DARTSVIEW, A TOOLKIT FOR DARTS IN LABVIEW

Ngo Khanh Hieu[(1)], Grolleau Emmanuel[(2)]
(1) University of Technology, VNU-HCM
(2) Laboratory of Applied Computer Science, LISI-ENSMA, France

ABSTRACT: *DARTS (Design Approach for Real Time Systems) [4] is a software design method for real time systems. LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a graphical application development environment developed by National Instruments Corporation based on the dataflow representation of the "G" language [6][2]. LabVIEW is implicitly multithreaded and has high level functions for communication/synchronization, allowing it to be used as a programming language for control/command and soft real-time applications. In order to help a designer to develop a real-time application, we propose the library DARTSVIEW, which simplifies the passage from the conception of a "multitasking" application to the implementation [8]. One can use DARTSVIEW in different phases of the life cycle of real-time system software. The last version of DARTSVIEW, allows to define in XML several real-time programming normalized languages, and to generate a part of the code for different specific programming languages (Ada, POSIX 1003.1, VxWorks, OSEK/VDX, etc.). The flexibility introduced by the use of XML allows a designer also to generate some code targeting real-time scheduling analysis tools in order to achieve the temporal validation. The objective of this article is to present an overview of DARTSVIEW, a Toolkit for DARTS in LabVIEW, the role of DARTSVIEW in the software life-cycle, and some perspectives for the extensibility of this Toolkit in the future.*

Keywords: *DARTSVIEW, Toolkit for DARTS in LabVIEW.*

1. INTRODUCTION

The "concurrency" is one of the problems that we have to face frequently in real time systems. A concurrent system has many activities (or tasks) occurring in parallel. Usually, the order of incoming event is not predictable and these events may overlap [4]. So several tasks may handle the data-acquisition at different rates, some other tasks may be dedicated to the calculation of commands, and some others to the commands of several devices. When these activities (or tasks) synchronize and communicate, the conformance with rules of the mutual exclusion, of the synchronization, and of the communication is actually a key issue to be addressed:

- Mutual exclusion is the mechanism for ensuring that only one process at a time performs a specified action. Hence, it guarantees shared access to data (or resources) to the tasks.
- Synchronization is the control of the execution of two or more interacting processes so that they perform the same operation simultaneously. It allows to block a task until another one awakes it,
- Communication is a mechanism permitting the tasks to exchange the data.

DARTS (Design Approach for Real Time Systems) is a software design method, which emphasizes the decomposition of a real-time system into concurrent tasks and defines the interfaces between these tasks. In a DARTS diagram, each task is presented by a parallelogram (Fig. 1). It can be either a hardware task (released by an external event, such as an interrupt or a real-time clock), or a software task (released by another task) [8]. DARTS can be used as a

conception method for multitask systems (including real-time and control/command systems), since it focuses on the task decomposition, and thus is really close to the implementation process [2].

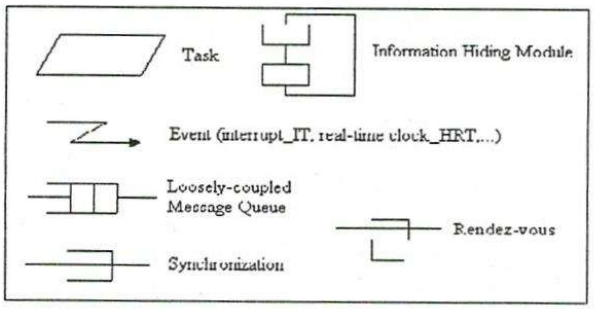

Figure 1. Elements of a DARTS diagram

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a graphical application development environment in the G language. LabVIEW is very well suited for data-acquisition, signal processing, and (soft real-time) control/command of process. The LabVIEW programming language is naturally parallel: when parts of the data flow are independent, the runtime can map them in several system threads. However, the difference between the notion of parallelism in LabVIEW and the semantics of dataflow associated in the G language does not allow to make a direct communication between them by a dataflow (in this case, the second function has to wait for the completion of the first one in order to start its execution). Therefore, following the work of [3], LabVIEW integrates intertask communication tools.

