., Johnson, R. E., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., and Irwin, J. (1997). Aspect-Oriented programming. In Aksit, M. and Matsuoka, S., editors, *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP '97)*, volume 1241 of *Lecture Notes in Computer Science*, page 220–242. Springer-Verlag.

Lohmann, D., Scheler, F., Tartler, R., Spinczyk, O., and Schröder-Preikschat, W. (2006). A quantitative analysis of aspects in the eCos kernel. In *Proceedings of the 2006 EuroSys conference on - EuroSys '06*, pages 191–204, Leuven, Belgium.

Object Management Group (2001). Complete uml 1.4 specification.

Prehofer, C. (1997). Feature-Oriented programming: A fresh look at objects. In Prehofer, C., editor, *ECOOP'97 — Object-Oriented Programming*, volume 1241, pages 419–433, Berlin/Heidelberg. Springer-Verlag.

Schulze, M., Fessel, K., Werner, P., and Steup, C. (2008). AVR-halib project website. online, `http://avr-halib.sourceforge.net`. [(online), as at: 25.02.2010].

Spencer, H. and Collyer, G. (1992). #Ifdef considered harmful, or portability experience with c news. In *the USENIX Summer 1992 Technical Conference*, page 185–197. USENIX Association Berkley.

Spinczyk, O., Gal, A., and Schröder-Preikschat, W. (2002). AspectC++: an aspect-oriented extension to the c++ programming language. In *Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications*, pages 53–60, Sydney, Australia. Australian Computer Society, Inc.

Tarr, P., Ossher, H., Harrison, W., and Sutton, S. (1999). N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)*, pages 107–119, Los Angeles, CA, USA.

Unruh, E. (1994). *Prime number computation*. ANSI. ANSI X3J16-94-0075/ISO WG21-462.

Veldhuizen, T. L. (1995). Using c++ template metaprograms. *C++ Report*, 7(4):36–43. Reprinted in C++ Gems, ed. Stanley Lippman.

Performance Characterization of Real-Time Operating Systems for Systems-on-Silicon

Douglas P. B. Renaux[1,3], Rafael E. De Góes[3], Robson R. Linhares[2,3]

[1] Departamento Acadêmico de Eletrônica (DAELN)
Universidade Tecnológica Federal do Paraná (UTFPR)
Av. Sete de Setembro 3165 – Curitiba – PR – Brasil

[2] Departamento Acadêmico de Informática (DAINF)
Universidade Tecnológica Federal do Paraná (UTFPR)
Av. Sete de Setembro 3165 – Curitiba – PR – Brasil

[3] eSysTech – Embedded Systems Technologies
Travessa da Lapa 96, cj 73 – Curitiba – PR - Brasil

douglasrenaux@utfpr.edu.br;
rafael@esystech.com.br;
robson@dainf.ct.utfpr.edu.br

Abstract. *An RTOS is a software component that is used in the majority of the real-time embedded systems. It has a significant effect on the system's performance and reliability. This paper addresses the issue of publishing parameterized performance characteristics of an RTOS in a platform independent manner.*

Concepts of parametric timing analysis were extended to consider the performance of the processor, memory and peripherals in a parameterized way. The proposed method was applied to a commercial RTOS. Validation of the method shows results with a precision better than 10%.

Key-words*: timing analysis, WCET (Worst Case Execution Time), RTOS (Real-Time Operating System) performance characterization, COTS (Commercial Off The Shelf) software component performance.*

1. Introduction

As embedded systems complexity and diversity are constantly increasing, developers face a number of opposing needs in the design cycle. The development time and effort can be reduced with the use of both hardware and software COTS components, however, these components must be appropriately integrated and characterized in advance. High-volume production costs can be reduced with the use of the high-integration Systems-On-Silicon (SOS), however, this poses stringent demands on the design process and tools, demanding the use of hardware-software co-design and heavy use of modeling, simulation and estimation, since the actual hardware is available only near the end of the design cycle.

Current fabrication technologies allow for dies with less than 30 mm^2 to implement over 50 million transistors using below-40 nm processes. Complete systems can be implemented in a single die, including multiple processor cores, RAM, Flash, dynamic

RAM, and several types of generic and special purpose peripheral units that implement communication channels, audio and video processing, interfaces to data storage devices, among many others.

An essential COTS component used in most of current embedded systems designs is the RTOS. It provides an abstraction of the hardware and manages its resources allowing the software development team to focus on the application specific software. Although the RTOS contributes heavily to the performance and robustness of the final system, its performance is usually described very coarsely, mainly citing the context switch times and latencies. In most embedded system's designs, and particularly in those using SOS, a much more detailed performance characterization is required, so that the final system´s performance can be accurately predicted in early phases of the design. Design cycles are too lengthy and costly to allow for a second attempt to achieve the desired performance.

The aim of this paper is to propose a means of describing the performance of an RTOS. The rationale for such a description is fourfold:

1. RTOS execution times and blocking times are essential information to be used in the schedulability analysis of real-time systems;
2. when comparing different RTOSes as alternatives for a design, specific RTOS configurations can be compared from a performance point of view;
3. the application programmer can identify which RTOS services are time consuming or have execution times that are not compatible with given response time requirements;
4. when combined with the performance data of the other components of the system, the performance of the whole system can be accurately predicted and checked against the performance requirements.

It is important to notice that the performance characterization of an RTOS is severely dependant on the HW platform where it runs. The embedded world is characterized by a very large variety of hardware platforms, as opposed to the standardized hardware platforms of general computing (such as PCs and Macs). To encompass such a large variety of hardware platforms we use parameterized generic hardware models, as no RTOS provider would be able to provide performance data for every possible embedded hardware platform and their various configurations.

2. Problem Statement

A typical embedded systems development process is illustrated at a high level in Figure 1. Such a process may be used both when silicon is designed for a specific application as well as when COTS HW components are used. It is important to realize that in this process, HW and SW are designed and implemented concurrently (HW/SW co-design). Actual measurements of performance can only be done on the final HW platform in the Integration phase, however, particularly when specific SOS is designed for this application, a good estimate of system performance must be already available during the System Design phase.

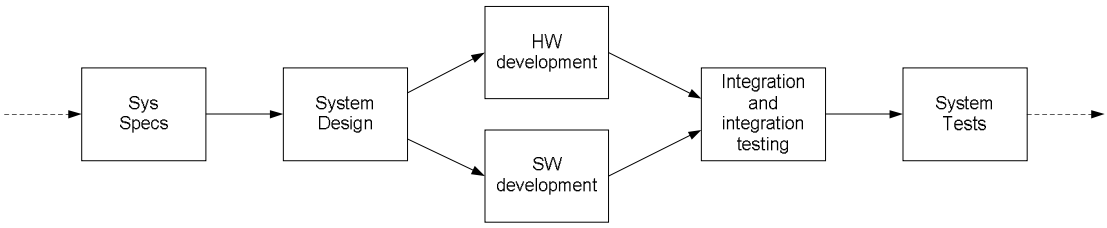


Figure 1 - Typical SOS development process

In Figure 1, the activities preceding Sys Specs and those after System Tests are not shown. In the System Design phase, the identification and characterization of COTS HW components (from libraries and catalogs of HW components), COTS SW components (including RTOS), custom HW component definition (including glue logic and application specific HW), and custom SW component definition (SW wrappers and application specific SW) is done. In this phase, the performance of the final system must be estimated based on the documented performance of its COTS components and estimated performance of the HW and SW components to be developed.

During the design and development phases, models, simulators and evaluation boards may be used to support prototyping and performance evaluations. The level of precision of the simulators and the similarities of the evaluation boards when compared to the actual hardware will define how accurate these performance estimates are.

The problem we are addressing in this research is to identify a way that an RTOS provider (or any other SW component provider) can document the performance characteristics of his product. Since RTOS are used in a wide range of HW platforms, and since the performance of the HW platform strongly affects the RTOS performance, a means is required to parameterize the performance characteristics of the RTOS. In this paper we propose such a means, and we evaluate the proposed means on a commercial RTOS.

3. Literature Review

The determination of the Worst Case Execution Time of real-time software is a subject of study for over a decade. Many conferences deal with this subject. Since 2001 a WCET Workshop is held along to the Euromicro conference. Among the vast literature available, the papers most closely related to our research are described in this section.

Colin, A., and Puaut, I. (2001) analyzed the RTEMS RTOS and identified several problems that made the analysis difficult and imprecise: unstructured code in the RTOS, use of dynamic function calls, and unknown bounds in loops. They reported an 86% overestimation on the WCET.

Sandell, D., Ermedahl, A., Gustafsson, J., and Lisper, B. (2004) reported other types of problems when analyzing the WCET of RTOS services: high dependency of execution times on the RTOS configuration; high dependency of loop bounds on the RTOS state; and high variation of execution times depending on the current mode of the RTOS.

Puschner, P. and Schoeberl, M. (2008) proposed a means of avoiding unpredictability of execution times by rethinking HW and SW implementations: (1) use of single path programming; (2) simplifying HW design; and (3) perform static scheduling of accesses

to shared memory. The achieved gain in predictability came at a high cost of always executing both the “then” and the “else” part of a decision, and discarding the results of one of the parts, as well as significantly reducing the performance of the HW.

Lv, M., Guan, N., Zhang, Y., Deng, Q., Yu, G., Zhay, J. (2009) present a survey on the five most prominent techniques for WCET analysis of RTOS. They also identified three challenges still to be resolved: (1) Parametrization of WCET; (2) Combining the WCET of the application with the WCET of the RTOS; and (3) Combining WCET analysis and schedulability analysis.

A significant step forward was achieved by Altmeyer, S., Hümbert, C., Lisper, B., and Wilhelm, R. (2008) who developed a parametric timing analysis. Instead of the tradicional way of representing the WCET of a service by a single value, they represent it by a formula that includes the service call parameters that affect the execution time. The dependency of the RTOS state was not modeled.

4. Proposed RTOS Performance Modeling

The aim is to characterize the performance of the kernel. To do that in a broad and precise manner the following information is required:

1. Execution characteristics of each service on a generic hardware platform. This is processor’s architecture specific since a change to the architecture implies in changes the machine code.
2. A model of how the service calls arguments and the RTOS state affect the service’s execution time.
3. A characterization of the memory regions that are used and their latencies, for single read, single write, burst read and burst write.
4. A model of blocking during the execution of each service. Blocking, if it occurs at all during calls to a given service, can be in forms such as: masking all interrupts, masking specific interrupts, and preventing context-switches, i.e. preventing preemption.

The RTOS performance characterization proposed here is based on a parameterized generic hardware model (Figure 2). Hence, all performance data that is provided is dependent of the parameters of the hardware, such as clock frequency, memory latencies, and peripheral latencies.

4.1. Performance Parameters Definition Process

The process to be used to determine the performance parameters of an RTOS is depicted in Figure 3. Starting with the source code of the RTOS under analysis, a static analysis is performed to determine the path that determines the worst-case execution time of each service. Then, a test case is build that exercises this path. This test case is executed and its execution is logged, at instruction level. The log, or execution trace, is then analyzed to extract information about the number of accesses to each peripheral or memory section (PRi and MRj in Figure 2). The sections of source code that cause blocking or that are dependent on arguments or state are identified during the static analysis and their execution parameters are identified as well.

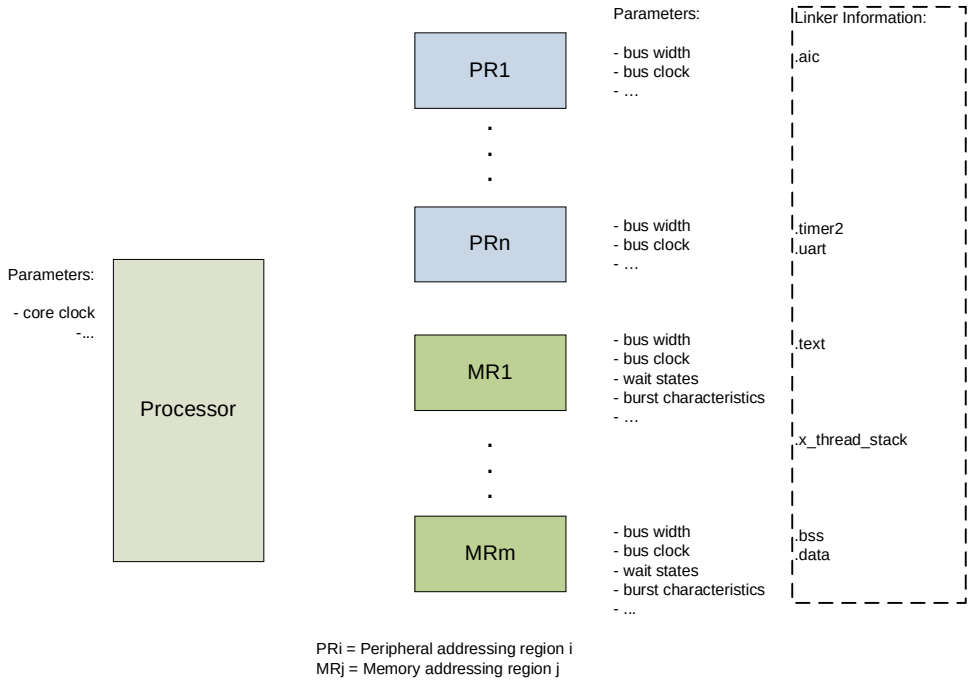


Figure 2 - Parameterized Generic Hardware Model

The process described in Figure 3 is executed by the RTOS provider (or any other SW component provider) to obtain the performance characterization of the RTOS. The use of this information is described in Section 4.2.

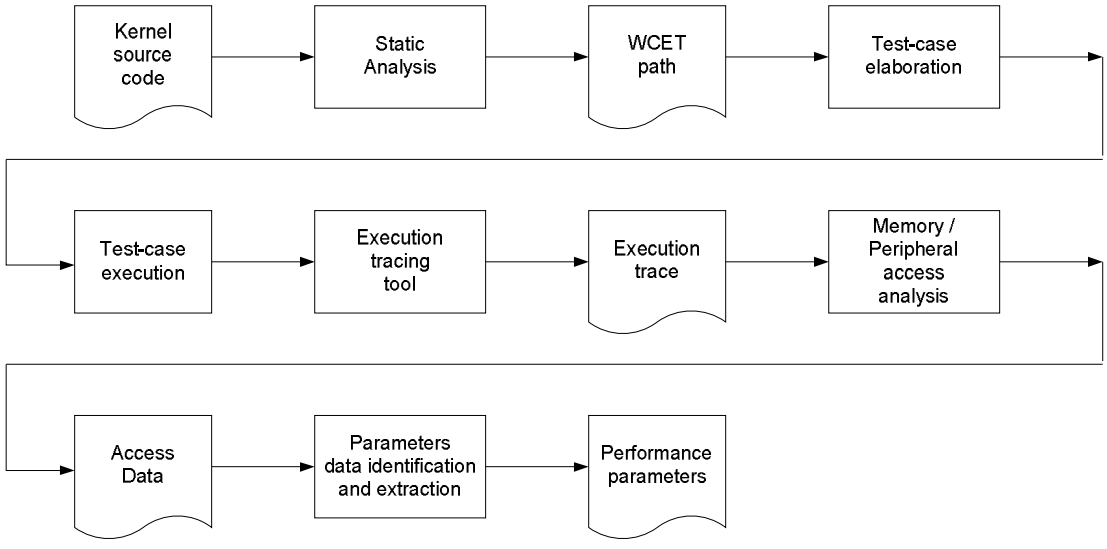


Figure 3 - Performance parameters definition process

This process combines static and dynamic techniques, aiming at obtaining the best possible results from each one. In our validation (Section 5), we used LDRA Testbed [LDRA Testbed (2010)] and PERF [Renaux, D. P. B. ; Góes, J. A. ; Linhares, R. R. (2002)] for the static analysis; here, the possible execution paths are extracted and the WCET path is identified. Then, a SW developer elaborates a test case that exercises the

WCET path. This test case is executed on real hardware, or on a instruction level simulator. The execution is monitored by a trace tool that records the execution trace. A Segger´s J-Trace unit [Segger(2010)] was used as well as the trace recording functionality of the IAR´s EWARM IDE [IAR(2010)].

At this stage of our research, we are evaluating the proposed method; hence, there are no tools yet to support the analysis of the trace. As such, the two final steps were performed manually. The execution traces of the service calls range from less than ten instructions execution to around 600 instructions. This is the case of a call to CreateThread, which is analyzed in Section 5.

4.2. System Performance Estimation

The performance of the final system can be estimated by combining information from three different sources:

1. The RTOS provider. Using the process depicted in Section 0, the RTOS provider publishes the performance parameters of all the RTOS services as well as RTOS internal activities.
2. The Hardware designer. During the design phase the hardware designer identifies the performance characteristics of the processor as well as the access characteristics to each memory/peripheral region.
3. The Software integrator. He provides the mapping between the linker sections (listed in the RTOS performance characterization) and the corresponding physical memory/peripheral regions.

Once these three sets of information are available, it is possible to estimate the WCET of each service of the RTOS on the final system, as well as the WCET of the internal activities of the kernel, such as the timer interrupt handler.

Furthermore, it should be noted that the RTOS traces change for every processor architecture, and for different configurations of the RTOS (if this is the case). Hence, the RTOS provider must perform this analysis, and publish their results, many times. For the case described here, the traces are for the X Real-Time Kernel [eSysTech(2008)] configured for ARM7TDMI processors. Hence, these traces represent the execution of this RTOS on any HW platform based on this processor core.

5. Validation

The RTOS used as testbed in this research is the X Real-Time Kernel, developed by eSysTech Embedded Systems Technologies. A technical cooperation agreement between UTFPR and eSysTech resulted in a long term collaboration between the LIT (Laboratory for Innovation and Technology in Embedded Systems) at UTFPR and eSysTech. The X Real-Time Kernel, or simply X, is representative of microkernels used in deeply embedded systems. It is basically composed of the following modules: microkernel, hardware abstraction layer (X-HAL), shell, event tracing, TCP/IP stack, USB stack, FAT 16/32 and graphics library. The structure of this kernel is depicted in Figure 4 – see eSysTech (2008).

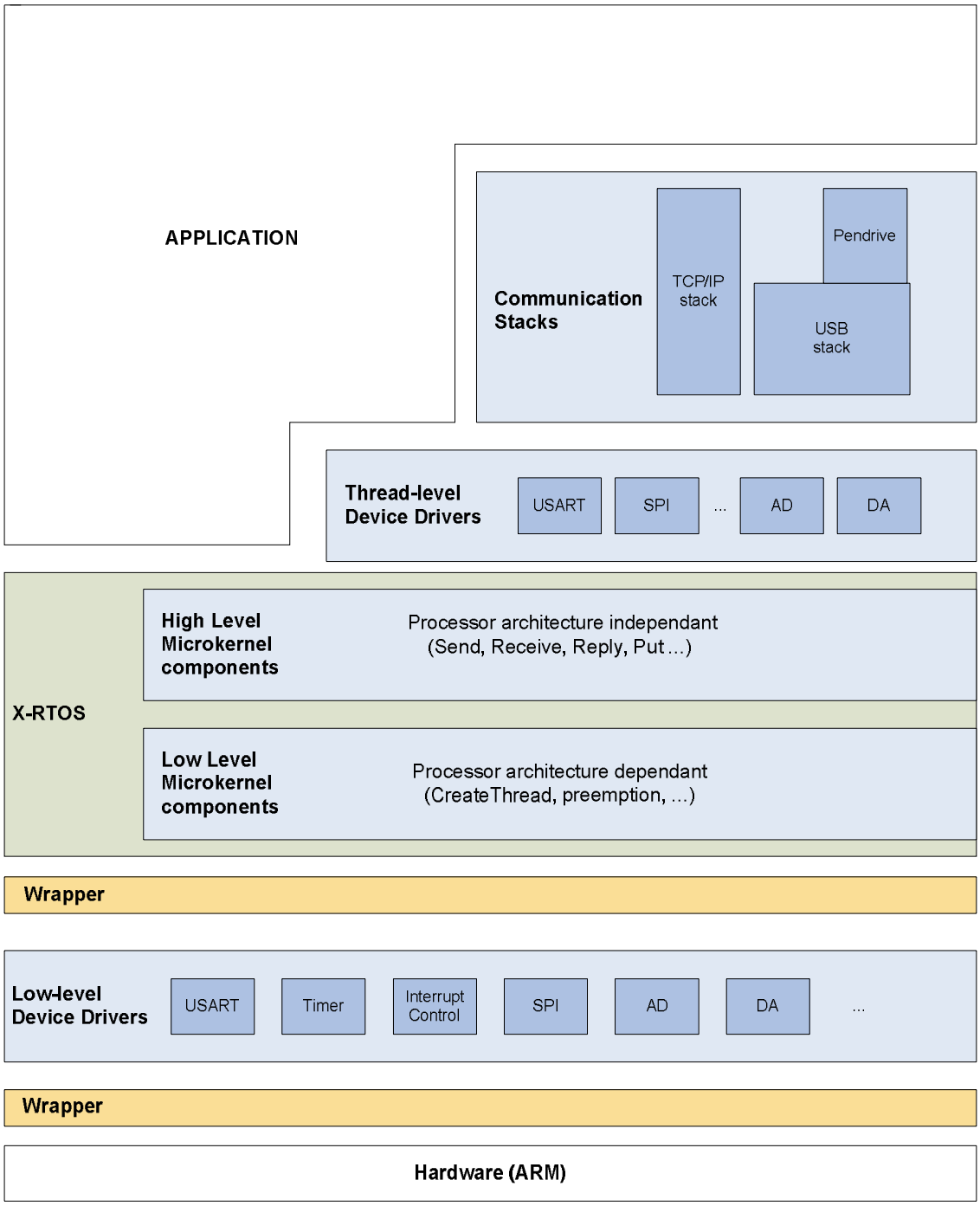


Figure 4 - X Real-Time Kernel Structure

X is being used extensively in embedded systems designed by eSysTech and by its customers. Concerning the first case (systems designed by eSysTech), the company reports [eSysTech(2010)] that, in given applications, over 95% of the embedded code is part of the kernel, hence less than 5% of the code needs to be developed to produce a new system. In such a case, where the application logic is relatively simple compared to the RTOS functionalities (scheduling, USB stack, etc) the latter plays an even more significant role in determining the performance of the system.

5.1. Performance Characterization of the CreateThread Service Call

As a result of this research, the characterization of all calls to the X's microkernel (called the microkernel´s methods in the X literature) is being performed. One of such methods was selected to be presented here: CreateThread. It concerns access to the internal data structures of the microkernel, as well as to the thread's stack. It is one of the most complex and lengthy of the microkernel's methods. It was selected as representative of the effectiveness of the proposed performance characterization.

Definition of the CreateThread Service Call:

```
TId X::CreateThread(
            void (* t_main) (uint32_t, uint32_t),
            uint32_t arg1,
            uint32_t arg2,
            const char * name,
            uint32_t stack_size,
            uint32_t put_queue_size,
            uint32_t config,
            uint32_t priority)
```

Following the process described in Section 4.1, the first activity is the static analysis. The flowgraph of CreateThread was extracted by the LDRA Testbed tool (Figure 5). The analysis of this flowgraph indicates no timing dependency to any of the parameters of the call. The only timing dependecy is to the state of the kernel's internal heap: to the current number of segments present in a linker section named x_heap. A test case is elaborated aiming at creating a given number of segments in the heap before the call to CreateThread is performed. The test case is then executed and its trace is recorded.

The kernel accesses the following linker sections, i.e. logical addressing spaces that are mapped to physical addressing spaces at link time are:

.iram_text: a code section with functions that require fast execution times;
.text: code section with most of the functions of the kernel;
.const: read only data section;
.stack: section for the stack;
.x_heap: a heap section used only for the internal data structures of the kernel.

The trace is then analyzed to identify its sections and the characterize the accesses done in each section. A trace section is a part of the trace that has the same repetition and blocking characteristics. The table below presents the result of the analysis of the trace of the execution of the test case of CreateThread. It is divided in three sections: the first is executed once without blocking; the second is executed once with blocking (interrupts are disabled), and the third is executed N times and also with blocking (again interrupts are disabled). The number of executions of the third section (N) is given by the number of segments in the x_heap.

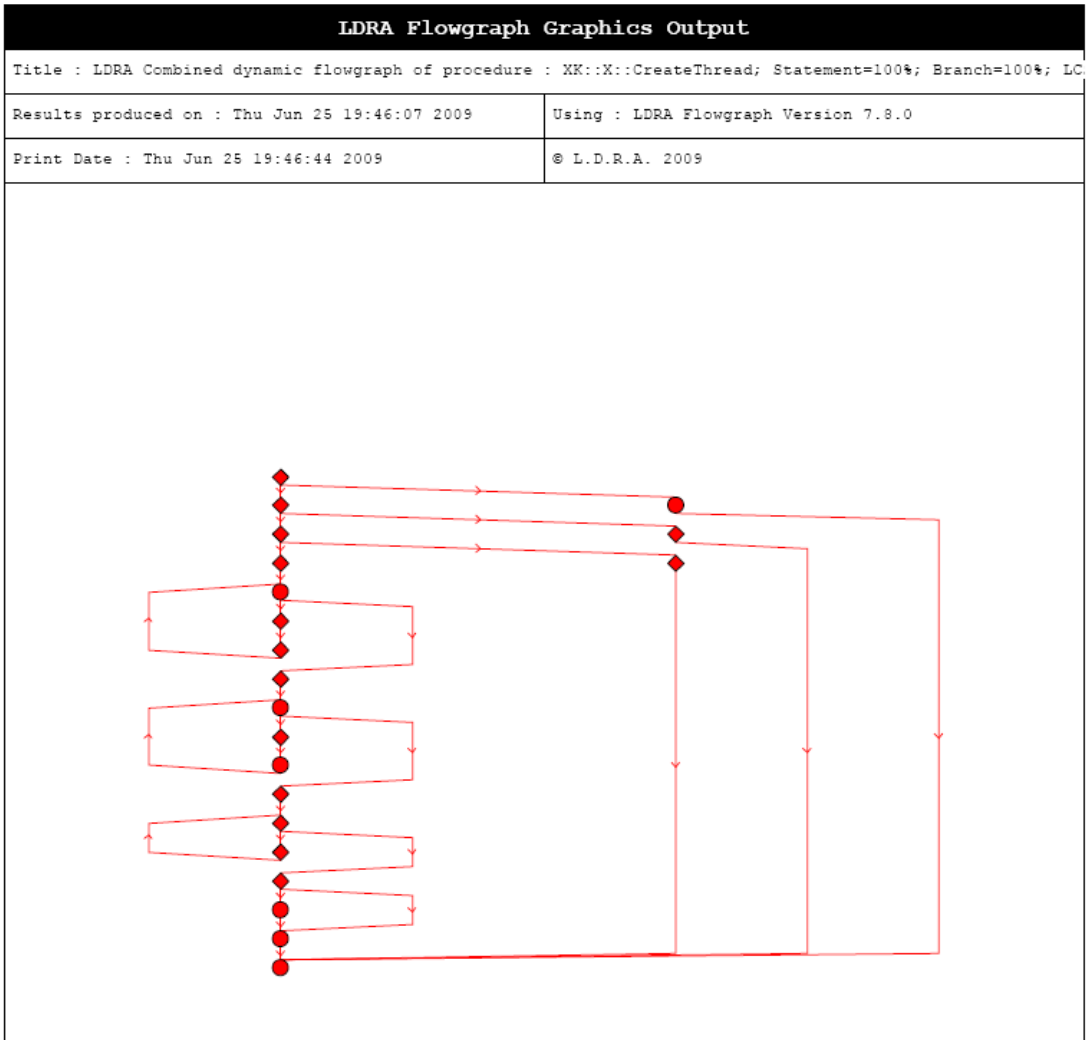


Figure 5 - LDRA Testbed Static Analysis - Flowgraph of CreateThread

The process in Section 4.1 finishes by publishing the following performance parameters for CreateThread. The values in columns “i” to “w8” are the number of accesses to each of these linker sections in each of the trace sections.

Performance characterization for the sections of CreateThread:

Sec	Rep	Block	i	t	c	s	b	r32	w32	r8	w8
1	1		3	13	1	17	4	1	0	0	0
2	1	Y	49	268	7	32	54	12	36	19	25
3	N	Y	0	18	0	0	2	2	0	0	0

Where:

Sec = code section of CreateThread; each section has different repetition and blocking charateristics.
Rep = number of times the execution of this section is repeated (N = represents the current number os segments in kernel’s internal heap).

Block = indicates if this section of code blocks preemption by disabling interrupts.

Logical sections (also known as linker sections):
i = iram_text – frequently used kernel code, usually allocated to fast memory;
t = text – code area;
c = const – constants in code area;
s = stack;
b = number of branches (jumps);
r32 = 32-bit wide read accesses to the x_heap area;
w32 = 32-bit wide write accesses to the x_heap area;
r8 = 8-bit or 16-bit wide read accesses to the x_heap area;
w8 = 8-bit or 16-bit wide write accesses to the x_heap_area.

5.2. Performance Validation

Given the performance parameters from Section 5.1, the values of WCET of a call to CreateThread were estimated for three different hardware platforms using the section mapping information provided by the software integrators of each test program (one test program per hardware platform) and the hardware access times provided by each hardware designer. The execution times were also measured on real hardware, since the three hardware platforms are available. The comparison of the estimated and measured results are shown below.

In these experiments, the call to CreateThread was performed when the kernel's heap was fragmented into 13 segments. Hence, N = 13 for the third section.

Hardware platforms description:

1. ARM7TDMI@72MHz with 16 MBytes of external 32-bit wide SDRAM and 32KBytes of internal SRAM;
2. ARM7TDMI@72MHz with 64KBytes of internal SRAM;
3. ARM7TDMI@72MHz with 1 MByte of external 16-bit wide SRAM and 32KBytes of internal SRAM.

Test on hardware platform 1:

	i	**t**	**c**	**s**	**b**	**r32**	**w32**	**r8**	**w8**	
Accesses	52	515	8	49	84	39	36	19	25	
Access time	14	14	70	70	250	200	200	200	200	ns
Total time	728	7210	560	3430	21000	7800	7200	3800	5000	ns

Total estimated execution time: 56,728 ns (addition of values in last row).

Measured execution time: 51,111 ns

Test on hardware platform 2:

	i	t	c	s	b	r32	w32	r8	w8	
Accesses	52	515	8	49	84	39	36	19	25	
Access time	14	14	14	14	45	14	14	14	14	ns
Total time	728	7210	112	686	3780	546	504	266	350	ns

Total estimated execution time: 14,182 ns (addition of values in last row).

Measured execution time: 14,137 ns

Test on hardware platform 3:

	i	t	c	s	b	r32	w32	r8	w8	
Accesses	52	515	8	49	84	39	36	19	25	
Access time	14	160	160	180	450	160	200	80	120	ns
Total time	728	82400	1280	8820	37800	6240	7200	1520	3000	ns

Total estimated execution time: 148,988 ns (addition of values in last row).

Measured execution time: 146,400 ns

An analysis of these results show that the estimated performance is both safe (estimated execution times are never lower than actual execution times) and have a low overestimation (maximum of 10%). When the memory accesses are more predictable, such as the case of internal SRAM, the overestimation was lower than 1%.

At the current stage of development, the effects of the SDRAM access buffers and cache memories are not modeled. Once this models are developed and included in our performance characterization data, we expect to achieve even better results.

6. Conclusion

This paper presents a significant contribution of considering the performance of hardware components in the characterization of the performance of an RTOS. This was achieved in a three step process: (1) the RTOS provider publishes the **parameterized performance characteristics** of his RTOS; (2) the HW designer provides the access time characterization of a specific hardware platform; and (3) the SW integrator provides the mapping of the logical sections listed by in the RTOS characteristics to the physical devices of the actual system's hardware. From the combination of the information provided by these three sources precise performance estimations can be obtained.

This research extends previous parametric timing analysis by considering the effects of the internal RTOS state and data structures and the characteristics of hardware components to obtain more accurate performance estimations.

To illustrate the method, a representative kernel service was selected: CreateThread. The same method applies to the other services of the kernel.

The process presented here was experimented on microcontrollers using an ARM7TDMI core. These cores do not use cache memories, hence, these were not considered in the model so far. Future versions of the proposed model will consider other architectures, such as the Cortex-M4 and the Cortex-A8 and will include the effects of the cache in the performance characterization. Also, we are evaluating the development of tools that would automate some of the activities (mainly Performance Parameters Identification and Extraction) that are currently performed by hand.

References

Altmeyer, S., Hümbert, C., Lisper, B., and Wilhelm, R. (2008) "Parametric timing analysis for complex architectures" In: The 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA).

Colin, A., and Puaut, I. (2001) "Worst-case execution time analysis of the RTEMS real-time operating system" 13th Euromicro.

eSysTech (2008) X Real-Time Kernel, Programmer´s Manual, 2008.

eSysTech (2010) "Code functionality assessment and performance measurements for the ISPI project", Internal Report, Feb 2010.

IAR (2010) "IAR Embedded Workbench for ARM" http://www.iar.com/website1/1.0.1.0/68/1/

LDRA (2010) "Software Development and Testing with LDRA Testbed" http://www.ldra.com/testbed.asp.

Lv, M., Guan, N., Zhang, Y., Deng, Q., Yu, G., Zhay, J. (2009) "A Survey of WCET Analysis of Real-Time Operating Systems", In: 2009 International Conference on Embedded Software and Systems. IEEE.

Puschner, P. and Schoeberl, M. (2008) "On composable system timing, task timing, and WCET analysis" In: WCET 2008.

Renaux, D. P. B. ; Góes, J. A. ; Linhares, R. R. (2002) "WCET Estimation from Object Code implemented in the PERF Environment" In: Second International Workshop on Worst-Case Execution Time Analysis, 2002, Viena. WCET'2002 - 2nd International Workshop on Worst-Case Execution Time Analysis, 2002. v. 1. p. 28-35.

Sandell, D., Ermedahl, A., Gustafsson, J., and Lisper, B. (2004) "Static timing analysis of real-time operating system code" In: 1st International Symposium on Leveraging Applications of Formal Methods.

Segger (2010) "J-Trace ARM". http://www.segger.com/cms/j-trace-arm.html

12th Brazilian Workshop on
Real-Time and Embedded Systems

♦

Technical Session 4

Sensor Networks and Wireless Communication

Coordination Mechanism and Customizable Hardware Platform to Provide Heterogeneous Wireless Sensor Networks Support*

Edison P. de Freitas[1,4], Rodrigo S. Allgayer[2], Tales Heimfarth[3], Flávio R. Wagner[4], Tony Larsson[1], Carlos E. Pereira[2], Armando M. Ferreira[5]

[1]School of Information Science, Computer and Electrical Engineering
Halmstad University, Halmstad – Sweden

[2]Electrical Engineering Department – Federal University of Rio Grande do Sul
Porto Alegre – Brazil

[3]Computer Science Department – Federal University of Lavras
Lavras – Brazil

[4]Institute of Informatics – Federal University of Rio Grande do Sul
Porto Alegre – Brazil

[5]Defense Engineering Graduate Program – Military Institute of Engineering
Rio de Janeiro – Brazil

edison.pignaton@hh.se, allgayer@ece.ufrgs.br, tales@dcc.ufla.br,
flavio@inf.ufrgs.br, tony.larsson@hh.se, cpereira@ece.ufrgs.br, armando@ime.eb.br

Abstract. *This paper presents an effort to support emerging Wireless Sensor Networks applications composed by different types of sensor nodes. The work is composed by two parts, in which the first is dedicated to provide cooperation abilities to sensor nodes, while the second is a customizable hardware platform intended to provide different types of sensor nodes, from those more resource constrained up to the resource-rich ones. A description of a testbed demonstrator of the proposed system is provided and comparisons with previous published simulation results denote the feasibility of the proposal.*

1. Introduction

A number of emerging applications are being developed having as basis Wireless Sensor Networks as driving technologies. An important feature of the new systems that implement these applications is the usage of different types of sensors working in a unique network, cooperating in order to accomplish with users' expectations. An example of such applications is area surveillance systems, which use sensor nodes with different sensing, computing and mobility capabilities to gather data of an area of interest.

The main issues in developing heterogeneous sensor networks are: (i) support for cooperation among heterogeneous nodes; and (ii) customization of sensor nodes [Erman et al. 2008]. The former is related to concerns such as message exchange synchronization, QoS requirements management, task (re-) allocation, network adaptation,

*E. P. Freitas thanks the Brazilian Army for the grant to follow the PhD program in Embedded Systems at Halmstad University in Sweden, in cooperation with UFRGS in Brazil.

among others. The later is related to the diversity of node platforms, which may be built upon very distinct hardware components controlled by very different pieces of software.

Considering (i), the use of a middleware services represent a suitable approach to address the mentioned concerns, since they can integrate the technologies used in different nodes by means of common communication interfaces and cooperation mechanisms. Regarding (ii), customizable architectures can be very useful to build platforms for different sensor network nodes, from the very simple to the more sophisticated ones. This kind of architecture can provide a common base capability for all nodes. However, for nodes that need more advanced capabilities, the required resources can be incorporated. Hence, even though all nodes have the same base capability, some of them could be equipped with additional resources, thus making the sensor network more powerful due to this allowed heterogeneity.

This paper presents testbed results of a flexible and adaptable platform infrastructure intended to support heterogeneous sensor network applications composed by static sensor nodes on the ground and mobile sensors carried by Unmanned Aerial Vehicles (UAVs). It is based on the proposal of (i) a flexible coordination mechanism [Freitas et al. 2009], and (ii) on a customizable hardware architecture aimed for sensor nodes, called FemtoNode [Allgayer et al. 2009]. The key idea is to use this customizable platform to deploy different kinds of sensor nodes, from very tiny and resource constrained up to more sophisticated ones. Both types of nodes run a common coordination software, which provides the desired interoperability that will allow the cooperative work among different sensor nodes. In the demonstrator presented, nodes may be built upon the FemtoNode architecture and alternatively upon nodes with another hardware platform, namely SunSPOT [Microsystems 2010].

The remaining of this paper is organized as follows: In Section 2 the application scenario is highlighted, characterizing the network heterogeneity. Section 3 presents a pheromone-based coordination mechanism used to promote the collaborative work among static and mobile sensors. In Section 4, the FemtoNode customizable hardware architecture is described. Section 5 presents a description of a case study and highlights previous obtained simulation results, while Section 6 presents demonstrator and the practical results achieved with it, as well as a comparison with the previous simulations. Section 7 discusses related work in the area. Finally, Section 8 draws concluding remarks and gives directions for future work.

2. Motivation: Application Scenario and Network Heterogeneity

In the following, heterogeneity means that nodes in the network may have different sensing capabilities, computation power, and communication abilities. Additionally, it means that they may run on different hardware and operating systems. Therefore, such sensor networks are made up of low- and high-end nodes. Moreover, sensor nodes may have fixed positions or be able to move, being carried by UAV platforms, which can also vary from very small, as in [Walter et al. 2005], up to huge aircraft platforms, like GlobalHawk [Leonard and Drezner 2002].

Low-end sensor nodes are those with constrained capabilities, for instance piezoelectric resistive tilt sensors, with limited processing support and communication resource capabilities. High-end sensor nodes include powerful devices like radar, high definition

visible light cameras, or infrared sensors, which are supported by moderate to rich computing and communication resources.

Mobility, as mentioned, is another important characteristic related to the heterogeneity addressed in this work and requires special attention. Sensor nodes can be statically placed on the ground or can move on the ground or fly at some altitude over the target area in which the observed phenomenon is occurring. Figure 1 graphically represents the idea of the three heterogeneity dimensions considered in this work, in which each axis represents one of the considered characteristics.

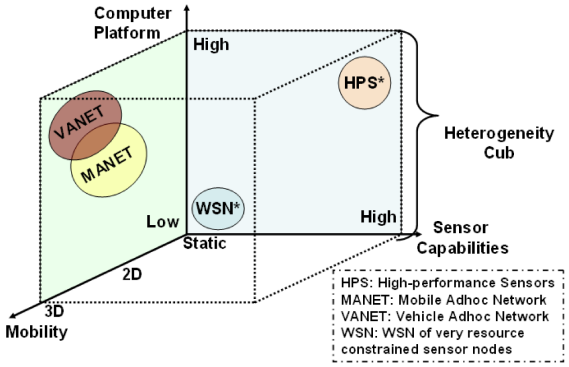


Figure 1. Heterogeneity Dimensions.

The reason for heterogeneity in the sensor nodes is to support a large range of applications that deal with very dynamic and challenging scenarios, which require different types of sensor capabilities in order to gather a wide diversity of data. Moreover, these different scenarios may require adaptations in the network, in terms of choosing suitable sensors for the tasks at hand as well as feasible QoS parameters, among others. The decisions related to these issues need additional data to be supported [Erman et al. 2008].

In order to illustrate the above idea, suppose that a network has the mission of providing a certain kind of information during a given period of time over an area of interest. The network must be able to choose a better alternative, among the set of all available options, in order to accomplish the mission. For example, an area surveillance system may receive the mission to observe if certain types of vehicles that are not allowed to pass through the surveyed area make any such violation and report if that is the case. To perform this in an efficient way ground sensors are set to alarm in the presence of unauthorized vehicles. If these sensors are not capable to confirm the violation, the alarms have to be delivered to more sophisticated sensors, carried by UAVs, in order to these last ones check and confirm the possible threat. Moreover, as the UAVs may be equipped with different types of sensor devices, they also should be able to decide which among them is the most suitable to respond to a given alarm. For example, UAVs equipped with visible-light cameras may provide poor results if employed in areas where the weather conditions are bad, e.g. in foggy or cloudy areas.

3. Pheromone-based Coordination Approach

The coordination strategy used in this work to make mobile sensor nodes cooperate with static sensor nodes is based on pheromone traces handed over by the mobile sensors to the static ones. Artificial pheromones are usually applied to distributed coor-

dination by means of stigmergy, the indirect communication using environment cues [Bonabeau et al. 1999]. A pheromone trail is deposited in the environment when the entities are moving.

The pheromone provides information to other entities when they pass over it. Artificial pheromone also looses its strength along the time, modeling the evaporation of the real pheromones. In the UAV research field, pheromones are used to guide the movement of UAV swarms, for instance in surveillance and patrolling applications [P. Gaudino et al. 2003, Sauter et al. 2005].

Differently from other existing approaches, in the present work pheromones are used to guide the selection and assignment of a suitable UAV to handle an alarm issued by a ground sensor node. When an alarm is issued by the detection of a target, the network is responsible for selecting an appropriate UAV to respond to the alarm. This is performed by routing a given alarm to the UAV that has the strongest pheromone trace over the area. Having this information, the UAVs will base their movement decisions in a way to respond to the received alarms. This strategy is called here heuristic-P. The main difference from the mentioned approaches is that they rely on global UAV-network connectivity to spread information about the pheromone map, while the present approach explores a local connectivity among UAVs and ground sensor nodes.

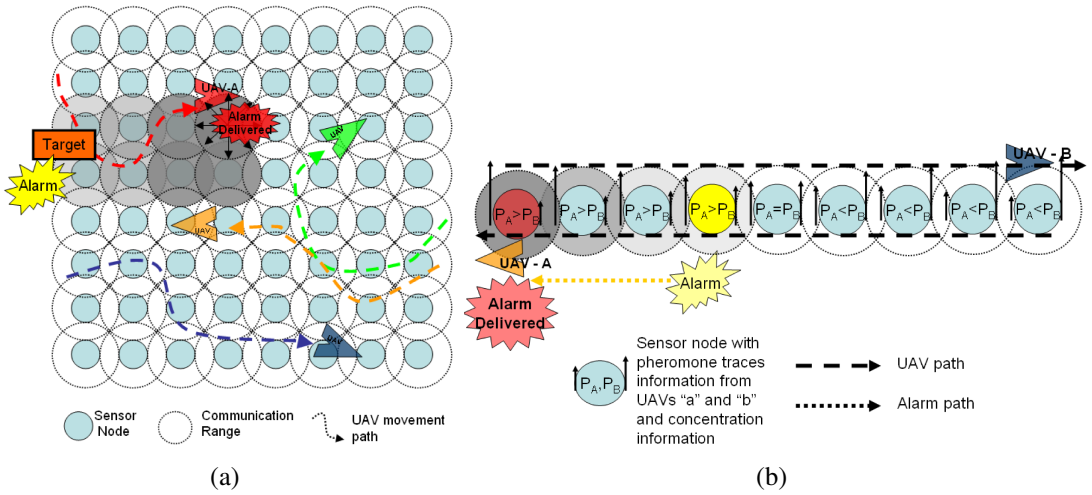


Figure 2. (a)Illustrative scenario for the pheromone strategy (b)Choice of a UAV based on the pheromone strategy.

Following the above outlined principles, the UAVs that are not engaged in the handling of any target will leave pheromone traces over the area which they cross. This pheromone trace is represented by a piece of information that is taken by the ground sensor nodes that are deployed in the area through which the UAVs have passed. When a target is detected by a ground sensor node, an alarm is issued. The decision about which UAV that will handle the potential target indicated by the issued alarm will be taken by the ground sensor nodes, by routing the alarm in the direction that points to the UAV which has the strongest pheromone trace over that area of the network. This process just considers the pheromone trace handed over by the UAVs to ground sensor nodes. This means that the only parameter taken into account is the time interval since a UAV passed by that specific location. Heuristic-P is inspired in [Heimfarth and Janacik 2008], which

presents a pheromone-based strategy to migrate services in a sensor network, in which the pheromone concentration determines the places where the services are required.

In heuristic-P, instead of services, alarms are moved through the network following the pheromone concentration. Figure 2(a) presents a scenario that illustrates the strategy. A ground sensor node in the left border of the area detects a target. Then it issues an alarm, which is received by its neighbors. However, only those which have pheromone information about a UAV stronger than that of the alarm issuer will forward the alarm. This way, the alarm will follow a path to the closest UAV, which is represented in the figure by the shaded sensors, until the alarm delivery.

Figure 2(b) illustrates the choice of the strongest pheromone trace to be followed by an issued alarm. It is possible to observe that the alarm follows the strongest trace, which corresponds to UAV-A, until its delivery to this UAV. The arrows illustrated besides each sensor node represent how strong the pheromone of each UAV is. As it is possible to see, the pheromone level of UAV-A is increasing to the left, while the pheromone level of UAV-B is increasing to the right.

When an alarm reaches the UAV indicated by the strongest pheromone trace, if this UAV is not engaged in the handling of another alarm it sends a confirmation message to the node that had delivered the alarm. If the suggested UAV is already engaged in another alarm, the current alarm follows the second strongest pheromone trace to find another UAV to engage. When an idle UAV detects a new target, it takes the responsibility for handling it. In case that the UAV is already busy with another alarm response mission, it relays the incoming alarm that will be routed to another UAV, according to the pheromone-based heuristic-P strategy explained above.

In order to increase the robustness of the proposal, in case an alarm is issued by a node that has no pheromone trace, a direction is randomly chosen and the alarm is sent in that direction until it finds a pheromone trace. When the trace is found, it follows the trace as explained above. This situation is more likely to occur in the initialization of the system, especially in cases in which the number of UAVs deployed in the system is very low with regard to the area under surveillance.

Figure 3 presents an example of how an alarm issued by a sensor node (Figure 3-A) is routed through the network, following the pheromone traces (Figure 3 from A to D), until it is delivered to a UAV (Figure 3-E). The pheromone traces in the nodes are represented by the numbers in the center of the circles representing the ground sensor nodes in the figure. The smaller the number is, the stronger the pheromone. This translates the idea of the time past since a ground sensor node received the last pheromone beacon from a UAV. When a ground sensor node receives this pheromone beacon, it sends this information to its neighbors with a pheromone one point weaker (a number one unit greater than the one representing the node's pheromone information). This is an indirect beacon that helps the other nodes find the traces to route the alarms. The nodes that receive the indirect beacons do not forward it. The symbol ∞ means that the node has no pheromone trace, i.e. the last beacon (directly from a UAV or indirectly from another ground node) was received a long time before, above a tunable threshold. The number representing the pheromone is periodically incremented, indicating that the pheromone trace becomes weaker when time elapses, until disappearing (become ∞).

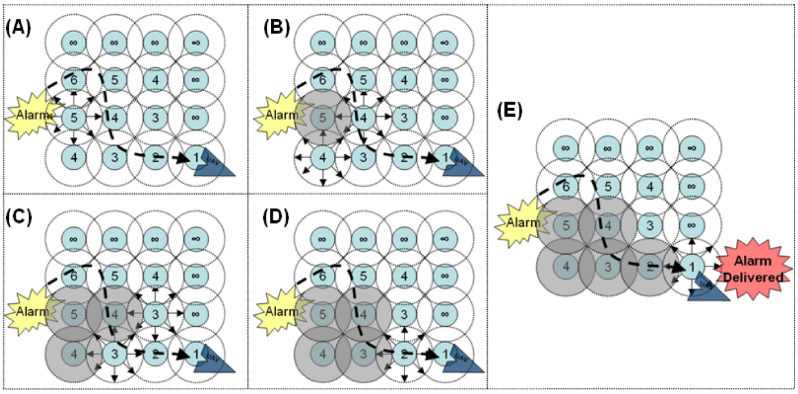


Figure 3. Alarm Routing and Delivery.

4. FemtoNode - Wireless Sensor Architecture

The architecture of a sensor node aims at efficiently supporting specific application needs. It requires a dedicated processing module, including a wireless communication interface, which meets both energy and performance requirements, as well as respects footprint constraints. The fact that application requirements as well as environment and other operational conditions may change during system run time imposes a major challenge [Hinkelmann et al. 2007]. In this context, the use of reconfigurable hardware [Garcia et al. 2006] appears as an interesting alternative.

Therefore, a customizable sensor node called FemtoNode is proposed. It contains a customizable ASIC and a wireless communication interface, which are configured according to application requirements.

The nodes use the RT-FemtoJava processor [Ito et al. 2001], a stack-based microcontroller that natively executes Java byte-codes. It implements an execution engine for Java in hardware, through a stack machine that is compatible with the specification of Java Virtual Machine. The customized application code is generated by the Sashimi design environment [UFRGS 2006]. The code also includes a VHDL description of the processor core and ROM (programs) and RAM (variables) memories. The Sashimi environment has been extended to incorporate an API that supports concurrent tasks, implementing the RTSJ standard [Wehrmeister et al. 2006].

As RT-FemtoJava is customizable, its code can be optimized according to the application requirements, reducing the occupied hardware area and also the energy consumption. The customizable hardware architecture of the FemtoNode allows the use of the sensor node as either a low- or high- end node. If the application requires higher performance resources to handle more complex data, such as image processing, additional resources can be included in the FemtoNode implementation. However, if the application is aimed at processing simple data, such as those from presence sensors, a reduced set of resources is used in the processor. This feature is important for the sensor node, because energy consumption is a great concern in wireless sensor networks, due to the nodes' limited energy resource. Besides, reducing the unused resources during its synthesis the sensor node architecture allows its implementation in reconfigurable circuits with fewer available logical units, which is a feature that provides a larger application portability between different reconfigurable architectures with fewer available resources.

In the current implementation, the FemtoNode includes a wireless transceiver of Texas Instruments CC2420, which utilizes the IEEE802.15.4 standard communication protocol targeted to wireless sensor network applications with a low data rate. A module adapter described in VHDL implements the interface with the wireless transceiver. The module uses data and address buses to communicate with the processor, performing the exchange of data and allowing the transceiver parameters configuration.

As the data transfer rate from the wireless transceiver is low, compared to the processor frequency, the wireless communication module implements a buffer to store data, preventing delays while providing the necessary data to the processor. The module uses an interrupt system to inform the processor when a reception occurred.

To facilitate the use of the wireless communication module by the application developers, a communication API has been developed. The Wireless-API abstracts details of the communication media between the sensor nodes, offering a simplified form for the configuration of the data transfer module.

5. Case Study Description and Simulation Results

In order to illustrate the use of the proposed platform infrastructure, including the customizable FemtoNode and the cooperation mechanism, an area surveillance application is studied. In this application, low-end sensors nodes are scattered on the ground along a borderline. In case an unauthorized vehicle crosses the borderline limit, the sensors on the ground issue an alarm which will trigger the use of UAVs, which are equipped with more sophisticated sensors, such as radars or visible light cameras, in order to perform the recognition of the vehicle, and confirm a possible threat. Figure 4 presents this scenario.

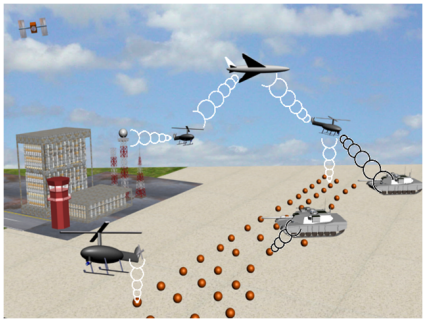


Figure 4. Area Surveillance Application Scenario.

The setup of this system is done in such way that a predefined threshold triggers the activation of the mechanism that issues the alarms. This is done when a static sensor is not able to assess the detected vehicle is or is not a possible threat. So, the usage of a more sophisticated sensor is required and so, the cooperation mechanism has to be employed in order to drive one of the UAV-carried sensor to the area where the alarm was issued.

According to the coordination strategy based on pheromones presented in Section 3, the sensor nodes on the ground route the alarms according to the pheromone trace left by the UAVs, choosing the strongest trace to follow. When the alarm achieves a node close to the UAV, the alarm is delivered. This mechanism addresses several problems related to the communication and interoperation between nodes, such as message routing, controlled delay, delivery assurance, and appropriate node selection to cooperate with.

For the demonstrator implementation presented in the following section, sensor nodes with two different architectures compose the described surveillance system, one based on the FemtoNode and another one on the SunSpot. Each of them includes all necessary resources to meet the requirements of their utilization. Thus, based on the application specifications, a customization of the FemtoNode architecture was implemented. The UAV's architecture is a FemtoNode with a large set of resources, capable of processing a large amount of data. Further details will be presented in Section 6.

5.1. Simulation Results for the Presented Scenario

As already mentioned, the presented scenario was already target of simulation experiments, which provided the results that will be presented in the following. These results will then be compared with the ones achieved with the demonstrator and discussed in Section 6.

Experiments using ShoX simulator [Lessmann et al. 2008] were performed and reported in [Freitas et al. 2009]. The accessed metrics presented in [Freitas et al. 2009] were: 1) the mean response time to the alarms generated in the system; 2) the number of alarms lost, due to communication failures; and 3) the utility in employing a given UAV to handle a given alarm.

For the purposes of this paper, the metrics 1 and 2 are taken into account, as the issue related to the evaluation of the utility in employing a UAV to handle a given target was not implemented so far in the demonstrator.

The simulation setup was the following: The surveillance area has dimensions 10 Km x 10 Km, in which 20,000 ground sensor nodes are randomly deployed with independent uniform probability, 500 meters communication range. Six UAVs of three different types, equally distributed, patrol the area, having a communication range of 1.5 Km and flying at the altitude of 250 meters and with speeds from 100 Km/h up to 120 Km/h. Three different runs were simulated, with one, three, and five targets respectively. The targets can further be of five different types, randomly chosen, with speeds from 50 Km/h up to 80 Km/h.

Figure 5 presents the simulation results in terms of the mean time required to respond to the alarms. Both raw data from each run (total of 20 runs for each number of targets) and the average value (lines with squared dots) are plotted in the figure. It is possible to observe that, in the worst case, the mean time to find a UAV that is idle to engage in the handling of an alarm is around 4 seconds, in the scenario with the maximum number of targets. On the other hand, in the best case, when there is just one target, the time needed to find a UAV is in average less than 1 second. An explanation for this behavior is that it is more probable to find an idle UAV when the number of targets is smaller. This may happen because, when there are more targets, an alarm message may follow a pheromone trace of a UAV that has just engaged in handling a target announced by another alarm, so the alarm must be retransmitted to the network and follow another trace. However, the solution does scale, as the increase in the mean time to find an idle UAV is linear with the increase in the number of targets, as can be concluded by taking the average values for all runs for each number of targets.

The second considered metric evaluates the system efficiency in terms of detecting a target and correctly routing the alarm message to an idle UAV. For all simulation runs,

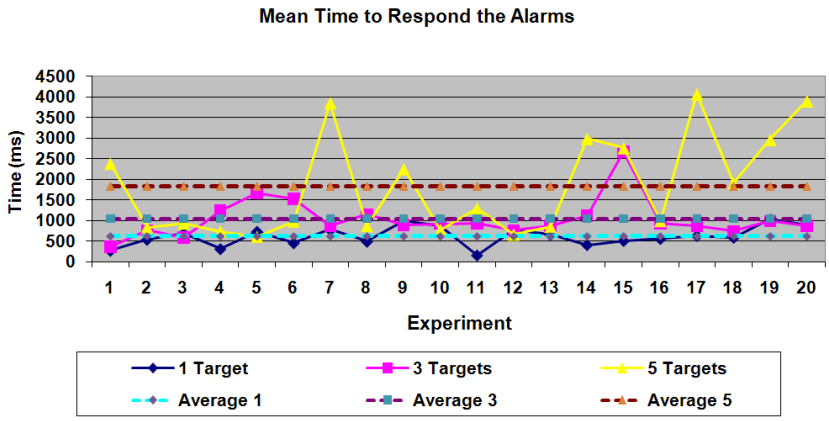


Figure 5. Alarm Response Time Achieve by the Simulation.

no alarm was lost, which means that the system had 100% efficiency for the simulated scenario and correctly found an idle UAV at all occasions when an alarm was issued.

6. Demonstrator Presentation and Results

The simulations reported in [Freitas et al. 2009] showed that the proposed approach works well in the described scenario. However, wireless communications are very sensible to interferences and unpredictable variations. This means that simulation data, such as communication reachability and delays, are not always confirmed in real deployments. This fact motivated the deployment of a demonstrator to assess if the properties of the proposed approach are also observable in a physical implementation.

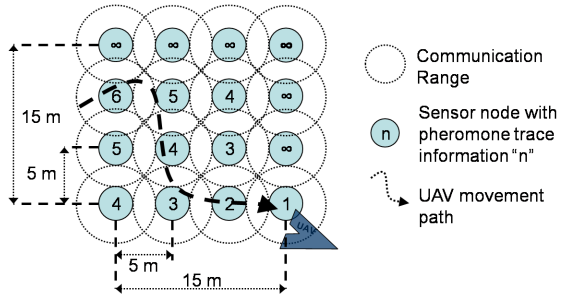


Figure 6. Demonstrator Setup.

The deployed demonstrator is composed as a network consisting of sixteen static ground sensor nodes (nine SunSpots and the others FemtoNodes) and one mobile node (FemtoNode). The ground sensor nodes are equally distributed in a grid in an area of 225 square meters. The mobile FemtoNode, moved manually, represents a UAV that "flies" over this area leaving pheromones over the ground sensor nodes via a periodic beacon message sent to the network. Upon the occurrence of an alarm, the nodes route it in the direction of the nodes with stronger pheromone traces, until it arrives at a node which has communication with the UAV, as explained in Section 3. Figure 6 presents the demonstrator setup. The radio in the nodes was adjusted to provide a communication range of 5 meters, such that the nodes are capable of communicating only with their immediate vertical and horizontal neighbors, which are 5 meters apart, but not with their

diagonal neighbors or any other node in the grid. The mobile node, representing the UAV, has the same communication range configuration as the static nodes.

Twenty runs were performed. In each of them, an alarm was generated by one of the static nodes, randomly chosen, which had to be routed to the UAV according to the pheromone mechanism described in Section 3 and implemented as described above.

In order to stress the network and test these mechanisms, random messages were generated by the static nodes, which competed with the beacon and alarm messages for the utilization of the communication resources.

The evaluated metric with the described testbed was the time to respond to the alarms generated in the system. By obtaining this metric, the delay of one hop communication was calculated and compared with the one achieved in the simulation results described before. Figure 7 presents the time taken by the system to deliver the alarm to the UAV.

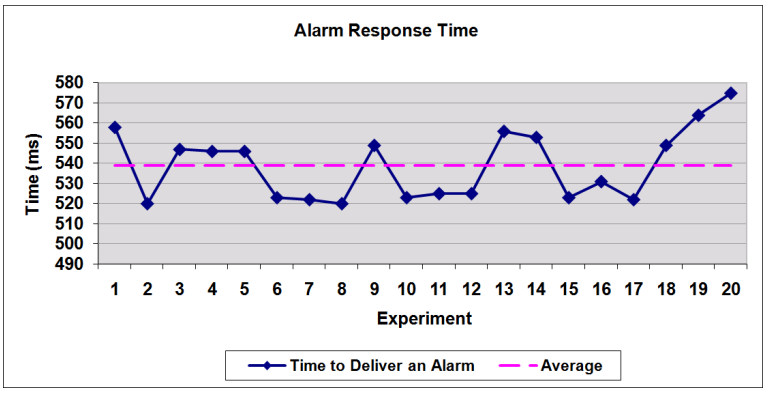


Figure 7. Alarm Response Time Achieved by the Demonstrator.

The average number of hops to deliver the alarm was 5 hops for the 20 runs of the testbed. Taking the average of the time to deliver an alarm, 538.85 ms, and the average number of hops, the average delay calculated is of 107.77 ms in each hop.

Considering the simulation results, taking the worst case scenario, the one with 5 targets, in average the number of hops for an alarm to be delivered was 13.78. Taking the average of worst case scenario, 1,821.65 ms to deliver an alarm, 132.14 ms is the delay for an alarm to be forwarded among the static nodes in each hop.

Comparing the delays obtained from the simulation runs and from the demonstrator, it is possible to observe that they are very close to each other. The delays obtained with the demonstrator are even better than the ones achieved by simulation, which is an evidence of the applicability of the proposed approach.

7. Related Work

AWARE [Erman et al. 2008] is a middleware whose goal is to provide integration of the information gathered by different types of sensors, including low-end sensor nodes in a wireless sensor network and mobile robots equipped with more sophisticated sensors. Our proposal not only addresses heterogeneous sensors and their coordination, but also concerns like QoS, e.g. message delay, which is missing in [Erman et al. 2008].

In [Walter et al. 2005], an approach using digital pheromones to control a swarm of UAVs is presented. The method proposed by the authors uses digital pheromones to bias the movements of individual units within a swarm toward particular areas of interest that are attractive, from the point of view of the mission that the swarm is performing, and away from areas that are dangerous or just unattractive. In the large sense, the pheromone-based strategy used in our work has a similar goal, driving the UAVs to areas of interest. However, differently from their approach, we use the pheromone traces to localize the UAVs when an alarm is issued by a ground sensor node informing an event of interest and then drive the UAVs to the location where the event happened.

In [Caldas et al. 2005] a sensor node was presented incorporating re-configurable hardware resources to improve and to expand the set of features executed by conventional sensor nodes. These features allow the processing of complex events that requires high computing efficiency and accuracy. Our proposal enhances type of flexibility by optimizing the microcontroller architecture by synthesize only the resources that the applications need.

8. Conclusions and Future Work

This paper presented a system solution to provide interoperability and coordination support for heterogeneous sensor networks composed by ground static sensor nodes and mobile sensors carried by UAVs, and a customizable hardware to implement the different types of sensors needed in such networks. The first part of the work is represented by a bio-inspired pheromone- based approach, while the second part is represented by the FemtoNode platform, which provides a support to the development of different types of nodes, customized according to specific requirements.

The assessment of the viability in using of the proposed approach in real networks was done by means of a comparison between results obtained by simulation experiments, representing a large scale scenario, with a small scale testbed demonstrator, which uses the pheromone mechanism and the FemtoNode. The evaluation provided evidences that the proposed solution indeed sounds.

As future works, a large demonstrator is being planned, in which we aim also to evaluate the selection of the utility in employing a given UAV to handle a given target, and like this, be able to compare the additional simulation results reported in [Freitas et al. 2009] with results from a demonstrator.

References

Allgayer, R. S., Götz, M., and Pereira, C. E. (2009). Femtonode: Reconfigurable and customizable architecture for wireless sensor networks. In *Proc. of 10th International Embedded Systems Symposium*, pages 302–309, Langenargen, Germany.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). Swarm intelligence: From natural to artificial systems. New York. Oxford University Press.

Caldas, R. B., Jr., F. L. C., Nacif, J. A., Roque, T. R., Ruiz, L. B., Fernandas, A. O., da Mata, J. M., and Jr., C. C. (2005). Low power/high performance self-adapting sensor node architecture. In *Proc. of 10th IEEE Conf. on Emerging Technologies and Factory Automation*, volume 2, page 976, Catania, Italy.

Erman, A. T., Hoesel, L., and Havinga, P. (2008). Enabling mobility in heterogeneous wireless sensor networks cooperating with uavs for mission-critical management. In *IEEE Wireless Communications*, pages 38–46.

Freitas, E. P., Heimfarth, T., Wagner, F. R., Ferreira, A. M., Pereira, C. E., and Larsson, T. (2009). Evaluation of coordination strategies for heterogeneous sensor networks aiming at surveillance applications. In *Proc. of 8th IEEE Sensors*, pages 591–596, Christchurch, New Zealand.

Garcia, P., Compton, K., Schulte, M., Blem, E., and Fu, W. (2006). An overview of reconfigurable hardware in embedded systems. In *EURASIP J. Embedded Systems*, p. 13–13, New York, USA.

Heimfarth, T. and Janacik, P. (2008). Experiments with biologically-inspired methods for service assignment in wireless sensor networks. In *IFIP Intl Federation for Information Processing*, pages 71–84, Boston, USA. Springer.

Hinkelmann, H., Zipf, P., and Glesner, M. (2007). A domain-specific dynamically reconfigurable hardware platform for wireless sensor networks. In *Int. Conf. on Field-Programmable Technology*, pages 313–316.

Ito, S. A., Carro, L., and Jacobi, R. P. (2001). Making java work for microcontroller applications. In *IEEE Design and Test of Computers*, pages 100–110, Los Alamitos.

Leonard, R. and Drezner, J. (2002). Global hawk and darkstar. Santa Monica, California. RAND Corporation.

Lessmann, J., Heimfarth, T., and Janacik, P. (2008). Shox: An easy to use simulation platform for wireless networks. In *Proc. of 10th International Conference on Computer Modeling and Simulation*, pages 410–415.

Microsystems, S. (2010). *SunSPOT - Sun Small Programmable Object Technology*. Sun Microsystems. <http://www.sunspotworld.com>.

P. Gaudino, B. S., Bonabeu, E., and Clough, B. (2003). Swarm intelligence: a new c2 paradigm with an application to control of swarms of uavs. In *Proc. of 8th International Command and Control Research and Technology Symposium*.

Sauter, J. A., Matthews, R., Parunak, H. V. D., and Brueckner, S. A. (2005). Performance of digital pheromones for swarming vehicle control. In *Proc. of 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 903–910. ACM Press.

UFRGS (2006). *Sashimi Manual*. UFRGS. <http://www.inf.ufrgs.br/~lse/sashimi/>.

Walter, B., Sannier, A., Reinerss, D., and Oliver, J. (2005). Uav swarm control: Calculating digital pheromone fields with the gpu. In *Proc. of The Interservice/Industry Training, Simulation & Education Conference*.

Wehrmeister, M. A., Pereira, C. E., and Becker, L. B. (2006). Optimizing the generation of object-oriented real-time embedded applications based on the real-time specification for java. In *Proc. of The Design, Automation, and Test in Europe Conference*, pages 806–811, Belgium.

A Free-Collision MAC Proposal for 802.11 Networks

Omar Alimenti [1,2], Guillermo Friedrich [1], Guillermo Reggiani [1]

[1]SITIC Group – Universidad Tecnológica Nacional – FRBB
Bahía Blanca –Argentina

[2] IIIE - Universidad Nacional del Sur – DIEC
Bahía Blanca –Argentina

iealimen@criba.edu.ar, {gfried,ghreggiani}@frbb.utn.edu.ar

Abstract. *Wireless technologies are a good choice for work in industrial environments, where it is necessary to interconnect mobile systems or it wants to avoid sensors and controllers wiring in plant. However, these technologies present reliability and timing problems inherent in the radio channels, mechanisms for medium access, etc. The standard 802.11e provides two alternatives for medium access (EDCA and HCCA) by differentiating traffic into four Access Categories (ACs). This paper proposes a mechanism for controlling the medium access, so-called WRTMAC, developed from the EDCA scheme of standard 802.11e. The handling of the arbitration inter frame spaces (AIFS) has been modified in order to make deterministic the medium access, even in terms of high traffic next to the saturation of the system.*

1. Introduction

Wireless technologies have become a very attractive option for industrial and factory environments. We can appoint the reduction of time and cost of installation and the maintenance of cabling industrial and their changes. The damage on the wirings and connectors due to the aggressive environments of certain types of industries is another reason. The applications of industrial control that involve some kind of mobile systems, in which data communications must meet requirements of real time and reliability, can benefit from the wireless interconnection [Willig A., Matheus K. and Wolisz A., 2005]. However, it is necessary to consider features of the wireless medium, such as the typical weaknesses of a radio frequency channel (RF), the mobility of some stations, the uncertainty in the time of physical medium access of some protocols, etc.

Despite other types of existing wireless interconnections, we are interested on wireless local area networks (WLAN) based on IEEE 802.11 standard. The Medium Access Control protocol (MAC) is decisive in the performance of the network [Vanhatupa T., 2008]. The 802.11MAC mechanism can operate in two ways: Point Coordination Function (PCF) and Distributed Coordination Function (DCF). PCF, also called free of contention, uses an Access Point (AP) as a network coordinator. In DCF, without centralized control, the nodes compete for the access to the physical medium. In spite of the differences, both modes use the Carrier Sense Multiple Access with Collision-Avoidance (CSMA/CA) mechanism to obtain the access to the medium and transmit. One of the weaknesses of the 802.11 MAC protocol is that it not support differentiated quality of service (QoS) for different types of traffic. For that reason,

802.11e [IEEE Std 802.11e; Part 11, 2005] was developed to support two QoS mechanisms: Enhanced Distributed Coordination Access (EDCA) and Hybrid Coordination Function Controlled Channel (HCCA). The EDCA scheme extends DCF, as it is known in the original standard [IEEE Std 802.11; Part 11, 2007], differentiating four prioritized Access Categories (AC) [Vittorio S. and Lo Bello L., 2007]. In spite of EDCA improves the throughput and the response time with regard to DCF, the reduced amount of AC limits the differentiation of traffic with temporary restrictions [Ferré P., Doufexi A., Nix A. and Bull D., 2004]. This paper proposes changes at the MAC level, based on the standard 802.11e, in order to adequate the EDCA mechanism for real-time industrial applications, generating a number of ACs as devices and/or messages are there in the network [Pereira da Silva M. and Becker Westphall C., 2005], making deterministic the time to access the medium. This mechanism has been called WRTMAC: *Wireless Real- Time Medium Access Control*.

2. DCF and EDCA

A wireless local area network (WLAN) 802.11 type is a broadcast network, characterized by the uncertainty in the medium access time.

DCF is a distributed medium access control scheme, based on the CSMA/CA mechanism. A station must sense the medium before starting a transmission; if the medium remains idle during a random time, the station transmits, otherwise its transmission must be postponed until the end of the current one. DCF distinguishes two techniques: the simplest, the station transmits the frame when it is obtained the access to the medium, and waits the acknowledge (ACK) from the receiver; the other uses an exchange of RTS/CTS frames between sender and receiver, prior to the dispatch of the data, in order to avoid collisions due to the hidden nodes [Bensaou B., Wang Yu and Chi Chung Ko, 2000]. This work is based on the first one.

A collision is difficult to detect in a wireless medium, so a given amount of time named inter-frame space (IFS) is used to control the access to the channel. When sensing indicates that the medium is free, a station must wait a time named distributed inter-frame space (DIFS) after the end of the previous transmission (Figure 1). Then there is a waiting time, named backoff window (BW), whose duration is a random quantity of slots time (ST), between a minimum of 0 and a maximum equal to CW–1. CW is the value of the contention window, which begins with a minimum value CW_{min}, and doubles this value after each collision up to a maximum CW_{max}. When the BW timer reaches zero and if the medium remains free, the station begins its transmission. If the medium becomes busy before BW expires, this timer is frozen until the channel remains idle during a DIFS time. If BW expires in two or more stations at the same time, there will be a collision. After a frame was received satisfactory, the receiver station must wait a time short inter-frame space (SIFS) to send an ACK (Figure 1). If the transmitter station didn't receive the ACK after a SIFS time from the end of its message, interprets that a collision has occurred and will be necessary retransmit. The collisions possibility of this mechanism causes uncertainty about the time needed to realize a transmission.

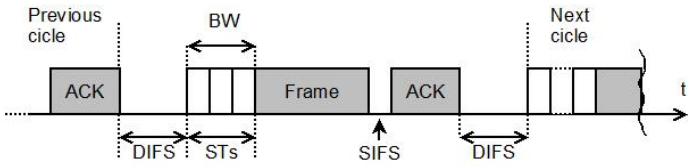


Figure 1. 802.11 DCF Timing

The 802.11e standard introduces the EDCA mode (Figure 2), which proposed a differentiated mechanism of QoS with four ACs: AC_BK (Background) for the lowest priority level (1-2), AC_BE (Best Effort) for the following levels (0-3), AC_VI (Video) for the priorities 4 and 5 and AC_VO (Voice) for the highest (6-7). According to its priority, a frame will be located in one of those four categories. Each AC uses specific values of arbitration IFS (AIFS), CWmín and CWmáx [Willig A., (2008)].

The difference between DCF and EDCA is that, the first does not distinguish types of traffic and, when the medium is free, all stations must wait for the same DIFS before starting its BW timer to contend for the medium access, using all the same CW. However, each type of traffic in EDCA, parameterized for its AC_i, will start its BW timer after sensing the medium idle for a while $AIFS_i$. The AIFS value depends on the AC of the message; therefore an AC of higher priority will have a lower value, having more probability to access the channel. Due to frames with the same AC can coexist in several nodes, collisions can occur and they are resolved in a similar way to DCF

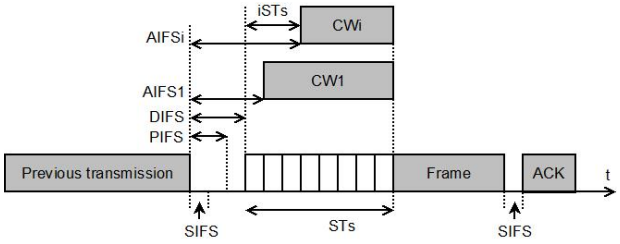


Figure 2. EDCA Timing

The goal of WRTMAC is to develop a collision free MAC method that guarantee the response time, defined as "the time measured from transmission request until the ACK reception". The basic proposal establishes one AC for each type of message, assigning a given $AIFS_i$ to each one. The waiting time prior to a transmission is equal to DIFS plus the $AIFS_i$ according to the type of message i.

3. WRTMAC: a Real Time variant to WLAN 802.11

3.1. Basic scheme

The objective that has been established for WRTMAC is a real time deterministic behavior. So, the maximum latency to transmit a frame must be ensured and must be necessary to remove those probabilistic elements own of DCF and EDCA.

EDCA has been the starting point for defining WRTMAC, introducing variants to achieve the target. In that sense, have been established the following patterns of operation:

- Each type of frame has assigned a certain priority, different from any other, in a similar way to the bus CAN [Bosch Robert GmbH, 1991].
- The priority is indicated by a numerical value from zero (maximum priority) and a certain positive number *N* for the minimum. The total amount of priorities should be established by the amount of types of messages that should be handled in the context of a particular application.
- If two or more simultaneous requirements arise, always the frame of the highest priority must be transmitted.
- The logic for controlling the access to the channel has been designed to avoid the occurrence of collisions. However, collisions can occur after intervals of prolonged inactivity, due to the drift between the local clocks of the nodes. The resolution of these collisions should be done in a bounded and predictable time. Also, it has been designed a simple strategy to allow a free-collision operation.

Figure 3 presents the basics of WRTMAC. When a station has a frame to send, it must wait until the medium becomes idle. After a while called "Real-Time Inter-Frame Space" (RIFS), if the medium is still free, the transmission starts. If during the wait, the channel becomes busy, the process will be stopped and should be restarted when the medium becomes free.

In WRTMAC each message has its exclusive RIFS value. Its duration is inversely proportional to the priority it represents. $RIFS_i$ is called the waiting time (backoff) for the priority *i* message.

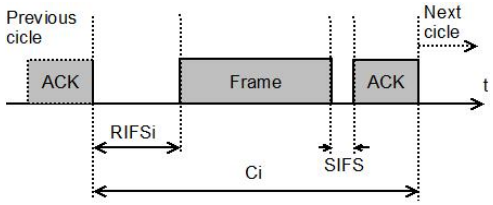


Figure 3. WRTMAC: Basic scheme

WRTMAC determinism is based on that, each message uses a $RIFS_i$ arbitration time, fixed and different from others. This tends to avoid the occurrence of collisions, while ensuring that, in case of contention, the winner will be the higher priority message.

Figure 4 shows the order of three frames, with priorities 2, 3 and 4, contending for the medium access. The three nodes begin the wait, but as $RIFS_2$ is the shortest, $Frame_3$ and $Frame_4$ attempts must be aborted. They are restarted after the end of $Frame_2$ cycle.

$RIFS_i$ duration is calculated based on the values of DIFS and ST. They are established by the selected physical layer (PHY) of the standard, according to (1):

$$RIFS_i = DIFS + i * ST \tag{1}$$

Table 1 shows the values of SIFS, DIFS and ST for different variants of physical layer (PHY):

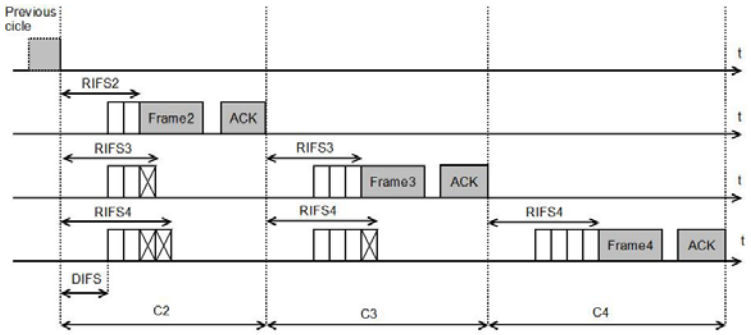


Figure 4. Transmission order according to priority of messages

Table 1. 802.11: PHY variants

PHY	Frec. (GHz)	Rate (Mbps)	SIFS (µs)	DIFS (µs)	ST (µs)
802.11b	2.4	11	10	50	20
802.11g	2.4	54	10	28	9
802.11a	5.8	54	9	16	34

The transmission cycle of priority i, composed by $RIFS_i$, the transmission time of the frame i (t_{FRAMEi}), SIFS and the transmission time of the ACK frame (T_{ACK}), is called C_i (2):

$$C_i = RIFS_i + t_{FRAMEi} + SIFS + t_{ACK} \quad (2)$$

The ACK instructs the MAC entity of the transmitter that the frame sent, reached its destination. In general, if ACK is not received a retransmission is not performed, but notifies the upper layers that the transfer has failed (or at least that there is no certainty that has been successful). The decision regarding what actions must be taken is left to the uppers layers; they know the logic and timing constraints of the application. WRTMAC is only responsible for providing deterministic communication service on the maximum latency. Only it would be performed a unique retransmission in case of collision, without affecting the deterministic behavior, as explained in 3.4.

One can see that WRTMAC allows implement a Real-Time scheme of Rate Monotonics (RMS) type [Liu and Layland, 1973], assigning priorities to messages in reverse order of their periods. Knowing t_{FRAMEi} for all messages of a certain real-time system, and assuming that they are periodic, one can set the minimum possible period between transmission requirements (T_i) for a given message m_i, in terms of all other messages m_j of higher priority than m_i (where $j<i$). Adapting the classic formula used to analyze the schedulability of a set of real-time periodic tasks on a processor [Lehoczky J., L. Sha, and Y. Ding, 1989], the minimum period possible for a message of priority i, is (3):

$$T_i \geq \sum_{\forall j<i} \left\lceil \frac{T_i}{T_j} \right\rceil C_j + C_i \quad (3)$$

Where: T_i, T_j, C_i, C_j: Period and transmission cycle of priority messages i and j.

Figure 4 shows messages with periods $T_2 \leq T_3 \leq T_4$. (3) is valid when the network is working with high traffic, i.e. when there is always at least one transfer request pending, awaiting the end of the current transmission. However, depending on

the total amount of messages and their periods, there may be long intervals of silence. This requires a special analysis, because the completion of a transfer is the event that reset the counter on each node and allows maintaining synchronism.

Exceeding the time corresponding to RIFS longer without having made any transmission, all nodes must restart their counters. Each node must maintain the timing of activity in the medium, even when there is no requirement, because even a moment prior to the expiration of its RIFS is able to receive a request and transmit it in the current cycle.

3.2. Operation on non-saturation conditions

When the network has extended moments of silence, all nodes must proceed as follows: wait a while corresponding to the duration of the message of the lowest priority plus a ST, and then restart their counters. In fact, as one takes into account the time ST elapsed after $RIFS_N$, the counters $RIFS_i$ are not restarted from zero, but with an initial value ST (equivalent to consider that the timers are reset at the expiry of $RIFS_N$).

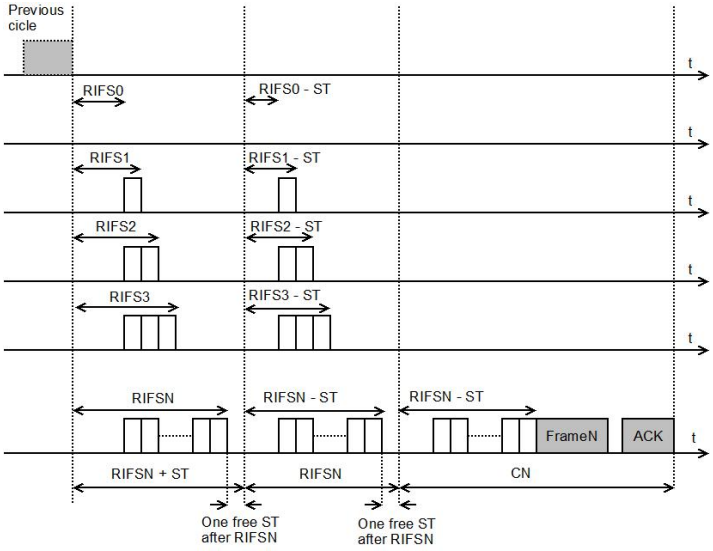


Figure 5. WRTMAC: counters in non-saturation conditions

Figure 5 shows the transmission of a frame after cycles of inactivity. In real operating conditions could succeed longer intervals of silence, maintaining the same concept to restart the counters after each RIFSN period of inactivity in the medium. In case of more extended inactivity, the drift between the locals clocks of each node can lead to a collision condition, as is discussed in 3.4. Another situation to consider is priority inversion, discussed in 3.3. Both, the collision and the priority inversion introduce delays, which must be added to (3) to generalize its expression.

3.3. Priority Inversion

Priority inversion is called the situation that occurs when the transmission of a frame must wait until the completion of a lower priority. Figure 6 shows the almost simultaneous transmission request from $Frame_2$ and $Frame_3$, however, as the requirement of $Frame_2$ released an instant after the end $RIFS_2$, its transmission must wait for the next cycle.

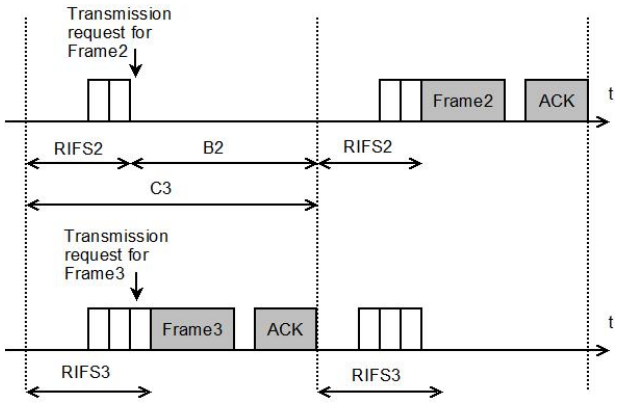


Figure 6. Priority inversion: *Frame$_3$* is transmitted before *Frame$_2$*

As the requirement *Frame$_3$* came before the expiry of RIFS3, it is transmitted and completes its cycle C_3. Thus *Frame$_2$* was blocked for a while B_2, whose maximum value is $B_2 = C_3 - RIFS_2$. If all frames are considered of lower priority than 2, the maximum block time of *Frame$_2$* is:

$$B_2 = máx(C_j) - RIFS_2 \qquad \forall\, j > 2 \tag{4}$$

In general, for any frame of priority *i*, time blocking by priority inversion is:

$$B_i = máx(C_j) - RIFS_i \qquad \forall\, j > i \tag{5}$$

Another deadlock occurs when a frame must wait until the next cycle to be transmitted, because its request arrived a moment after the expiration of its RIFS, having requirements that cause a priority inversion (Figure 7).

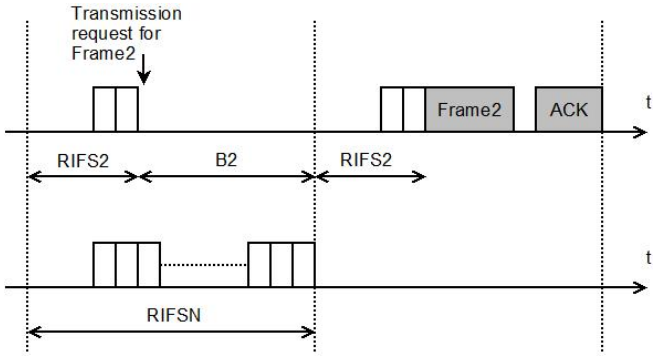


Figure 7. Deadlock of a frame of priority 2 to the end of *RIFS$_N$*

In this case, the deadlock time is $B_2 = RIFS_N - RIFS_2$. As is lower than the priority inversion blocking, remains valid (5).

Based on these considerations, (3) is extended as follows (6):

$$T_i \geq \sum_{\forall j<i} \left\lceil \frac{T_i}{T_j} \right\rceil C_j + C_i + B_i \tag{6}$$

In (6), it isn't taking into account the occurrence of collisions, which is discussed in 3.4.

3.4 Collision by drift of local clocks

When takes place an idle interval of duration greater than or equal to $RIFS_N$, all the nodes must restart their RIFS timers with a periodicity $RIFS_N$.

In case of almost simultaneous requests of consecutive priorities, due to the asynchrony that could exist between clocks of different nodes, it could have cancelled (or reduced almost totally) the difference of one ST between adjacent priority levels, giving rise to a collision (Figure 8).

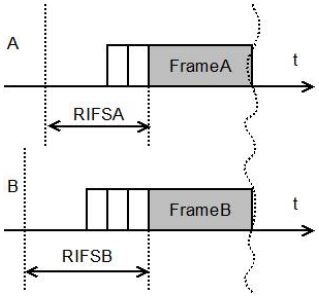


Figure 8. Collision by drift of local clocks

The way in which each node detects a collision depends on the time gap between the ends of the collided transmissions. Nevertheless, after detected a collision and resynchronized the RIFS timers, each node that has been involved in the collision, restarts the process to try a new transmission. This is the unique situation for which a retransmission is allowed, i.e., after a collision following a silence longer than $RIFS_N$.

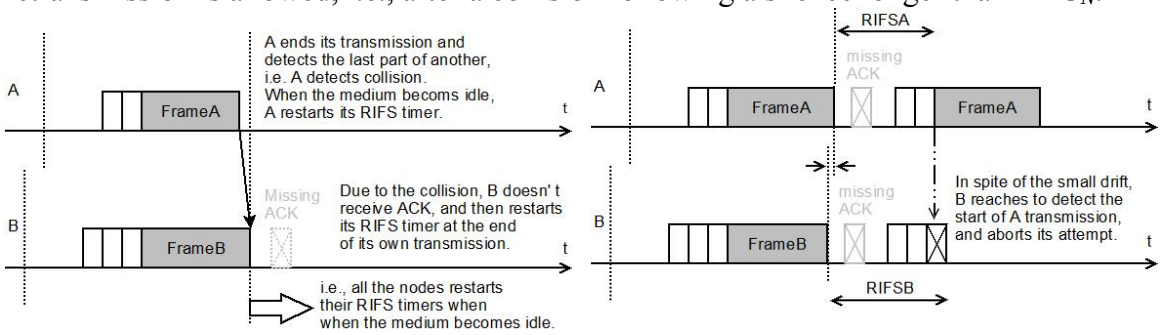


Figure 9. Collisions Type 1 (left) and Type 2 (right)

Figure 9 shows both types of collisions. In Type 1, $Frame_B$ finalizes an amount of time after $Frame_A$, enough to enable node A to sense carrier during the SIFS after its transmission. Node A restarts its RIFS timer with the end of transmission of B. However, B detects the collision by the lack of ACK, and therefore it assumes that the new beginning of its RIFS timer is the end of its own transmission. The remaining nodes that have not been involved with the collision, restart their timers when the medium becomes idle, i. e. after the end of $Frame_B$. In this case, the end of transmission of $Frame_B$ is the event that allows the clock synchronization between all the nodes.