During several case studies, we realized that it was quite interesting to help the designer to create a multitask application in LabVIEW with DARTS based bricks provided as a Toolkit named DARTSVIEW, in order to get past of the classic multitask implementation process, and to focus on the behaviour of the tasks.

One of the important roles of DARTSVIEW is to help the designer to represent a DARTS diagram directly in LabVIEW. And in the software life-cycle like the classic V model given on Figure 2, the functional aspect of the system may be tested.

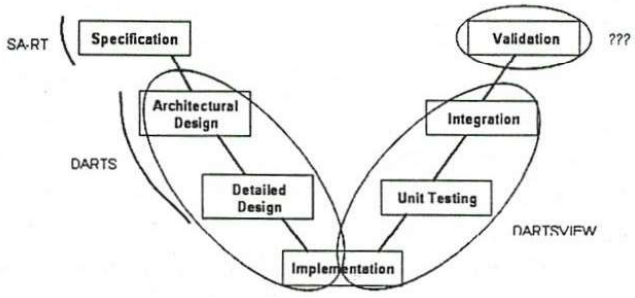

Figure 2. Software life-cycle in V

The temporal validation usually consists in a schedulability analysis based on a temporal model of the tasks [1]. The target programming language will likely be an imperative language (like C-based, Ada), and several tools can be used, Response-Time Analysis based, like MAST [MAST01], or building a feasible schedule, like PeNSMARTS [5]. The choice that has been made for DARTSVIEW was to use the flexibility of XML for the temporal validation in the

same way as it has been used to generate program in several programming standards: an XML model can be used to output a task model in the required format of several validation tools.

Therefore, in this case the software life-cycle based on a classic V model would be extended with the second V porting the workstation code whose parallel behavior has been tested on the workstation, and on the embedded target. The Figure 3 represents a software life-cycle in W, and the role of DARTSVIEW in this life-cycle.

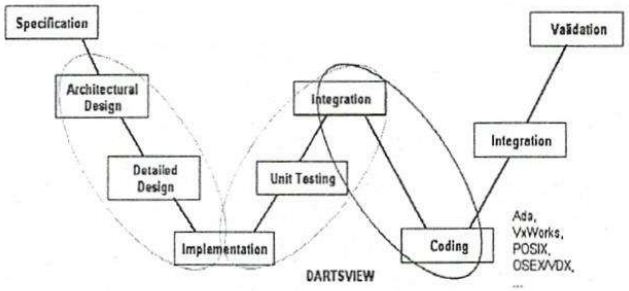

Figure 3. Software life-cycle in W

In the sequel, the following aspects of DARTSVIEW approach will be presented: section 2 presents the main multitasking LabVIEW concepts, and how these concepts are used in DARTSVIEW v7.1. The DARTSVIEW Toolkit and a case study are presented in section 3. Section 4 presents some perspectives.

2. DARTS AND MULTITASKING LABVIEW CONCEPTS

The implicit notion of parallelism inherent in LabVIEW allows multitask programming transparently: in fact, two loops running in parallel are mapped on different threads, and hence are executed in parallel. This characteristic permits to implement directly an abstraction of the tasks based on a DARTS representation. So, in LabVIEW a task DARTS can be simply modelled by an infinite While loop. However, the difference between the notion of parallelism in LabVIEW and the notion of data flows in G language does not allow exchanging the data directly between the tasks. Then this section presents how LabVIEW implements the interfaces between these tasks.

3. TASK SYNCHRONIZATION

Synchronizing two tasks consists in introducing a precedence constraint at a certain place of code: the destination task has to wait for an event sent by the source task in order to execute an action. In DARTS, synchronization between two tasks is presented on Fig. 4. For this type of synchronization, programming languages usually use a counting semaphore [2]. However the Semaphore tools proposed after LabVIEW v7.1 are bounded semaphore (LabVIEW requires that a semaphore can not have a count greater than its initial count). In order to solve this problem, we had to modify the implementation of the task synchronization in the way that firstly we decrease the count of a semaphore to 0, and then the release of semaphore must be verified to insure that the count is always smaller than or equal to the initial count.