In Type 2, the involved frames finalize with a tiny difference of time, which not allows the detection of the later ending frame by the other node. Then, all the nodes synchronize with the last end of transmission ($Frame_A$ in the example), with the exception of the node that ends in first place its collided transmission ($Frame_B$ in the

example). But this gap is smaller than that it is needed to cause a new collision, because it did not allow the sensing of the later end-of-frame by the first finishing node.

After collision detection and clock synchronization, the pending transmissions will be dispatched according to their $RIFS_i$, following the rules of WRTMAC.

3.5 Worst-case delay due to a collision

Figure 10 shows that, after a long idle period a request arrives for the sending of $Frame_i$, but it comes immediately after the expiration of $RIFS_i$. Then, the node must wait until the next cycle to try again. Since there are no pending requests of priorities lower than i, all the RIFS the timers will be restarted after the expiration of $RIFS_N$. Until that moment, the delay accumulated by $Frame_i$ is:

$$D_i = RIFS_N - RIFS_i \tag{7}$$

During the next cycle starts the transmission of $Frame_i$, but a collision occurs. Figure 10 shows the worst-case collision delay for $Frame_i$, because is involved the longest frame (the effect would be the same if $Frame_i$ is the longest one).

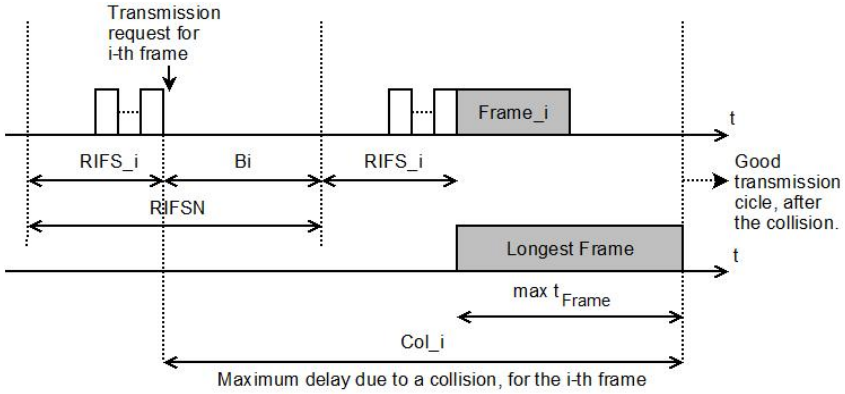


Figure 10. Worst-case delay due to a collision

After the collision, a new cycle starts. If $Frame_i$ continues being the highest priority among those that are awaiting transmission, the delay due to the collision is:

$$Col_i = D_i + RIFS_i + max(t_{FRAME}) \tag{8}$$

Replacing D_i from (7), and as the resulting delay does not depend on frame priority, it is designated generically as Col:

$$Col = RIFS_N + max(t_{FRAME}) \tag{9}$$

Now it should be modified the formula (6) to include the delay due to a collision. However, the priority inversion blocking, B_i, and the delay due to a collision, Col, are mutually exclusive. It could happen one or the other but not both. Thus, the minimum possible period between requests of the ith-message is:

$$T_i \geq \sum_{\forall j<i} \left\lceil \frac{T_i}{T_j} \right\rceil C_j + C_i + \max(B_i, Col) \tag{10}$$

The formula (10) allows the schedulability analysis for a particular system, based on the amount of messages, their duration and periodicity.

Although the Rate Monotonic scheduling requires that priorities must be assigned in inverted order with respect to the request periods, if some (or eventually all) messages have the same period, it must be assigned different $RIFS_i$ to each one, to allow the medium access arbitration.

3.6. Collision-Free operation

Given that idle intervals greater than $RIFS_N$ can lead to the occurrence of collisions, one way to avoid them could be to avoid the occurrence of such long intervals without transmissions.

A simple strategy could be that, the node designated for the transmission of the lowest priority frames, always should perform a transmission. Therefore, if at $RIFS_N$ timer expiration it does not have a pending request for the transmission of a message, it must send a dummy frame, whose sole purpose is to occupy the medium, allowing the synchronization of all nodes with the end of this transmission.

Using this simple strategy, becomes valid the formula (6) to establish the minimum possible period for a given message *i*.

4. Performance evaluation

Given the initial motivation that led to the development of WRTMAC, its application in industrial control systems, usually based on small periodical messages, they was evaluated the maximum number of messages that could be driven by a network of this type and / or the shortest feasible period for each one. Also, the utilization factor was calculated, defined it *as the fraction of the total time that the medium is used to transport data*.

It has been considered that the network is used for the transmission of a given set of messages of equal size and period. It has been selected the 802.11b physical layer at 11 Mbps, with long preamble (192 microseconds); message payload of 50, 100 and 1500 bytes (plus 36 bytes of header), and 14 bytes of ACK. Also, it has been evaluated two options: one of this based on Formula (10) –with collisions– and the other based on Formula (6) –collision-free–. The utilization factor has been calculated for the free-collision mode. The results are showed in Table 2.

It is observable that there are no significant difference between payloads of 50 and 100 bytes (typical sizes for supervisory and control systems).

Also, there is a small improvement by applying the free-collision model. Hence, it could be simplified the MAC mechanism, eliminating the handling of collisions and its associated retransmissions.

Moreover, it can be seen that for long messages (1500 bytes) the minimum period does not increase proportionately (eg. 77 ms for 64 messages of 100 bytes each; 143 ms for 64 messages of 1500 bytes each). It is allowed to estimate that this network could be used with a mixing of short messages (monitoring and control) and long messages (data, images, etc.), with a small impact on the real-time performance.

Table 2. Minimum period and utilization factor, based on quantity and size of messages

		Minimum Period (ms)		Utilization Factor (%)
Payload (bytes)	N° of messages	With collisions	Collision-free	
50	8	6	4	4.8 %
	16	12	9	4.8 %
	32	28	27	4.3 %
	64	75	74	3.1 %
	128	232	230	2.0 %
100	8	6	6	9.7 %
	16	12	12	9.7 %
	32	29	29	8.0 %
	64	78	77	6.0 %
	128	237	234	3.9 %
1500	8	15	15	58.1 %
	16	30	30	58.1 %
	32	63	62	56.3 %
	64	144	143	48.8 %
	128	368	366	38.1 %

As can be expected, the utilization factor increases with the payload size, because the influence of the overhead due to headers, ACK, etc. is reduced. However, the utilization factor decreases significantly as the total number of messages increases. This is due to the growing overhead associated with the different RIFSs needed to arbitrate the medium access. Based on this situation, a way to improve the network performance could be to group several messages, sharing the same RIFS. However, as RIFS is used to arbitrate the medium access, the RIFS sharing could be possible only when there are more messages than nodes. In these cases, as it is not possible a collision between messages originated in the same node, it could be possible that several messages share the same RIFS value. This will be analyzed in a future work.

5. Conclusions

WRTMAC (Wireless Real Time Medium Access Control) is a proposal that implements a MAC based on EDCA concepts of the standard 802.11e, in order to achieve a mechanism for distributed wireless medium access, allowing predictable access time to the medium of the devices in the network. Changes are proposed to make suitable the EDCA mechanism, in order to generate as Categories Access (AC) as devices and/or messages are in the network. This will be deterministic the access time to the medium. This proposal could bring the EDCA mechanism for real-time industrial applications.

It was shown that WRTMAC allows the implementation of a Rate Monotonic Scheduling (RMS) scheme, since it is possible to know the minimum feasible period of a message, according to all messages.

The proposal also shows that, despite collisions can occur, the collided frames will be retransmitted, and the collision will be solved in a bounded time. Moreover, WRTMAC can operate in a free-collision mode.

Furthermore, when evaluating the performance of WRTMAC over traffic patterns typical of industrial application networking, it was noted that this mechanism provides an adequate performance without times uncertainties. In addition it was shown that the network could be used by combining short and long messages (for monitoring, control and general data, images, etc.), without greatly real-time performance degradation.

As it was analyzed, the increase in the number of RIFS diminishes the utilization factor of the network, due to the RIFS overhead. Hence, future works will be oriented to share the same RIFS between several messages originated in the same node. The goal will be to develop a methodology to establish the minimum number of RIFS needed for a given set of messages.

References

Willig A., Matheus K. and Wolisz A. (2005), "Wireless Technology in Industrial Networks", Proceedings of the IEEE, Vol. 93, No. 6 (June), pp. 1130-1151.

Vanhatupa T. (2008), "Design of a Performance Management Model for Wireless Local Area Networks", PhD Thesis.

"IEEE Std 802.11e; Part 11 (2005): Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications and Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements".

"IEEE Std 802.11; Part 11 (2007): Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications".

Vittorio S. and Lo Bello L. (2007), "An Approach to Enhance the QoS Support to Real-Time Traffic on IEEE 802.11e Networks", 6th Intl Workshop On Real Time (RTN07) Pisa Italy.

Ferré P., Doufexi A., Nix A. and Bull D. (2004), "Throughput Analysis of IEEE 802.11 and IEEE 802.11e MAC", WCNC, IEEE Communications Society.

Pereira da Silva M. and Becker Westphall C. (2005), "Performance Analysis and Service Differentiation in the MAC SubLayer of IEEE 802.11e Ad Hoc Networks", Proceedings of the Advanced Industrial Conference on Telecommunications, IEEE.

Bensaou B., Wang Yu and Chi Chung Ko (2000), "Fair Medium Access in 802.11 based Wireless Ad-Hoc Networks", IEEE/ACM The first Annual Workshop on Mobil Ad hoc Networking e Computing (MobiHoc´00), Boston, EUA.

Willig A.(2008), "Recent and Emerging Topics in Wireless Industrial Communications: A Selection", IEEE Transactions On Industrial Informatics, V4 Nº 2.

Robert Bosch GmbH (1991), "CAN Specification 2.0". *www.semiconductors.bosch.de*

Liu and Layland (1973), "Scheduling algorithms for multiprogramming in a hard real-time environment", Journal of the ACM, Vol.20 Nº 1, pp. 46-61.

Lehoczky J., L. Sha, and Y. Ding (1989), "The rate monotonic scheduling algorithm: Exact characterization and average case behavior". Proc. IEEE Real-Time Systems Symposium, pp. 166-171.

Performance Evaluation of a Real-Time MAC Protocol for MANETS

Marcelo M. Sobral[1], **Leandro B. Becker** [1]

[1] Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina (UFSC) – Florianopolis, SC – Brazil

{sobral,lbecker}@das.ufsc.br

Abstract. *A wireless ad-hoc network for mobile nodes is characterized by a highly-dynamic topology that cannot predict the duration of the links among the nodes and neither the density of nodes within the network. Our previously proposed HCT (Hybrid Contention/TDMA) Real-Time MAC protocol provides a kind of short-range resource reservation policy for groups of nodes, which lasts while the participant links are available. Moreover, it adapts to continuous topology modifications. This paper analyses the performance of the HCT MAC protocol in special networks scenarios that deal with nodes mobility. Obtained results show that exists a direct relation between mobility degree and expected clustering performance of the protocol, which influences its real-time performance.*

1. Introduction

A new generation of applications will require communication capacity in environments without any infrastructure. To make the problem more complex, such applications might be composed by mobile nodes, which rely on wireless links to achieve communicability. The literature recently proposed the term MANET (*Mobile Ad-Hoc Networks*) to represent such application domain. Some MANET applications present an additional complexity because they require real-time guarantees with respect to the communication medium. An example of such applications is vehicle-to-vehicle (V2V) systems [Voelcker 2007], like platooning, which helps to reduce traffic congestions and provide safe driving. New areas of research in the space community have similar communication requirements, as for example in distributed satellite systems (DSS) [Bridges and Vladimirova 2009] where multiple spacecraft in varying configurations are used to achieve a mission's goals collaboratively.

In this context, we recently proposed a hybrid medium access control mechanism named HCT (Hybrid Contention/TDMA-based) MAC [Sobral and Becker 2008], which aims to provide a deterministic medium access by means of resource reservation. It also supports reconfiguration and mobility by using a contention-based approach. Therefore, it assumes that the mobile nodes are self-organized in clusters, used exclusively to support the allocation of time-slots within the corresponding member nodes. Ideally, clusters should be defined in a way that it allows as many nodes as possible to operate in resource-reservation mode. In [Sobral and Becker 2009] we addressed the problems related to the dynamic self-organization of clusters. More specifically, we presented an approach to establish the clusters in a distributed and autonomous way, taking into account the *neighborhood quality*, which stands for the communication quality among neighbour nodes.

An excellent neighborhood quality would imply a high number of nodes communicating to the cluster-head with suitable link quality, thus improving the amount of clusterized nodes and cluster *longevity*. By definition, more clusterized nodes and more lasting clusters increase the operation in resource-reservation mode of the involved nodes.

In this paper we present a performance evaluation of the HCT-MAC to assess the resulting cluster formation in specific scenarios dealing with nodes mobility. Our previous performance evaluation used only static scenarios. We focused this new evaluation in quantifying (i) cluster longevities and (ii) rate of clusterized nodes. Rate of clusterized nodes gives the average number of nodes that are cluster members. We also investigate the relation between both results and the neighborhood size, which gives the number of neighbors that receive frames sent by a node.

The remainder of the paper is structured as follows: section 2 gives a brief overview of the HCT, including its clustering strategy and how it computes the neighborhood quality. Section 3 presents preliminary results that show the resulting clustering rates and longevities for small networks which nodes move according to some mobility scenarios, and the consequent neighborhood sizes. Finally, section 4 concludes the paper and points out some future directions of our work.

2. Overview of HCT MAC Protocol

In [Sobral and Becker 2008] we presented the Hybrid Contention/TDMA-based (HCT) MAC, which aims to provide a time bounded medium access control to mobile nodes that communicate through an ad-hoc wireless network. A key issue in this protocol is to self-organize nodes in clusters (i.e: set of neighbor nodes), as a mean to solve the problem of timely transmission of messages. Our protocol assumes as basic requirements a periodic message model, where the assignment of time-slots must be done within clusters. A competition strategy is adopted, without the need of a global coordinator nor scheduler, in such a way that time-slots are iteratively allocated by the nodes.

The HCT-MAC is a hybrid protocol because it has both contention-based and resource-reservation characteristics. A TDMA-based MAC protocol divides time in so-called time-slots, being responsible to assign one or more time-slots to each node. To solve this problem, the HCT-MAC uses a contention-based approach to allocate slots: if a node knows which slots are idle, it can try to use some of them, chosen randomly, and then verifies if any collision has occurred. For each chosen slot, if there was no collision, the node may assume that the slot is allocated. For the remaining slots, it can repeat the procedure until all the needed slots are allocated or, in the worst-case, no slots are available anymore. This protocol assigns the time-slots iteratively, until the allocation stabilizes, i.e. the nodes allocate all needed slots. The already allocated time-slots can be used just like in a TDMA protocol, revealing the resource-reservation aspect of the HCT-MAC.

The timing in the HCT has a periodic and hierarchical structure, as ilustrated in figure 1. A *cycle* is the basic period for transmissions, thus it works like a time unit for the protocol. It is an interval of time that is common to all clusters, and is divided in *superframes*, which are assigned to the clusters. Superframes are all equal in size, which means that they contain the same number of *time-slots*, plus two control frames called *beacons*. Therefore, clusters need to allocate superframes, and cluster members allocate

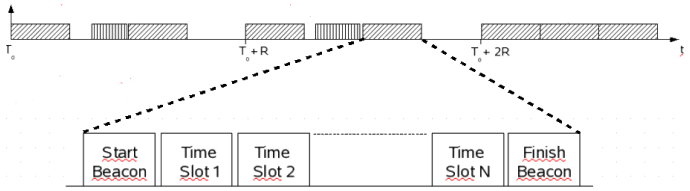


Figure 1. Timing in the HCT: cycles of length R divided in superframes

time-slots within those superframes.

The TDMA component of the HCT, described in [Sobral and Becker 2009], depends on the clustering of the nodes, which must be obtained in a self-organized manner. This is due to the fact that the HCT protocol is designed to be used in mobile ad-hoc networks, where the nodes are not previously aware of the topology, neither of their neighborhoods. The chosen approach relies on initial contention-based access, that shifts gradually to time-based as clusters are formed and become stable. That means, as nodes self-organize in clusters, they can reserve bandwidth and transmit messages in a timely manner. This implies procedures for the neighborhood discovery, the collection of information to support the independent choice of the best candidate nodes to start clusters, and the announcement of new clusters and ingress of interested nodes.

2.1. The Clustering Approach

As described in [Sobral and Becker 2009], clusters are simply sets of nodes that agree to share a superframe, which represents a portion of the network bandwidth. A key element in the cluster topology is the cluster-head, a special node responsible to start cluster transmissions, to account for idle and used time-slots, and to report successful transmissions. Ideally, the cluster-head should be the node with the best neighborhood quality within the region to be covered by the cluster, in order to minimize the probability of errors in transmissions inside the cluster. In our proposed HCT it is not possible to determine exactly the nodes with best neighborhood quality, since no node has a global view of the network, and no global information is maintained by the protocol. But the information about the neighborhoods, estimated based on the received frames, can be collected and shared locally among the nodes to help them to decide to become or not cluster-heads. Thereby, the cluster-heads can be self-elected according to the information they are able to obtain about the nodes around them.

The rule for establishing clusters is guided by the fact that the cluster-head should be the node with the best neighborhood quality (NQ). The quality of a neighborhood of a node is defined in this work as a function of the qualities of links between this node and each of its neighbours. It expresses both the quantity of neighbours and their link qualities. A high NQ means few or no transmission errors (missed or corrupted frames) or, in other words, a high frame reception rate. Therefore clusters have higher probability to be stable, because more frame losses would force member nodes to try to bind to other clusters.

2.2. Neighborhood Quality Computation

HCT needs to compute continuously the Neighborhood Quality (NQ). The NQ of a node depends on the quality estimation of individual links to its neighbours.

In [Sobral and Becker 2009], NQ was defined as a moving average of the sum of RSSI-based measures of received frames. The chosen link quality estimator (LQE) derived from the RSSI of the received frames because, as demonstrated in [Zuniga and Krishnamachari 2004], there is a relation between the expected PRR (Packet Reception Rate) and the RSSI. However, as discussed in [Baccour et al. 2010], this kind of LQE does not fully capture the properties of a link. To overcome this problem we are moving towards the adoption of F-LQE [Baccour et al. 2010], a new link quality estimator that combines several link properties to better characterize its quality.

The HCT protocol uses LQE also to increase the probability of nodes with better link qualities (relative to the cluster-head) to become members of a cluster. When a node tries to ingress a cluster, it waits for the Start Beacon control frame, chooses randomly an idle time-slot and then transmits in the selected time-slot. If the cluster-head receives such transmission, it acknowledges it in the Finish Beacon control frame but with probability p_q; it means that the cluster-head ignores with probability $1-p_q$ the received transmission. This probability p_q is higher for transmissions received with high link quality. Thus, HCT tries to establish clusters with higher neighborhood quality, increasing the probability that their member nodes are those with better link quality.

2.2.1. F-LQE

[Baccour et al. 2010] defines F-LQE as a link quality estimator based on four measured properties: packet delivery (SPRR), asymmetry (ASL), stability (SF) and channel quality (ASNR). By combining these properties, it aims to provide a more accurate link quality estimation. The instantaneous quality $LQ(i)$ of the link of node i is expressed as a membership in the set of good links, according to a fuzzy rule shown in equation 1. The overall quality $FLQE_i(\alpha, w)$ of node i is averaged over a window of w received frames (suggested to be 30), and smoothed according to a parameter α (suggested to 0.6), as can be seen in equation 2. Its attractiveness to the HCT resides both in the ability to capture important aspects of link quality and the smoothness and stability of generated values.

$$\begin{aligned} LQ(i) &= \beta \cdot min(\mu_{SPRR}(i), \mu_{ASL}(i), \mu_{SF}(i), \mu_{ASNR}(i)) + \\ &\quad (1-\beta) \cdot mean(\mu_{SPRR}(i), \mu_{ASL}(i), \mu_{SF}(i), \mu_{ASNR}(i)) \end{aligned} \tag{1}$$

$$FLQE_i(\alpha, w) = \alpha \cdot FLQE + (1-\alpha) \cdot LQ(i) \tag{2}$$

2.2.2. F-LQE metrics within HCT

To compute its four link properties (packet delivery, asymmetry, stability and channel quality), F-LQE needs to keep a history of measured values. They must be calculated for each individual link, i.e., a node computes their values individually for each one of its neighbours. But since HCT transmits always in broadcast, some adjustments are needed to compute these measures:

- **Packet delivery:** it depends on SPRR (Smoothed PRR), that accounts for the actually received frames compared with the transmitted ones. HCT can obtain this estimator by the inclusion of a sequence number within each frame. Therefore, receiver nodes can compare the number of actually received frames and the interval of sequence numbers to compute the PRR.
- **Channel quality:** easily obtained from the RSSI of received frames subtracted by the noise floor.
- **Stability:** it corresponds to the variability of the PRR, as obtained in *Packet delivery*. The F-LQE defines the stability as the coefficient of variation, i.e. the reason between standard deviation and mean.
- **Asymmetry:** the hardest to obtain, because it depends both on *PRRup* and *PRRdown*. *PRRdown* is straightforward, since it is the same as the *Packet delivery*, but *PRRup* is the *Packet delivery* as seen by the neighbour. It is not feasible to make the neighbour to transmit its *Packet delivery*, but *PRRup* could be derived if the neighbour would acknowledge each received frame. Unfortunately, since HCT uses only broadcasts, this is also not feasible. Therefore, it was decided to exclude this metric from F-LQE within HCT

2.2.3. Using F-LQE to Compute the Neighborhood Quality

The Neighborhood Quality (NQ) was modified to better explore the F-LQE. Obviously, a good neighborhood should be composed by neighbours with good link quality estimations. It means it depends both on the quantity of neighbours and their corresponding LQE values. As defined in [Sobral and Becker 2009], NQ was computed as the sum of the LQE values, but with the adoption of F-LQE it would be better to calculate the product of the F-LQE values (L_{ij}), as shown in equation 3. The parameter L_{REF} (*Link Quality Reference*) was chosen to emphasize good links. Thus, NQ_{ij} value increases if $L_{ij} > L_{REF}$, otherwise it decreases.

$$NQ_j = \|N_j\| \prod_{i \in N_j} (1 + L_{ij} - L_{REF}) \tag{3}$$

Finally, to avoid sudden changes in NQ_j, which can appear when frames from neighbours are missed for just few cycles (for instance due to external interferences), a smoothing of NQ_j is applied as seen in equation 4 (parameter $\beta \in (0, 1]$). The resulting metric is called SQ_j (Smoothed Neighborhood Quality).

$$SQ_j = \beta \cdot NQ_j + (1 - \beta) \cdot SQ_j \tag{4}$$

3. HCT Evaluation

HCT was designed to be used in networks containing mobile nodes. In such scenarios, it is expected that clusters arise and disappear dynamically, as the network changes its topology. Depending on the mobility degree of the network, reorganization of clusters are expected to occur more or less frequently. Since nodes transmit in resource-reservation mode only when they are members of clusters, the clustering performance has great importance to the HCT. The clustering performance can be expressed as:

- **Clustering rate:** rate of nodes that are members of cluster.
- **Cluster longevity:** expected longevity of clusters, represented by the interval between the moment a node becomes a cluster-head and the moment it reverts to single node (this occurs when its cluster becomes empty).

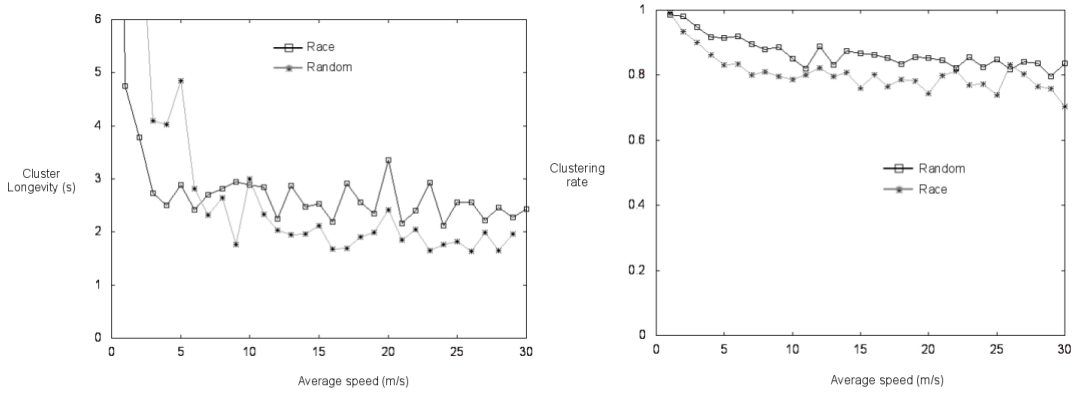
An important aspect of HCT performance is the neighborhood size, which gives the number of neighbours that receive frames sent by a node. This metric can be investigated both through its average over all transmission cycles, and by a ratio to the potential neighborhood (i.e. the nodes that can receive the frame). We investigate here a possible relation between clustering rate, longevity, and neighborhood size. This might hold because as more nodes are members of clusters, and consequently operate in resource-reservation mode, less collision is expected and more nodes receive each frame.

The HCT performance in networks with mobile nodes was investigated by means of simulations. They were executed using a simulation model of HCT developed using the Omnet++ simulation framework [Varga 2001], assuming a IEEE 802.15.4 physical layer. The HCT models uses as physical layer the project Castalia, maintained by the National ICT at the University of Australia [Pham et al. 2007], which implements the signal model proposed in [Zuniga and Krishnamachari 2004] and simulates a IEEE 802.15.4 compatible radio. This model derives the PRR (Packet Reception Rate) of each link according to a path loss model as function of the corresponding SNR (Sound to Noise Ratio). The Castalia model needed to be modified to support mobility, in such way it recomputes its internal PRR matrix each time a node moves.

Two special simulation scenarios were created:

1. **Random:** the simulations are composed by networks with changing topologies according to random node speeds and increasing speed averages. Nodes were scattered and enclosed within a square region, moving in constant directions and reflecting on the walls. These scenarios should demonstrate a pessimist situation for the HCT, since few patterns can exist in such networks. Indeed, only the average speed of the nodes dictates the duration of their links. This mobility model can be related to movements in the urban space, like people moving in squares or malls.
2. **Race:** the networks were composed by nodes that move all in the same direction, but with slight different speeds. In these scenarios, the relative speeds between nodes are small and thus it is expected a less frequent cluster reorganization. This mobility model can be related to cars moving along a street or highway, people runnning in a competition, and other cases where comunnicating devices move in the same direction.

In both cases the networks were composed by 20 nodes. For each of these scenarios, the maximum speeds varied between 1 and 40 m/s. For the random mobility model, the speed of each node was chosen from an uniform distribution between 0 and the maximum speed, and the direction was chosen randomly and only changed when the node reached the boundaries of the square area. For the race model, the speeds were chosen from a normal distribution, using as mean half of the maximum speed and a standard deviation such that the speeds stay below the maximum speed with probability 0.99. In both models the speeds do not change along the simulations. Each simulation generated statistics for the clustering rate and cluster longevity. The cluster longevity was expressed as



(a) Cluster longevity as function of speed (b) Clustering rate as function of speed

Figure 2. Clustering performance

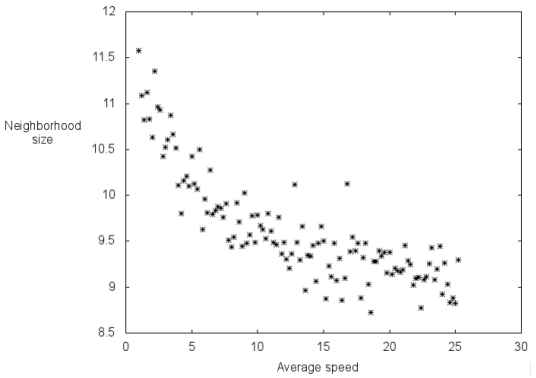
a histogram and also as an expected value, calculated from the histogram. The clustering rate was shown as the number of clusterized nodes as function of the average speed.

3.1. Obtained results

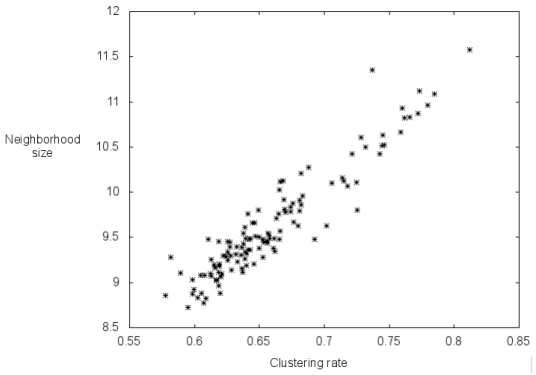
The results for the cluster longevities are shown in figure 2(a). The plots show that cluster lifetimes drop faster from 0 to 5 m/s, and for greater average speeds it stays around 1.5 s. The race scenario gives a slight better cluster longevity for higher speeds. Since in this kind of simulation the nodes move all in the same direction, but with slight different speeds, their smaller relative speeds allow more durable links. In the case of the random scenario, although the relative speeds are potentially higher because nodes can move in any direction, the more dense resulting network favors the cluster longevities. In the simulation, the network density effect prevails for smaller average speeds.

Clustering rate, another relevant metric of performance of HCT, is shown in figure 2(b). There is a slight difference between random and race scenarios, with the random one giving higher clustering rates. It must be noted that in the race scenario the relative speeds between nodes are lower than in the random scenario. This leads to more durable memberships, since it takes longer to a node to get out of sight of its cluster-head. But since the nodes are continuously scattering along the race line, it is more probable that once a node gets single, it will remain longer in this state due to the lower network density.

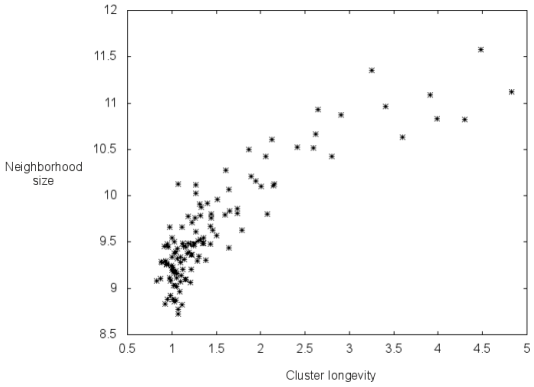
Both clustering rate and longevity expresses the behaviour of clustering within the HCT protocol in the simulated scenarios. The resulting performance, in the point of view of an application, can be seen in the rate of messages each node can transmit in resource-reservation mode, and the number of neighbours that receive those messages, known as neighborhood size. Therefore, the neighborhood size becomes an important way to investigate the success of HCT in delivering frames. In the simulations of the random scenario, the neighborhood sizes were averaged for each simulated speed and the results are shown in figure 3(a). It decreases as the speeds get higher, that can be related to the clustering performance. In figure 3(b) the neighborhood size is related to the clustering rate, revealing a linear relation. As shown in figure 3(c), a similar relation to the cluster longevity was not found.



(a) As function of average speed (m/s)



(b) Relation to clustering rate



(c) Relation to cluster longevity

Figure 3. Average neighborhood size

4. Conclusions and Future Work

This paper extended previous performance evaluation already conducted for the HCT MAC by adopting scenarios that deal with the mobility of nodes. Previous evaluation only considered static nodes.

The evaluation presented in this paper was performed by means of simulations using two different mobility scenarios: (i) random movements and (ii) race emulation. The measured clustering performance was expressed as clustering longevity, clustering rate, and neighborhood size. As expected, the clustering performance decreases as nodes move at higher speeds, as this creates more dynamic topologies, demanding more cluster reconfigurations. It is also observed that both clustering rate and cluster longevity decreases with an exponential-like curve as function of the speed of nodes.

Curiously, the race-emulation scenario presented worse results in these respect, although the smaller relative speeds between nodes can favor longer cluster lifetimes. This is due to the fact that the race-emulation generates a more sparse network, so it is less probable that a single node has a satisfactory neighborhood to clusterize.

Another observation is that, as expected, the neighborhood size also decreases as node speed gets higher. However, it shows a linear relation of the neighborhood size to the clustering rate, that gives the rate of nodes that are operating in resource-reservation mode. This relation was not found when compared to the cluster longevity. The neighborhood size corresponds to the quantity of nodes that successfuly receive frames from their neighbours, and is a measure of performance of the data delivery in the protocol. Therefore, a higher clustering rate improve the data delivery, but the same cannot be said about the cluster longevity.

Despite this evaluation, there exists a number of open questions regarding the efficiency of the HCT protocol in mobile networks. Firstly, it must be further investigated the scalability of the protocol compared to the network size and mobility degree. There must exist a maximum node speed below which the protocol has a satisfactory performance. The overall performance of the protocol depends on the clustering performance, which is influenced by the neighborhood quality. Currently, the neighborhood quality is calculated as a function of the quantity of neighbours and their link qualities, and its instantaneous values are used to support clustering decisions. The neighborhood quality could be changed to try to anticipate the neighborhood behaviour, increasing if the neighborhood is becoming better or decreasing otherwise. Finally, clusters could change their cluster-heads to adapt to changes in the neighborhood qualities of the cluster members. A cluster-head which detects that its neighborhood quality is significantly lower than one of its cluster members could delegate the cluster-head role to that node and revert to single node.

5. ACKNOWLEDGMENTS

The authors would like to thanks the Brazilian Research agencies FAPESC, CAPES, and, in special, CNPq (Brazilian National Counsel of Technological and Scientific Development) for the grant 486250/2007-5.

References

Baccour, N., Koubaa, A., Youssef, H., Jamaa, M. B., do Rosario, D., Alves, M., and Becker, L. (2010). F-lqe: A fuzzy link quality estimator for wireless sensor networks. In *The 7th European Conference on Wireless Sensor Networks (EWSN 2010)*, Coimbra, Portugal.

Bridges, C. P. and Vladimirova, T. (2009). Agent computing applications in distributed satellite systems. In *9th International Symposium on Autonomous Decentralized Systems (ISADS 2009)*, Athens, Greece.

Pham, H. N., Pediaditakis, D., and Boulis, A. (2007). From simulation to real deployments in wsn and back. In *IEEE International Symposium on World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007*, pages 1–6.

Sobral, M. M. and Becker, L. B. (2008). A wireless hybrid contention/tdma-based mac for real-time mobile applications. In *ACM Symposium on Applied Computing 2008, Real-Time Systems Track*, Fortaleza, Brazil.

Sobral, M. M. and Becker, L. B. (2009). Towards a clustering approach to support real-time communication in ad-hoc wireless networks. In *Brazilian Workshop on Real-Time Systems 2009 (WTR 2009)*, Recife, Brazil.

Varga, A. (2001). The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference*, pages 319–324, Prague, Czech Republic. SCS – European Publishing House.

Voelcker, J. (Oct. 2007). Cars get street smart. *IEEE Spectrum*, 44(10):16–18.

Zuniga, M. and Krishnamachari, B. (2004). Analyzing the transitional region in low power wireless links. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 517–526.

MASIM: A Tool for Simulating Mobile Agent Applications on Wireless Sensor Networks

Marcos Camada[1], **Carlos Montez**[1], **Flávio Assis**[2]

[1]Pós-Graduação em Eng. Automação e Sistemas – Univ. Federal de Santa Catarina (UFSC)
88040-900 – Florianópolis – SC – Brazil

[2]Programa de Pós-Graduação em Mecatrônica – Univ. Federal da Bahia
40170-110 – Salvador – BA – Brazil

{mcamada, montez}@das.ufsc.br, fassis@ufba.br

Abstract. *A mobile agent is an autonomous software entity that is able to migrate between nodes of a distributed system, carrying its code and execution state. Recently, mobile agents have been proposed to be used in wireless sensor networks, as an approach to reduce energy consumption of wireless sensor nodes. The advantages provided by mobile agents, however, depend on specific application aspects and on the particular way how they are used to accomplish some task. Due to the lack of suitable tools for the simulation of mobile agent-based applications over wireless sensor networks, this paper introduces a tool for this purpose, called MASIM. This tool extends the toolboxes of MATLAB/TrueTime. This paper describes the general features of MASIM and illustrates its use to a specific scenario, where a protocol based on mobile agents and one based on message passing are compared.*

1. Introduction

A Wireless Sensor Network (WSN) is a type of wireless network whose nodes are computing devices equipped with sensors, such as temperature, light or humidity sensors [Yu et al. 2004]. These networks might be composed of tens or even thousands of nodes, that can be deployed randomly in a environment, present mobility and be self-organizable [Malik and Shakshuki 2007]. Sensor nodes are typically small autonomous devices with strongly limited processing power and memory capacity, with a short-range transceiver and with non-replaceable batteries [Ye et al. 2001]. Therefore, they should be designed so as to spend as less energy as possible, so that network lifetime can be increased.

Mobile agents have been proposed to be used in WSN as an approach to conserve energy of nodes (see, for example, [Chen et al. 2006]). A mobile agent is an autonomous software entity that has as its main feature the ability to migrate between nodes, carrying its code and execution state. Mobile agents may provide additional advantages such as reduction of network load, complexity reduction in the design of interfaces provided by nodes, and capacity to autonomously adapt to changes in the system, thus contributing to systems robustness and fault tolerance [Lange and Oshima 1999]. However, the advantages provided by mobile agents depend on specific application aspects and on the particular way how they are used to accomplish some task [Jansen et al. 1999].

Simulation is one important tool to help evaluating the potential advantages of using mobile agents in specific application scenarios. There are currently many simula-

tion tools for WSN, but, to the best of our knowledge, none of them provides specific abstractions and functionality to simulate mobile agents over WSN.

This paper describes a tool to fulfil this need. We describe MASIM (Mobile Agent SIMulator in Wireless Sensor Network), a toolbox for simulating mobile agents on MATLAB [MathWorks 2009]. MASIM provides abstractions and functionality for modelling mobile agents, their environment and interactions between the many system components involved. The model used in MASIM is compatible with the mobile agents model defined by FIPA (Foundation for Intelligent Physical Agents) [FIPA 2004] and OMG (Object Management Group) [OMG 1997], two important standardized efforts in the context of mobile agent systems. MASIM uses an energy model that defines how each component spends energy which is based on the hardware specification of Mica2 [Crossbow 2009], one of the most important sensor node platforms. MASIM can also be easily extended to use energy models based on different platforms. We believe that extending MATLAB has the particular advantage of adding functionality to a system which is very frequently used for computer simulation.

This paper is structured as follows. In Section 2 we compare our approach to other existing wireless network simulation environments. In Section 3 we describe MASIM. In Section 4 we describe a particular use of MASIM to evaluate the advantages that mobile agents might bring for a specific scenario. Finally, in Section 5 we conclude the paper.

2. Related Work

MATLAB is a tool for the development and analysis of algorithms and data visualization with main focus on numeric computation [MathWorks 2009]. The MATLAB environment provides a set of libraries, called toolboxes, which can be developed in the native language of MATLAB or C++ MEX (based on ANSI C++). MATLAB can be used integrated with Simulink, which provides a graphical environment and a set of libraries for the design, simulation, testing and implementation of different types of systems, including communication, control and signal processing systems [Simulink 2009]. Simulink can be extended by libraries. A library of particular importance for this work is TrueTime [Ohlin et al. 2007]. TrueTime provides resources for the simulation of real-time control systems. It provides a set of functions and programming building blocks for specifying such systems: kernel block, network block, Wireless Network and a battery block [Ohlin et al. 2007]. MATLAB with Simulink and TrueTime thus provide resources for defining an energy model for a WSN and support IEEE 802.15.4 [IEEE 2006], which is becoming a *de facto* standard for the physical layer and medium-access control sublayer of sensor networks.

There are currently many different simulation tools for wireless sensor networks. Some of them, closely related to our system, are: TOSSIM [Levis et al. 2003], Atemu [Polley et al. 2004], SENSE [Chen et al. 2004], ns-2 [Fall and Varadhan 2008] and J-Sim [Sobeih et al. 2005].

TOSSIM is a discrete event simulator for wireless sensor nodes which are based on the TinyOS operating system. TOSSIM simulates the behavior of hardware components, such as the ADC, clock and EEPROM memory, the flash boot sequence, and components of the wireless communication protocol stack. The main feature of this simulator is the fact that the code used for simulation can be installed with no modifications on real

devices.

Atemu (Atmel Emulator) is a sensor network simulator which is based on the standard Mica2 [Crossbow 2009] architecture and the AVR (Advanced Virtual RISC) microprocessor (although it can simulate heterogeneous WSN). The system provides components to be used in the specification of the simulation scenario, such as processor, clock and radio device. It allows the simulation of low-level operations on sensor nodes.

SENSE is a discrete event simulator for sensor networks which is based on the IEEE 802.11 standard with DCF (Distributed Coordination Function) as its physical / MAC sublayer. This simulator implements a battery model, which enables a control over energy spent based on the used electrical current.

NS-2 (Network Simulator 2) is a discrete event simulator that has a broad support for simulation of wired and wireless networks. Its support for modelling energy is, however, very limited.

J-Sim is a simulation tool for WSN developed in Java. It provides components for modelling typical WSN elements, such as battery, processor and radio models, and a phenomena generator. It is based on the IEEE 802.11 standard at the physical layer and MAC sublayer.

Among the systems cited above, MATLAB was the system which provided most adequately basic building blocks for constructing MASIM, specially due to its support for IEEE 802.15.4 and bulding blocks for specifying energy models.

3. MASiM

In this section we describe the main features and some implementation aspects of MASiM. MASiM was built using MATLAB with Simulink and TrueTime. We first provide a description of the system from a conceptual point-of-view, presenting the adopted WSN model (Section 3.1) and agent model (Section 3.2). We then describe the main system classes (Section 3.3), the system task model (Section 3.4).

3.1. WSN Model

In MASiM, a WSN is composed of two types of nodes: sensor node and base station. A WSN might have more than one base station. Base stations and sensor nodes have different hardware and software characteristis, but all sensor nodes are homogeneous. All nodes, including the base station, are capable of moving in the environment. Thus communication links between nodes are dynamic.

The main function of a sensor node is to monitor some physical characteristic of an environment and transmit the sensed data to the base station through wireless links. Each node maintains information about its energy level, its position in the environment (localization) and about time. The clocks of nodes are all synchronized. Each node is uniquely identified in the network.

Sensor nodes are organized in a mesh network (mesh) as defined by Zigbee[1]. Not all sensor nodes might reach the base station in a single communication hop. Thus, communication between sensor nodes and the base station will be primarily multihopping.

[1]http://www.zigbee.org

The range of the base station radio device is assumed to be larger than the range of sensor node radios. Thus, the resulting communication network is asymmetric, in the sense that a base station might reach a sensor node with a single hop, but this node might not reach the base station directly. The nodes are aware of their active neighbouring nodes and each node periodically broadcasts its location to its neighbours. Additional data, such as the energy level of the node, might be piggybacked in these periodical messages.

The network uses IEEE 802.15.4 without beacon [IEEE 2006] as the physical layer and MAC (Medium Access Control) sublayer. Since there is no beacon, there is no formation of superframe, and (unslotted) CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance) is used as the medium access control mechanism.

3.2. Mobile Agent Model

The mobile agent model adopted in MASiM is based on the model defined by FIPA [FIPA 2004]. A mobile agent can be in one of the following states: **Started**, **Active**, **Suspended**, and **Waiting**. The meaning of these states and the possible state transitions are described below.

On each node, mobile agents execute on logical places called here *agencies*. From a functional point-of-view, an agency represents the needed functionality that must be present on a node so that agents can execute there. We will, however, use the terms agency and node here interchangeably to denote the place where agents execute.

A mobile agent is instantiated at a node in the **Start** state. In this state, the agent receives its mission and its unique identifier. The user of MASiM is responsible for ensuring the uniqueness of this identifier. When the agent starts execution, it enters the **Active** state.

While at the **Active** state, an agent performs the operations defined in its mission. Two special operations are the movement and cloning operations. The implementation of these operations involves the execution of a specific protocol for mobility and cloning. Before an agent can move from an agency to another or before an agent can be cloned at a specific node, a negotiation protocol is carried out between the agency where the agent currently is, which will be called here the *original agency*, and the *target agency*, i.e., the agency to where the agent wants to move or the agency where the new instance of the agent will be created. The agent stays in the **Suspended** state from the beginning of the movement or cloning process until its end.

The mobility protocol is illustrated in the activity diagram in Figure 1. This protocol starts when an agent calls a movement operation, defined at the interface of the original agency. One of the parameters of this call is the identifier of the target agency. At this point, the agent enters the **Suspended** state. The original agency sends a message that contains a copy of the agent (code and state) for the target agency. The target agency will check whether there are sufficient resources for instantiating the agent locally. If the resources are sufficient, the agency creates the new instance of the agent. This instance is, however, created in the **Suspended** state. The target agency then sends a message to the original agency confirming that the movement operation was accepted. When the original agency receives this message, it destroys the local instance of the agent and, after that, sends a message to the target agency, allowing it to start the execution of its instance of the agent. When the target agency receives this message, it resumes the execution of the

agent. The agent enters again the **Active** state. If the target agency does not have enough resources to execute the agent, it sends a message to the original agency denying receiving the agent.

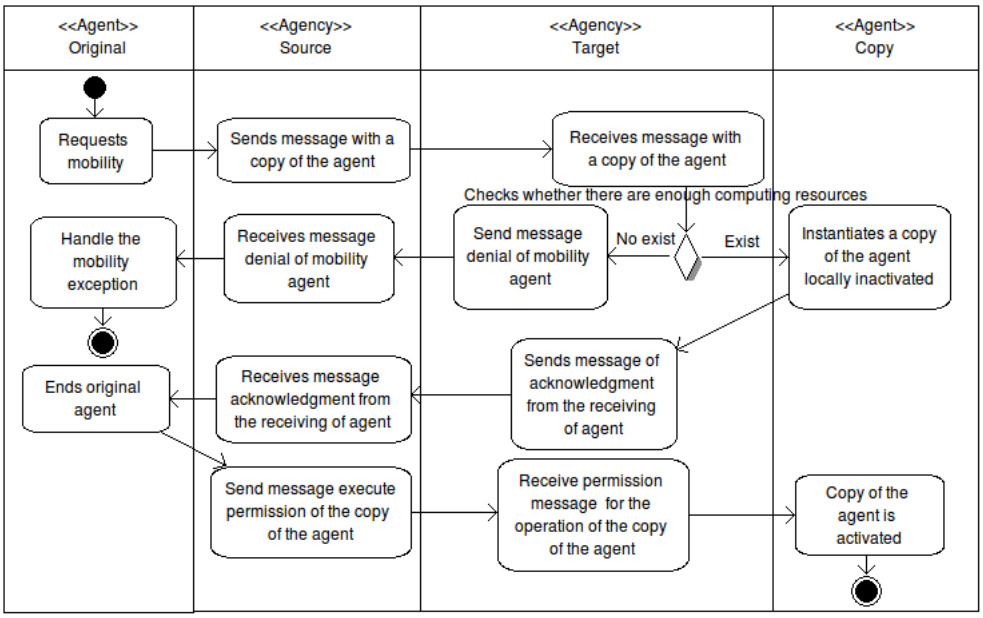


Figure 1. Activity Diagram of the agent mobility protocol.

The cloning protocol is illustrated in the activity diagram depicted in Figure 2. It begins in a way similar to the mobility protocol. The cloning operation has as one of its parameters the identifier of the remote agency where the clone agent shall be created (the target agency). After the agent calls the cloning operation, the agency where it is (original agency) sends a message to the target agency. The agent enters the **Suspended** state. This message contains the code and state of the agent to be cloned. The target agency checks if there are enough resources to create the local copy of the agent. If yes, it sends a message to the original agency confirming the creation of the clone agent, but, differently from the mobility protocol, the agent instance at the target agency immediately enters the **Active** state. When the original agency receives the confirmation message, the local instance of the agent enters the **Active** state too. As in the mobility protocol, if there are not enough resources for creating the agent at the target agency, the target agency sends a message to the original agency denying creating the clone.

During an agent movement or cloning operation, the agent execution and data state are transformed into a byte stream (a process called serialization), to be transfered to the target node. At the target node, this byte stream is used to reinstantiate the agent or create a new copy of it (deserialization) [Guenes et al. 2003]. This process is performed by the TrueTime toolbox, and how this is done is out of the scope of this work. However, the agent size can increase during its execution, due to data collected along its path. In MASiM, the user is responsible for determining how the agent size will grow at each migration.

An agent can send and receive messages. The process of receiving messages is blocking. When an agent is waiting for a message, it enters the **Waiting** state. It only

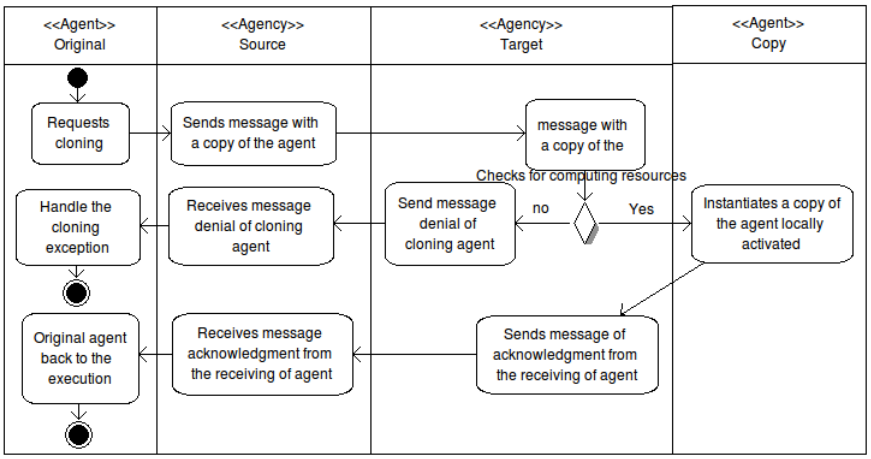


Figure 2. Activity Diagram of the protocol for cloning the agent.

enters the **Active** state again when a message arrives. If a message is sent to a specific agent at an agency and this agency is not at this agency, the message is discarded.

3.3. Main MASiM Classes

The main classes in MASiM are: **NodeMessage**, **AgencyMessage**, **MessageInformationAgency**, **AgentMessage**, **InformationAgency**, **Phenomenon**, **Agent**, **Mission**, **State**, **Nodes** and **Agency**.

Nodes, agencies and agents communicate with each other by exchanging messages. In MASiM messages are exchanged asynchronously. If an entity sends a message to another non-existent entity, the message is discarded. MASiM defines a message class hierarchy. Three types of messages are defined: **NodeMessage**, **AgencyMessage** and **AgentMessage**.

The **NodeMessage** class models messages exchanged between nodes. The main attributes of this class are: *sender*, *target* and *identifier*. The *sender* refers to the node that sends the message. The *target* attribute represents the node to which the message is sent. The *identifier* attribute refers to the unique identifier of the message. The contents of a **NodeMessage** is the data that is sent to the destination node. This content is represented by the **AgencyMessage**.

An agency message, represented by the **AgencyMessage** class, like **NodeMessage**, has as attributes the agency that sent the message (*sender*), the agency to which the message is being sent (*target*) and the unique identifier of the message (*identifier*).

MessageInformationAgency is an abstract class that can be extended by the classes **Agent**, **InformationAgency** and **AgentMessage**. The **MessageInformationAgency** class contains the phenomenon of the sender agency. Class **Phenomenon** represents the phenomenon that a given node monitors. This class has an attribute that identifies the type of phenomenon (*type*).

The **Agent** class models the agents. This class has as attributes a unique (agent) *identifier*, an attribute that contains the set of agencies that the agent has already visited (*trace*) and, if the agent is a clone of another agent, it contains also an attribute that

identifies the agency from where the agent was created (*parentAgency*). This reference becomes obsolete if the original agent moves and does not send a message to its cloned agents informing its new location. At the time an agent is instantiated, the agent receives its identifier and its mission. The **Agent** class contains an additional attribute which represents this mission (*mission*). This attribute can not be changed.

In addition to its attributes, the **Agent** class has also the following methods: *move*, to move the agent to a target agency; *clone*, to create a clone of the agent; *sendMessage*, to send a message to one or a set of agents - if a particular agent is not specified, the message is sent to all agents in a particular agency; *getMessage*, to receive a message; *enabled*, to verify if the agent is in the **Suspended** state; and *executeMission*, to initiate agent execution.

The mission of the agent is represented by the **Mission** class. This class has a set of states. These states determine the set of operations that the agent must execute. These operations are defined by the user. The **Mission** class contains an identifier of the current execution state (*currentStateIdentifier*) and the identifier of the next execution state (*nextStateIdentifier*) of the agent. These attributes are used to determine what state the agent must enter after moving or being cloned.

The **MessageAgent** class models messages that an agent sends to another agent at a certain agency. This class has the following attributes: *sender*, which identifies the agent that has sent the message; *receiver*, which identifies the agent to which the message is being sent; *targetAgency*, which identifies the agency where the receiver agent must be to receive the message; and *messageContents*, which conveys the actual information to be transmitted. When the target agency receives a message, it stores it in an appropriate memory buffer, where the agent can read it. If the receiver agent is not at the target agency when the message arrives, the agency discards the message. No confirmation of receipt is sent to the sender agent.

The **AgencyInformation** class represents the data that **AgencyMessage** might contain during the execution of the mobility or the cloning protocol.

The **Node** class represents a network node. Each node is uniquely identified by the attribute *identifier*. The battery energy level of the node is represented by the attribute *battery*. The value of the local clock of the node is represented by the attribute *clock*. The attribute *memory* determines the amount of free memory available on the node. A node in a WSN can be of one of two types: base station or sensor node. The base station is modelled with the **Node** class. A sensor node is modelled with the **SensorNode** class. This class extends the features of the **Node** class and contains an additional attribute, called *Sensor*, which is the current value of the sensor. The type of phenomenon monitored by the node is represented by the attribute *phenomenon*.

An agency is modelled by the **Agency** class. This class is uniquely identified by the attribute *identifier*. The value of this attribute will match the same value of the identifier of the node where the agency is. The methods it provides are used by agents to access information about the resources of the node. To get information on the energy level of the node, the method *getEnergyLevel* is invoked. The method *getFreeMemory* returns how much free memory is available on the node. The method *getValueSensor* returns the current value of the sensor. The type of phenomenon monitored by the node is

represented by the method *getPhenomenon*.

The role of the controller class **Simulation** is performed by **MATLAB** with **Simulink** and the **TrueTime** toolbox. The **Simulator** is responsible for executing the simulation, controlling time and the simulation variables, like the level of the nodes batteries, nodes mobility and the execution of the methods of the network components.

In all classes described above there are methods for reading and writing all the attributes of the class.

3.4. Programming Tasks

MASIM was programmed using C ++ MEX programming language and was based on the resources provided by the TrueTime toolbox. The use of TrueTime requires that the programming follow a model based on tasks, which can be periodic or aperiodic. A task can communicate with other tasks through shared memory regions called **Mailboxes**. To follow this programming model, MASIM tasks were organized into two groups: Node Tasks and Agency Tasks. Node Tasks are those related to operations present in all nodes, and Agency Tasks are those existing only in nodes that run the agency. Thus, the Node Task are:

- **Node's Received Message Handler:** aperiodic task triggered by an interrupt when a message arrives. This task has the role of receiving and forwarding the message to the task responsible for it;
- **Neighborhood Message Handler:** aperiodic task triggered by Node's Received Message Handler with a role to process messages with information about neighboring nodes. This neighborhood information are stored in memory for the use of agents;
- **Simple Message Handler:** task triggered by Node's Received Message Handler with a role of dealing with messages defined by a user. The way that this message should be handled is specified by the user;
- **Sent Message Handler:** aperiodic task responsible for sending messages to another node. This task is triggered after a particular task to put a information to be sent in a Mailbox. This task gets the information and creates a message to be sent;
- **Notification to the Neighborhood Node:** periodic task whose role is to send a message to neighboring nodes with information about the current node. The information that are sent are: the energy level, the node identifier and coordinates. This task puts these information in the Mailbox and triggers an event that activates the task Node's Received Message Handler. This last task will be responsible for sending messages on the node to all neighboring nodes.

The tasks that have a role to perform operations related to the agency are as follows:

- **Agency's Received Message Handler:** aperiodic task triggered by the Node's Received Message Handler. This task has the role of receiving and processing messages in its Mailbox. If the message destination is the agency, this task will process this message. This occurs when a message is related to mobility or cloning protocol. However, if the message contains an agent, this task forwards it to Creator Agent task, which is responsible for instantiating the agent on the local node.

The message destination may a local agent. Thus, this message is forwarded to the Controller Execution Agent task, which is responsible for delivering the message to the agent, if one exists. Otherwise, this message is discarded;

- **Creator Agent:** aperiodic task triggered by the task Agency's Received Message Handler. This job role is to instantiate a particular agent that has been received during mobility or cloning for the current node;
- **Controller Execution Agent:** periodic task whose role is to run the agents following a run queue. This task executes and controls the cycle of the mission of each agent.

4. Assessment of The MASIM Tool through Simulation Scenarios

4.1. Manufacturing environment scenarios

Aiming to build scenarios with mobile agents in a distributed system, a manufacturing environment was adopted, similar to that proposed by [Krishnamurthy and Zeid 2004]. In such environments, it is necessary to monitor the operation of various equipments, collect data and forward them to a centralized operation room where decisions are taken based on individual or collective information obtained by this monitoring.

A set of sensor nodes distributed in some factory environments can monitor equipments, and mobile agents could be used both to collect data in a intelligent and selective way and to convey information about the plant to destination (a base station). It was considered scenarios where a node can communicate directly with the base station only if it is in a certain distance of the base station (they are neighbors). Moreover, there is the possibility that there are unconnected nodes, that is, nodes without neighbors in the network. Finally, the data collected can be related to information of alarms from equipment malfunctioning, and there may be a need for a maximum time (deadline) for which this information reaches the operations room, and the data has a freshness constraints.

Therefore, through the use of Simulink/Matlab files (MDL files) in MASIM, some manufacturing scenarios were modeled based on two different approaches: (i) a simple diffusion-based approach; and (ii) mobile agent-based approach.

4.1.1. First Approach: A Simple Diffusion Protocol (without mobile agents)

In this approach, when a node receives a message with a monitored event from another node, it attaches the received data to its message and forwards it to neighbors. After that, when the first message arrives to the base station, it is considered that the mission is complete.

4.1.2. Second Approach: Agents that Moves Beyond The Range of Base Station

The protocol based on agents has the goal to achieve a trade-off between energy consumption and network coverage (Figure 3). In **State 0**, there is an agent in the base station that clones itself to all neighboring nodes. In **State 1**, clones choose to move to their neighboring nodes of higher energy level until they find a particular node where the target phenomenon was observed. When this happens, in **State 2**, each agent tries to send

a message to the agent in the base station. However, it is possible there may be a clone agent far from the base station. In this case, this agent moves back toward base station, until it reaches a node neighboring the base station, before it send the message:

To simplify the understanding of this algorithm, states **State 3, 4, 5 and 6** only carry out the operation to complete the mission of the agent. The **States 4 and 5** handle exceptions that may occur in the cloning and mobility, respectively. When the operations of receiving target phenomenon and find phenomena are carried out successfully, the final **States 3 and 6** are invoked.

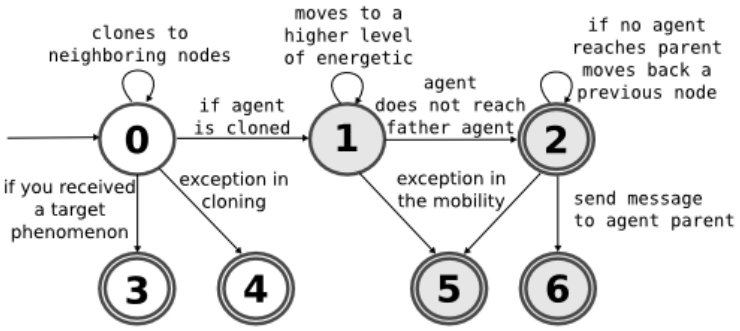


Figure 3. Specification of the agent's mission.

4.2. Comparison of agent-based and message-based approaches

In order to exercise the simulator, both approaches based on agents and based on diffusion messages were implemented with two densities of nodes: 50 and 100 nodes. For each density, 20 simulations were performed, each one with the duration of $100s$. In each simulation, the nodes are redistributed randomly in area.

The nodes used in the simulations are based on the hardware configuration of the Mica2. Thus, the energy source of the sensor nodes are two batteries with $3V$ and $27000J$ of energy. These nodes were randomly scattered over a fully plan area of 300m^2, and after the deployment, the nodes are considered fixed (non-mobile nodes). In each simulation scenario, it is also considered that only 10% of nodes monitors the target phenomenon. The radio signal from the antenna has an operating range of $99m$ and there may be disconnected nodes in the network. In each node there is energy consumption due to data processing and communication (sending and receiving messages). In this work, is not considered the energy consumption due to sensor or actuator tasks. Thus, the receipt and sending of one message and processing performed by the processor spends $16.62 * 10^{-4}J$, $9.6 * 10^{-4}J$ and $4.8 * 10^{-4}J$, respectively [Polastre et al. 2004].

Simulations show that the proposed approach based on agents is the better choice in respect to energy consumption (Table 1). There was a considerable increase in energy consumption by the nodes in the approach based on diffusion, because, as the number of nodes increase, the amount of messages received for each node also increase. However, the diffusion approach is more effective in coverage, because, at the end of the simulation, all nodes have been achieved by the protocol in all scenarios. The coverage parameter means the amount of network nodes that were recognized by the base station through a

Table 1. Energy consumption and coverage.

	Energy consumption			*Coverage*	
	50 *nodes*	100 *nodes*		50 *nodes*	100 *nodes*
Agent-based	0,238J	0,239J		14%	12%
Diffusion-Based	0,418J	0,601J		100%	100%

specific protocol. The average coverage for the agent-based approach were 14% and 12% for densities of 50 and 100 nodes, respectively.

In respect of time constraints, the diffusion-based protocol may be more effective to attend tight deadlines, because messages arrive at base station in a minimum time ($2s$). This value was much smaller than that obtained by agent-based approach (average of $15s$).

5. Conclusions

This work aimed to develop a tool for simulating agents in Wireless Sensor Networks called MASIM. With this tool it is possible to specify simulation scenarios using static or mobile agents. The flexibility in MASIN was achieved through the provision of functions and block diagrams, which facilitate the programming of scenarios and agents mission, determining different behaviors of the agent during its lifetime.

The programming interface developed allows the use of functions and data structures facilitating users to program the simulation models: the network topology may be defined using the block diagrams available; the programming of nodes and defining the mission of agents are made through the C++ MEX; and mission of agents can be modeled as a state machine. To exercise the simulator, showing its flexibility, was built simulations of scenarios based on mobile agents and dissemination of messages in a WSN. Results from these simulations shown possible advantages and disadvantages of using the approach of mobile agents concerning energy saving and network coverage metrics.

As future work, the tool is going to allow the user to specify custom information about the neighborhood. Moreover, it is going to allow the definition of custom protocols for cloning and mobility of agent. It is expected too that the user is going to able to program the settings using the MATLAB language, in addition to C++ MEX.

References

Chen, G., Branch, J., Pflug, M. J., Zhu, L., and Szymanski, K. (2004). Sense: A sensor network simulator. http://www.cs.rpi.edu/ szymansk/papers/wpcn.04.pdf. Accessed on Jan. 2010.

Chen, M., Kwon, T., Yuan, Y., and Leung, V. C. (2006). Mobile agent based wireless sensor networks. *Journal of Computers*, 1.

Crossbow (2009). Mica2 - wireless measurement system. "`http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf`". Accessed on Jan. 2010.

Fall, K. and Varadhan, K. (2008). *The ns Manual*. UC Berkeley and LBL and USC/ISI and Xerox PARC.

FIPA (2004). Fipa agent management specification. "`http://www.netlib.org/lapack`". Accessed on Jan. 2010.

Guenes, M. H., Tiersem, M. E., Yildiz, M., and Kuru, S. (2003). Performance analysis of mobile agents using simulation. In *Proc. of the Advanced Engineering Design Conference (AED2003)*, Praga, Czech Republic.

IEEE (2006). Part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans). Technical report, IEEE Computer Society.

Jansen, W., Mell, P., Karygiannis, T., and Marks, D. (1999). Applying mobile agents to intrusion detection and response. Technical report, National Institute of Standards and Technology Computer Security Division, Washington, D.C, EUA.

Krishnamurthy, S. and Zeid, I. (2004). Distributed and intelligent information access in manufacturing enterprises through mobile devices. In *Journal of Intelligent Manufacturing*, pages 175 – 186. Kluwer Academic.

Lange, D. B. and Oshima, M. (1999). Seven good reasons for mobile agents. *Commun. ACM*, 42(3):88–89.

Levis, P., Lee, N., Welsh, M., and Culler, D. (2003). Tossim: accurate and scalable simulation of entire tinyos applications. In *Proc. of the 1st Int. Conf. on Embedded Networked Sensor Systems*, pages 126 – 137, Los Angeles, California, EUA. ACM.

Malik, H. and Shakshuki, E. (2007). Data dissemination in wireless sensor networks using software agents. In *Annual Int. Symp. on High Performance Computing Systems and Applications*, page 28, Saskatoon, Saskatchewan, Canadï¿½. IEEE Computer Society.

MathWorks, T. (2009). *MATLAB Getting Started Guide*. The MathWorks, Inc.

Ohlin, M., Henriksson, D., and Cervin, A. (2007). *TrueTime 1.5 - Reference Manual*. Department of Automatic Control, Lund University.

OMG (1997). Mobile agent system interoperability facilities specification. "`http://www.omg.org`". Accessed on Jan. 2010.

Polastre, J., Hill, J., and Culler, D. (2004). Versatile low power media access for wireless sensor networks. In *Proc. of the 2nd Int. Conf. on Embedded Networked Sensor Systems*, pages 95 – 107, Baltimore, MD, EUA. ACM.

Polley, J., Blazakis, D., McGee, J., Rusk, D., and Baras, J. (2004). Atemu: a fine-grained sensor network simulator. In *First Annual IEEE Comm. Society Conf. on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004*, pages 145 – 152.

Simulink (2009). *Simulink 7 - Simulation and Model-Based Design*. The MathWorks. Accessed on January 2010.

Sobeih, A., Chen, W.-P., Hou, J. C., Kung, L.-C., Li, N., Lim, H., ying Tyan, H., and Zhang, H. (2005). J-sim: A simulation and emulation environment for wireless sensor networks. *IEEE Wireless Communications magazine*, 13:2006.

Ye, W., Heidemann, J., and Estrin, D. (2001). An energy-efficient mac protocol for wireless sensor networks. pages 1567–1576.

Yu, Y., Krishnamachari, B., and Prasanna, V. K. (2004). Issues in designing middleware for wireless sensor networks. *IEEE Network*, 18:15 – 21.

12th Brazilian Workshop on Real-Time and Embedded Systems

♦

Work in Progress Session

Framework para Integração entre Ambientes Inteligentes e o Middleware do Sistema Brasileiro de TV Digital

Reiner F. Perozzo[1], Carlos E. Pereira[1]

[1]Departamento de Engenharia Elétrica – Universidade Federal do Rio Grande do Sul
Av. Osvaldo Aranha, 103 – 90035-190 – Porto Alegre – RS – Brasil

reiner.perozzo@ufrgs.br, cpereira@ece.ufrgs.br

Abstract. *Since December 2007, Brazil is implementing the Brazilian Digital Television System (SBTVD). Moreover this new system to provide high definition images, offers interactivity through the Ginga middleware, which transforms the digital TV receivers - set-top boxes (STBs) – in the computing platforms of interaction between viewers and applications that run on the middleware. These STBs are becoming increasingly present in homes, which enables its integration with the Ambient Intelligence (AmI). This paper proposes a framework for integration between AmI and the SBTVD, in order to allow the management of automated services in AmI, by means of television sets and STBs with Ginga middleware.*

Resumo. *Desde dezembro de 2007 o Brasil está implementando o Sistema Brasileiro de TV Digital (SBTVD). Além desse novo sistema proporcionar imagens em alta definição, oferece a interatividade, através do middleware Ginga, o qual transforma os receptores de TV digital – set-top boxes (STBs) – em plataformas computacionais de interação entre os telespectadores e as aplicações que são executadas sobre o middleware. Esses STBs estão se tornando cada vez mais presentes nas residências, o que possibilita a sua integração com os Ambientes Inteligentes (AmI). Este trabalho propõe um framework para integração entre AmI e o SBTVD, a fim de permitir que os serviços automatizados do AmI possam ser gerenciados através de televisores e STBs com o middleware Ginga.*

1. Introdução

A chegada da TV Digital no Brasil está trazendo consigo pelo menos três grandes vantagens: (i) melhor qualidade de som e imagem; (ii) portabilidade, pois permite ao telespectador assistir TV em movimento, utilizando dispositivos móveis como telefones celulares e TVs portáteis; (iii) interatividade, a qual permite que o telespectador consiga interagir com a programação e acessar uma variedade de serviços, tais como: *t-government*, *t-banking*, *t-commerce* e outros [Silva 2008]. O acesso a esses serviços e a TV interativa (TVi) é possível no Sistema Brasileiro de TV Digital (SBTVD) graças a especificação de um *middleware* [Ginga 2009], composta por três módulos principais: Ginga-CC, Ginga-NCL e Ginga-J. O Ginga consiste na camada de *software* posicionada entre o código das aplicações e a infraestrutura de execução (plataforma de *hardware* e sistema operacional) dos *set-top boxes* (STBs), oferecendo suporte às aplicações interativas [Soares 2008].

Tendo em vista os recursos oferecidos pelo Ginga e a presença crescente dos STBs nas residências, estes podem ser utilizados como plataformas computacionais integrantes dos chamados Ambientes Inteligentes (AmI) - ambientes automatizados com dispositivos capazes de se auto-gerenciarem e cientes de tudo que os cercam [Ark e Selker 1999], [Ducatel 2001]. No AmI predomina a visão de um mundo cercado por uma grande quantidade de dispositivos que oferecem suporte inteligente nas atividades diárias dos usuários. Nesse caso, há uma relação direta entre os AmI e a automação predial/residencial, pois o desenvolvimento de projetos nessa área necessita de espaços físicos automatizados, o que inclui sensores, atuadores e diversos sistemas de controle [Nazari 2009], [Edwards 2006]. Uma ampla variedade de serviços em áreas como segurança (controle de acesso, identificação de usuários), conforto (controle de temperatura e umidade, iluminação) e entretenimento estão surgindo diariamente, trazendo consigo muitos desafios a serem resolvidos, como, por exemplo, o gerenciamento dos serviços de automação e os sistemas de interação homem-máquina (IHM) existentes nesses ambientes [Tsourakis 2006], [Perozzo e Pereira 2008].

Dessa forma, este trabalho propõe a integração entre AmI e o SBTVD, através de um *framework* que permita aos usuários utilizar o televisor não somente para assistir a programação das emissoras em alta definição, mas também, para gerenciar os serviços e os dispositivos de automação existentes no AmI.

2. Trabalhos Relacionados

O desenvolvimento de aplicações para TV Digital vem sendo bastante explorada recentemente no Brasil e principalmente em países onde a TV interativa já está presente há mais tempo [Dolan 2001], [Hopkins 2009], sendo possível encontrar diversos grupos de pesquisa trabalhado em projetos cujo objetivo está focado na construção de aplicações interativas [Peta5 2009], [Filgueiras e Giannoto 2009]. Dentre esses trabalhos destaca-se a proposta de [Simioni e Roesler 2006], que apresenta um *framework* para criação de aplicações de TV Digital e cujo objetivo é facilitar e agilizar o desenvolvimento desse tipo de aplicação. Esse *framework* é baseado na estrutura *Globally Executable - Multimedia Home Platform* (GEM - MHP) [DVB 2010]. Em linhas gerais, [Simioni e Roesler, 2006] concentram a proposta no desenvolvimento de aplicações interativas voltadas ao comércio eletrônico e ao acompanhamento de informações referente ao mercado financeiro. O *framework* é responsável por gerenciar as informações recebidas pelo STB e está dividido em três camadas: i) Catálogo: responsável pelo gerenciamento da base de dados; ii) Painel: responsável por montar a interface da aplicação; iii) Aplicação: utiliza as informações geradas pelas outras camadas. A camada Catálogo irá manipular os arquivos recebidos pelo carrossel de dados e fornecer à camada Painel os itens que sejam solicitados. Esta, por sua vez, disponibiliza uma interface para a Aplicação. Basicamente, as aplicações que utilizam esse *framework* recebem dados de um arquivo *eXtensible Markup Language* (XML) que servem para a atualização das interfaces de interação. De acordo com o conteúdo da informação recebida, a aplicação recria a interface, substituindo um determinado item.

Quanto à utilização de STBs e TVs para gerenciamento de residências, é apresentado por [Cabrer 2006] uma proposta de integração entre duas tecnologias: a MHP para TV Digital e a *Open Services Gateway Initiative* (OSGi) [OSGi 2009] como uma plataforma para configurações de *gateways* residenciais. Com os avanços tecnológicos que vem ocorrendo nos últimos anos, os *gateways* residenciais são

fundamentais para a criação de pontes de comunicação entre os ambientes inteligentes, seus dispositivos e o mundo externo. Na proposta de [Cabrer 2006] o usuário pode interagir com os serviços existentes na residência, acionando dispositivos através do televisor. Dessa forma, há a integração entre o MHP e o OSGi, sendo que o primeiro está orientado a funções, e o segundo orientado a serviços. Como as duas tecnologias possuem arquiteturas distintas foi desenvolvido um XbundLET, uma aplicação que permite a interação natural entre o MHP e o OSGi. Um Xbundlet não apenas define uma ponte de comunicação entre essas plataformas, como também constitui em um elemento de *software* híbrido que pode ser executado em ambas as arquiteturas. A integração entre diferentes plataformas também é apresentada na proposta de [Viana 2009], cujo objetivo é a criação de uma interface de acesso chamada Ginga OSGi Bridge, capaz de oferecer às aplicações Java e NCL uma ponte de acesso transparente ao *framework* OSGi. Nesse caso, as aplicações desenvolvidas poderiam verificar estados de sensores, acionar dispositivos e interagir com os serviços residenciais que estivessem disponíveis no ambiente e, consequentemente, no *framework* OSGi. Por outro lado, a proposta de [Viana 2009] necessitaria que a especificação do *middleware* Ginga fosse ampliada para suportar a comunicação transparente entre as aplicações Ginga e o *framework* OSGi.

3. Framework Proposto

Nesta seção é apresentada a proposta de um *framework* responsável pela integração entre AmI e o SBTVD, a fim de permitir que os serviços e os dispositivos de automação presentes no AmI possam ser acessíveis por aplicações interativas que são executadas sobre o *middleware* Ginga. Nesse caso, a idéia principal é agregar valor aos receptores de TV Digital, permitindo que estes possam, além de decodificar o sinal da TV Digital, se tornarem plataformas computacionais de gerenciamento do AmI, oferecendo aos usuários uma outra possibilidade de interação.

3.1. Modelo Conceitual

O modelo conceitual do *framework* tem por objetivo facilitar a compreensão da proposta, através da identificação de alguns requisitos do domínio da aplicação e que o *framework* deve atender, tais como:

- (i) Auxiliar na construção de aplicações interativas para acesso aos serviços e dispositivos de automação existentes nos AmI;

- (ii) Permitir a criação de projetos de aplicações interativas independentes de plataforma de *hardware*, através de modelos orientados a objetos;

- (iii) Possibilitar a reutilização de projetos, otimizando o tempo de desenvolvimento de novas aplicações;

- (iv) Permitir a geração automática de código, criando aplicações baseadas no perfil do *hardware* da plataforma alvo e no perfil da linguagem de programação suportada pelo *middleware* de interatividade;

- (v) Permitir que as aplicações construídas sejam capazes de, em tempo de execução, realizarem a descoberta de novos dispositivos e serviços que são inseridos no AmI e adaptarem as interfaces de acesso com base nos resultados obtidos dessa descoberta.

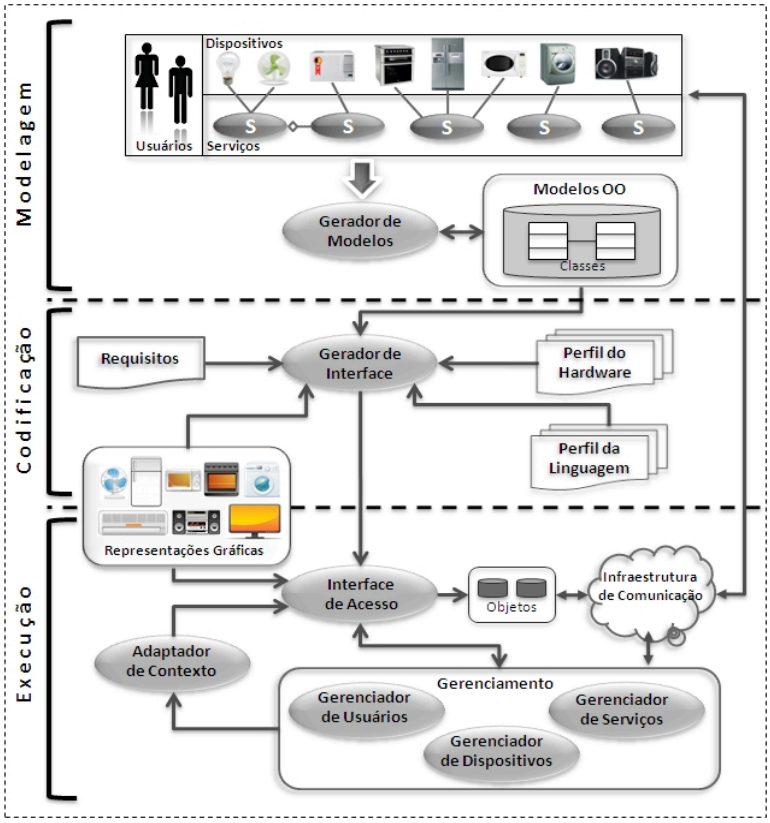


Figura 1. Arquitetura do Framework Proposto

Dessa forma, conforme ilustrado na Figura 1, a arquitetura do *framework* proposto está dividida em três níveis: **(i) Modelagem**: esse nível contém uma representação computacional do mundo real, ou seja: os usuários, os dispositivos físicos de automação e os serviços disponíveis no AmI são representados através de modelos orientados a objetos; **(ii) Codificação**: baseado na modelagem e nos requisitos do projeto, esse nível é o responsável pela codificação das aplicações interativas que serão executadas nas plataformas alvo com suporte ao Ginga, criando uma interface de acesso aos dispositivos e serviços que se deseja instanciar; **(iii) Execução**: nesse nível são executadas as aplicações interativas e os módulos de adaptação e gerenciamento do AmI, os quais realizam desde funções mais simples, como o acionamento de dispositivos de automação e a verificação do estado atual de algum sensor, até a realização de funções mais complexas, como a descoberta e a disponibilização de novos serviços ou dispositivos que são inseridos dinamicamente no AmI.

Em cada nível da arquitetura existem diversos componentes que realizam funções específicas, colaborando uns com os outros, através de um fluxo de informações que é iniciado pelo **Gerador de Modelos**, o qual é o responsável por transformar em modelos computacionais todos os usuários, serviços e dispositivos existentes no AmI, tais como: sensores, atuadores e controladores. Os modelos orientados a objetos são extremamente importantes pelo fato de serem independentes das plataformas de *hardware*, o que facilita a reutilização de projetos. A partir da modelagem, o componente **Gerador de Interface** realiza uma leitura dos modelos orientados a objetos, criando uma associação entre as classes existentes nesses modelos e as representações gráficas disponíveis em uma biblioteca de imagens, cujo resultado

dessa operação é uma **Interface de Acesso** aos objetos (instâncias das classes). Essa interface é uma aplicação interativa que é executada nos receptores de TV Digital e que permite aos usuários gerenciar os serviços e os dispositivos de automação presentes no AmI, onde esses receptores utilizam da mesma infraestrutura de comunicação de dados disponibilizada para os serviços e dispositivos que compõem o ambiente. Além disso, o Gerador de Interface permite que a partir de um único projeto seja possível ter a mesma aplicação sendo executada em diferentes receptores de TV Digital, tanto os fixos (TVs e STBs) quanto os móveis (telefones celulares), de acordo com a especificação do *middlware* Ginga, a qual prevê a utilização das linguagens NCL e Java para o desenvolvimento de aplicações.

Quando a Interface de Acesso é inicializada, entram em ação os outros quatro componentes dinâmicos do *framework*: (i) **Gerenciador de Dispositivos**: o qual é o responsável por monitorar o estado de operação de cada dispositivo de automação e realizar a descoberta de novos dispositivos que possam ser inseridos no AmI; (ii) **Gerenciador de Serviços**: responsável pela descoberta dos novos serviços que são disponibilizados pelos dispositivos ou informar à Interface de Acesso caso algum novo serviço seja composto por outros serviços (*services composition)*; (iii) **Gerenciador de Usuários**: componente que identifica a inserção de um novo usuário ou de um novo perfil de usuários que utiliza o AmI, tais como: perfis de convidado, administrador do sistema, emergência, etc.; (iv) **Adaptador de Contexto**: com base nas informações obtidas pelos gerenciadores, esse componente adapta a Interface de Acesso atualizando as informações referentes aos usuários, aos serviços e aos dispositivos de automação. Por exemplo, no caso de ser inserido um novo dispositivo de automação no AmI, o Adaptador de Contexto notifica a Interface de Acesso para que seja criada uma nova instância daquele objeto, permitindo que o novo dispositivo fique disponível, automaticamente, na Interface de Acesso do usuário.

4. Conclusões e Trabalhos Futuros

Neste trabalho foi proposto um *framework* para integrar os AmI e o SBTVD, o qual é utilizado no desenvolvimento de aplicações interativas que servem para gerenciar os serviços e os dispositivos de automação existentes nos AmI. A principal contribuição deste trabalho está na arquitetura do *framework* e nos seus componentes que permitem agregar valor aos STBs e TVs, os quais além de decodificarem o sinal da TV Digital podem ser utilizados como plataformas computacionais de automação predial/residencial. Atualmente, o *framework* está em fase de implementação e estão sendo construídos cenários que servirão como base para estudos de casos e validação da proposta. Dentre os projetos que estão no domínio desse trabalho destaca-se o acordo de pesquisa firmada com uma empresa multinacional, fabricante de eletrodomésticos de linha branca, cujo escopo do projeto consiste em permitir que um refrigerador inteligente consiga ser gerenciado por receptores de TV Digital com suporte ao *middlewre* Ginga.

Referências

Ark W. S and Selker, T. (1999) “A Look at Human Interaction with Pervasive Computers”. In: IBM Systems Journal, v. 38, n. 9, p. 504 – 507.

Cabrer, M. R. et al. (2006) “Controlling the Smart Home form TV. In: International Conference on Consumer Electronics, 2006, Las Vegas: Proceedings… New York: IEEE, p. 421 - 429.

Dolan, M. A. (2001) "Report on Television Data Applications". <http://www.itl.nist.gov/div897/staff/barkley/tv-data-apps-mdolan.pdf>

Ducatel, K. et al. (2001) “Scenarios for Ambient Intelligence (ISTAG Report)”. In. Institute for Prospective Technological Studies (European Commission), Seville.

DVB (2010) " Open Middleware for Interactive TV". <http://www.mhp.org/>.

Edwards, W. K. (2006) “Discovery systems in ubiquitous computing”. In: Pervasive Computing, IEEE. Vol. 5, Issue 2, April-June, pp. 70-77.

Filgueiras, L. V. L. e Giannoto, E. C. (2009) "Estudo de aplicações interativas na TV usando rastreamento do olhar". In: 1 Simpósio Internacional de Televisão Digital (SIMTVD), Nov, Bauru, Brasil. pp. 391-409.

Ginga (2009) “Ginga Digital TV Middleware Specification for SBTVD”. <http://www.ginga.org.br>

Hopkins, B. (2009) "Creating Interactive TV Applications With the Tru2way Platform and OCAP". <http://java.sun.com/developer/technicalArticles/javame/iptv-tru2way>

Nazari, A. A. S. et al. (2009) “3DSim: Rapid Prototyping Ambient Intelligence”. <http://www.igd.fhg.de/igd-a1/projects/amilab/index.html>.

OSGi (2009) "Open Services Gateway Initiative". <http://www.osgi.org/>.

Perozzo, R. F. e Pereira, C. E. (2008) “Management of Services in Intelligent Environments for Mobile Devices”. In: 4th IET International Conference on Intelligent Environments, 2008, Seattle, USA. v. 1. p. 1-6.

Peta5, (2009) "Aplicações para TV Digital Interativa". <http://www.peta5.com.br>

Silva, A. (2008) “Receptores de TV Digital”. In. Revista da Sociedade Brasileira de Engenharia de Televisão, ISSN 1980-2331, n. 105, pp. 27-29.

Simioni, A., Roesler, V. “Um framework para o desenvolvimento de aplicações interativas para a Televisão Digital”. In: ERRC - Escola Regional de Redes de Computadores, 2006, Passo Fundo, Brasil. UPF. v. 1. p. 10-16.

Soares, L. F. G. (2008) “TV interativa se faz com Ginga”. In. Revista da Sociedade Brasileira de Engenharia de Televisão, ISSN 1980-2331, n. 105, pp. 30-35.

Tsourakis, N. et al. (2006) “An Architecture for Multimodal Applications over Wireless Data Networks”. In: Int. Conf. on Intelligent Environments, IEEE, p. 221 - 227.

Viana, N. S. et al. (2009) “A Convergence Proposal between the Brazilian Middleware for iDTV and Home Network Platforms”. In: 5th IEEE International Workshop on Networking Issues in Multimedia Entertainment (NIME'09), Las Vegas-USA. p. 1-5.

Modelo de Arquitetura para Construção de Plataformas de Software Embarcado

Gustavo A. F. B. Melo, Sérgio V. Cavalcante

Centro de Informática – Universidade Federal de Pernambuco (UFPE)
CEP. 50740-540 – Recife – PE – Brazil

{gafbm,svc}@cin.ufpe.br

Abstract. *The development of embedded software is still very difficult. A few tools assist the embedded application design with visual modeling and automatic code generation, but they fail to support the HAL. This paper proposes an open architecture for reusable and easily portable software platform design. The architecture is composed of nine layers which encapsulate and separate features and functionalities of the microcontroller, peripherals, electronic components, kernel and middleware. This work aims standardization for embedded software development which allows you to automate the process of elaborating software platforms.*

Resumo. *O desenvolvimento de software embarcado ainda é muito deficiente. Algumas ferramentas auxiliam o projeto de aplicações embarcadas com modelagem visual e geração automática de código, mas falham no suporte a HAL. Este artigo propõe uma arquitetura para projetar plataformas de software reusável e de fácil portabilidade. A arquitetura é composta por nove camadas que encapsulam e separam características e funcionalidades do microcontrolador, periféricos, componentes eletrônicos da placa, kernel e middleware. Este trabalho visa uma padronização no desenvolvimento de software embarcado que permite automatizar o processo de construção de plataformas de software.*

1. Introdução

O desenvolvimento de sistemas embarcados é um processo que envolve duas abordagens contrastantes. Por uma lado o desenvolvimento de hardware é suportado por métodos analíticos e ferramentas de síntese automática (EDA) [1], como IDEs para construção de ASICs e PCBs. Por outro lado o desenvolvimento de software embarcado não tem acompanhado a mesma velocidade e eficiência, principalmente no que diz respeito à automatização. Algumas poucas ferramentas auxiliam o projeto de software embarcado com modelagem visual e geração automática de código [2][3], mas têm uma deficiência no suporte às interfaces de hardware. Boa parte disso deve-se ao fato de que as plataformas de hardware utilizadas por sistemas embarcados apresentam arquiteturas bastante distintas [4]. Além disso, esses dois paradigmas (projeto de hardware e software) têm sido tratados de forma ortogonal. Grande parte das metodologias de co-design [5][6] baseiam-se em dividir o projeto em hardware e software, e depois desenvolvê-los separadamente. No entanto, alguns componentes de software podem ser modelados de modo que eles sejam diretamente relacionados a componentes de hardware.