4. LOOSELY-COUPLED COMMUNICATION

The communication is the transfer of data from one task to another. It is either based on a send and forget paradigm (see Fig. 4) when the size of the message queue is unbounded, or when a recent message replace the oldest one, or on a producer/consumer paradigm when the

size of the queue is bounded and no message can be lost (default behaviour of the message queue tool in LabVIEW).

We notice that after LabVIEW 7.1, the data is casted to a variant data type, and allows sending a message of any data type to the message queue. Retrieving the message consists in casting back the variant to the original data type. LabVIEW is checking the coherence when casting from a variant type to another data type, so the user can not make any typing mistake without being warned at runtime.

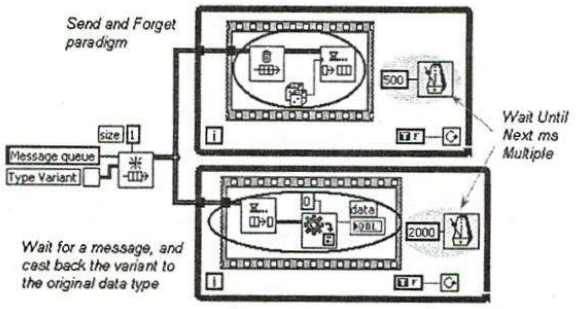

Figure 4. Message queue with replacement

In the send and forget paradigm (see Fig.4), the writing consists in emptying first, and then writing into the message queue in order to replace the oldest message in the case it would not have been read by the time the new message is sent. So the producer could send a message and then continue its execution without care of the reception of it in the consumer. This communication is very useful in the case of dense task producer; it allows to control the reception-rate in the task consumer.

Tightly-coupled message communication

The synchronous message communication, represented by the tightly-coupled message communication (Ada 83 rendez-vous), is a mechanism with which the producer sends a message to the consumer, and then immediately waits for a response (a message, or an event). In LabVIEW, this kind of communication can be implemented by two message queues (hence one for the producer, another for the consumer). A model of producer is presented on Figure 5: the producer firstly sends a message to the queue of the consumer, and then waits for the response sent by the consumer in order to continue its execution. The model of consumer is symmetric.

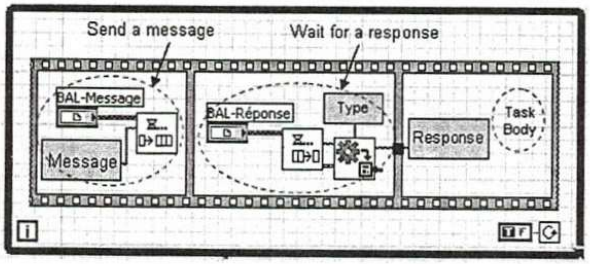

Figure 5. Producer, Tightly-coupled message communication with response

In the case of the tightly-couple message communication without response, the message queue for the consumer might be replaced by a tool of synchronization (i.e., a semaphore in LabVIEW) for the signal of the consumer to the producer when it receives successfully the message sent by the producer.

5. INFORMATION HIDING MODULE (IHM)

Information hiding is used as a criterion for encapsulating data stores. In DARTS, IHMs are used for hiding the contents and representation of data stores and state transition tables. When an IHM is accessed by more than one task, the access procedures must synchronize the access to the data [4]. The Figure 6 represents a simple implementation of IHM in LabVIEW. An IHM for encapsulating data stores of type "Reader/Writer" is compounded of two atomic operations, Read and Write, acting on an internal data. Note that we use a message queue of size 1 to store this data, and a counting semaphore of size 1 too in order to insure atomicity.

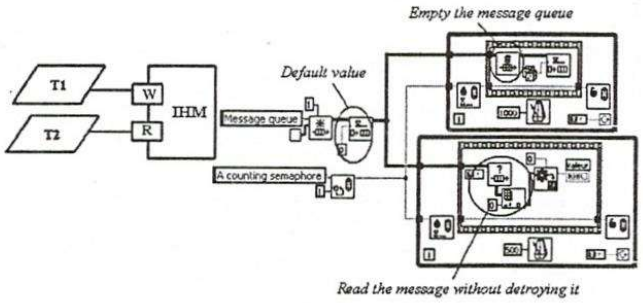

Figure 6. Communication by IHM

Hence, writing consists in emptying the message queue firstly, and then inserting the new value, while reading consists in getting the data value without destroying it. For the other IHMs (i.e., IHM de type Multiple-Readers/Writer, State Transition Modules, Device Interface Modules, etc.), thanks to the VIs in the palettes "Queue" and "Semaphore" of LabVIEW, we could implement them easily and intuitively.