AUTOSAR [7] é uma parceria de empresas a fim de estabelecer uma padronização aberta à indústria para arquiteturas elétrico-eletrônicas automotivas. O principal conceito de AUTOSAR é separação entre aplicação e infra-estrutura. Entretanto, esta abordagem é muito direcionada a aplicações automotivas, além de ser bastante restritiva. EPOS [4] é um sistema operacional baseado em componentes desenvolvido para permitir portabilidade da aplicação. Contudo, as abstrações de hardware designadas por esta abordagem não separam interface de acesso e controle de acesso, o que acarreta em menos modularidade.

Este artigo propõe uma arquitetura para construção da plataforma de software no nível de componentes. Ele descreve uma estrutura de software embarcado reusável e de fácil portabilidade, que é composta por nove camadas que encapsulam e separam características e funcionalidades do microcontrolador, periféricos, componentes eletrônicos da placa, kernel e middleware.

A seção 2 apresenta a arquitetura proposta para o desenvolvimento de software embarcado, bem como descreve cada uma das suas nove camadas. Um exemplo real de utilização da arquitetura é apresentado na seção 3. As conclusões do trabalho são expostas na seção 4.

2. Arquitetura

A arquitetura proposta utiliza os padrões de projeto Layered Pattern e Five-Layer Architecture Pattern [8]. Ela organiza o software embarcado em camadas, de modo a facilitar o reuso e a manutenabilidade dos componentes do sistema (Figura 1). Esta estrutura permite ao projetista abstrair a plataforma em termos de hardware e sistema operacional, proporcionando um alto grau de portabilidade da aplicação embarcada.

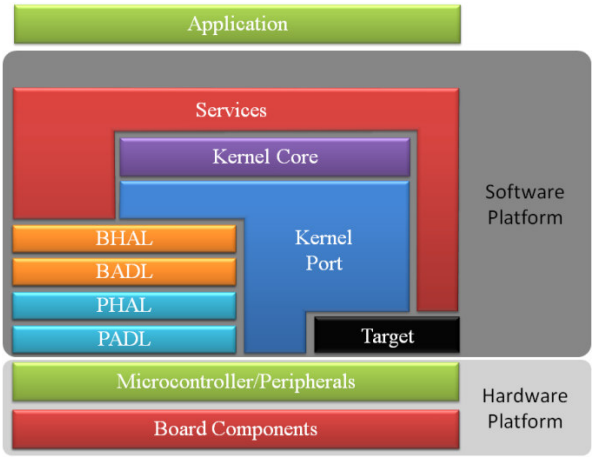


Figura 1. Embedded Software Architeture

A plataforma de hardware constitui-se dos componentes eletrônicos os quais o software é capaz de acessar. Exemplo: chip com o core do processador e seus periféricos, memória flash, LCD, LED ou FPGA. A plataforma de software provê as funcionalidades necessárias para lançar uma aplicação, dando suporte a escalonamento, gerenciamento de memória, acesso ao hardware e outros. Os componentes da plataforma de software são divididos em nove camadas: Target, PADL, PHDL, BADL, BHAL, Kernel Core, Kernel Port, Services e Application. As subseções seguintes descrevem com maiores detalhes cada uma das camadas da plataforma de software.

2.1 Target

Existem diferentes fabricantes que fornecem diferentes famílias de processadores. Cada uma dessas famílias possui suas peculiaridades. Além disso, vários compiladores distintos dão suporte a uma família. O código necessário para o startup do microcontrolador, definição dos vetores de interrupção e os tipos básicos são parcelas do software que normalmente diferem entre os compiladores e/ou processadores. Toda esta parte do software dedicada à combinação processador/compilador é encapsulada na camada Target. Dessa forma é possível mudar o compilador usado no projeto trocando apenas o componente Target. A abstração que esta camada proporciona também permite a portabilidade do resto do software para o caso de troca do microcontrolador por outro da mesma família e com diferente capacidade de memória.

2.2 PADL

Camada Dependente da Arquitetura do Periférico (*Peripheral Architecture Dependent Layer*) inclui os componentes de software que permitem o acesso direto aos periféricos do microcontrolador por meio de seus registradores. Cada família de microcontrolador possui um conjunto de periféricos com uma arquitetura particular. É função dos componentes desta camada definir a interface de acesso, independente do compilador, aos registradores de cada um dos periféricos. Assim podemos desenvolver as interfaces para cara periférico do microcontrolador sem se preocupar com as peculiaridades do compilador.

2.3 PHAL

Camada de Abstração de Hardware do Periférico (*Peripheral Hardware Abstraction Layer*), como o próprio nome diz, abstrai especificidades da arquitetura dos periféricos do microcontrolador e permite que as camadas superiores sejam independentes do hardware. Este conceito já é bem difundido no campo de sistemas operacionais [9][10][11].

O motivo de a camada PHAL ser separada da camada PADL é que um mesmo periférico pode ter mais de uma abstração de hardware. Um exemplo é a UART1 do microcontrolador LPC2368 [12]. Esta UART (*Universal Asynchronous Receiver Transmitter*) possui todas as funcionalidades das outras UARTs e mais a funcionalidade de controle de fluxo. Um componente *phal_uart* pode implementar todas as UARTs deste chip, inclusive a UART1, enquanto o componente *phal_modem_uart* pode contemplar as características específicas de controle de fluxo da UART1.

2.4 BADL

Camada Dependente da Arquitetura da Placa (*Board Architecture Dependent Layer*) possui os módulos que permitem o acesso, através dos periféricos, aos elementos da placa externos ao microcontrolador. Os componentes desta camada implementam o protocolo específico para acessar determinado hardware, como por exemplo uma determinada memória flash serial da ou um simples LED, por meio dos componentes PHAL.

2.5 BHAL

Camada de Abstração de Hardware da Placa (*Board Hardware Abstraction Layer*) é análoga a PHAL, abstrai as características específicas dos componentes eletrônicos da

placa, discutidos na seção 2.4. Uma memória flash serial Atmel família AT45 *badl_atmel_45_flash* é encapsulada por um módulo *bhal_flash_memory*, que possui uma interface genérica para todas as memórias flash.

2.6 Kernel Port

Esta camada tem o papel de garantir a portabilidade do kernel. Várias funções do kernel utilizam rotinas que dependem da arquitetura do microcontrolador. Como muitas vezes estas rotinas são escritas em *assembly*, elas dependem também do compilador. O *port* do kernel tem o objetivo de abstrair esta porção que é específica a microcontrolador/compilador [9]. Os componentes desta camada podem utilizar módulos da camada PHAL, Target ou ainda acessar diretamente a CPU.

2.7 Kernel Core

É a essência do kernel. Todas as funções do kernel, independentes do hardware e compilador, são implementadas nesta camada. Rotinas de criação de tarefas, troca de contexto, entrada e saída de seção crítica e incrementação do tempo são exemplos da infra-estrutura que o kernel fornece para os componentes da camada Service (seção 2.8), como, por exemplo, serviço de controle de tempo (*delay*), serviço de semáforo e serviço de fila, além do serviço de escalonamento.

Já que cada kernel apresenta funcionalidades distintas, fica difícil determinar interfaces genéricas para todos os kernels. Cada componente possui uma API diferente. Por outro lado alguns kernels suportam interfaces padronizadas para RTOS, como POSIX [13] e µITRON [14], o que pode aumentar a reusabilidade.

2.8 Services

Esta camada encapsula todo o acesso da aplicação à plataforma de software. Funcionalidades como escalonamento, gerenciamento de memória, acesso a dispositivos periféricos, storage, dentre outros, são disponibilizadas para as aplicações por meio de componentes chamados serviços. Esta idéia baseia-se no conceito de middleware [15], na qual um conjunto de serviços permite que múltiplos processos rodando em uma ou mais plataformas interajam independentemente da arquitetura (interoperabilidade).

Como foi dito, os dispositivos do sistema, sejam eles IO, comunicação, armazenamento de dados ou outros, têm interface com a aplicação por meio de serviços, que gerenciam o recurso de uma determinada forma (fila, semáforo,...). Se existem duas formas diferentes de gerenciar um recurso, como uma UART, deve haver dois componentes de serviços diferentes, um para cada tipo de gerenciamento. O projetista que deve decidir qual serviço deve ser instanciado para gerenciar a UART da forma mais apropriada às necessidades da aplicação.

2.9 Application

Esta camada designa os componentes que implementam as funcionalidades do sistema embarcado. Os módulos da aplicação dependem das interfaces fornecidas (serviços instanciados) pela camada Service, e.g. escalonamento de tarefas, mas são totalmente independentes da arquitetura de hardware da plataforma. Para rodar um componente da aplicação em outra plataforma basta que esta outra plataforma possua os mesmos serviços que este componente utiliza.

3. Exemplo

Esta seção apresenta um exemplo de utilização da arquitetura proposta para o projeto de um painel de LEDs. A plataforma de hardware é a placa MCB2300 da Keil [12] (microcontrolador LPC2368 da NXP) com uma memória flash serial da Atmel, família AT45 [16]. O RTOS aplicado foi o FreeRTOS [17]. O projeto de software possui 73 arquivos contendo 6553 linhas úteis de código.

Primeiro, nós mapeamos os componentes de hardware da placa em componentes de software. Como visto, os recursos da placa utilizados são o microcontrolador LPC2368, uma flash e um LED. Para o primeiro, um componente da camada Target foi instanciado. Para os outros, dois componentes da camada BHAL são instanciados. Os módulos BHAL proveêm abstração de hardware, mas eles precisam acessar funcionalidades específicas dos respectivos componentes eletrônicos. Portanto, os módulos *bhal_led* e *bhal_flash_memory* são implementados, respectivamente, sobre os módulos *badl_led* e *badl_atmel_45_flash* (implementação BADL para a memória AT45). Os componentes da placa são conectados ao microcontrolador através de pinos que são controlados pelos periféricos. Cada módulo BADL depende dos componentes PHAL relacionadas a cada um dos periféricos utilizados. O módulo *badl_led* depende do módulo *phal_io_port*, enquanto o módulo *badl_flash_memory* depende de *phal_io_port*, bem como de *phal_spi*. O componente *phal_spi* acessa *padl_spi* (SPI), *padl_pincon* (Pin Connect Block), *padl_sysctrl* (System Control Block) e *padl_vic* (Vectored Interrupt Controller), todos periféricos do LPC2368. O componente *phal_io_port* também depende de *padl_pincon* e *padl_sysctrl*, além de usar o módulo *padl_gpio* (General Purpose Input/Output).

O *port* do FreeRTOS, além de outros detalhes, utiliza o timer. Portanto, o componente *kernel_port_FreeRTOS* torna-se dependente de *phal_timer*. Este, por sua vez, faz uso das interfaces implementadas pelos componentes *padl_timer*, *padl_sysctrl* e *padl_vic*. O componente *kernel_FreeRTOS*, a essência do kernel, usou a implementação original do núcleo do FreeRTOS. Após instanciar os componentes do kernel nós criamos os serviços necessários à aplicação. O primeiro serviço, *service_led* fornece um suporte simples para acessar o LED. O acesso à memória flash é garantida por *service_lightdot_storage*. O componente *service_frt_panel* fornece acesso ao painel de LEDs. O serviço *service_scheduling* dá suporte a escalonamento baseado em prioridade para tarefas periódicas, usando a infra-estrutura de *kernel_FreeRTOS*. O serviço de tempo (*service_time*) é essencial para tarefas que exigem a função de *delay*. Daí então criou-se a aplicação. O módulo *LedTest* é responsável por fazer o LED piscar. O componente *DrawPanel* destina-se a exibir rolando uma mensagem lida da unidade de armazenamento.

4. Conclusão

Atualmente há uma deficiência de modelos de arquitetura de software embarcado apropriados para a composição de plataformas de software por meio da conexão de componentes. Este artigo apresentou uma arquitetura para o desenvolvimento de software embarcado visando maximizar a reusabilidade e portabilidade dos componentes de software. Esta arquitetura proposta foi exemplificada por um projeto de um painel de LEDs, que demonstra a implementação de componentes para cada camada.

Pretendemos evoluir este trabalho a fim de elaborar uma padronização para o projeto de software embarcado que permita a criação de repositórios de componentes, como acontece com plataformas de software desktop (Java, C #,...). Dessa forma, a construção de uma plataforma exige apenas instanciar os componentes de cada camada do repositório correspondente.

5. Referências

[1] Hazinger, T.A. and J. Sifakis. (2006) "The Embedded Systems Design Challenge", In: *Proc. of the 14 International Symposium on Formal Methods*.

[2] The MathWorks, "Simulink - Simulation and Model-Based Design", www.mathworks.com/products/simulink/, 1994.

[3] Ha, S., Kim, S., Lee, C., Yi. Y., Kwon, S. and Joo, Y.-P. (2007) "PeaCE: A hardware-software codesign environment for multimedia embedded systems", In: *ACM Transactions on Design Automation of Electronic Systems*, Vol. 12, No. 3, pp. 1-25.

[4] Marcondes, H., Junior, A., Wanner, L., Cancian, R., Santos, D., and Fröhlich, A. (2006). "EPOS: Um Sistema Operacional Portável para Sistemas Profundamente Embarcados". *Workshop de Sistemas Operacionais*.

[5] Cavalcante, S. (1997) "A Hardware-Software Codesign System for Embedded Real-Time Applications", PhD Thesis, Department of Electrical and Electronic Engineering, University of Newcastle.

[6] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, B. Tabbara. (1997) "Hardware-software co-design of embedded systems: the POLIS approach". Norwell, MA: *Kluwer Academic Publishers*.

[7] AUTOSAR GbR. (2006) AUTOSAR – Technical Overview V2.0.1.

[8] B. P. Douglass, (2002), "Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems", *Addison-Wesley.*

[9] A. J. Massa, (2002), "Embedded Software Development with eCos"*, Prentice Hall*.

[10] A. Tanenbaum, (1992), "Modern Operating Systems", *Prentice Hall*, Englewood Cliffs, NJ.

[11] "Windows NT Hardware Abstraction Layer (HAL)", Microsoft, http://support.microsoft.com/kb/99588, (2006).

[12] Keil, "MCB2300 Evaluation Board", www.keil.com/mcb2300/, abril/2010.

[13] D. R. Butenhof, (1997), "Programming with POSIX threads"*, Addison-Wesley*.

[14] µITRON4.0 Specification, (1999), ITRON Committee, TRON Association, Japan.

[15] J. M. Myerson, (2002), "The Complete Book of Middleware", *Auerbach Publications*.

[16] Atmel, "Atmel DataFlash", www.atmel.com/products/DataFlash/, abril/2010.

[17] Real Time Engineers Ltd., "FreeRTOS", www.freertos.org, acessado em abril/2010.

Communication Middleware For Hospital Automation: Send Alerts and Monitoring of Vital Signs

Cicília R. M. Leite[12], Bruno G. De Araújo [1], Ricardo A. M. Valentim[1], Gláucio B. Brandão[1] and Ana M. G. Guerreiro[1]

[1]Department of Computer and Automation Engineering, Federal University of Rio Grande do Norte – 59075-750 - Natal – RN - Brazil

[2] State University of Rio Grande do Norte and College of Science and Technology Mater Christi - Departament of Computing – State University of Rio Grande do Norte (UERN) 59.625-620 – Mossoró – RN – Brazil

{ciciliamaia,brunogomes, valetim, glaucio,anamaria}@dca.ufrn.br

Abstract. *The development of middleware has emerged as an area of research. It is expanding and is focused the integration of services available for distributed applications. In this context, many challenges have also arisen with the use of middleware, such as communication, flexibility, performance, integration with the Web and the computer itself. The development of middleware for mobile computing incorporates new challenges to developers because of the limitations of the mobile devices. Thus, they have to understanding the new technologies of mobile computing and middleware, in order to integrate them. The objective of this work is to develop a context-aware middleware and service-oriented for data management in real-time using mobile devices.*

1. Introduction

The remarkable advances in wireless communication and distributed systems, and the popularization of mobile devices have turned mobile computing into a reality. Mobile computing is exemplified by devices having some form of processing, such as laptops, PDAs, cell phones, and smart sensors, among others. These devices are related if they are connected through a wired network.

Thus, one can see that the link between the ubiquitous devices favors, which can be understood as the ability to participate in distributed computing regardless of where the device is located. The differences in network connectivity, capacity and platform resource availability can significantly affect application performance. Traditional middleware systems such as CORBA and DCOM have been successful in dealing with heterogeneity in hardware and software platforms that enable portability, thereby facilitating the development of distributed applications.

In this context, we offer appropriate support to address the dynamic aspects of mobile systems. Modern distributed applications require middleware that is able to adapt to environmental changes and is compatible with the level of service quality needed. Using Java technology we developed middleware that will provide data monitoring services, such as real time database access, and send alerts to mobile devices. Thus, the methodology consists of two phases: theoretical and experimental. The theoretical phase consists of reading scientific articles to understand the system, whereas the experimental phase involves system development.

This article is organized as follows: In Section II, we describe the related work. In Section III, we present an overview of the proposed middleware. In Section IV, we present middleware development, its design concept, its implementation, a case study to validate the specific middleware, as well as experiments and results. And finally, in Section V, concluding remarks and a discussion are given regarding future research.

2. Related Work

Continuous data stream processing, emerging as a new research field, concerns the processing of information from sources that produce data at a fast and continuous rate.

For example, information from sensory devices can be considered as a continuously expanding and unlimited sequence of data items without any boundaries. Traditionally, such information required special monitoring applications and equipment that processed and reacted to continual inputs from several sources, such as in a weather monitoring station, patient monitoring equipment, etc.

Advances in electronics have contributed to an increased demand in distributed applications that allow the use of devices with embedded processing power [4]. Examples are industrial networks that use intelligent node controlling processes. According to Pedreiras [12], this occurs because of the trend towards decentralized computing, currently converging to a distributed environment. The functionalities of many processing elements will be processed and compared to centralized computing, which encapsulates functionalities in a single processor with more processing power.

Several studies have been conducted to address various problems faced by industrial automation systems that can potentially be adapted and used in hospital automation. In 1976 Nitzan and Rosen [11] foresaw that industrial automation concepts could be automated using programmable systems such as data acquisition for process control; signal monitoring and processing, providing cost reduction and process optimization. These concepts have been incorporated into the medical environment, making them feasible for use in hospital automation [2].

3. Overview of Middleware

This article presents the development of service oriented context-aware middleware for real-time data management using mobile devices. Middleware tracks data in real time, providing the following services: real-time access to databases from the mobile device; constant database updating from continuous reading data acquisition devices; automatic integration of the various Database Management Systems (DBMS), regardless of their platforms; and the sending of alerts to mobile devices.

After specification and analysis, a prototype will be developed and integrated with a hospital system to detect risk-patients in order to trigger the timely intervention of a Medical Emergency Team (MET).

Hospital systems, highly dependent on information management, must reduce costs and increase efficiency and agility, in addition to ensuring the readiness and effectiveness of the procedures performed. Thus, this project will be of great importance for public health, contributing directly to enhancing services and medical diagnoses, thereby improving the quality of products and services provided to the population.

3.1 System Framework

Capable robust systems are needed to effectively control large volumes of information. Accordingly, middleware was designed with functionality for monitoring information through real-time database queries, in addition to being able to send alerts in critical situations defined by the user. The middleware is based on five stages, as shown in Figure 1.

The first stage is acquisition, responsible for obtaining the data and sending them to a local server, which will in turn relay them to a central server; the second stage involves pre-processing the recovered data, which will be processed by a Digital Signal Processing (DSP) technique such as wavelet transform. In the classification stage, the pre-processed data will be transformed to classify them according to the problem; post-processing is where the data is formatted to be compatible with the system device and are prepared and sent to the registered user of this device. In this stage the middleware is capable of monitoring these data and sending alerts to pre-defined devices.

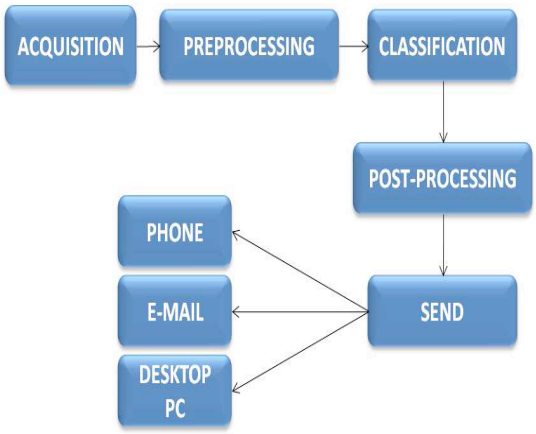


Figure 1. System Framework

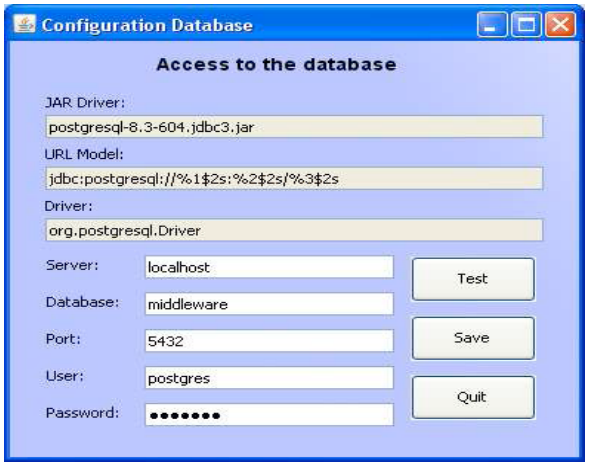


Figure 2. Automatic Configuration Service of the Database

4. Middleware Development

The system offers access to different databases, and the idea is to automate the processes, providing automatic database configuration. The user has only to transmit database information (server, base, port, driver and url model), to be connected automatically, as shown in Figure 2.

Finally, there is a service that sends alerts when critical information is detected. It is able to send Short Message Service (SMS) and alert messages to mobile phones and e-mail addresses, with a predefined message from of a registered user.

To send an SMS, the modem (port, speed, model and brand) must be configured, as shown in Figure 3. To send one by e-mail, server configuration (SMTP, e-mail, user and password) must be performed, as illustrated in Figure 4.

The middleware was developed in Java using the 1.6.0-14 JDK standard library and Eclipse 3.4 environment (Eclipse, 2009). A number of API's and frameworks were also used. The real-time database access was developed with temporal characteristics of the real-time language (LC-RTDB), specified in Leite, 2009 [8].

An automatic database configuration uses JPA, as shown in Figure 4. Configuration of the E-mail technology (Java Persistence API) and sends alerts by SMSLib Framework, which requires the computer to be connected to a GSM modem to send the SMS. An internet connection is the only requirement to send an e-mail.

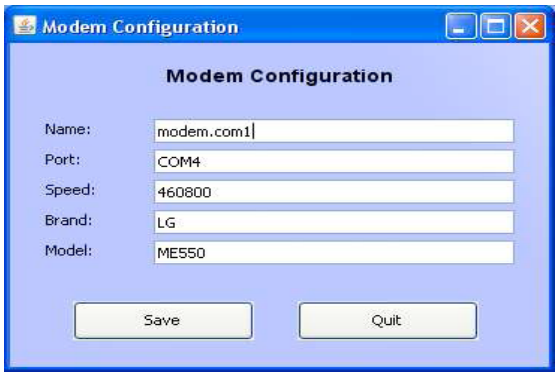


Figure 3. Configuration of the Modem

Figure 4. Configuration of the E-mail

The same middleware specification was used for the development application. The only difference is that the application contains no implemented business logic; it accesses the middleware methods by using its own services.

The middleware developed was applied to the Intensive Care Unit (ICU), where patients require continuous monitoring of vital signs. The application aimed at early detection of risk situations, allowing health professional intervention. This middleware emphasizes the fast transmission to the medical staff of information regarding a patient's vital signs, because it allows prediagnosis when a possible risk is detected. To achieve efficient control, a secure and robust application is required, as illustrated in Figure 5. The middleware developed can be used for this purpose because of the services it offers. The first stage involves registering information on physicians and all the possible ways of sending them alerts, such as email, phone number, etc. This represents the test of middleware services through application.

After registration, an alert will be sent to the doctor by e-mail or SMS if any patient condition falls outside normal limits. A periodic query is set if the patient goes to the ICU, as illustrated in Figure 6. The monitoring screen is best viewed in Figure 7, where the patients conditions are displayed.

$$T_G = \frac{RTT}{2} \qquad (1)$$

In conclusion, the main idea of the middleware tests was to satisfactorily consolidate the functions developed, and validate the services offered through a case study. The middleware can be used in different situations, since domestic monitoring for critical applications shows promising potential.

To measure the performance and efficiency of the experiment were tested by sending several alert messages between a server and a mobile device and vice versa. We used a computer with the following configuration Intel Core 2 Duo 2.4 Ghz, 4GB RAM, Network Card Wireless Intel. The device was a mobile phone with wifi access. Two tests were conducted, each on a different network.

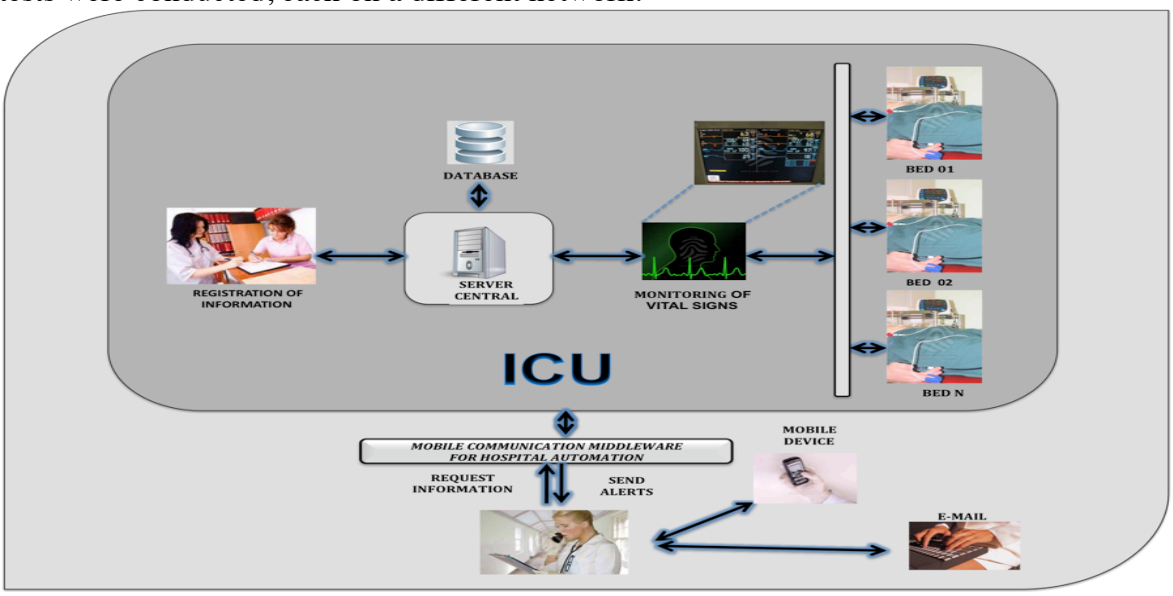


Figure 5. Application the Mobile Communication Middleware for Hospital Automation

To get an estimate of the total broadcasting time occurred during the message transmission system, we use the RTT (Round Trip Time), which corresponds to the time spent and return a message through the network. Thus, using equation (1) can perform this calculation.

Thus, the time spent (Tg) of sending only corresponds to half the RTT. In the test, several items were made and receipts of alert messages to take an average time spent. With the outcome of this experiment, the order to have a idea of the feasibility of using this system in a network environment. The first test was conducted in a residential local area network using a router, 54 Mbps, with the server connected to the router via cable. The second test was performed using a 150 Mbps network with several computers and traffic, in which the server is connected via wireless network. In both tests the mobile device to connect through the wireless network. The result is shown in Table 1

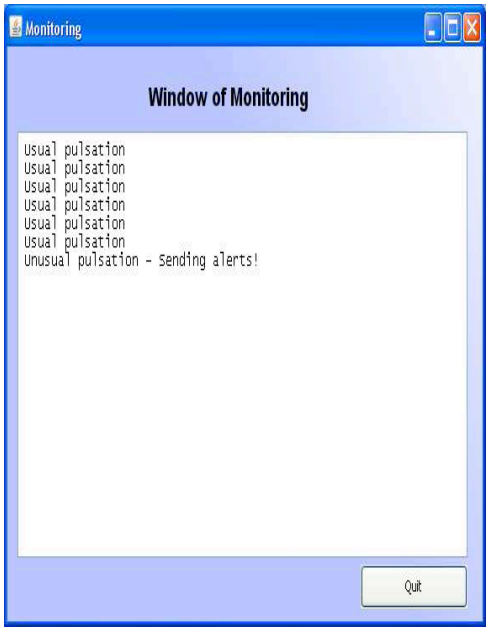


Figure 6. Window of Monitoring

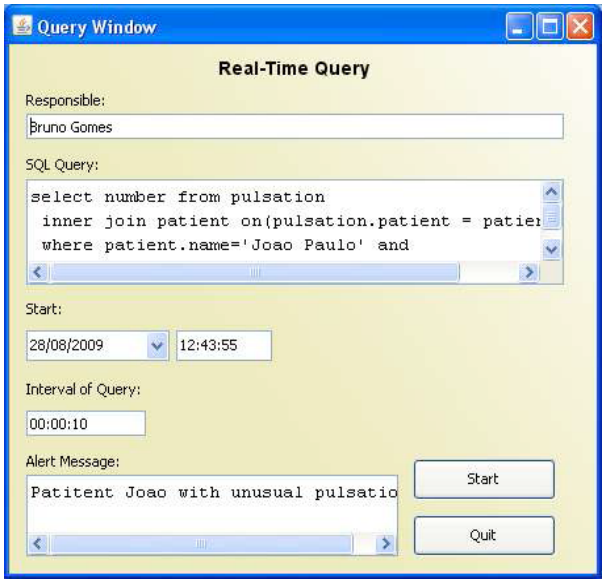


Figure 7. Real-Time Periodic Query

Table 1. Analysis of the peformance - Tests

Network	TEST 01 **(Milisegundos)**	TEST 02 **(Milisegundos)**
1	2	16
2	4	14

5. Conclusions

In this article, we sought to put forth the main concepts of middleware and mobile computing and the mobile computing challenges facing middleware. We developed, a service-oriented context-aware middleware for real-time data management using mobile devices. A case study monitored medical diagnoses by sending the results presented to validate the middleware, services and functions developed. The middleware was used in a hospital setting.

This work, once implemented, will help meet the demand for innovative software and hardware required in automation systems, more specifically in hospital automation. The results of this research project may contribute significantly to information and communication systems in hospitals. Hospital systems are too costly for public institutions to acquire, so our alternative was to develop an open system that enables hospital procedure management. This project will have an important impact on scientific and hospital automation, because it allows the development of an intelligent

system that will help not only doctors but also the entire health team to monitor and diagnose their patients.

Acknowledgment

This work relied on the support of the Laboratory of Hospital Automation and Bioengineering (LAHB) of the Department of Automation and Computer Engineering of the Federal University of Rio Grande of the Norte.

References

[1]Brito, A. E. M., Brasileiro, F. V., Leite C. E., Buriti, A. C. Ethernet Communication in Real-Time to a Netowork of Microcontrollers, Annals of XV Brazilian Conference on Automatica (CBA 2004)-Brazil, September 2004.

[2]Brooks, J.; Brooks, L. Automation in the medical field. Engineering in Medicine and Biology Magazine, IEEE Volume 17, Issue 4, July-Aug. 1998 Page(s):76, 81.

[3]Carreiro, F., Moraes, R., Fonseca, J. A e Vasques, F. Real-Time Communication in Unconstrained Shared Ethernet Networks: The Virtual Token-Passing Approach, submitted at Emerging Technologies and Factory Automation - ETFA, Catania, Italy, 2005.

[4]Dietrich, D., Sauter, T. Evolution Potentials for Fieldbus Systems. WFCS 2000, IEEE Workshop on Factory Communication Systems. Porto, Portugal, September 2000.

[5]Dolejs, O., Smolik, P., e Hanzalek Z. On the Ethernet use for real-time publish-subscribe based applications. In 5th IEEE International Workshop on Factory Communication Systems, Vienna, Austria, Sep. 2004.

[6]IEEE 802.3/ISO 8802-3 - Information processing systems - Local area networks - Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, 2nd edition, 21 September 1990.

[7]Gullikson, M. L. Risk Factors, Safety, and Management of Medical Equipmet. IEEE The Biomedical Engineering Handbook. ISBN 0-8493- 8346-3. USA, 1995.

[8]Leite, C. R. M. Linguagem de Consulta para Aplicacõoes em Tempo-Real [dissertacão], Campina Grande: Universidade Federal de Campina Grande, 2005.

[9]Moraes, R.; Vasques, F., Real-time traffic separation in shared Ethernet networks: simulation analysis of the h-BEB collision resolution algorithm, Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on, vol., no.pp. 89- 92, 17-19 Aug. 2005.

[10] Murakami, Alexandre; Gutierrez,M. A.; Lage, Silvia Helena Gelas; Rebelo, Marina de Fátima de Sá; Ramires, José Antonio Franchini. A Continuous Glucose Monitoring System in Critical. IEEE Computers in Cardiology, v. 32, p. 10-14, 2006.

[11] Nitzan, D.; Rosen, C.A. Programmable Industrial Automation. Transactions on Computers. Volume C-25, Issue 12, Dec. 1976 Page(s):1259 - 1270.

[12] Pedreiras, P., Almeida, L., Gai, P. and Giorgio, B. FTTEthernet: A Flexible Real-Time communication Protocol That Supports Dynamic QoS Management on Ethernet-Based Systems. IEEE Transactions on Industrial Informatics, Vol. 1, No. 3, August 2005.

[13] RFC 4541. Christensen, M., Thrane &Thran, Kimball, K. IGMP and MLD Snooping Switches Considerations. Hewlett-Packard, May, 2006.

[14] Thomesse, J.-P. Fieldbus and interoperability Contr. Eng. Pract., vol. 7, no. 1, pp. 81-94, 1999.

[15] Valentim, R. A. M.; Morais, A. H. F.; Brandao, G. B.; Guerreiro, A. M. G. A performance analysis of the Ethernet nets for applications in real-time: IEEE 802.3 and 802.3 1 Q. Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on 13-16 July 2008 Page(s):956 - 961. 2008.

[16] Várady, P., Benyo, Z. and Benyo, B. An open architecture patient monitoring system using standard technologies. IEEE Transactions on Information Technologies in Biomedicine, Vol. 6, No. 1, pp.95?98, 2002.

[17] Varshney, U. Patient monitoring using infrastructure-oriented wireless LANs. International Journal of Electronic Healthcare, Volume 2, Number 2 / 2006, 149-163, 2006.

[18] Viegas, R. Valentim, R. A. M. Texeira, D. G. Guedes, L. A. Performance Measurements of Protocols to Ethernet Real-Time Applications Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on 20-22 Sept. 2006 Page(s):1352 - 1355. 2006.

Projeto de Redes Intraveiculares Híbridas

Rodrigo Lange[1], **Rômulo de Oliveira**[1], **Nestor Roqueiro**[1]

[1]Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

Abstract. *O FlexRay é um protocolo de comunicação que tem sido fortemente promovido como o futuro padrão de fato para sistemas automotivos de tempo real. Entretanto, a maioria dos dispositivos como sensores e atuadores para fins automotivos atualmente disponíveis utilizam outros protocolos como Controller Area Network (CAN) e Local Interconnect Network (LIN). Caso exista troca de informações entre dispositivos localizados em segmentos de redes que utilizam protocolos diferentes, será necessário um gateway entre estes segmentos. Este artigo descreve os primeiros resultados e observações de um trabalho onde são estudados aspectos relativos ao projeto de redes veiculares que utilizam gateways para trocas de dados.*

1. Introdução

Desde a década de 1970 tem sido observado um crescimento exponencial no número de dispositivos eletrônicos incorporados em veículos automotores, sendo que no ano de 2004 um veículo *high-end* chegava a utilizar mais de 60 microprocessadores que trocavam 2.500 tipos de mensagens de dados. Uma das razões para este crescimento tecnológico é o surgimento de novos *hardwares* e *softwares* que facilitaram a introdução de funções cujo desenvolvimento seria oneroso ou mesmo impossível utilizando apenas componentes mecânicos e hidráulicos. Neste ambiente, o *Controller Area Network* (CAN) tem sido o padrão de fato com mais de 400 milhões de nodos vendidos a cada ano. Mas segundo alguns autores, o CAN não é adequado para uso nos futuros sistemas X-By-Wire que possuem características *hard real-time* por natureza e exigem comunicação de alta velocidade determinística e robusta. Uma das iniciativas para atender às necessidades dessas aplicações foi o desenvolvimento do protocolo FlexRay, frequentemente citado como sendo o protocolo que vai se tornar predominante para as comunicações em sistemas automotivos [Navet and Simonot-Lion 2008].

No entanto, atualmente a disponibilidade comercial de sensores e atuadores com interface FlexRay é reduzida ou mesmo inexistente. Pode ser citado como um exemplo dessa deficiência um catálogo da Bosch Motorsport [Bosch 2010], que oferece diversos dispositivos com interfaces CAN mas nenhum com FlexRay. Um projeto em que o FlexRay seja definido como principal protocolo e no qual se pretenda utilizar sensores e atuadores comerciais possivelmente necessitará de *gateways* para gerenciar a troca de dados entre dispositivos interligados a segmentos de rede que utilizam protocolos diferentes. Para um bom projeto de uma rede veicular, deve-se levar em consideração não apenas o impacto destes *gateways*, mas também as características dos próprios protocolos, como sua velocidade e o formato dos quadros.

Este artigo apresenta os primeiros resultados de um projeto cujo principal objetivo é propor técnicas para o projeto de redes veiculares onde *gateways* gerenciam a troca

de dados entre dispositivos localizados em segmentos de rede que utilizam os protocolos FlexRay e CAN. O trabalho foi motivado pelas necessidades de um projeto maior no qual está sendo desenvolvido um veículo conceitual chamado Flue: um triciclo inclinável para uso urbano com capacidade para dois passageiros (Figura 1). O objetivo é desenvolver um veículo leve e que utilize sistemas *X-by-Wire* para melhorar as características de consumo e baixa emissão de poluentes [de Souza Vieira et al. 2009]. Sendo que a literatura indica o FlexRay para uso em sistemas *X-by-Wire* [Navet and Simonot-Lion 2008], este foi escolhido como sendo o principal protocolo de comunicação do veículo. Mas devido a dificuldade de obtenção de sensores comerciais com interface FlexRay optou-se por utilizar dispositivos CAN disponíveis no mercado. As leituras são enviadas para as *Electronic Control Units* (ECU's) localizadas no segmento FlexRay através de *gateways*, evitando assim o esforço necessário para o desenvolvimento e validação de sensores específicos.

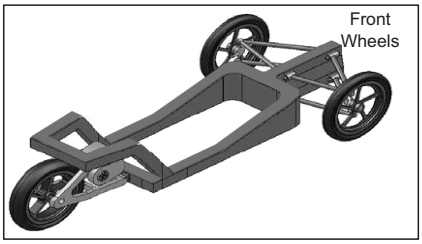


Figura 1: Ilustração do veículo conceitual [de Souza Vieira et al. 2009]

2. FlexRay

O FlexRay é um protocolo desenvolvido por um consórcio de diversos fabricantes com o objetivo de suceder ao CAN em aplicações que exigem determinismo, sincronização, largura de banda e confiabilidade, tais como sistemas X-by-Wire.

O controle de acesso ao meio (*Media Access Control*, MAC) do FlexRay é baseado em um ciclo de comunicação periódico com duração predefinida T_{bus}. Um ciclo do FlexRay contém um Segmento Estático (*Static Segment*, ST), um Segmento Dinâmico (*Dynamic Segment*, DYN) e dois segmentos de protocolo *Network Idle Time* (NIT) e *Symbol Window* (SW). Tanto o ST como o DYN são compostos por vários *slots* de tempo, controlados por contadores exclusivos. Durante a fase de projeto é atribuído um identificador $FrameID$ para cada mensagem que pode ser transmitida em um *cluster*. Em um dado *slot* ST ou DYN apenas um nodo é autorizado a enviar dados para o barramento, e esse é o nodo que contém a mensagem com o identificador igual ao valor atual do contador de *slots*. A Figura 2 ilustra um ciclo de comunicação FlexRay.

Um ciclo de comunicação FlexRay sempre contém um ST. Dentro do ST a arbitragem das mensagens é feita através do Time Division Multiple Access (TDMA). O ST contém um número configurável de *slots* estáticos, sendo que todos têm uma duração idêntica. Se uma mensagem não estiver pronta no início de seu *slot*, este permanece vazio e nenhuma outra mensagem pode ser enviada [FlexRay 2005].

Assim como no ST, o DYN tem tamanho fixo e é dividido em *slots*, sendo os identificadores dos *slots* atribuídos aos nodos da rede. No entanto, a coordenação de transmissão utiliza um método mais flexível, chamado por alguns autores de *Flexible TDMA* (FTDMA). Neste método, o DYN é dividido em *minislots* (MS) que possuem tamanho

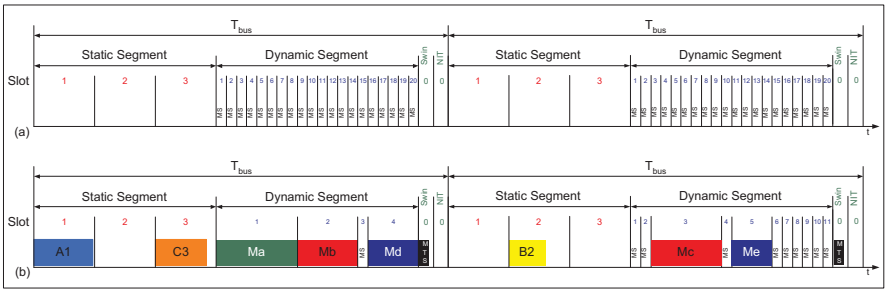


Figura 2: Ciclo de Comunicação FlexRay [Pop et al. 2008]

igual. Assim como no ST, é atribuído um identificador para cada mensagem, e este identificador reflete a prioridade da mensagem. O comprimento de uma mensagem dinâmica pode ser arbitrário mas esta deve caber dentro do DYN. No início de um segmento dinâmico, é permitida a transmissão da mensagem de maior prioridade. Se a mensagem não estiver pronta no início do seu *slot* um quadro vazio com o tamanho de um MS é transmitido. Após a transmissão o acesso é dado para a próxima mensagem, que é enviada se estiver pronta e se couber na parte restante do segmento DYN. Caso contrário, é enviado um quadro vazio com tamanho de um MS. Este processo é repetido para todas as mensagens, ou até o fim do segmento DYN.

Na Figura 2.b o comportamento do FlexRay é exemplificado. As mensagens $A1$, $B2$ e $C3$ são alocadas no ST, e Ma, Mb, Mc, Md, Me são alocadas no DYN. Em primeiro lugar, considere-se o comportamento do ST. No primeiro ciclo $A1$ e $C3$ são enviadas. No segundo ciclo, $B2$ (que por algum motivo não estava pronta no primeiro ciclo) é enviada. Considerando-se o DYN, no primeiro ciclo são enviadas Ma e Mb. Para Mc não existe o número necessário de MS, de forma que esta é postergada para o ciclo seguinte e apenas um MS é utilizado. A mensagem Md é enviada, não restando mais MS no ciclo. No segundo ciclo, dois MS são utilizados por não existirem mensagens para Ma e Mb. Mc é enviada, outro MS é transmitido devido a Md e, por fim, Me é enviada.

Na literatura o ST é abordado entre outros em [Schmidt and Schmidt 2008b] e [Grenier et al. 2008]. Aspectos relativos ao DYN são discutido em [Pop et al. 2008], [Schmidt and Schmidt 2008a] e [Chokshi and Bhaduri 2010].

3. CAN

O *Controller Area Network* (CAN) é um barramento de rede projetado para proporcionar uma comunicação simples, robusta e eficiente e é atualmente o padrão de fato para aplicações automotivas. O CAN está presente nos diversos domínios de um veículo, com velocidades variando de 125kbit/s até 1 Mbit/s. Cada mensagem possui um identificador numérico cujo valor determina sua prioridade. Um quadro CAN padrão pode conter até 8 bytes de dados. Um quadro CAN padrão pode conter até 8 bytes de dados de carga útil.

Falando de forma livre, projetar uma rede CAN consiste em atribuir identificadores para as mensagens e verificar se o sistema é escalonável. Um método comum de verificar a escalonabilidade do sistema é calcular o tempo de resposta (*worst-case response time*, wcrt) de cada mensagem e então comparar o wcrt com seu deadline. Um método bem conhecido para o cálculo do wcrt de mensagens CAN é apresentado em [Davis et al. 2007]. Já a atribuição dos identificadores pode ser feita, por exemplo, com a

implementação Ada `Assign_Pri` apresentada em [Burns and Wellings 2001].

4. Gateway

Dos trabalhos existentes na literatura que são relacionados a *gateways* para redes veiculares com FlexRay, pode-se citar [Da 2007] onde é abordado o projeto de *gateways* seguindo o padrão AUTOSAR, [Seo et al. 2008a] e [Seo et al. 2008b], onde são reportadas implementações de *gateways* em placas de desenvolvimento. Entretanto, nenhum dos trabalhos citados aborda questões como análise do tempo de resposta ou de escalonamento.

5. Questões Relacionadas ao Projeto de Redes Veiculares com *Gateways*

Devido às características dos componentes presentes em uma rede veicular, existem vários pontos e problemas que devem ser considerados no projeto de um sistema que utilize *gateways*. Existem aspectos relacionados especificamente aos *gateways*, como por exemplo os algoritmos dos softwares que precisam considerar as diferenças de quadros e velocidades dos protocolos, ou questões relativas ao uso de padrões como o AUTOSAR [Da 2007]. Outros aspectos dizem respeito especificamente aos sensores e atuadores. Este trabalho não aborda questões relacionadas especificamente aos *gateways* ou dispositivos de forma isolada, mas sim questões relacionadas ao projeto da rede como um todo. Apesar da literatura apresentar trabalhos que abordam todos os pontos que serão levantados aqui, pelo nosso conhecimento não existem trabalhos que apresentem uma solução completa para o problema.

Para um melhor entendimento dos aspectos que serão descritos abaixo, será utilizada como exemplo uma rede veicular composta por um segmento FlexRay de 1Mbps que também atua como *backbone*, um segmento CAN-B com velocidade de 50kbp/s e um segmento CAN-C de 500kbp/s, todos interligados por *gateways* (Figura 3).

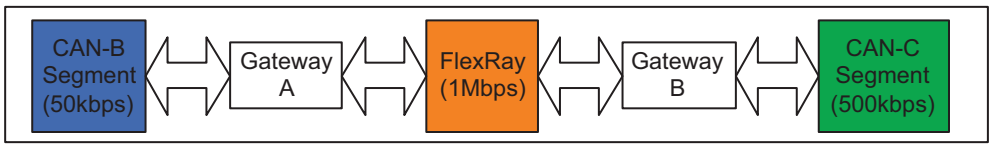


Figura 3: Exemplo de rede com três segmentos

Pode-se dizer que o projeto de uma rede é uma busca de definições para diversos parâmetros, sendo que as escolhas para um dos parâmetros podem influenciar diretamente as escolhas que poderão ser feitas para outros parâmetros. No exemplo, considere-se uma mensagem m gerada por um dispositivo conectado ao segmento CAN-C. A mensagem m também é utilizada por um dispositivo conectado ao segmento CAN-B. Tanto o tamanho do ciclo FlexRay como o modo de transmissão dos dados m no FlexRay (no ST ou no DYN) tem impacto direto na escolha da prioridade dos quadros que conterão m nos barramentos CAN, visto que estas prioridades devem ser altas o suficiente para evitar que m perca seu deadline.

Uma opção para a definição de todos os parâmetros da rede seria através de Programação Linear Inteira onde, obedecendo a restrições fornecidas e informações sobre as mensagens, a resolução de uma formulação desse como resultado definições ótimas

para um sistema completo. No entanto este tipo de técnica é computacionalmente cara, podendo ser inviável para sistemas grandes. Além disso, um sistema resultante pode ser válido apenas para o conjunto inicial de mensagens, sendo necessária a obtenção de um novo conjunto completo de definições mesmo para pequenas modificações. Por estes motivos esta abordagem não será explorada.

Neste trabalho optou-se por procurar uma abordagem com custo computacional baixo que permita a criação de sistemas mais flexíveis (que não necessitem a alteração do sistema como um todo no caso de mudanças). Para que este objetivo seja atingido, uma das etapas foi o levantamento de questões que podem influenciar nas definições para os diversos parâmetros de uma rede. Até o presente momento, as questões identificadas foram: **a)** Como considerar a interferência dos *gateways* no cálculo do wcrt de uma mensagem? **b)** É possível a formação de filas de mensagens nos *gateways* devido ao fato dos segmentos possuírem velocidades diferentes? Se sim, como isso pode ser evitado? **c)** Como definir os tamanhos para o ciclo e os segmentos do FlexRay, se considerada a rede como um todo? **d)** Em qual segmento do FlexRay alocar uma mensagem que é originada em um dos barramentos CAN e que deve ser enviada através de um *gateway*? Quais os fatores devem ser levados em conta na decisão sobre a alocação? **e)** Para as mensagens que são transmitidas de um barramento CAN para outro, sem serem aproveitadas por dispositivos no barramento FlexRay, vale a pena fazer algum tipo de agrupamento de mensagens? Qual o impacto deste agrupamento?

No atual estágio deste projeto estão sendo feitos estudos sobre o uso de técnicas de particionamento de deadlines como ferramenta para a definição dos parâmetros de uma rede veicular. Para isso, as técnicas apresentadas em [Kao and Garcia-Molina 1997] tem sido modificadas de forma a abranger os pontos apresentados no parágrafo anterior. Entretanto, os estudos tem esbarrado em um problema relacionado à literatura disponível sobre o FlexRay: até o presente momento, todos os estudos que abordam o cálculo do wcrt de uma mensagem do DYN tem sido questionados por apresentarem abordagens que resultam em valores ou otimistas ou muito pessimistas [Chokshi and Bhaduri 2010].

6. Resultados Experimentais

Visando ilustrar alguns dos resultados já obtidos neste trabalho, apresentamos um pequeno exemplo que utiliza alguns dos sensores e atuadores do projeto Flue.

Msg	Descrição	C	T	D	Bus	R
m1	Speed Sensor	8	10.0	10.0	CAN $\Rightarrow$ FR	0.0570 ms
m2	Yaw-rate Sensor	3	10.0	10.0	CAN $\Rightarrow$ FR	0.0544 ms
m3	Incln and Acc	8	10.0	10.0	CAN $\Rightarrow$ FR	0.0570 ms
m4	DataLog 1	8	20.0	20.0	CAN $\Rightarrow$ FR	0.0570 ms
m5	DataLog 2	6	20.0	20.0	CAN $\Rightarrow$ FR	0.0560 ms
m6	Left Tilt Ac.	2	5.0	5.0	FR	0.0528 ms
m7	Right Tilt Ac.	2	5.0	5.0	FR	0.0528 ms
m8	Steering Pos.	2	10.0	10.0	FR	0.0528 ms

Tabela 1: Exemplo de Sistema (C em bytes, T, D e R em ms)

Neste exemplo, sensores conectados a um barramento CAN de 50kbp/s enviam suas leituras para dispositivos em um barramento FlexRay de 1Mbp/s. Para o cálculo do wcrt R de cada mensagem foi utilizada a estratégia UD [Kao and Garcia-Molina 1997]

em conjunto com as equações presentes nos trabalhos citadas neste documento. Foi considerado um wcrt R_{GW} do *gateway* como sendo 0,1ms. Para o FlexRay, foram considerados um T_{bus} de 5ms, um ST com 3ms e um DYN com 2ms. Todas as mensagens foram alocadas no ST. A Tabela 1 apresenta os resultados deste exemplo.

7. Agradecimentos

Ao CNPq, CAPES e PPGEAS/UFSC pelo suporte financeiro, e ao Dr. Rodrigo Vieira e demais membros do LI pela ajuda sobre aspectos relacionados ao triciclo.

Referências

Bosch (2010). Bosch Motorsport Catalog Edition 2010/1. Available at `http://www.bosch-motorsport.de/`.

Burns, A. and Wellings, A. (2001). *Real-time Systems and Programming Languages: ADA 95, Real-time Java, and Real-time POSIX*. Addison Wesley, third edition.

Chokshi, D. and Bhaduri, P. (2010). Performance Analysis of FlexRay-based Systems Using Real-Time Calculus, Revisited. In *Symposium On Applied Computing*.

Da, Z. W. (2007). Performance Analysis of AUTOSAR vehicle Network Gateway. Master's thesis, Ireland.

Davis, R., Burns, A., Bril, R., and Lukkien, J. (2007). Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Real-Time Systems*.

de Souza Vieira, R., Padilha, R., Nicolazzi, L., and Roqueiro, N. (2009). Five Degrees of Freedom Model for a Tilting Three-Wheeled Narrow Vehicle. *IEEE Transactions on Vehicular Technology*.

FlexRay (2005). FlexRay Communications System Protocol Specification Version 2.1. Available at `http://www.flexray.com/`.

Grenier, M., Havet, L., and Navet, N. (2008). Configuring the Communication on FlexRay - The Case of the Static Segment. *Proceedings of ERTS*.

Kao, B. and Garcia-Molina, H. (1997). Deadline Assignment in a Distributed Soft Real-Time System. *IEEE Trans. Parallel Distrib. Syst.*

Navet, N. and Simonot-Lion, F. (2008). *Automotive Embedded Systems Handbook*. CRC.

Pop, T., Pop, P., Eles, P., Peng, Z., and Andrei, A. (2008). Timing Analysis of the FlexRay Communication Protocol. *Real-Time Systems*.

Schmidt, E. and Schmidt, K. (2008a). Message Scheduling for the FlexRay Protocol: The Dynamic Segment. *IEEE Transactions on Vehicular Technology*.

Schmidt, K. and Schmidt, E. (2008b). Message Scheduling for the FlexRay Protocol: The Static Segment. *IEEE Transactions on Vehicular Technology*.

Seo, S., Kim, J., Moon, T., Hwang, S., Kwon, K., and Jeon, J. (2008a). A Reliable Gateway for In-vehicle Networks. *Proceedings of the 17th World Congress The International Federation of Automatic Control*.

Seo, S., Moon, T., Kim, J., KIM, S., Son, C., Jeon, J., and Hwang, S. (2008b). A Gateway System for an Automotive System: LIN, CAN, and FlexRay. *6th IEEE International Conference on Industrial Informatics INDIN*.

Método para Diminuir o Tempo de Interferência de Tarefas de Tempo Real

Ítalo Campos de M. Silva[1], **Rômulo Silva de Oliveira**[1], **Luciano Porto Barreto**[2]

[1]Departamento de Automação e Sistemas – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

[2]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
CEP 40170-110 – Salvador – BA – Brasil

{italo,romulo}@das.ufsc.br, lportoba@ufba.br

Abstract. *Real-time tasks run on Linux with PREEMPT-RT patch won best accuracy from the use of high resolution timers. These timers are processed through interrupts and interfering with all tasks that are running, whether real time or not. Some of these timers are processed in hard irq unnecessarily interfering with real time tasks for a longer time than necessary. This article proposes a method to solve this problem and reduce the time of preemption of these tasks.*

Resumo. *As tarefas de tempo real executadas no Linux com o patch PREEMPT-RT ganharam melhor precisão desde a utilização dos temporizadores de alta resolução. Estes temporizadores são processados através de interrupções, interferindo em todas as tarefas que estejam executando, sejam elas de tempo real ou não. Alguns destes temporizadores são processados em hard irq desnecessariamente, interferindo em tarefas de tempo real por um tempo maior do que necessário. Este artigo propõe um método para resolver este problema e diminuir o tempo de preempção destas tarefas.*

1. Introdução

O Linux vem se tornando cada vez mais popular, pois ele é um sistema operacional de propósito geral que fornece bom desempenho, estabilidade e baixo tempo médio de resposta. Ele está sempre em constante evolução devido ao grande conjunto de desenvolvedores que estudam seu código e tentam melhorá-lo. Devido a ser um sistema operacional de código aberto, o seu *kernel* pode ser estudado e alterado por qualquer pessoa, sendo ele muito utilizado no meio acadêmico por tais motivos.

O desenvolvimento do *kernel* padrão do Linux não tem como prioridade suportar aplicações de tempo real, mas mesmo assim ele já implementa o padrão POSIX.4 [POSIX.13 1998]. Mas para suportar de forma mais completa aplicações de tempo real, existem *patches* que alteram o código do *kernel* padrão. Um deles e o estudado para este artigo é o *PREEMPT-RT*, que tem como objetivo prover determinismo para tarefas de alta prioridade no *kernel* [Molnar 2005].

Este artigo irá propor alterações no *kernel* do Linux com *PREEMPT-RT*, na tentativa de diminuir o tempo de interferência causada sobre tarefas de tempo real em determinadas situações. Demonstrando através de medições e equações matemáticas o problema

existente, logo depois demonstra-se através de exemplos e equações o que deve ocorrer com as alterações propostas.

2. Hard irq e SoftIRQ

O Linux trabalha com o conceito de chamar a atenção do processador quando necessário através de interrupções, as quais são utilizadas para alterar o fluxo de execução normal do sistema [Mauerer 2008]. Quando a interrupção é gerada, o processador pára tudo o que está fazendo e executa um código responsável por tratá-la (tratador de interrupção) em contexto de interrupção. Após esta interrupção ter sido tratada, o processador volta a executar o que estava sendo executado.

O tratador de interrupções em geral é dividido em duas partes: *Top Half* (atividades executadas assim que ocorre a interrupção) e *Bottom Half* (atividades relacionadas à interrupção que podem ser postergadas para executarem em um momento mais oportuno) [Love 2005]. As atividades executadas em *Top Half* também podem ser ditas que estão sendo executadas em *hard irq*. Já a *softIRQ* é uma maneira de postergar trabalho dentro do kernel, ou seja, é um dos tipos de tarefas da *Bottom Half*.

3. Temporizadores de Alta Resolução

O Linux utiliza-se de temporizadores para realizar muitas tarefas, como atualizar a hora do sistema, detectar falhas de envio de pacotes pela rede, escalonar tarefas, entre outras. Algumas destas tarefas não têm necessidade de uma grande resolução temporal, mas outras podem oferecer um serviço bem melhor dependendo da resolução temporal que o kernel forneça a elas [Etsion et al. 2001].

No intuito de melhorar o desempenho de muitas tarefas, as quais necessitavam de melhor precisão temporal, o subsistema de tempo do kernel do Linux foi alterado, facilitando a manutenção e desenvolvimento de temporizadores, tendo também sido desenvolvido um tipo de temporizador com alta resolução (*High Resolution Timer*) [Gleixner and Niehaus 2006].

A diferença fundamental entre estes temporizadores e os já existentes (temporizadores clássicos) é que estes utilizam-se de interrupções de disparo único (*one-shot*). Desta forma um dispositivo de relógio é programado para gerar uma interrupção no momento exato em que o temporizador deve expirar, sendo este dispositivo reprogramado para o momento da próxima expiração sempre que for gerada a interrupção.

Estes temporizadores possuem um ponteiro para uma função, a qual deve executar sempre que eles expirem. Como eles são executados através de interrupções, eles impõem sua execução acima de qualquer tarefa que esteja sendo processada. Mas alguns temporizadores de alta resolução não precisam ser processados em *hard irq*, tendo seu processamento postergado, sendo executados através de *softIRQs*. Assim os temporizadores de alta resolução na sua maioria aumentam de forma considerável o tempo gasto no processamento de interrupções, garantindo que as funções ligadas a eles são processadas no menor tempo possível depois que eles expiram.

4. Descrição do Problema

Os temporizadores de alta resolução preemptam as tarefas que estão sendo executadas no sistema por um tempo maior ou menor, dependendo da quantidade de temporizado-

res expirando ao mesmo tempo e se eles devem ser executados via *hard irq* ou *softIRQ*. Mas ainda existem alguns temporizadores dos que são executados via *hard irq* que são utilizados como *sleeps* por determinadas tarefas, ou seja, utilizados para acordar a tarefa depois de um período de tempo. Através de medições realizadas no kernel do Linux, verificou-se que o maior número de temporizadores de alta resolução são os executados em *hard IRQ*, mas que não são definidos explicitamente como temporizadores de *sleep*, tendo na sua maioria de execuções 475 temporizadores por segundo como pode ser verificado na figura 1. Seguido deles estão os que são definidos explicitamente como *sleep*, com uma maioria de 100 temporizadores por segundo e os executados via *softIRQ* são pouquíssimos.

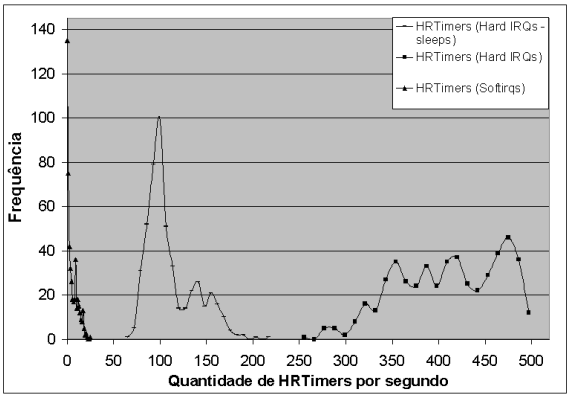


Figure 1. Quantidade de HRTimers executados por segundo

Os temporizadores definidos explicitamente como *sleep*, tem o objetivo de retirar uma tarefa da fila de espera do processador e colocá-la na fila de prontos, desta forma a tarefa será escalonada pelo processador e entrará em execução assim que for possível. Mas esta tarefa que será acordada pelo temporizador possui uma prioridade, a qual é utilizada para definir quando ela será executada pelo processador. Desta forma mesmo ela sendo acordada pelo temporizador em *hard IRQ*, ela deverá esperar ser executada de acordo com sua prioridade. Sendo assim, caso existam outras tarefas de maior prioridade na fila de prontos, elas serão executadas antes desta tarefa. Assim acordá-la neste momento não causará nenhum benefício para ela, pois ela terá de aguardar sua execução de qualquer forma e ainda atrapalha por um tempo maior as tarefas de prioridades maiores que a dela.

Como exemplo supõe-se três tarefas escalonadas pela política de escalonamento para tempo real SCHED_FIFO (T1, T2 e T3), as quais têm prioridades 90, 80 e 70 respectivamente. Considera-se a tarefa T1 com *deadline* e período igual a 32 e as tarefas T2 e T3 com *deadline* e período igual a 30. Considera-se também o tratador de interrupção dos temporizadores como uma tarefa de mais alta prioridade, já que executa em *hard IRQ*. Então como demonstrado na figura 2, T1 começa a executar e logo após dois temporizadores responsáveis por acordar T2 e T3 expiram, gerando uma interrupção no processador que executa o tratador de interrupção, o qual preempta todas as tarefas em execução e conduz T2 e T3 para a fila de prontos. Depois da interrupção ser tratada as tarefas voltam a executar de acordo com suas prioridades. Pode-se verificar na figura 2 que o tratador de interrupção gasta um tempo considerável, justamente para acordar as tarefas T2 e T3 que não vão poder executar ainda, preemptando de forma desnecessária T1.

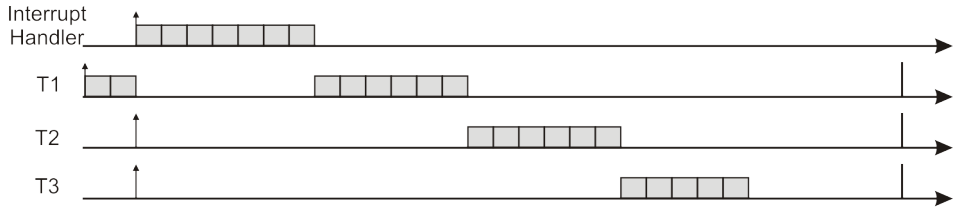


Figure 2. Exemplo de como o tratador de interrupção atrapalha tarefas de alta prioridade

Pode-se calcular o tempo que o tratador de interrupção dos temporizadores gasta, preemptando as tarefas em execução no sistema, utilizando a equação (1).

$$C_{IRQ} = C_{TC} + C_{FG} + C_{TIH} \tag{1}$$

Onde C_{IRQ} é o tempo total gasto pela chamada de interrupção do processador, C_{TC} é o tempo gasto na troca de contexto, C_{FG} é o tempo gasto nas funções gerais de interrupção (como atualizar estatísticas e variáveis) e C_{TIH} é o tempo gasto pelo tratador de interrupção dos temporizadores de alta resolução, o qual pode ser calculado pela equação (2).

$$C_{TIH} = \sum_{i=1}^{n} \Big(C_{FT}(i) \Big) \tag{2}$$

Onde $C_{FT}(i)$ é o tempo de computação da função atrelada ao temporizador i, o qual varia bastante de acordo com o que ele executar, podendo ter vários fatores responsáveis pela variação da mesma função.

5. Abordagem Proposta

Este trabalho propõe diminuir o tempo de preempção que uma tarefa pode sofrer pelo tratador de interrupções em alguns casos. Mais especificamente quando a interrupção é para tratar temporizadores expirados e que tenham como função acordar alguma tarefa. Pois a proposta é fazer com que as funções ligadas a estes temporizadores sejam executadas em momentos posteriores, não atrapalhando assim as tarefas com prioridades maiores do que a que criou o temporizador. Mas ainda assim acordar a devida tarefa no momento que ela possa ser executada sem atrapalhar outras de prioridade maior.

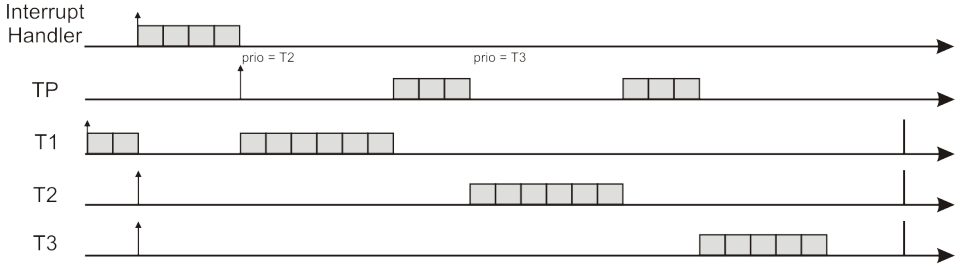


Figure 3. Exemplo de como deve funcionar após a implementação

A proposta é criar uma tarefa com prioridade dinâmica, ou seja, sempre que necessário ela terá sua prioridade modificada. Ela será responsável por executar todas as funções ligadas aos temporizadores definidos explicitamente como *sleeps*. Esta tarefa então assumirá a prioridade da outra que criou o temporizador, executando assim apenas quando não existir nenhuma tarefa com prioridade maior. Desta forma ela acordará a devida tarefa no momento que esta já possa executar, respeitando assim as prioridades.

Para isso ser possível devem ser feitas algumas alterações no kernel do Linux, como criar uma árvore vermelha e preta, a qual armazenará estes temporizadores pela prioridade ligada a eles. Então em vez de processar esses temporizadores expirados em *hard IRQ*, eles devem ser transferidos da árvore vermelha e preta onde são ordenados por tempo de expiração para a árvore criada. Logo em seguida a tarefa criada para processar estes temporizadores deve ser acordada, tendo sua prioridade alterada de acordo com os temporizadores expirados.

Desta forma, caso haja uma tarefa T com prioridade P executando no processador e existam n temporizadores com prioridades menores que P, as quais irão expirar antes que a tarefa T termine de executar, esta tarefa não vai sofrer uma preempção tão grande. Isso pode ser verificado pela alteração do cálculo da variável C_{TIH} da equação (1) demonstrado pela equação (3).

$$C_{TIH} = \sum_{i=1}^{n} \Big(C_{TA}(i) \Big) \tag{3}$$

Onde $C_{TA}(i)$ é o tempo gasto para trocar o temporizador i de uma árvore vermelha e preta para outra. Lembrando que o tempo para remover ou inserir um temporizador em uma árvore vermelha e preta é $log(n)$, mas para a árvore criada para este trabalho o n tem um valor máximo de 100 nodos ($log(100) = 2$), pois os temporizadores são classificados entre as 100 prioridades de tempo real. Desta forma o tempo para realizar esta troca varia mais devido a quantidade de nodos na primeira árvore. Através da figura 4, pode-se verificar que o tempo gasto para trocar o temporizador de fila é bem menor que o tempo gasto para acordar a tarefa, como isso faz parte da proposta deste trabalho, realmente o tempo gasto em *hard irq* vai ser diminuído.

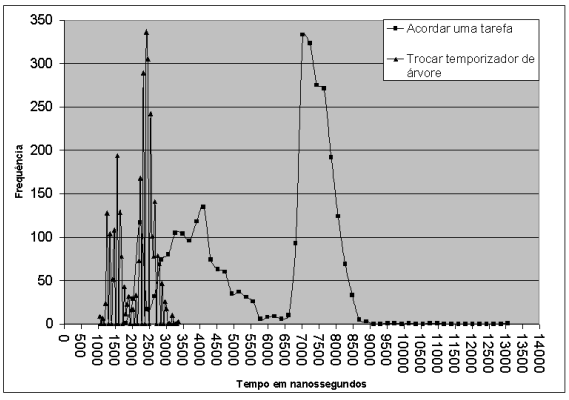


Figure 4. Variação de tempo para acordar uma tarefa e trocar um temporizador de árvore

Com estas alterações pretende-se diminuir o tempo em certas execuções do tratador de interrupção, diminuindo assim o tempo de preempção que ele causa às tarefas. Mas as tarefas que devem ser acordadas por estes temporizadores acabam recebendo algum acréscimo no atraso de sua liberação (*release jitter*) como pode ser visto na figura 3. Este acréscimo se dá porque a tarefa que já deveria estar na fila de prontos, ainda está na fila de espera e ainda deve mudar de fila através da execução da tarefa criada para isso. Então devido a execução desta tarefa que processará o temporizador, o atraso na liberação deverá ser pouco incrementado.

Aplicando-se as alterações sugeridas ao exemplo da figura 2, pode-se supor que o escalonamento deste exemplo ficaria semelhante ao exemplo da figura 3. Onde TP é a tarefa proposta para este trabalho e durante sua execução no exemplo, ele tem sua prioridade alterada para a prioridade da tarefa T2 e T3 respectivamente. Então através do exemplo pode-se verificar a diminuição do tempo de preempção que a tarefa T1 sofre, já a tarefa T2 não sofre alteração na sua execução, mas a tarefa T3 sofre um atraso maior na sua liberação.

6. Conclusão

Neste artigo foi proposto um método de postergar a execução de alguns temporizadores utilizados para acordar tarefas. De acordo com os cálculos apresentados neste artigo, pode-se ter uma prévia da diminuição do tempo de preempção de algumas tarefas, onde quanto maior a carga de temporizadores deste tipo, melhor se aplica este método, pois diminui consideravelmente o tempo gasto no tratador de interrupção.

O próximo passo deste trabalho será realizar as alterações necessárias no kernel do Linux, como criar a tarefa responsável por processar os temporizadores, adicionar alguns campos na estrutura dos temporizadores de alta resolução e alterar alguns trechos de código relacionado a eles. Depois testar, medir os tempos de execução, verificar a diferença na execução entre o *kernel* base e o alterado e por fim validar os cálculos baseado nos tempos da alteração realizada.

References

Etsion, Y., Tsafrir, D., and Feitelson, D. G. (2001). Effects of clock resolution on the scheduling of real-time and interactive processes. *School of Computer Science and Engineering*.

Gleixner, T. and Niehaus, D. (2006). Hrtimers and beyond: Transforming the linux time subsystems. *Proceedings of the Linux Symposium*.

Love, R. (2005). *Linux Kernel Development*. Novell.

Mauerer, W. (2008). *Professional Linux Kernel Architecture*. Wiley Publishing, Inc., Indianapolis.

Molnar, I. (2005). *PREEMPT-RT*. Disponível em: http://www.kernel.org/pub/linux/kernel/projects/rt - Último acesso em: 27 fev 2010.

POSIX.13 (1998). *IEEE Std. 1003.13-1998. Information Technology -Standardized Application Environment Profile-POSIX Realtime Application Support (AEP)*.

Análise da Plataforma SunSPOT para Programação de Sistemas de Controle Distribuído em Rede de Sensores sem Fio*

André Cavalcante[1], Rodrigo Allgayer[1], Ivan Müller[1], Jovani Balbinot[1], Carlos E. Pereira[1]

[1]Departamento de Engenharia Elétrica – Universidade Federal do Rio Grande do Sul (UFRGS)
Av. Osvaldo Aranha, 103 – Porto Alegre/RS – Brasil

andrecavalcante@ufam.edu.br, allgayer@ece.ufrgs.br, ivan.muller@ufrgs.br,
jovani.balbinot@gmail.com, cpereira@ece.ufrgs.br

Abstract. *This work presents a temporal analysis for a diferential control system of a mobile robot in SunSPOT platform. In such system, which is programmed in Java, each part of it is executed in distinct processor device. These devices form a wireless sensor network. The results of simulations on PC are compared with experimental data and reveal inserted errors by the communication system and by the processing load of the platform.*

Resumo. *Este trabalho apresenta uma análise temporal para um sistema de controle diferencial de um robô móvel em uma plataforma SunSPOT. Neste sistema, programado em Java, cada parte é executada em dispositivos de processamento distintos. Estes dispositivos formam uma rede de sensores sem fio. Os resultados de simulações em PC são comparados com os dados experimentais e revelam os erros introduzidos pelo sistema de comunicação e pela carga de processamento da plataforma.*

1. Introdução

Sistemas de controle distribuídos são desejáveis por diversas vantagens tais como o aumento na robustez do sistema, redundância e a migração de tarefas entre os sistemas computacionais da rede [Zampieri 2008]. Também, a carga de processamento por máquina é balanceada, não sobrecarregando somente uma máquina da rede, ao contrário de sistemas centralizados onde o servidor pode estar sobrecarregado.

Atualmente, diversas tecnologias e protocolos de comunicação possuem capacidade de proporcionar a interligação de sistemas de forma distribuída, incluindo comunicações cabeadas e comunicações sem fio. Com o avanço da comunicação sem fio, há agora a possibilidade da utilização deste tipo de tecnologia em ambientes industriais, reduzindo-se custos devido a não necessidade de uma estrutura física (cabos e barramentos) para transmissão de dados entre os dispositivos do sistema [Kawka and Alleyne 2005, Liu and Goldsmith 2004].

Por outro lado, a utilização de sistemas distribuídos demanda uma arquitetura descentralizada muito dependente do comportamento do sistema de comunicação.

Este artigo analisa o desempenho de um sistema de controle distribuído utilizando-se Rede de Sensores Sem Fio (RSSF). Como plataforma de desenvolvimento foi utilizado

*Este trabalho é parcialmente financiado pelas instituições CAPES, CNPq e FAPEAM.

nodos SunSPOTs [Microsystems 2010b]. Uma API (Application Programming Interface) que possibilita a chamada de métodos remotos também foi desenvolvida a fim de facilitar a programação de sistemas distribuídos.

O texto está dividido da seguinte forma, na Seção 2, apresenta-se as características e as dificuldades que surgem na programação de sistemas de controle distribuídos utilizando Java e SunSPOT. A seguir, na Seção 3, é descrita a API desenvolvida para chamada de métodos remotos. Na Seção 4, é apresentada a aplicação desenvolvida para posterior análise, e na Seção 5, os resultados obtidos. Por fim, na Seção 6, são colocadas as conclusões finais deste trabalho e apresentados os trabalhos futuros.

2. Programação de aplicações distribuídas

Em um ambiente industrial, é importante a realização de distribuição completa em uma rede de sensores, isto é, que todos os nodos da rede possam acessar as respectivas aplicações distribuídas em todos os demais nodos, independente de qual tipo é tal nodo. Isso significa que as aplicações que são executadas em uma rede industrial, abstraem a própria rede, de acordo com a sua aplicação. Não é interessante para o programador ter que criar código para tratar diferentes tipos de nodos (sensores, atuadores, repetidores, controladores, entre outros).

Além disso é esperado que, em uma rede industrial, as aplicações e a rede tenham certas características, como a auto-organização, onde os nodos têm a capacidade de organizar-se autonomamente na rede. Para realizar a auto-organização, o desenvolvedor deve prover a rede com a capacidade de distribuição automática de endereços e portas de comunicação, bem como a descoberta de serviços. A rede deve também ser capaz de se comunicar com dispositivos não anteriormente planejados, desde que respeitadas algumas interfaces padronizadas.

Uma arquitetura promissora que permite distribuição e possui um registro dos objetos (e serviços) disponíveis na rede é o RMI (Remote Method Invocation) [Waldo 1998]. O RMI é uma interface de programação que permite a execução de chamadas remotas no estilo RPC (*Remote Procedure Call*) em aplicações desenvolvidas em Java. Através da utilização da arquitetura RMI, é possível que um objeto ativo em uma máquina virtual Java possa interagir com objetos de outras máquinas virtuais Java, independentemente da localização dessas máquinas virtuais na rede. Dessa forma, algumas das características de RMI são desejáveis para sistemas industriais.

3. Chamadas de métodos remotos na plataforma SunSPOT

A plataforma de escolha para este trabalho foi a SunSPOT. Esta foi desenvolvida para a programação de aplicações para rede de sensores sem fio utilizando funcionalidades da linguagem Java. Possui sensores embarcados e um rádio de comunicação sem fio, representados por objetos Java cujas funcionalidades são acessadas através de métodos.

Enquanto é simples e rápido o acesso aos periféricos do SunSPOT, o mesmo apresenta algumas limitações importantes no que tange o desenvolvimento de uma rede de sensores e atuadores completamente distribuídos. Por ser baseado na CLDC 1.0 [Microsystems 2010a], o SunSPOT não possui as características de Java necessárias à chamadas de métodos remotos e também não possui nenhum tipo de serialização de objetos. Porém, tais características podem ser inseridas no sistema através da programação

de uma API. Com o objetivo de prover ao programador tanto a serialização quanto a chamada remota de métodos foi desenvolvida uma API denominada mRMI (minimum Remote Method Invocation), na tentativa da utilização de um SunSPOT em ambientes industriais.

Assim definiu-se uma interface denominada de `Externalizable`. Quando objetos de uma classe necessitarem ser serializados (externalizados, no jargão do RMI), basta que a classe implemente a interface. Assim, os objetos das classes especiais `ObjectOutputStream` e `ObjectInputStream` podem operar sobre um objeto que implemente `Externalizable`, realizando o transporte dos objetos pela rede. As classes `ObjectOutputStream` e `ObjectInputStream` também necessitaram ser criadas. Contudo cabe à classe que implementa `Externalizable` realizar a escrita e leitura adequada de seus parâmetros a partir dos *streams* fornecidos. Essa abordagem foi chamada de "manual", porque deixa ao programador toda a carga do protocolo de serialização.

Foi igualmente definida uma interface `RemoteSever` para permitir a construção de servidores que possibilitem a recepção de chamadas remotas. No lado do cliente foi criada uma interface `RemoteStub` que define o comportamento padrão para todos os "stubs" da API. Foram criadas classes com um comportamento e protocolo padrão a fim de minimizar os esforços de programação, contudo o programador deve ao menos conhecer o protocolo de rede para a utilização correta do sistema.

Como não há nenhuma implementação padrão no SunSPOT para o RMI, então não há igualmente um registro de objetos remotos (o *registry*). Isto também teve de ser criado e o gerenciamento de endereços e portas também teve de ser realizado manualmente.

4. Estudo de Caso: Simulação de um robô móvel distribuído

Como estudo de caso de um sistema de controle distribuído utilizando uma malha de controle fechada, foi escolhido um robô móvel que se movimenta no plano [Lages 2008]. O robô apresenta um acionamento diferencial que pode ser descrito pelo modelo no espaço de estados dado pela Equação 1.

$$\dot{x}(t) = \begin{bmatrix} \cos(x_3) & 0 \\ \sin(x_3) & 0 \\ 0 & 1 \end{bmatrix} u(t) \qquad y(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} R\cos(x_3) \\ R\sin(x_3) \end{bmatrix} \tag{1}$$

onde $x(t) = [x_c\ y_c\ \theta]^T$, sendo $(x_c,\ y_c)$ a posição do centro de massa do robô e θ a sua orientação. $u(t) = [v\ \omega]^T$ é a entrada do sistema, sendo v a velocidade linear e ω a velocidade angular do robô. A saída do sistema é $y(t)$, correspondendo a frente do robô cujo diâmetro é $D = 2R = 0.6m$.

Para este sistema, tem-se:

$$\dot{y}(t) = \begin{bmatrix} \cos(x_3) & -R\sin(x_3) \\ \sin(x_3) & R\cos(x_3) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = L(x)u(t) \tag{2}$$

e portanto, como $L(x)$ é não singular, o sistema pode ser linearizado por realimentação fazendo-se $u = L^{-1}(x)v$, e sendo v a nova entrada do sistema linearizado e desacoplado

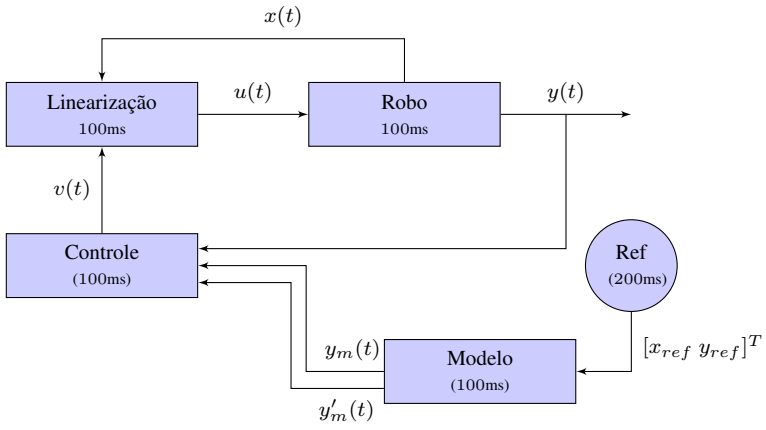


Figura 1. Diagrama em blocos para a simulação do robô.

dado por $\dot{y}(t) = v(t)$. Então, para controlar o sistema é utilizado um controlador por modelo de referência com $v(t)$ dado por:

$$v(t) = \begin{bmatrix} \dot{y}_{mx} + \alpha_1(y_m x - y1) \\ \dot{y}_{my} + \alpha_2(y_m y - y2) \end{bmatrix} \tag{3}$$

A referência é dada por:

$$\begin{cases} x_{ref}(t) = \frac{5}{\pi}\cos(0.1\pi t) \\ y_{ref}(t) = \begin{cases} \frac{5}{\pi}\sin(0.1\pi t) & \text{, para } 0 <= t < 20 \\ -\frac{5}{\pi}\sin(0.1\pi t) & \text{, para } t >= 20 \end{cases} \end{cases} \tag{4}$$

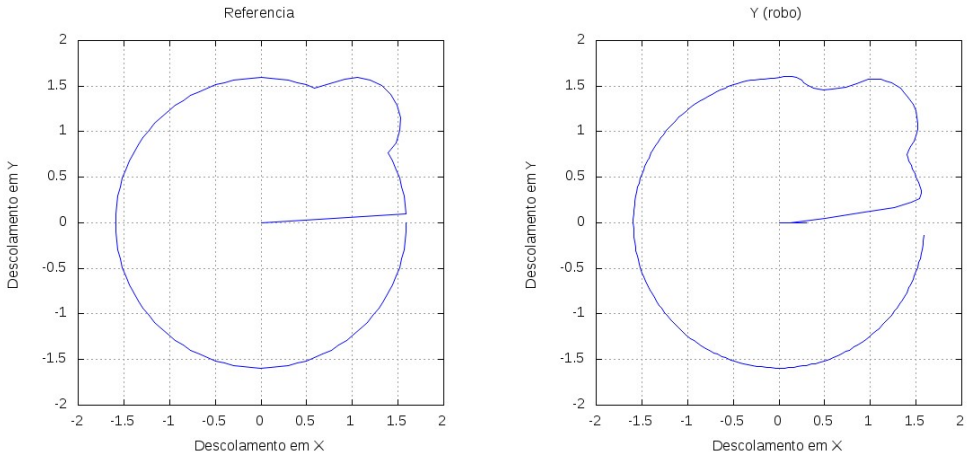
O robô descrito pela Equação 1 foi simulado no SunSPOT e visa refletir o comportamento da frente do robô (a saída do sistema) dada a entrada da próxima posição do centro de massa. A partir das equações mostradas, pode-se montar um diagrama em blocos do sistema do robô móvel controlado, sendo este representado na Figura 1.

O sistema do robô foi separado em dois nodos distintos: o nodo Robô e o nodo Referência. O nodo Robô executa as equações do robô dado pela Equação 1. Por outro lado, o nodo Referência executa os demais blocos funcionais apresentados na Figura 1.

5. Resultados

Com base no estudo de caso desenvolvido na Seção 4 e a fim de que se tenha uma visão clara da plataforma SunSPOT como nodo em uma aplicação de sistema de controle distribuído, algumas simulações foram realizadas com períodos de amostragem de 50ms, 80ms, 100ms e 200ms. Também foram realizadas simulações em que todas as *threads* do sistema executam somente em um PC, ou somente no SunSPOT, ou utilizando uma abordagem híbrida. Como análise dos resultados, neste artigo são abordados dois casos.

No primeiro caso os nodos Robô e Referência executam em uma máquina PC. Assim, a comunicação entre os nodos não apresenta interferência da rede física, e ainda ocorre em uma plataforma capaz de suportar muito processamento. Na Figura 2(a) está representada a referência gerada para o robô e, na Figura 2(b) a trajetória que o robô

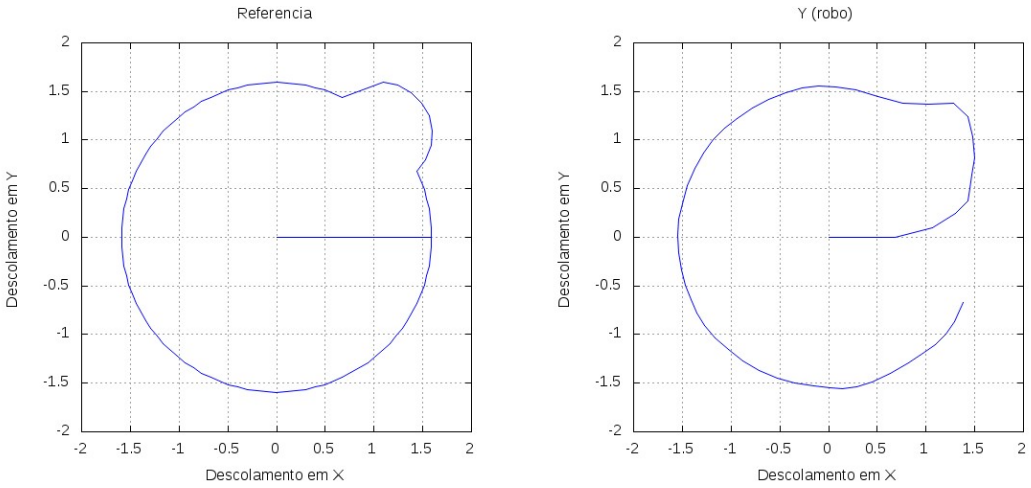


(a) Sinal de referência (b) Sinal de saída

Figura 2. Simulação em uma plataforma PC

realiza ao ser simulado o seu modelo. As figuras representam um plano (x, y) sobre o qual o robô se move.

Em outra situação, o nodo Robô é executado em um dispositivo SunSPOT, e o nodo Referência é executado em uma máquina PC comunicando-se com o nodo Robô através de rádio-frequência. O objetivo desta simulação é verificar o desempenho do sistema e o *jitter* dos períodos de execução dos blocos funcionais em um sistema de controle distribuído utilizando comunicação sem fio. Na Figura 3(a) está representada a referência gerada para o robô executado em um PC e, na Figura 3(b), a trajetória que o robô realiza ao ser simulado o seu modelo na plataforma SunSPOT.



(a) Sinal de referência (b) Sinal de saída

Figura 3. Simulação em uma plataforma SunSPOT

A análise estatística para o período de amostragem, dos dois casos, está apresentado na Tabela 5.

Tabela 1. Medidas estatísticas do jitter na simulação do robô nas plataformas.

	Simulação PC	Simulação SunSPOT
Período	0.100000s	0.100000s
Amostras	202	177
Mínimo	0.097000s	0.057000s
1o Quartil	0.099000s	0.092000s
Mediana	0.100000s	0.110000s
Máximo	0.105000s	0.180000s
Media	0.099801s	0.113744s
Desvio padrão	0.000860s	0.027153s

6. Conclusões

Comparando os gráficos da referência e aqui colocados, como era de se esperar as simulações utilizando a rede sem-fio apresentam erros maiores. As simulações com período de amostragem inferior a 100ms apresentam erros ainda maiores. As simulações no PC contudo não apresentam os mesmos erros, o que mostra que a plataforma escolhida (SunSPOT) carece de performance de processamento para este caso.

Igualmente o esforço necessário para a programação da API mRMI torna apenas mais simples a construção da aplicação em si, contudo o cuidado com o protocolo de comunicação ainda é relevante no esforço de programação. Portanto, verificou-se que o SunSPOT pode ser utilizado para suportar uma API que implemente chamada de métodos remotos, porém isto somente é possível para aplicações pequenas, onde existe um número limitado de nodos e de malhas de controle. Na mRMI, a complexidade do código aumenta em função do número de nós e da complexidade da aplicação, sendo necessário uma plataforma de hardware com maior capacidade de processamento para executar aplicações mais complexas.

Como trabalho futuro está previsto o desenvolvimento de estimadores do atraso do sistema para que o seu impacto possa ser minimizado.

Referências

Kawka, P. and Alleyne, A. (2005). Stability and feedback control of wireless networked systems. In *Proceedings of American Control Conference*.

Lages, W. F. (2008). Projeto. Technical report, Universidade Federal do Rio Grande do Sul.

Liu, X. and Goldsmith, A. (2004). Wireless network design for distributed control. In *Proceedings of 43rd IEEE Conference on Decision and Control*.

Microsystems, Sun (2010a). *Connected Limited Device Configuration - CLDC*. Sun Microsystems. Disponível em: <http://http://java.sun.com/products/cldc/>. Acesso em: março 2010.

Microsystems, Sun (2010b). *Sun Small Programmable Object Technology - Theory of Operation*. Disponível em: <http://www.sunspotworld.com/docs/index.html>. Acesso em: março 2010.

Waldo, J. (1998). Remote procedure calls and java remote method invocation. In *IEEE Concurrency*, volume 6, pages 5 – 7.

Zampieri, S. (2008). Trends in networked control systems. In *Proceedings of 17th IFAC World Congress*.

Uma Proposta para Visualização Aumentada em Tempo Real aplicada a Indústria

Danúbia Espíndola[1,2], Carlos E. Pereira[1], Renato V. Henriques[1], Silvia S. Botelho[2]

[1]Departamento de Engenharia Elétrica – Universidade Federal do Rio Grande do Sul (UFRGS)
Av. Osvaldo Aranha, 103 CEP: 90035-190 – Porto Alegre – RS – Brazil

[2]Centro de Ciências Computacionais – Universidade Federal do Rio Grande (FURG)
Av. Itália km 8 Bairro Carreiros – Rio Grande – RS - Brazil
{danubia,cpereira,rventura}@ece.ufrgs.br, silviacb@furg.br

Abstract. *This paper describes a proposal to visualization, using techniques of Augmented Reality that enable the real-time visualization of industrial processes.*

Resumo. *Este artigo descreve uma proposta para visualização, através de técnicas de Realidade Aumentada que possibilitam a visualização em tempo real de processos industriais.*

1. Motivação e objetivos

Integração no campo industrial significa atualmente "engenharia digital" e eficiente gerenciamento da informação. Esta afirmação é evidenciada pela crescente quantidade de sistemas digitais que surgem diariamente no campo industrial [Buccioli et al, 2006]. No entanto a falta de padrões em formatos digitais, a usabilidade da interface de comunicação entre homem-computador e o entendimento da informação vem tornando-se aspectos críticos com o surgimento acelerado dessas tecnologias.

Neste sentido, a complexidade de tarefas de manutenção, montagem e desmontagens de máquinas e seus sistemas digitais são um excelente domínio para as aplicações de Realidade Virtual, Mista e Aumentada (RV, RM e RA) [Georgel et al, 2009]. A visualização de sistemas com suporte a Realidade Aumentada representam uma potencial solução em interfaces homem-máquina, auxiliando no fornecimento de instruções contínuas, autônomas, em tempo real apresentadas no local certo e na hora certa. As pesquisas apontam para a RA como uma técnica de visualização em tempo real que pode ser fortemente aplicada ao contexto industrial [Regenbrecht et al, 2005].

A possibilidade de compartilhar e modificar dados remotamente; a interação em tempo real com o equipamento assistido através de dispositivos de RA; são objetivos claros da implantação de um sistema de visualização aumentada. Os ganhos com a implantação destas tecnologias atuam diretamente nos requisitos de tempo, qualidade e custo. Entre as possíveis aplicações de RA na indústria citam-se: projeto, colaboração, treinamento, manufatura e manutenção [Schoenfelder et al, 2008] [Henderson et al, 2009].

Sendo assim este estudo pretende propor uma metodologia bem como uma solução para validação de um sistema de visualização aumentada que atenda os requisitos de tempo para aplicações industriais.

2. Contextualização e trabalhos relacionados

Este estudo irá explorar aspectos de Realidade Aumentada como forma de misturar informações virtuais ao ambiente real da fábrica/equipamento para obtermos uma visualização em tempo real guiada pelo computador. A união de objetos virtuais com o cenário real da indústria, produzirá um ambiente único, sobreposto ao ambiente físico. O uso de RA permitirá análise, interação e exploração de aspectos cognitivos, relatados com a compreensão da informação, facilitando as tarefas de manutenção, treinamento, operação e a tomada de decisão.

Entre os principais desafios na implantação de técnicas de RA para visualização em tempo real, verificou-se a dificuldade na capacidade de sincronizar e alinhar num mesmo sistema de coordenadas, o movimento da câmera, o ambiente do utilizador e os objetos virtuais inseridos nesse ambiente. Ou seja, o alinhamento e a sincronização entre o real e o virtual em tempo real. No entanto, a utilização da biblioteca ARToolkit com a uso de marcadores para rastreamento apresenta bons resultados em termos de tempo de processamento. Por outro lado, a complexidade de modelos de equipamentos e plantas requer que a etapa de modelagem do ambiente virtual seja realizada previamente.

Entre os principais trabalhos encontrados para visualização aumentada aplicada a processos industriais destacam-se: o projeto ARVIKA, AMIRE, STARMATE, INT-MANUS. Outros estudos relevantes são: STUDIESTUBE, DWARF, ARBA e AAR.

O ARVIKA (*Augmented Reality für Entwicklung, Produktion und Service*) é um dos projetos de RA mais citados no estado da arte e é aplicado a indústrias de manufatura da Alemanha, liderado pela Siemens [Friedrich et al, 2002]. Basicamente utilizam-se capacetes virtuais (HMDs) para auxílio a operação de equipamentos da indústria através da interação com objetos virtuais sobrepostos a cena real.

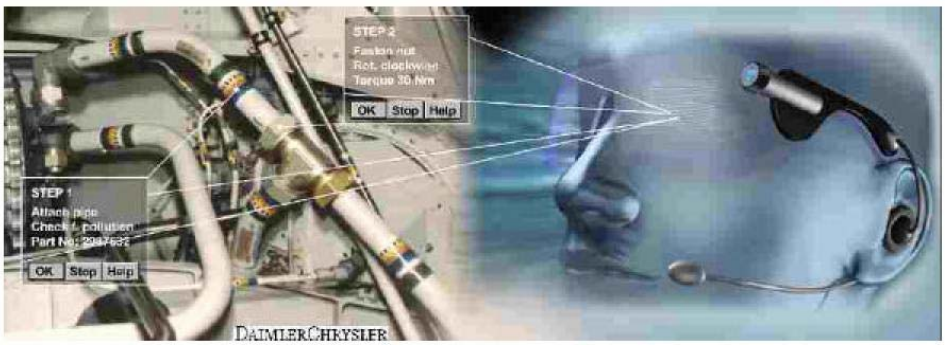


Figura 1 ARVIKA [Friedrich et al, 2002]

A utilização de HMD (*Head Mounted Display*) - *optical see-through* - possibilita visualizar o ambiente real tendo como apoio a sobreposição de elementos virtuais sobre a cena real, ou seja, o usuário tem como predominância a visualização do mundo real. As informações dos sistemas digitais podem ser requisitadas por diversos meios de interação, entre as soluções encontradas o comando de voz é uma alternativa

interessante uma vez que as mãos do usuário podem estar livres para manipulação do ambiente.

Sendo assim, buscaram-se ferramentas de realidade aumentada que possibilitassem a sobreposição de elementos virtuais sobre o ambiente real de forma contínua em tempo real conforme a requisição do usuário. Para isto uma arquitetura conceitual e uma metodologia para as etapas da modelagem à visualização foram desenvolvidas, e serão apresentadas na próxima sessão, com intuito de descrever e modelar a solução proposta.

3. Arquitetura e metodologia da proposta

A arquitetura conceitual ilustrada na figura 2 descreve os módulos envolvidos no desenvolvimento de um ambiente misto para assistência contínua de operadores durante a interação com equipamentos (ou plantas) industriais.

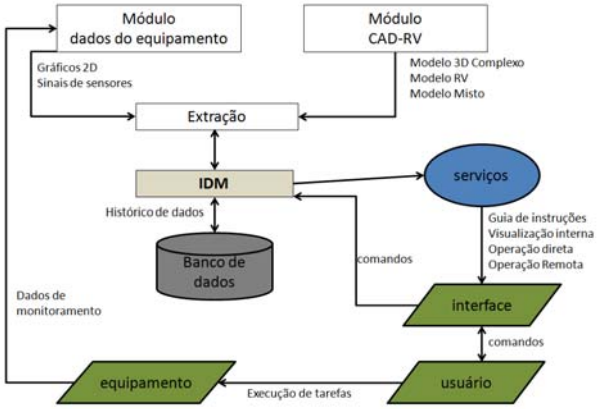


Figura 2 Arquitetura conceitual para a solução proposta.

Inicialmente dois módulos devem fornecer as entradas para o sistema de visualização aumentada: Módulo dados do equipamento e Módulo CAD-RV. O primeiro deve fornecer as informações sobre o equipamento a ser analisado. Estas informações podem ser: gráficos 2D, sinais de sensores, informações textuais; fornecidos por sistemas digitais de infotrônica utilizados na indústria .

O segundo, o módulo CAD-RV, deve gerar o modelo de realidade virtual a partir de modelos CAD complexos ou simples, gerados por ferramentas DMU (*Digital Mock-Up*). Da etapa de modelagem até a etapa de visualização devem ser realizados diversos estágios, tais como: redução de complexidade (quando necessário) e conversão de formatos.

As entradas fornecidas por estes módulos devem ser extraídas e integradas em um Modelo Descritivo Integrado (IDM) que irá relacionar os diferentes modelos para gerenciamento da visualização de acordo com os comandos do usuário. O modelo IDM está sendo descrito na linguagem XML (eXtensible Markup Language), que é uma linguagem de marcação para descrição hierarquizada de componentes.

O IDM além de descrever a integração de diferentes modelos, ou seja, relacionar informações CAD, RV, RM e MD (modelo de dados do equipamento), gerencia a apresentação de serviços na interface através de algoritmos de tomada de decisão

baseados nas informações armazenadas colaborativamente no banco de dados e nos comandos do usuário.

A Figura 3 descreve a metodologia aplicada para desenvolvimento do módulo CAD-RV. Primeiramente na etapa de modelagem faz-se a aquisição do modelo CAD. Sendo o modelo complexo, passa-se a etapa de redução de complexidade para obter-se um CAD simples. De posse do modelo simples pode-se então gerar um modelo RV (de realidade virtual) através da etapa de conversão de formatos. Por fim, passa-se então a etapa de visualização, ou seja, deve-se gerar o modelo misto, onde real e virtual misturam-se e a geração do conteúdo virtual (gráficos 2D, 3D, guias textuais de tarefas e etc) é agregada ao ambiente real.

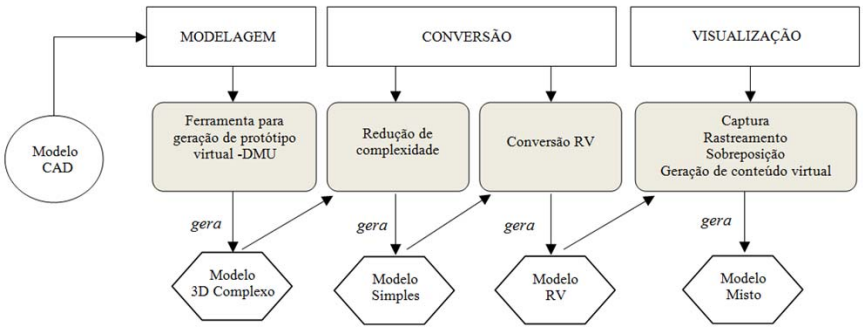


Figura 3 Metodologia para implementação do Módulo CAD-RV.

Por fim, uma metodologia para implementação do modelo IDM, bem como, uma descrição de suas funcionalidades: integração e gerenciamento da visualização, deve ser descrita futuramente.

4. Implementação

Para implementação da arquitetura conceitual e da metodologia proposta acima foram utilizados um conjunto de ferramentas descritos na Figura 4 abaixo. A visualização *on-line* é quando o conteúdo virtual é gerado automaticamente sobrepondo na cena mista às solicitações do operador e as leituras dos sistemas digitais (tais como: sistemas de automação, sistemas de manutenção, sistemas de manufatura). Foram utilizados: Visual Studio - C++ e as bibliotecas gráficas de Realidade Aumentada (ARToolkit), áudio (OPENAL) e renderização (OPENGL).

A possibilidade de visualizar em tempo real objetos virtuais misturados ao ambiente real de fábrica, deu-se principalmente pela excelente rapidez de processamento da biblioteca gráfica ARToolkit. Esta biblioteca com suporte a linguagem Java e C++, possui alta velocidade devido a sua boa integração com processadores de multimídia MMX da Intel. Sendo assim, a escolha pela utilização da ARToolkit deu-se devido a sua possibilidade de captura, rastreamento e sobreposição em um tempo de resposta que atende aos requisitos de operações específicas de montagem e manutenção industrial onde a seqüência de visualizações é feita por demanda e não automaticamente.

Por outro lado, a visualização *off-line* consiste nas visualizações que não são geradas de maneira autônoma pelo sistema, e sim, geradas anteriormente e armazenadas

no banco para posterior utilização. Cabe salientar que a idéia é automatizar o processo de visualização aumentada através da descrição XML, do modelo IDM, de forma a tornar este processo independente de plataformas e ferramentas.

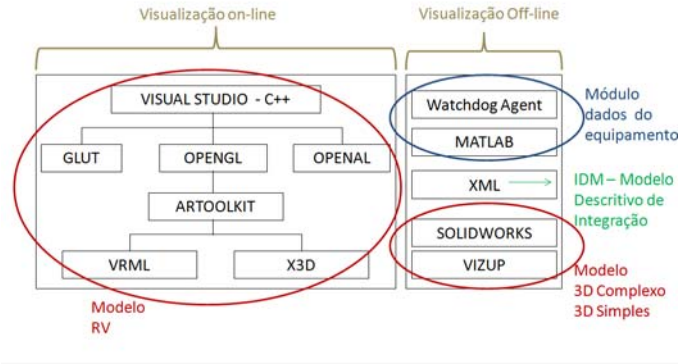


Figura 4 Arquitetura de software utilizada para validação.

A ferramenta Vizup foi utilizada para redução de complexidade do modelo CAD do Atuador CRS. Através da Solidworks importou-se o modelo CAD para conversão em modelo VRML (*Virtual Reality Modeling Language*) – modelo de realidade virtual. Outra possibilidade de formato para modelos virtuais é o X3D que deverá ser testado futuramente.

O sistema digital que forneceu informações a respeito do Atuador CRS foi o *Watchdog Agent* [Djurdjanovic et al, 2003]. Este sistema, denominado sistema de manutenção inteligente *IMS system*, é utilizado na manutenção preditiva de equipamentos críticos da indústria e pode ser observado na Figura 5 b.

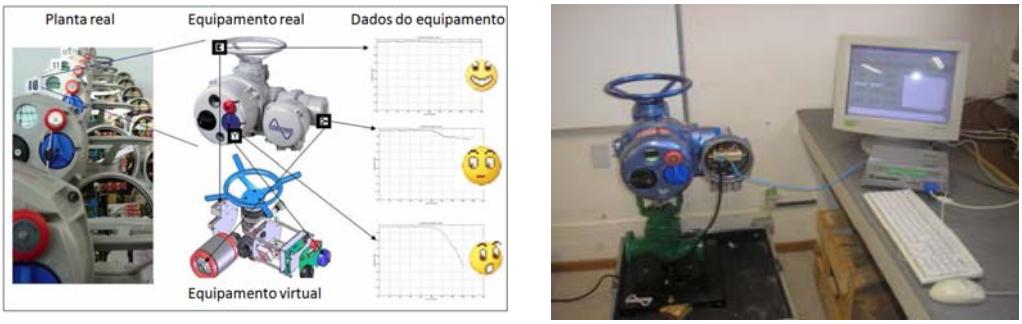


Figura 5 Integração de diferentes dados (esq); Experimento em laboratório (dir).

A figura 5 a esquerda representa a integração de diferentes modelos relacionados através de marcadores de realidade aumentada (etiquetas impressas no ambiente real). Nesta figura podem ser visualizados o equipamento virtual (modelo virtual), o equipamento real, os gráficos 2D adquiridos do sistema IMS (Watchdog Agent) e os três marcadores no atuador real.

5. Resultados e conclusões

Este trabalho apresentou um estudo sobre a integração de sistemas digitais da indústria em ambientes de visualização aumentada para aquisição da informação em tempo real durante as operações em chão de fábrica. De forma mais precisa, foi apresentado um

estudo de caso simulado em laboratório, para visualização aumentada de informações oriundas de um sistema de manutenção preditiva de uma válvula utilizada na indústria de petróleo e gás.

Após a identificação das limitações, restrições e necessidades associados a proposta foram descritos duas etapas para implementação: arquitetura conceitual e uma metodologia para o módulo CAD-RV da arquitetura. Estas etapas pretendem descrever procedimentos para visualização aumentada que integram as diferentes informações e modelos existentes. Ressalta-se que atualmente não existem ferramentas que, de forma integrada, realizem tais procedimentos.

Para o estudo de caso apresentado a ARToolkit atendeu aos requisitos temporais de 14 frames/segundo (em média) na solicitação das informações do Atuador CRS pelo usuário. A utilização de marcadores para o rastreamento de objetos virtuais apresentou bons resultados de tempo para ambientes simulados, estáticos e com baixo ruído. No entanto, a necessidade de validação em ambientes de baixa iluminação, reais e dinâmicos faz-se necessária.

Finalmente, além da conclusão das etapas de extração e de descrição da integração no modelo IDM, estabelecem-se a necessidade de avaliar o uso de outros sistemas digitais da indústria.

Referências

Buccioli, A., Bastos, A., Zorzal, E., Kirner, C. (2006) “Usando Realidade Virtual e Aumentada na Visualização da Simulação de Sistemas de Automação Industrial” In: SVR2006 - VIII Symposium on Virtual Reality, 2006, Belém-PA, 2006.

Djurdjanovic D., Lee J. And Ni J. (2003). Watchdog Agent, an infotronics-based prognostics approach for product performance degradation assessment and prediction. *In Advanced Engineering Informatics*, 2003 - Elsevier: pp. 109–25.

Friedrich W., Jahn D., Schmidt L., (2002), ARVIKA – Augmented Reality for Development, Production and Service, *In Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR’02)* pp.3-4.

Georgel, P., Schroeder, P., and Navab, N. (2009) “Navigation Tools for Augmented CAD Viewing” in IEEE Computer Graphics and Applications.

Regenbrecht H., Baratoff G. And Wilke W., (2005). Augmented Reality Projects in the Automotive and Aerospace Industries, *Published by the IEEE Computer Society, IEEE*.

Schoenfelder, R. and Schmalstieg, D. (2008) “Augmented Reality for Industrial Building Acceptance”, IEEE Virtual Reality 2008.

Henderson, S. and Feiner, S. (2009) “Evaluating the Benefits of Augmented Reality for Task Localization in Maintenance of an Armored Personnel Carrier Turret” in IEEE International Symposium on Mixed and Augmented Reality 2009 Science and Technology Proceedings.

Slow Down or Race to Halt: Towards Managing Complexity of Real-Time Energy Management Decisions *

Stefan M. Petters **Muhammad Ali Awan**

[1]CISTER
ISEP-IPP
Porto – Portugal

{smp,maan}@isep.ipp.pt

Abstract. *Existing work in the context of energy management for real-time systems often ignores the substantial cost of making DVFS and sleep state decisions in terms of time and energy and/or assume very simple models. Within this paper we attempt to explore the parameter space for such decisions and possible constraints faced.*

1. Introduction

Currently we observe a number of trends in the way embedded systems and in particular embedded real-time systems are deployed. Firstly, such systems have become ubiquitous and we become increasingly dependent on them. The computing capability of the processing cores is increasing at a dramatic pace leading to the change towards multi-functional and multi-criticality devices, where hard real-time and best-effort tasks share resources. Finally, while traditionally embedded systems were isolated single purpose devices, they have become often networked and/or mobile.

In particular mobile devices have inherently limited energy supply in terms of batteries. Also tightly packed multicore systems have increasingly thermal issues. In the past we have developed a method to accurately predict time and energy consumption under dynamic frequency and voltage scaling (DFVS) based on online measurements [Snowdon et al. 2007, Lawitzky et al. 2008, Snowdon et al. 2009]. However, a fundamental challenge is the substantial search space for online decision making. For example, our implementation [Lawitzky et al. 2008] of using DVFS in a RBED [Brandt et al. 2003] required 14 multiplications and eight additions per frequency set-point, leading for 22 set points available in our XScale [XScale 2004] platform to a maximum observed of 26,000 cycles. Our observations also indicated frequency and voltage scaling times of 500 to 600 μs.

On the other hand it has been shown that models used in theoretical DVFS work on real-time systems are substantially off the mark. This covers the following assumptions: The power model, where the power consumption is solely dependent on frequency and voltage as given in Equation 1 does not take application dependencies into account.

$$P \propto fV^2 \quad (1)$$

$$C = c * f \quad (2)$$

*This work was supported by CISTER FCT-608 and CooperatES QoS-Aware Cooperative Embedded Systems PTDC/EIA/71624/2006

A constant number of cycles required to execute a task as provided in Equation 2 does not take into account the number of wait states on external devices like main memory, which changes when the core frequency is changed. Frequency switch costs are frequently considered negligible in terms of time and energy, which it is clearly not. Often frequency and voltage is considered continuously scalable, while it in fact only changeable in discrete steps. In particular frequency is often only coarsely scalable for higher frequency set-points, which lower frequency set points can be site on a finer granularity. Individual issues have been tackled in mostly academic work, but a comprehensive approach taking all these factors into account is outstanding. For example, [Aydin et al. 2006] used a non constant number of execution cycles by considering an off-chip and an on-chip component of the execution time, but did not consider the overhead of transitioning into a different frequency set point and using an evaluation by simulation the problem is not exposed. [Cheng and Goddard 2005] used discrete frequencies, but a application independent power consumption, as well as assuming negligible voltage and frequency switch cost.

The problem is acerbated when sleep states are considered. The number of different sleep states available is processor dependent. This reaches from a simple clock gating, where the CPU core is separated from the system clock, to deep sleep states, which power off substantial parts of the chip. The former is available instantaneous (within a CPU core clock cycle), while the latter requires substantial time and energy to get into and out off.

Within this work we will identify all the parameters to be considered to make energy efficient frequency decisions in a real-time context, as well as looking into ways to reduce the search space for such decisions. In the following section we will introduce our system model before discussing implications for DVFS and sleep state decision taking on a static basis and taking knowledge about system dynamic slack into account.

2. System Model

In terms of power management models we draw on our previous work [Snowdon et al. 2007], with a few generalizations. While it is not essential for the discussion to use these models, they give an overall sense of the complexity of the task faced.

The worst-case execution time (WCET) of a task under DVFS is subject to various components. Some time is spent actively performing computations in the CPU core, some time is spent waiting for access to a bus, another part is contributed by the response time of memory, which can be changed by modifying the memory frequency, some time is spent waiting for I/O devices and so forth. Assuming all of these are subject to individual frequency settings, we get an overall execution time C being a function of various application dependent coefficients C_{coeff}.

$$C = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{bus}}{f_{bus}} + \frac{C_{mem}}{f_{mem}} + \dots \tag{3}$$

In the original work, we have identified the coefficients using measurements of certain hardware events, but for generality sake we limit ourselves to identify the makeup of the base equations. Throughout the paper we assume C_i to be the execution time at top speed in the understanding that for DVFS considerations the actual $C(\mathbf{f})$ of the frequency setpoint $\mathbf{f}$ has to be considered.

The energy consumed by a task on a given execution path is also subject to various components as identified in Equation 4. The first component is the energy spent in the CPU core with the core voltage squared. In relation to the XScale processor the circuitry in the CPU core is switched with 3 different frequencies and essentially covers the switching cost of the clock circuitry. The execution time dependence is encoded in the multiplication of C, but beyond that the coefficients γ_x are only architecture and not application dependent. Certain operations are dependent on the core voltage, but not on the CPU core frequency. An example is a multiplication function, where the same number of transistors switches irrespective of the core frequency.

Certain other operations are independent of the core voltage and frequency. An example could be a DMA packet transfer to the network interface, which will have a memory frequency dependent and a memory frequency independent part. The memory frequency independent part might still be subject to a specific frequency, for example internal to the network interface, which might not be scalable. Similar the voltage of the memory would be part of the coefficients. In Equation 4 this is encoded in the parameters α and β respectively. As final components are the static power consumption P_{static} and the memory is subject to switching over the time C. Again the voltage of the memory is considered immutable and is part of the coefficient γ_4.

$$E = V_{cpu}^2(\gamma_1 f_{cpu} + \gamma_2 f_{bus} + \gamma_3 f_{mem}) * C + V_{cpu}^2\alpha + \beta + \gamma_4 f_{mem} C + P_{static} C \quad (4)$$

We have shown [Snowdon et al. 2007, Lawitzky et al. 2008] how the parameters can be obtained runtime. In later work [Snowdon et al. 2009] we have demonstrated an approximation scheme for non RT systems, but this is concentrating on managing the performance to battery life trade-off with no consideration for RT systems. We assume a number of sleep states, which have monotonically increased power reduction, as well as monotonically increased time required to enter t^s and leave t^w the respective state.

Beyond the model for the power management aspects we also assume a sporadic task model with x independent tasks in which the minimum inter-arrival time T_i of a task τ_i is known. We assume form of temporal isolation, which may be achieved using constant bandwidth servers (CBS) [Abeni and Buttazzo 1998] or RBED [Brandt et al. 2003]. Slack is caused by the difference in worst-case assumption used during analysis and actual behavior at run-time. In later section we assume slack management as an important tool. It identifies the room to maneuver for power management in rate based real-time environments [Lin and Brandt 2005, Lawitzky et al. 2008]. For ease of representation we also assume implicit deadline model, where a job has to be completed before a new release is initiated.

3. Static Decision Taking

The least costly decision base is to statically assign frequencies and/or sleep states per task to be performed at start or on completion of the execution respectively. Generally speaking we have to consider the following scenarios: In a *race to halt* solution tasks are executed as fast as possible and the CPU is sent to a sleep state on conclusion of a job of the task. Alternatively a task may be subjected to DVFS to consume less execution time.

In a first step we rule out sleep states which lead to a potential violation of a deadline even when considering only a single task. This can be motivated by a job release of a task being triggered just after a transition into a given sleep state has been

initiated. Once initiated the transition into the sleep state has to be completed, before a wakeup is triggered by the ISR corresponding to the task release. This implies a blocking time of up to $t_n^s + t_n^w$ for sleep state n. Since it does not matter which task has initiated the sleep state transition we can used the rule expressed in Equation 5 to express this. Any sleep state violating Equation 5 may not be used, as it may cause a deadline miss regardless of scheduling algorithm or schedulability analysis used.

$$\min((T_1 - C_1), \ldots, (T_x - C_x)) \leq t_n^s + t_n^w \tag{5}$$

In a second step we examine whether it is economical to use such a measure. One observation in this context is that all tasks in the system have to follow the same rule, as to which sleep states are economical. This is caused by the general assumption that while one task initiates a sleep state, the processor may be transitioned into the waking state by any other task as we do not assume in this section any knowledge of past releases of any given task.

As such we compute the average expected idle time $\bar{G}$ produced by the system. Let l denote the hyper period and r_i represents release time per task τ_i as given in equations Equation 6, Equation 7 respectively. During a hyper period l of the average-case inter-arrival time's $\bar{T}_j$ of all tasks we have a total of r_i releases per task τ_i. We also need to compute how many releases of tasks will happen while another task is executing, as this means that the previous job completion will not lead to a transition into a sleep state.

$$l = \texttt{lcm}(\bar{T}_1, \ldots, \bar{T}_x) \tag{6}$$

$$r_i = \frac{l}{\bar{T}_i} \tag{7}$$

$$p_i = \frac{\sum_{\forall j \in \{0,\ldots,x\} \setminus i} r_j * \bar{C}_j}{l} \tag{8}$$

$$r_i^* = (1 - p_i) * r_i \tag{9}$$

As such we expect r_i^* to be the number of releases of τ_i which happen during idle intervals. Equation 9 determines the r_i^*. Where p_i given by Equation 8 compute the probability of one release of τ_i happening while any other task is executing.. The number of idle time task releases r_i^* will be a real number. However, for the average idle time computation this is of no concern. Now $\bar{G}$ can be computed by Equation 10 where $\bar{C}_i$ denotes the average-case execution time. In this we assume to get approximately $\sum_0^x r_i^*$ transitions into a given sleep state.

$$\bar{G} = \frac{l - \sum_0^x r_i * \bar{C}_i}{\sum_0^x r_i^*} \tag{10}$$

In order to check on the economic switching to sleep states we need to consider the break even time t_n^e of a sleep state. As the transitions into and out of a sleep state do not produce progress in the computation, but consume energy and time, the system needs to spend at least the break even time in that state.

The break even time can be estimated using the energy quantum to transition into and out of the sleep state E_n^s, E_n^w, the power consumed while in a given sleep state P_n, and the idle power consumption of the system P_{idle}. Here Equation 11 quantify the energy of sleep states along with energy of switching cost. Now a valid sleep state which saves on energy when switching to this sleep state needs to satisfy Equation 12.

$$P_{idle} * (t_n^e + t_n^s + t_n^w) = E_n^s + E_n^w + t_n^e * P_n \tag{11}$$

$$\bar{G} \leq t_n^s + t_n^w + t_n^e \tag{12}$$

Additionally, the energy consumption of the system needs to be checked against the energy consumption under DVFS, as well as against the general schedulability test used in the system. For the latter only the switching time required to enter and leave sleep states need to be considered.

4. Slack Management

In this section we assume that online slack information is available. In general we only assume the difference between WCET and actual execution time is known, while the slack caused by sporadic release of tasks is much harder to identify at runtime. A particular concern is that tasks may obtain slack after being preempted and thus already having some of their execution completed. This is of relevance as it adds another dimension in the decision space. In this section we will not assume that the previous release times of all other tasks is known and considered.

Similar to the previous section we assume that infeasible sleep states, which fail to guarantee that all deadlines are met because of the extensive transition time, have been removed. Opposed to the previous section, where a global decision on whether to scale or sleep can be taken, the system has more degrees of freedom for this. We assume the amount of slack passed to as task τ_i from previous tasks is labeled S. We will first discuss the scenario where a task receives S at release time and there are no other tasks in the ready queue. In this scenario the DVFS solution has to be computed using this S. In order to avoid the costly computation of the optimal frequency setpoint, which has to take into account the switching overhead etc, we assume this can be obtained with a number of comparisons S against pre-computed values, deciding on the frequency setpoint to be used. Furthermore the number of setpoints can likely be reduced, depending on the type of application. In our example hardware platform, a memory intensive application will use generally higher memory and bus frequencies when compared to a CPU bound application.

In the case of sleep modes, the algorithm can not only consider the slack being passed in, but also the expected slack generated by the task in question, $(C_i - \bar{C}_i)$ or even the average idle time $\bar{G}$ generated in the previous section. Since the expected slack generated by the task is not changing we can consider that as a constant in the trade-off between DVFS and using sleep modes.

When slack is passed in after preemption by a higher priority task we have to distinguish two scenarios. If the previous decision was to opt for a *race to halt* than the situation got actually more in favor of continuing the approach. However, if the previous decision was to opt for DVFS than the situation needs to be reassessed. In a first approximation the same trade off rules as used in the previous case may be used, ignoring any but the slack passed in after the preemption. When keeping track of the amount of execution completed it may happen that the actual execution time used so far is already more than the average-case execution time $\bar{C}_i$ has been executed in which case the trade-off would be more tuned towards DVFS.

In the case where there are more tasks in the ready queue, the situation becomes slightly more complicated. In this case we see the following possible scenarios. Establishing from the tasks in the ready queue the cumulative average-case slack generated by the tasks can be used to drive the decision. It is a fairly minimal extra effort, as the ex-

pected slack to be generated is inserted and removed, the same way a task is added and removed from the ready queue.

Another approach we envisage is to keep track of the number of jobs executed without any idle interval to reason about the expected idle interval after all jobs in the ready queue have been completed. The reasoning for this behavior is that the presence of many jobs executed in succession indicates that the next releases of the corresponding tasks will only happen at a later time. Note, that this is only a heuristic, which does not require to keep track of exact points of jobs releases in the past. However the exploration of these issues is future work.

5. Conclusion

In this paper we have explored a number of parameters that we need to consider for energy efficient DVFS and sleep states decision in a real-time context. Our proposed approach aims to reduce the design space for making such decision and lighten the runtime overhead for making energy efficient decisions. Future research will tackle work in the area of online DVFS and sleep state decisions and possible heuristics which allow for an efficient trade-off between the two. In particular we will explore how more information about past releases of tasks can be incorporated without causing prohibitive decision making overhead. Furthermore we will explore the impact the proposed methods will have on schedulability analysis methods.

References

Abeni, L. and Buttazzo, G. (1998). Integrating multimedia applications in hard real-time systems. In *19th RTSS*, pages 4–13.

Aydin, H., Devadas, V., and Zhu, D. (2006). System-level energy management for periodic real-time tasks. In *27th RTSS*, pages 313–322, Rio de Janeiro, Brazil. Comp. Soc. Press.

Brandt, S. A., Banachowski, S., Lin, C., and Bisson, T. (2003). Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *24th RTSS*, Cancun, Mexico.

Cheng, H. and Goddard, S. (2005). Integrated device scheduling and processor voltage scaling for system-wide energy conservation. In *2005 WS Power Aware Real-time Comput.*

Lawitzky, M. P., Snowdon, D. C., and Petters, S. M. (2008). Integrating real time and power management in a real system. In *4th OSPERT*, Prague, Czech Republic.

Lin, C. and Brandt, S. A. (2005). Improving soft real-time performance through better slack management. In *26th RTSS*, Miami, FL, USA.

Snowdon, D. C., Le Sueur, E., Petters, S. M., and Heiser, G. (2009). Koala: A platform for OS-level power management. In *4th EuroSys Conf.*, Nuremberg, Germany.

Snowdon, D. C., Petters, S. M., and Heiser, G. (2007). Accurate on-line prediction of processor and memory energy usage under voltage scaling. In *7th EMSOFT*, pages 84–93, Salzburg, Austria.

XScale (2004). *Intel PXA 255 Processor Developer's Manual*. Intel Corp. URL http://www.xscale-freak.com/XSDoc/PXA255/27869302.pdf.

The Effects of Initial Offset and Clock Drift Errors on Clock Synchronization of Networked Control Systems

Eloy M. Oliveira Junior[1], Marcelo L. O. Souza[1]

[1]Space Mechanics and Control Division (DMC)
National Institute for Space Research (INPE) – São José dos Campos, SP – Brazil
eloy@dem.inpe.br, marcelo@dem.inpe.br

Abstract. *Control systems of satellites, aircrafts, automobiles, traffic controls, etc., are becoming increasingly complex and/or highly integrated due to their use of computers networked via communication devices and protocols working in real time. In these systems, the time requirements should be followed strictly, with great precision and synchronization; otherwise the controls degrade until instability. This paper analyzes the effects of initial offset and clock drifts errors on clock synchronization of Networked Control Systems (NCS). To do so, it simulates a typical NCS in the TrueTime/Matlab/ Simulink environment using a FTM correction algorithm and a TDMA communication protocol. The preliminary results suggests that: 1) the TDMA protocol is more susceptible to errors in the initial offset than to errors in the clock drift; 2) the FTM algorithm corrects the clock drift error better than the initial offset error; 3) In a NCS with a TDMA protocol, a fault in the time management can turn the control laws temporarily unstable.*

1. Introduction

Control systems of satellites, aircrafts, automobiles, traffic controls, etc., are becoming increasingly complex and/or highly integrated as defined by the SAE-ARP-4754 Standard. Such systems use, among other key technologies, computers networked via communications devices and protocols working in real time to form Networked Control Systems (NCS). In these systems, the time requirements should be followed strictly, with great precision and synchronization; otherwise the controls degrade until instability. This creates the need to work with high-precision clocks corrected by periodical algorithms to achieve a good time management.

This paper analyzes the effects of initial offset and clock drifts errors on clock synchronization of Networked Control Systems (NCS). To do so, it simulates a typical NCS in the TrueTime/Matlab/Simulink environment using a Fault-Tolerant Mid-Point (FTM) correction algorithm and a Time Division Multiple Access (TDMA) communication protocol. In the first simulation, one of the nodes of a NCS is with an initial offset error in its clock. The initial offset error generates an initial delay that affects the clock synchronization with the FTM algorithm, and then, the communication and control. In the second simulation, one of the nodes of a NCS is with a drift error in its clock. The drift error generates a delay that affects the computing, communication and control. The objective is to analyze: 1) how the TDMA protocol is affected by such errors; 2) the efficiency of the FTM algorithm in correcting such errors; 3) how the NCS is affected by such errors.

The FTM algorithm is fault tolerant: each node reads the value of the clock of the other nodes in the network and estimates the drift of the clocks by a convergence function. In this paper, two control loops sharing the same databus in a network were simulated. Each control loop has a sensor, a controller and an actuator/plant. The nodes of the control loops use the FTM algorithm to synchronize the clocks of the nodes on the network. We used a TDMA communication protocol. The plant used is an electrical/hydraulic actuator of second order controlled by a Proportional, Integrative, and Derivative (PID) controller. We simulated it using the TrueTime toolbox, based on Matlab/Simulink, and we synchronized the nodes using the FTM algorithm.

2. Clock Model

There are many models to represent a physical clock: for example, Varnum (1983) proposed a simple stochastic model of a physical clock. In this paper, we used the geometric model of Figure 1, where the clocks are represented by straight lines. In this model the effects of fluctuation (jitter) will be discarded. More information about this model can be found in Oliveira Junior (2010).

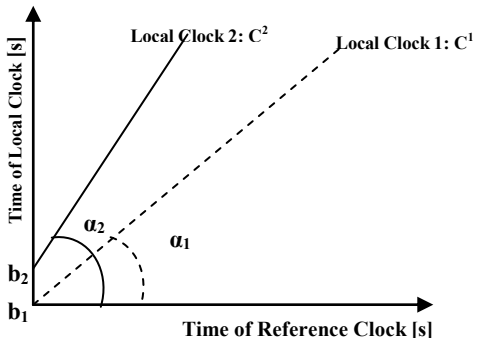


Figure 1. Geometric Clock Model.

In Figure 1, two lines are drawn, one representing the local clock 2, C^2, and other representing the local clock 1, C^1. Clock 2 has an initial offset and drift in relation to clock 1. The horizontal axis represents the time of the reference clock used to measure the time of both clocks. The vertical axis represents the time of local clocks. The clock model of Figure 1 is expressed by:

$$\Delta C_k(t) = C_k^2 - C_k^1 = (a_k^2 - a_k^1)(t_k^R) + (b_k^2 - b_k^1) = \Delta a_k t_k^R + \Delta b_k \quad (1)$$

In Equation 1, C^1 represents clock 1 and C^2 represents clock 2. The coefficient 'b' is the initial offset and the coefficient 'a' represents the inclination of a straight line. Deriving Equation 1 with respect to time, we have:

$$\dot{\Delta C}_k(t) = \Delta a_k \quad (2)$$

Equation 2 shows the rate of change of the difference between straight lines 2 and 1; we can conclude that Δa is the drift of clock 2 with respect to clock 1. With the discrete geometric model of the clock, we will set the model measures. This is given by:

$$\Delta a_k^{2,1} = \frac{b_k^2 - b_{k-1}^2}{b_k^1 - b_{k-1}^1} \quad (3)$$

$$\Delta b_k = b_k^2 - b_k^1 \tag{4}$$

$$t_k^R = kg^R \tag{5}$$

$$y_k = \Delta b_k + \Delta a_k^{2,1} t_k^R \tag{6}$$

Where Equation 3 represents the measurement of drift of clock 2 with respect to clock 1, Equation 4 represents the measurement of the instantaneous offset, Equation 5 represents the measurement of the reference clock where the g^r is the resolution of such clock, and Equation 6 represents the model of discrete measures.

3. FTM Algorithm

The FTM (Fault-Tolerant Mid-Point) algorithm, also known as Welch-Lynch algorithm provides fault tolerance for Byzantine clock synchronization of distributed systems (Lundelius & Lynch, 1984). To ensure that all nodes have a consistent view of time, we need to re-synchronize the clocks regularly (periodically). For this the algorithm follows a logical sequence shown in the flowchart of Figure 2. Each node applies this sequence with the objective of reaching a correction term. With this correction term, the deviations caused by the drift of the clocks are adjusted so that all system clocks are within a certain precision. In such flowchart:

Number 1 in Figure 2 indicates a loop condition. This condition means that if the local clock time of the node is equal to the time of re-synchronization, then synchronization has to start somewhere. R_{int} is the predetermined period of re-synchronization and k is its instant. Number 2 in Figure 2 indicates where one has to read data of the databus, meaning that the local clocks exchange information among themselves, that is, all clocks send a broadcast with the timestamp of its own clock. Number 3 in Figure 2 indicates the ordering of data. Each node sees only its row of the matrix with the timestamps values forming a vector of values A. This vector is sorted in ascending order. At Number 4 of Figure 2 the Welch-Lynch algorithm calculates the convergence function, after ordering the data in ascending order. It discards the highest and lowest value of A; so, it is the arithmetic mean of the highest and lowest value of the remaining elements in the vector, according to Equation 7. Equation 8 calculates de adjustment function with the value of the convergence function.

$$cfn(A) = \frac{A[f+1] + A[n-f]}{2} \tag{7}$$

$$Adj = kR_{int} + \delta - cfn(A) \tag{8}$$

And finally at Number 5 of Figure 2, the virtual clock fixes its value by adding their adjustment function.

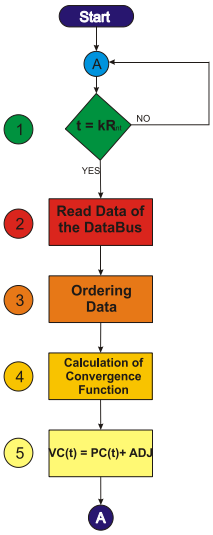


Figure 2. Flowchart of FTM Algorithm.

4. Design of Simulations

In the TrueTime/Matlab/Simulink environment, we simulated two sets of controls, i.e., a system with two control loops connected by a common databus. The controller used was a PID. The actuator/plant is described as continuous in time, according to the following transfer function:

$$G(s) = \frac{1000}{s(s+1)} \tag{9}$$

The controller and sensor nodes had logical clocks given by the virtual computer of the TrueTime Kernel; and they used the databus to exchange data among them and from their clocks. The actuators/plants used the databus only to receive the control data. Each control node implemented a periodic control task and a clock synchronization task. Each sensor node implemented a task for sending the measured data to the controller; and a task for clock synchronization. Each actuator/plant was activated by events when the control task arrived in the actuator by the databus. All nodes had an interruption caused by data arrived from the databus. The model of the simulated control system is given at Figure 3.

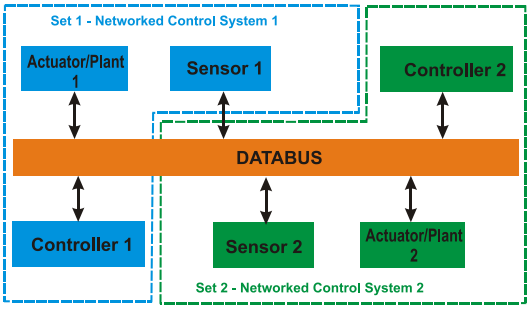


Figure 3. Model of the Simulated Control System.

For these simulations we used the TDMA philosophy for the communication protocols. The communication network is configured by the TrueTime Toolbox.

5. Results

We simulated 2 control subsystems sharing the same databus with a TDMA protocol. The sensors, actuators/plants and controllers are connected via the databus. Errors of clock were inserted in Sensor 1. The objective was to synchronize Controller 1, Sensor 1, Controller 2 and Sensor 2 together; and to compare the cases. In these simulations we varied: the clock drift in Figure 4; and the initial offset of node 2 (Sensor 1) in Figure 5. The databus used a communication protocol with TDMA. The drift rate applied was 1%. This may seem enormous in terrestrial and controlled environments. But, in this paper, we are interested in space environments, where variations of temperature of -10 a 50 degrees, can cause drift rates of up to 1% due to the extreme sensitivity of the quartz clock to temperature, according to Henderson et. al. (2000). In Figure 4, we observe that the FTM algorithm efficiently synchronized the NCS. In Figure 5, the synchronization algorithm did not correct the effects of clock and the system became temporarily unstable. This occurred because the FTM algorithm supposes that all nodes are initially synchronized.

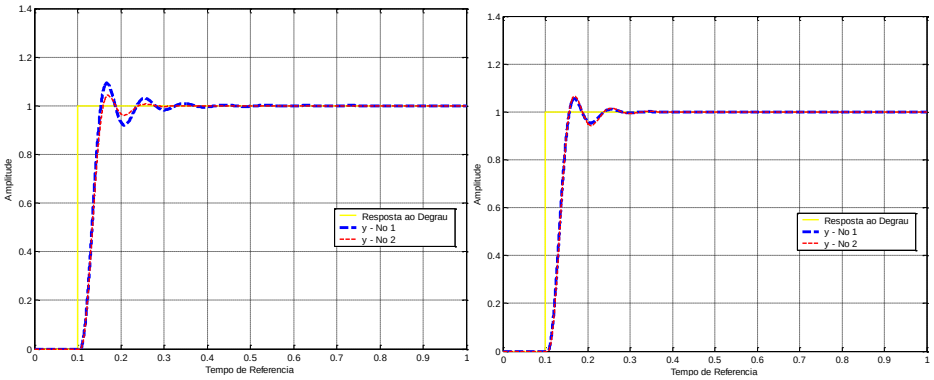


Figure 4. a) Step Response with 1% of drift rate. b) Step Response Synchronized.

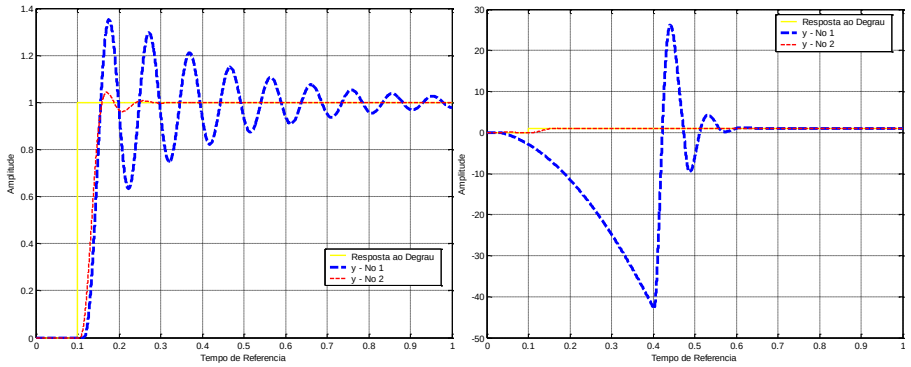


Figure 5. a) Step Response with 0.39 s initial offset. b) Step Response Synchronized.

The temporary instability in the simulation of Figure 5 occurred because sensor 1 starts with an initial positive offset. The virtual computer of sensor 1 identifies overdue tasks and it begins to perform all tasks delayed. The task of sensor 1 is to measure and submit data via the databus to the controlller. By sending the data via the TDMA databus, the volume of tasks is much larger than the slot of transmission time of sensor 1. Thus the tasks of a sensor will be suspended when its slot of transmission time is exceeded. This

suspension of tasks generates a large delay. Figure 6a shows the corresponding scheduler of the TDMA communication network, where we can observe this phenomenon: in blue we observe controller 1, in red we observe sensor 1, in green we observe controller 2 and in brown we observe sensor 2. The delay generated by the TDMA network is larger enough that the control system reverses its phase.

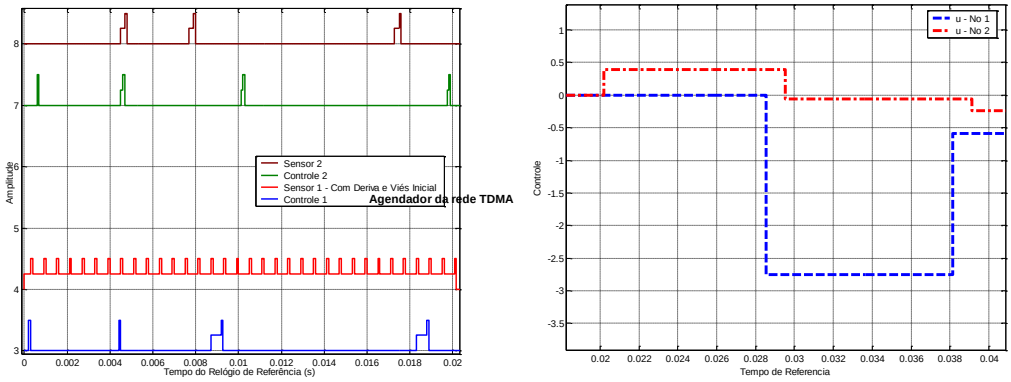


Figure 6. a) Scheduler of Databus Synchronized. b) Aproximation of Figure 5b.

By reversing the phase, the system becomes temporarily unstable, as shown in Figure 6b. For sensor 1, all tasks have already been done. Due to that, until a new task of measurement enters, the system remains unstable. When entering a new task, the system recovers from its instability.

6. Conclusions

This work still is in progress. But the preliminary results suggests that: 1) the TDMA protocol is more susceptible to errors in the initial offset than to errors in the clock drift; 2) the FTM algorithm corrects the clock drift error better than the initial offset error; 3) In a NCS with a TDMA protocol, a fault in the time management can turn the control laws temporarily unstable.

References

Henderson, W.; Kendall, D.; Robson, A. Accounting for Clock Frequency Variation in the Analysis of Distributed Factory Control Systems. Proceedings of the 2000 IEEE International Workshop on Factory Communication Systems, Porto, Portugal, 2000.

Kopetz, H. Real Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, Holland, 1997.

Lundelius, J.; Lynch, N. A New Fault-Tolerant Algorithm for Clock Synchronization. Proceedings of the third annual ACM Symposium on Principles of Distributed Computing, Vancouver, Canada, 1984.

Oliveira Junior, E. M. Study of FTM, FTA and Kalman Filter (KF) Algorithms for the Clock Synchronization and their Influences on a Control System. 2010. 457 p. Dissertation – INPE, São José dos Campos, Brazil, 2010.

Varnum, F. B. Kalman Filtering with a Two-State Clock Model. Proceedings of the 15th Annual Precise Time and Time Interval (PTTI) Application and Planning Meeting, Meeting, Washignton, USA, 1983.

Analysis, Design and Simulation of a Reconfigurable Control Architecture for the Contingency Mode of the Multimission Platform

Jairo C. Amaral[1], Marcelo L. de O. e Souza[1]

[1]Division of Space Mechanics and Control – National Institute for Space Resarch (INPE)
Av. dos Astronautas, 1.758 Jd. Granja – São José dos Campos – SP – Brazil
jairo.amaral@dem.inpe.br, marcelo@dem.inpe.br

Abstract*. Currently, Reconfigurable Control Systems (RCS) are becoming more and more widespread and studied in the aerospace community. This work presents the analysis, design and simulation of reconfigurable control architecture for the Contingency Mode of the Multimission Platform (MMP), a generic service module currently under design at INPE. Its* ***embedded real time control system*** *can be switched among nine main Modes of Operation, according to ground commands or information (mainly alarms) coming from the control system. The implementation follows the specifications when they were found; when specifications could not be found, they were designed ad hoc. The tests are based in simulations with the MATRIXx/SystemBuild software. They focus mainly on the worst cases that the satellite is supposed to endure in its mission; this can be during modes, or during transitions between modes and submodes.*

1. Introduction

Control systems of satellites, aircrafts, automobiles, traffic controls, etc., are becoming increasingly complex and/or highly integrated due to their use of computers networked via communication devices and protocols working in real time. In these systems, the reconfiguration of control modes and law are increasingly being used, to meet diverse phases of the mission or even faulty operations. This should happen smoothly, with fast and minimum transients and stable and precise steady states; otherwise the controls could enter strange modes, degrade performance and even reach instability.

This work presents the analysis, design and simulation of the reconfigurable control architecture for the Contingency Mode of the Multimission Platform (MMP). The MMP is a generic service module currently under design at INPE. Its **embedded real time control system** can be switched among nine main Modes of Operation and other submodes, according to ground commands or information (mainly alarms) coming from the control system. The MMP can aquire one and three axes stabilization in generic attitudes, with actuators including magnetotorquers, thrusters and reaction wheels.

The implementation followed the specifications when they were found; when specifications could not be found, they were designed ad hoc. The MMP enters in the Contingency Mode right after the launcher separation, or if there is an emergency, according to the following sequence: it stops any rotation using magnetotorquers; opens

the solar pannels, if it is not done yet; points them to the Sun using propulsors; and acquires gyroscopic rigidity using reaction wheels. If the stopping with magnetotorquers is not achieved in a predetermined time, the MMP will enter in a submode for trying to achieve it with propulsors. As there is propulsor control for only two axes, it will also wait for the best moment to make a maneuver.

1.1 State of Art Comparison

Currently, Reconfigurable Control Systems (RCS) are becoming more and more widespread and studied in the aerospace community. Examples related to our work are:

The Oersted [Boegh and Blanke,1997] is a Dannish microsatellite of aproximatelly 60 kg launched in 1997. Its main objective is to collect measures of Earth`s magnetic field and high energy particles in this vicinity. Being small and low costing, it was not possible to deal with failures adding redundancies, so the integrity of the attitude control needed to be waranted by an automous supervising system. The Oersted`s architecture needed to accommodate the implementation of many functions, and they were implemented in a supervisory structure of three levels: an inferior with I/O of the control net, a second level with algorithms for detection and acocomodaton of faults, and a third level with supervisory logic. The many control modes are consideed separately, while the supervisory level needs to choose the correct mode for each situation.

The Open Control Plataform [Wills et al. 2000] is an open software architecture developed at Georgia Tech for distributed, reconfigurable, hierarchical control systems. Complex control systems for autonomous vehicles require components that are often supported on different types of hardware platforms and operating systems. They must often interact in a distributed environment, and at the same time, the configuration and integration must be flexible enough to allow rapid online adaptation to react to unpredictable events. The specifc drive of this project was to support the autonomous control of unmanned vehicles with capacity of vertical take off and landing (helicopter).

1.2 Operation Modes

Due to the diversity of conditions that the MMP will face during its entire life, there is a separation in many Operational Modes, where each mode is defined by the environment and conditions in which the satellite will be. Those modes are shown in Figure 1, and are divided in two major groups, defined by the environment where the satellite is:

Ground Modes:

- Off Mode (OFM). In this mode, all the equipments are shut off (with disconnected batteries). This mode is to storage and transport.
- Integration and Test Mode (ITM). This mode is used during the assembly and integration tests, or in the launch platform. During the assembly and integration, all the tests are done, while at the launch platform, only the tests of functional verification will be done.

Flight Modes:

- Start Mode (STM). This mode can be used on the ground, during the flight phase, and at any time during the useful life of the satellite.

- Contingency Mode (COM). The objective of this mode is to automatically take the satellite and its payload from STM to a safe mode after the launcher separation, or in case of an anomaly.
- Fine Navigation Mode (FNM). This mode is used to acquisition of attitude, position and time in a precise way to allow the transition from the COM to the nominal mode.
- Nominal Mode (NOM). This is the operational mode of the satellite, where the payload can perform its objectives. In this mode the wheel desaturation with magnetic actuators also happens.
- Wheel Desaturation Mode with Thrusters (WDM). In this mode the reaction wheel desaturation is done by the action of thrusters. This proceeding aims to reduce the angular speed of the wheels back to nominal levels of operation.
- Orbit Correction Mode (OCM). It is used to execute orbital maneuvers on the orbital plane, or from it.
- Orbit Correction Mode Backup (OCMB). If one of the thrusters fails, the orbital maneuvers will be executed with only two of the symmetric thrusters, to minimize the disturbing torques.

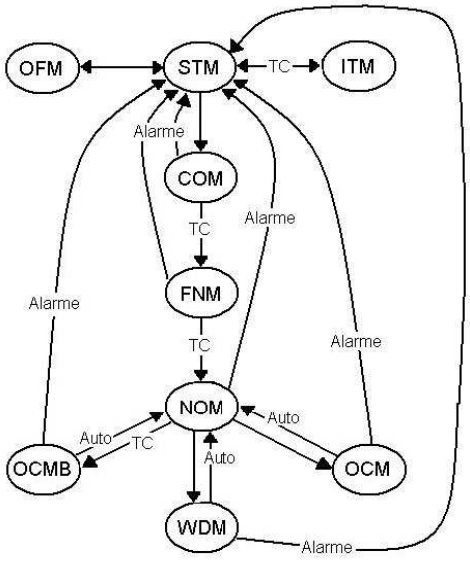


Figure 1. Transition logic of the operation modes of the MMP.

Source: INPE (2001).

2. Implementation

According to the specifications, the Contingency Mode is charged of executing a stop with magnetotorquers and the solar pointing in two axes. It may be accessed normally by the Start Mode, and by any other mode in case of an alarm.

We could not find in the available literature about the MMP a control law for the magnetotorquers in the Contingency Mode. Thus we choose a control law derived from the work of [Prudêncio, 2000] for the satellite SACI-1, which in its turn, is based in the work of [Shigehara, 1972] and is analogous to the same law used in the dessaturation. As there is no need of a specific pointing, all the magnetotorquers use a version of this control policy for reduction of rotation velocity. If the magnetotorquers cannot stop the satellite in a specified time, it enters in a submode for trying a stop with thrusters. In any case, when the satellite accquires an angular speed under 0.2 degrees/second, it will open its pannels and execute a fine stop with reaction wheels.

A law for the solar pointing was not found in the available literature about the MMP. Therefore, the law adopted was for two separated rotations: the first around the y-axis, and the second around the x-axis. Each one is a bang-bang control law.

The control signal is converted in polar coordinates, so that the poles would be in the y axis and the vector z would be between (0, π/2) rad. It activates the propulsors of the y axis so that the rotation of the satellite would be reduced to zero. The point where the torque direction is reversed is obtained by the Torricelli equation. As the Sun has a slow aparent movement, and the pointing does not need to be very precise, the signal which indicates the angular speed comes from the inertial unit in this model.

The transition between modes and submodes are controlled by a state machine, which enables different control laws accordingly. It is fed with sensors and outputs from other control blocks.

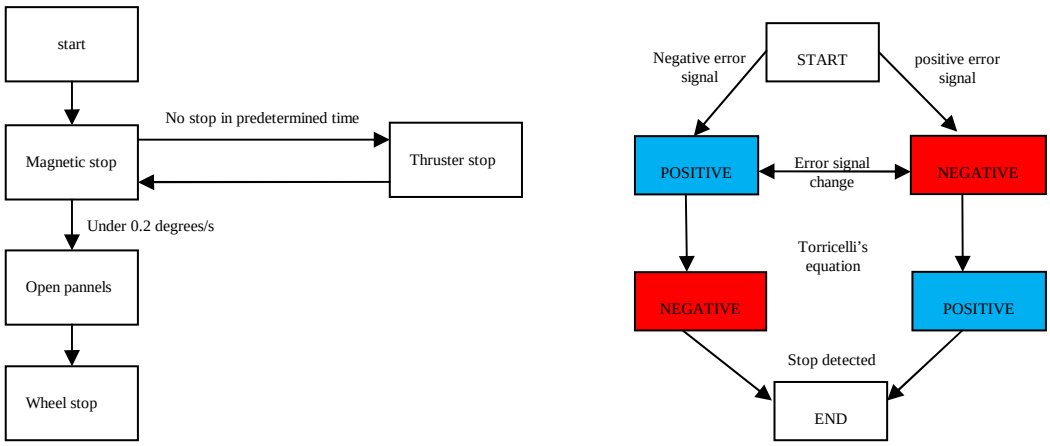


Figure 2. Block diagram of the detumbling (left); and block diagram of an axis alignment with thrusters (right).

3. Tests

The tests are based in simulations with the MATRIXx/SystemBuild software, from National Instruments, which supports developers with tools to model, analyse and test a control system. Their general objective is to check if the MMP satisfies the design requirements, but our main interest is to check the stability during the transition of control laws from modes and submodes.

Even if each mode of operation has a stable control law, this conclusion cannot be extended for the resulting system when the subsystems are not linear. This is known as the **Problem of Hyperstability**, and might turn the validation of reconfigurable control systems extremely difficult. There are analytical approaches for such, but their practical applications are limited. As an alternative, we focused in numerical simulations of the worst cases which the satellite is supposed to endure in its mission; it might be during modes of operation or during their transitions. If the results are satisfactory, it will be reasonable to conclude that they will do so in the other cases.

The plant includes simulations such as orbit propagation, air drag, and variations in inertia moment, and it was reused from [Amaral 2008]. We expect to show that the MMP is able to satisfy the official requirements found and the ad hoc requirements.

4. Results

The two example cases considered a circular orbit with 7000 km of radius and 45 degrees of inclination.

Figure 3 shows the module of the angular speed being reduced by the magnetotorquers from 1.7 degrees/s to under 0.2 degrees/s. This was achieved in aproximately 2 hours.

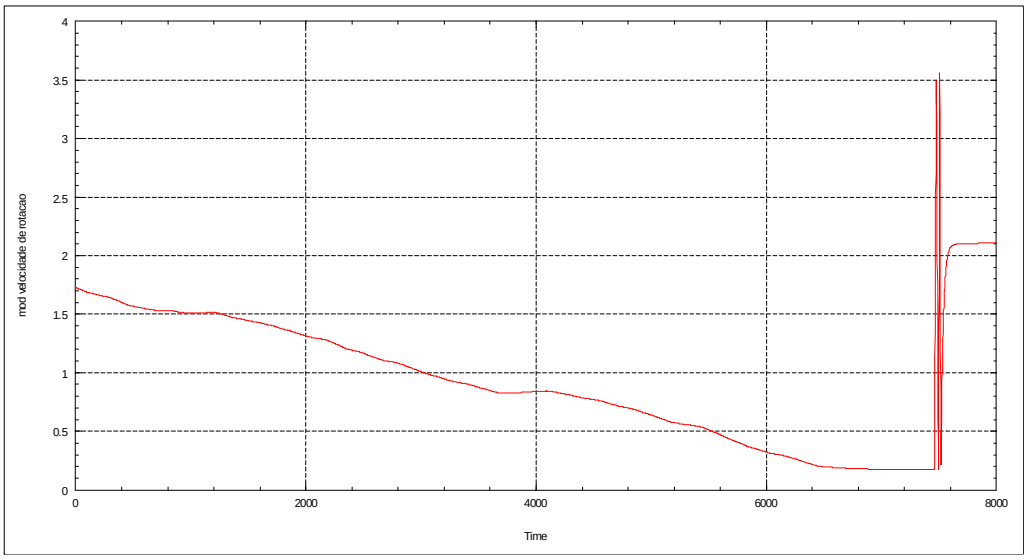


Figure 3. Module of angular speed during magnetic stop.

Figure 4 shows the same scenario, but focuses in the solar pointing stage. After the stop, the thrusters are activated, one axis at a time, for executing two rotations towards the Sun. Then the satellite acquires gyroscopic rigidity using a reaction wheel.

These two examples transit through all the submodes detailed in each diagram of Figure 2. Transits like the magnetic stop, pannels oppening and reaction wheel stop were not much noticeable in the angular speed in Figure 3 [6000 s – 7500 s]; but the inversion of thrusters' control signal are clearly seen as the two spikes in Figure 4 [7450 s -7550 s].

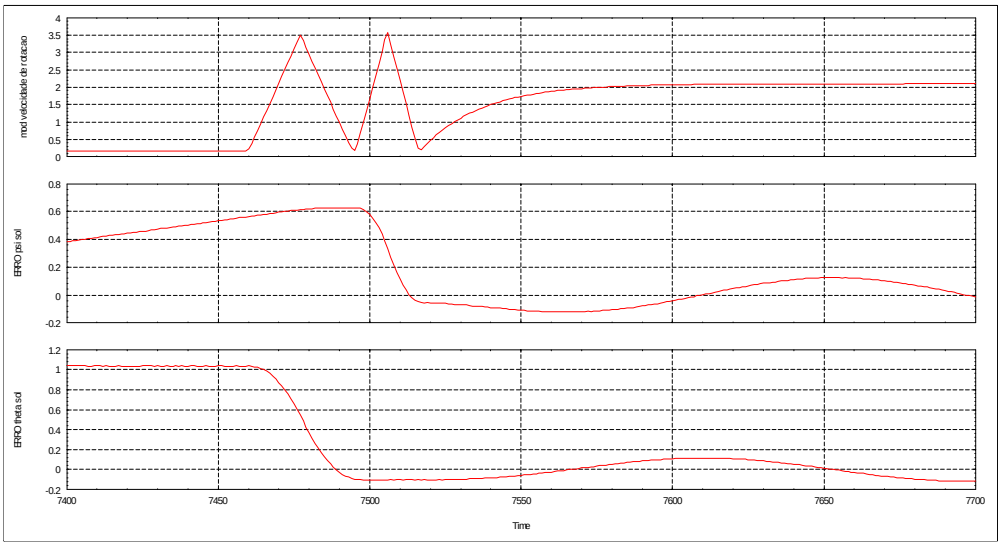


Figure 4. Module of angular speed and errors of solar pointing.

5. Conclusions and Future Objectives

The results until now show that the implementation satisfies the project requirements. For low inclinations of orbit, the detumbling with magnetotorquers loose effectivity and fails to stop in the predetermined time, but the submode with thrusters was able to force a stop. Besides, the solar pointing submode worked well even when the initial position was opposite to the Sun. We expect to estimate in what kind of orbits, and in what rotation axis, the magnetotorquers can stop the rotation in a given time. We also expect to identify what situations could delay or prevent a solar pointing after the stop. Later, we intend to investigate other transients in the control system caused by the switching between modes of operation.

References

AMARAL, J. C. Análise, Projeto e Simulação de uma Arquitetura de Controle Reconfigurável para a Plataforma MultiMissão. Dissertação (Mestrado em ETE/CMC) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2008.

BØGH, S. A., BLANKE, M. Fault-Tolerant Control — A Case Study of the Ørsted Satellite, Compenhagen, DEN, 1997.

INPE. A822000-DPK-01/D5a – Multimission Platform Data Package for System Requirements Review (SRR). 2001. São José dos Campos – SP.

PRUDENCIO, S. V. Simulação Digital em Tempo Real de um Sistema de Controle de Atitude Magnético Autônomo de um Satélite. Dissertação (Mestrado em ETE/CMC) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2000.

SHIGEHARA, M. Geomagnetic Attitude Control of an Axisymmetric Spinning Satellite. Journal of Spacecraft and Rockets, 9(6):391-398, 1972.

WILLS, L; SANDER, S; KANNAN, S; KAHN, A; PRASAD J V R; SCHRAGE, D. An Open Control Platform for Reconfigurable, Distribute, Hierarchical Control Systems. — Proceedings of the Digital Avionics Systems Conference, Philadelphia, PA, October 2000.

Preliminary Results of Global Time Petri Net Analysis Applied to Embedded Software Prototyping

Leticia Mara Peres[1], **Eduardo Todt**[1], **Luis Allan Kunzle**[1]

[1]Computer Science Department, UFPR, Curitiba, Brazil

lmperes@inf.ufpr.br,todt@inf.ufpr.br,kunzle@inf.ufpr.br

Abstract. *We propose in this paper an approach based on Time Petri Nets (TPN) to analyze time aspects and to schedule embedded software during the prototyping activities. Firstly, a TPN that represents the interaction between system tasks is modeled using Petri net design patterns. Secondly, a class graph is built with the Global Time Method (GTM) and interval algebra operations. Finally, firing sequences on GTM graph is found. The firing sequences represent behavior itineraries of software, and their global time based on cyclic scheduling, fixed priority and earliest deadline first scheduling policies. The example shows simple and precise time analysis of behavior itineraries which use a powerful theory based on interval algebra, different of conventional models.*

1. Introduction

Real time embedded systems are constrained about functionalities and resources. In these systems, time constraints are so important as functional constraints.

Petri nets (PN) and their algebraic properties are used to model and to analyze systems involving parallelism, concurrency and synchronization. Several extensions of the basic formalism have been proposed to increase their power of modeling. In this work we are interested in Time Petri nets (TPN), where quantitative time restrictions can be considered [1]. The classic technique of analysis is based in an enumerative method to generate the reachable state space of a TPN [2]. This method finds the relative time interval which the system remains in each state, being therefore efficient to verify net properties and to analyze time constraints relative to a given class. This method itself does not present global time information of the net behavior since its initial marking. In order to obtain more accurate results, Dill [3] describes data structures Difference Bounds Matrix (DBM) and proposes a method for using delay information in state graph verification of finite state concurrent systems. These structures have been used in association with model checking and timed automata approaches [4], [5], [6]. We propose a global time alternative method based on interval algebra.

TPN are used in different applications of embedded system verification, scheduling and synthesis methods. Cortes et al. [4] define a method of embedded software modeling and verification using PRES+, a type of TPN. Lime and Roux [5] propose SETPN to model real-time systems, especially embedded systems. They present a set of PN design patterns to model tasks with preemptive scheduling. In [6], Lime and Roux use their scheduling TPN design patterns of [5] to model tasks and deal with Fixed Priority and Earliest Deadline First policies.

We present in this paper the Global Time Method (GTM) which is an approach to generate a new type of class graph for TPN. This method finds relative and global time information about the current state of the net. The global time information is correct for both limits of time interval of any class, even when the net represents concurrency among events [7]. The remainder of this paper is organized as follows. The Section 2 defines basic concepts of interval operations, TPN and class graph. Section 3 establishes the class graph based on GTM, and presents an example of application. Section 4 presents an application of GTM on the context of embedded software scheduling, while Section 5 concludes the article.

2. Basic Concepts

Let two rational numbers, a and b, such that $a \leq b$. We denote $[a, b]$ as the set $\{x \in R : a \leq x \leq b\}$, defined as closed interval from a to b. An interval $[c, d]$ is denoted as not proper when $d < c$, with c and d rationals. Given the intervals $[a, b]$ and $[c, d]$, proper or not proper, we define the following operations: $[a, b]+[c, d] = [a+c, b+d]$; $[a, b]-[c, d] = [max\{0, a - d\}, max\{0, b - c\}]$; $[a, b] \ominus [c, d] = [max\{0, a - c\}, max\{0, b - d\}]$. The interval subtraction $\ominus$ is used to adjust time coefficients which can be represented by not proper time intervals.

A Time Petri net (TPN) is a tuple $TPN = (P, T, Pre, Post, M_0, I)$ [8], where:

- P is a finite set of places, $p \in P$,
- T is a finite set of transitions, $t \in T$,
- Pre is an input application such as $Pre : (P \times T) \rightarrow \mathbb{N}$,
- $Post$ is an output application such as $Post : (P \times T) \rightarrow \mathbb{N}$,
- M_0 is the initial marking, and
- $I : T \rightarrow (\mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\}))$ such as $(t, e(t)) \in I$ and $e(t) = [a, b]$, where a and b are positive rational numbers.

A marking is an assignment of marks to places of the net. The marking of a place $p \in P$ is denoted $M(p)$. A TPN has one static time interval $e(t) = [a, b]$, with $a \leq b$, associated to each transition $t \in T$. The limits a and b represent, respectively, the earliest and the latest possible firing time of transition t, counted from the instant when t is enabled ($\forall p, M(p) \geq Pre(p, t)$). When t fires in a marking M_{k-1} a new marking of the net is given by marking $M_k = M_{k-1} + Post(t) - Pre(t)$ ($c_{k-1}[t_f > c_k$).

A state class $c_k = (M_k, W_k)$, where M_k is the current marking of the TPN obtained by the firing of a transition t. W_k is the set of time information for this class. A transition t, enabled in a class c_k, is a *persistent* transition in c_k if t was enabled in a class c_{k-1} and t did not fired in c_{k-1}. A transition t, enabled in c_k, is *newly enabled* if t was not enabled in class c_{k-1} or, the firing of t originated the class c_k and it was re-enabled in c_k.

The state class graph is a directed graph where each node is a state class and each arch is labeled with one transition. The root node of class graph is the start class c_k with level $k = 0$ and has the initial marking M_0 of TPN.

3. Global time method (GTM)

The information set W_k of class c_k has two types of time information: relative and global. The relative time information is the accumulated time for each transition since its enabling in class c_k. Global time information refers to the accumulated time since the initial marking (class c_0).

Let $r_k(t_i)$ a relative time interval of a transition t_i calculated in a class c_k such that $c_{k-1}[t_f > c_k$, and defined as:

$$r_k(t_i) = \begin{cases} e(t_i) & \text{if } t_i \text{ is } \textit{newly enabled} \text{ in } c_k \\ r_{k-1}(t_i) - r_{k-1}(t_f) & \text{if } t_i \text{ is } \textit{persistent} \text{ in } c_k \end{cases}$$

The relative time interval $r_k(t_i)$ of each enabled transition t_i of a class c_k is used to identify which ones are fireable. A transition t_f with $r_k(t_f) = [a_f, b_f]$ is fireable in c_k if, and only if, t_f is enabled in c_k and there is not another transition t_i with $r_k(t_i) = [a_i, b_i]$ enabled in c_k such that $b_i < a_f$.

The persistence adjustment coefficient $ac_k(t_i)$ of an enabled transition t_i in a class c_k such that $c_{k-1}[t_f > c_k$, is defined as:

$$ac_k(t_i) = \begin{cases} r_{k-1}(t_i) \ominus r_{k-1}(t_f) & \text{if } t_i \text{ and } t_f \text{ are both newly enabled in } c_{k-1} \\ ac_{k-1}(t_i) \ominus r_{k-1}(t_f) & \text{if } t_i \text{ is persistent and } t_f \text{ is newly enabled both in } c_{k-1} \\ r_{k-1}(t_i) \ominus ac_{k-1}(t_f) & \text{if } t_i \text{ is newly enabled and } t_f \text{ is persistent both in } c_{k-1} \\ ac_{k-1}(t_i) \ominus ac_{k-1}(t_f) & \text{if } t_i \text{ and } t_f \text{ are both persistent in } c_{k-1} \end{cases}$$

The persistence adjustment coefficient $ac_k(t_i)$ prevents the increase of imprecision to computing the global time.

The global time interval $g_k(t_i)$ of a fireable transition t_i in a class c_k such that $c_{k-1}[t_f > c_k$ is:

$$g_k(t_i) = \begin{cases} e(t_i) & \text{if } k = 0, \text{ i.e. if is the initial class} \\ g_{k-1}(t_f) + r_k(t_i) & \text{if } k \neq 0 \text{ and } t_i \text{ is newly enabled in } c_k \\ g_{k-1}(t_f) + ac_k(t_i) & \text{if } k \neq 0 \text{ and } t_i \text{ is persistent in } c_k \end{cases}$$

The global time interval $g_k(t_i)$ is counted from the initial marking until the firing instant of t_i in class c_k.

Let t_f be the fired transition in a class c_k such that $c_k[t_f > c_{k+1}$. The upper bound of global time interval $g_k(t_f) = [a_f, y]$ of fired transition t_f, must be adjusted by the lowest upper bound of intervals calculated to all t_i in class c_k, where $y = min\{b_i \mid g_k(t_i) = [a_i, b_i], \forall t_i$ fireable in $c_k\}$.

The successive firing of two or more transitions in a TPN from any class c_k to any other class c_{k+n}, also called firing sequence, is represented by $c_k[s > c_{k+n}$. The global time of a firing sequence s of $c_0[s > c_k$ is the resulting of global time interval $g_{k-1}(t_f)$, being t_f the last transition fired to reach class c_k, that is, $c_{k-1}[t_f > c_k$.

The class graph can express the staying time in each class, i.e., how long the system remains in the state represented by the class. The staying time of a net in a certain reachable class c_k is given by: $i_k = [x, y]$, where $x = min\{a_i \mid r_k(t_i) = [a_i, b_i]\}$ and $y = min\{b_i \mid r_k(t_i) = [a_i, b_i]\}$, $\forall t_i$ fireable in c_k.

4. Application

We based the application of GTM on the work of Lime and Roux, 2009 [6]. It defines a special TPN with scheduling layer and, among other things, allows to map each place of

the net to a task. We propose to use parts of this layer to model TPN according design patterns of work of Lime and Roux, 2003 [5], associating tasks to transitions and places of the net. Then, we generate the GTM state class graph and analyze firing sequences that satisfies "Earliest Deadline First" and "Fixed Priority" scheduling policies.

Let, according [6]:

- $\tau \in$ *Tasks*, being *Tasks* the set of tasks of the system, where there is no task migration between processors.
- *Sched:Procs* $\mapsto \{$*FP,EDF*$\}$ the function that maps a processor to a scheduling policy, being FP "Fixed Priority" and EDF "Earliest Deadline First" ;
- Π*: Tasks* $\mapsto$ *Procs* the function that maps a task to its processor;
- ϖ*: Tasks* $\mapsto \mathbb{N}$, for $Sched(\Pi(\tau)) = FP$, gives the priority of the task on the processor;
- δ*: Tasks* $\mapsto (Q^+ \times (Q^+ \cup \{\infty\}))$, for $Sched(\Pi(\tau)) = EDF$, gives the deadline interval of the task on relative to its activation time.

In order to map each place of the TPN to a task, we use the function $\gamma : P \mapsto$ Tasks $\cup\{\phi\}$, where ϕ denotes that the place is not mapped to any real task. We, as in [6], assume that for each transition, there is at most one place p such that $p \in Pre(t)$ and $\gamma(p) \neq \phi$. If $\forall p \in Pre(t)$, $\gamma(p) = \phi$, then t is not bound to any real task and we say that it is *part of* ϕ (denoted by $\gamma(t) = \phi$). Otherwise, for each transition t, we say that t *is part of* the task τ, and we denote it $t \in \tau$ if one of its input places is mapped to $\tau : t \in \tau \Leftrightarrow \exists p \in Pre(t)$, s.t. $\gamma(p) = \tau$. So, $\gamma(t)$ is the task s.t. $t \in \tau$.

As in [6], each task τ is thus modeled by a subnet of the TPN composed of places mapped to τ by γ and of transitions with static time, which are parts of τ. As in [6] we assume that at most one instance of each task is active at a given instant, which is expressed by the restriction that at most one place mapped to τ by γ is marked at a given instant. Let $B(\tau)$ be the set of transitions which *start* the task τ and similarly, let $E(\tau)$ be the set of transitions which terminate τ. These two sets are user-defined as part of the modeling phase. After TPN was modeled, we propose generate a GTM state class graph, as presented at section 3. The definitions of scheduling layer reflect on TPN and on each scheduling policy.

The mapping between scheduling policies and GTM graph defines a criterion for path enumeration on GTM state class graph, where the path is the firing sequence, satisfying some scheduling policy. We have established criteria for "Earliest Deadline First" and "Fixed Priority" scheduling policies. For *Sched:Procs* $\mapsto \{$*CE*$\}$, where CE is "Cyclic Executive", the TPN model represents only one task which is typically realized as an infinite loop in *main()*. Because *CE* has not a specific criterion in order to satisfy, this policy is achieved only by modeling TPN and it is not necessary formalize this function in relation to GTM state class graph enumeration.

Fixed Priority ($Sched(\Pi(\gamma(t))) = FP$**)**: After class graph building, each transition t_i has a priority of the task on the processor associated to it ($\varpi(t_i)$). Then, the function ϖ*: Tasks* $\mapsto \mathbb{N}$ guides the firing sequence enumeration. We choose the fireable transition which has the higher priority. At the end of this enumeration, we already have the total time for a firing sequence according GTM. In the case of tasks with the same priority at some point, one of these criterion can guide the firing sequence enumeration between the processes with the same priority: a FIFO choice; an earliest deadline first considering the

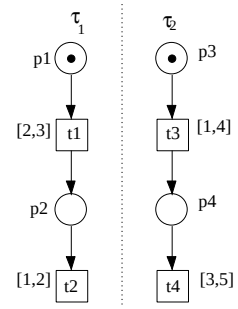


Figure 1. TPN of two tasks on one processor, from [5]

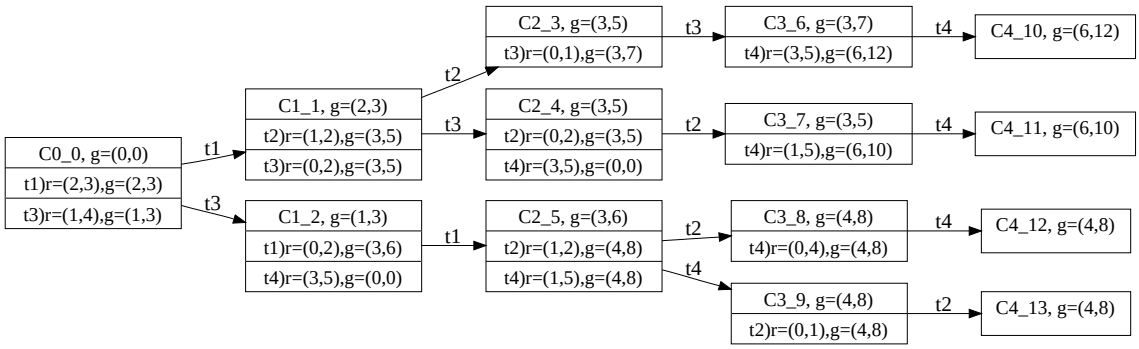


Figure 2. Class graph of TPN representing two tasks on one processor, of Fig. 1

static time $e(t_i)$ for each transition t_i, as we present in the following section; and a random choice.

Earliest Deadline First ($Sched(\Pi(\gamma(t))) = EDF$**)**: Another type of firing sequence can be enumerated on class graph, using the Earliest Deadline First scheduling policy. Then, the function δ*: Tasks* $\mapsto (Q^+ \times (Q^+ \cup \{\infty\}))$ guides this enumeration. Our criterion is to choose the transition which has the lower deadline given by $\delta(\tau)$ as following. Let $\delta(\tau)$ of a transition t_i calculated in a class c_k such that $c_{k-1}[t_f > c_k$, and defined as: $\delta(\tau) = LFT(r_k(t_i))$. The latest firing time (LFT) of the time $r_k(t_i)$ for each transition t_i is the guide for firing sequence enumeration. As the FP policy, at the end of enumeration already has the total time for a firing sequence.

Considering the TPN of Figure 1. The task τ_1 has priority $\varpi = 1$ and one preemption point. The task τ_2 has also one preemption point, but priority $\varpi = 2$. Then, $\varpi(t_1) = 1$, $\varpi(t_2) = 1$, $\varpi(t_3) = 2$ and $\varpi(t_4) = 2$. The TPN class graph according GTM is presented in the Figure 2. For $Sched(\Pi(\tau)) = FP$, the firing sequence is: t_3, t_1, t_4, t_2. It is interesting to note that t_1 is the only one fireable in the class $C2\_5$ (class C, at level 2, with unique identification 5), even t_4 being enable and t_3 being in the same task that t_4; t_4 executes after t_1 because t_4 has highest priority. The global time of this sequence is $g_{4\_13} = [4, 5]$. For $Sched(\Pi(\tau)) = EDF$, the firing sequence can be t_1, t_2, t_3, t_4, with global time $g_{4\_10} = [6, 9]$, or t_1, t_3, t_2, t_4, with global time $g_{4\_11} = [5, 9]$.

5. Conclusions

GTM avoids the imprecision increase in time information when analyzing transition firing sequences which represents the time interval of system behavior itineraries. This happens

when the modeled system presents many concurrent or persistent transitions. Also, GTM state classes describe intervals in both global, based on the simulation beginning, and relative, based on the class entry moment, time information. This increases the analysis power of our approach.

The essence our approach is to verifiy scheduling scenarios generated using GTM from TPN modeled using design patterns. This design patterns represent a set of tasks and their interactions as proposed by [5] and can be tasks on one processor, cyclic tasks synchronized *via* a semaphore, semaphore for mutual exclusion and CAN bus access.

The main contribution of our work is to apply the global time method to real-time software based on tasks with fixed priority and earliest deadline first. The main limitation of the proposed approach is the endless enumeration of classes in cyclic nets, according to the indefinite increase the global time. For the application in the context of prototyping this problem is currently treated by limiting the number of execution cycles of tasks, reflected by the class levels during the generation of graph. Even with this limitation the analysis is still useful as corresponds to the repetition of the initial critical instant for real-time systems based on cyclic tasks.

References

[1] P. Merlin, “A study of recoverability of computer systems,” Ph.D. dissertation, University of California IRVINE, 1974.

[2] B. Berthomieu and M. Menasche, “A state enumeration approach for analyzing time petri nets,” in *3rd European Workshop on Applications and Theory of Petri Nets*, Varenna, Italy, sep 1982.

[3] D. L. Dill, “Timing assumptions and verification of finite-state concurrent systems,” in *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*. London, UK: Springer-Verlag, 1990, pp. 197–212.

[4] L. A. Cortés, P. Eles, and Z. Peng, “Verification of embedded systems using a Petri net based representation,” in *ISSS ’00: Proceedings of the 13th international symposium on System synthesis*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 149–155.

[5] D. Lime and O. H. Roux, “Expressiveness and analysis of scheduling extended time Petrinets,” in *5th IFAC Int. Conf. on Fieldbus Systems and Applications,(FET’03)*. Aveiro, Portugal: Elsevier Science, Jul. 2003, pp. 193–202.

[6] ——, “Formal verification of real-time systems with preemptive scheduling,” *Journal of Real-Time Systems*, vol. 41, no. 2, pp. 118–151, 2009.

[7] E. A. Lima, R. Lüders, and L. A. Künzle, “Uma abordagem intervalar para a caracterização de intervalos de disparo em redes de petri temporais,” *SBA. Sociedade Brasileira de Automática*, vol. 19, no. 4, p. 379, 2008, in Portuguese, http://dx.doi.org/10.1590/S0103-17592008000400002.

[8] B. Berthomieu and M. Diaz, “Modeling and verification of time dependent systems using petri nets,” *IEEE Transactions on Software Engineering*, vol. 17, no. 3, pp. 259–273, March 1991.

Author Index