6. DARTSVIEW TOOLKIT

The DARTSVIEW Toolkit is presented on the Figure 7. It is a LabVIEW library abstracting the DARTS concepts into LabVIEW programming elements. The library has four mains palettes, named "Hardware Task" (task is activated by either a real-time clock or an external event), "Software Task" (task is activated by another by means of the synchronization/communication tool), "Communication Tool", and "Generate Code".

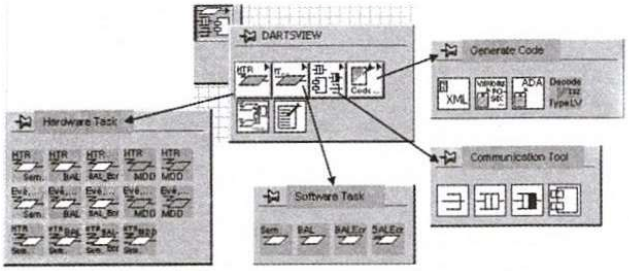

Figure 7. DARTSVIEW Toolkit

The designer of a real-time application can program his system directly from the DARTS conception, and will obtain a program that can be tested, and used in order to generate some code targeting different real-time programming standards or temporal validation tools. To illustrate the role of DARTSVIEW in a W life-cycle, a simple example is presented on Figure 8. This is a building's heating system; its brief behavior is the following: the ignition system is

run if the air (controlled by a fan) and the gas (controlled by a valve) are supplied. If during the operation of the system one of these two sources is closed, or the combustion is turned off, an alarm will be raised.

Thus, depending on the states of the sensors (the toggle switch state, the thermostat, the thermocouples, the flow meters, etc.), a central control task decides to send the commands to the tasks commanding the actuators, while another task is in charge of calculating the difference between the actual temperature and the required one.

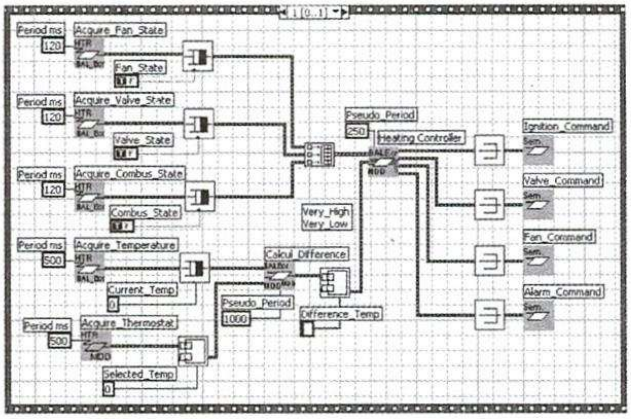

Figure 8. DARTSVIEW diagram of a heating system

The DARTSVIEW diagram on Fig. 8 is really similar to the DARTS diagram. Thanks to the simple and intuitive implementation, the designer could create in LabVIEW a software simulator in order to test the global behavior (the functional aspect) of the tasks system in the first V of the W life-cycle by means of the DARTSVIEW diagram. Moreover, all of the information about the tasks system will be recorded, and will be generated to the designer in form of a XML document (see Fig. 9) for the use in the second V of the W life-cycle: code generation.

LabVIEW allows a rapid development of a control/ command or soft real-time application, but it is less used for embedded hard real-time systems. Several standards and proprietary extensions are used, depending on the application area (aerospace, aeronautics, car, manufacturing, UAV, electronic devices…): Ada, ARINC 653, OSEK/VDX, POSIX 1003.1, VxWorks, TRON, etc.

Figure 9. XML representation

So it is convenient to find a flexible way to be able to generate the specific multitask code parts targeting these languages/standards from DARTSVIEW. The same problem arises when we want to generate the code in order to validate the application by a third-party tool. A flexible choice seems to be the use of XML in a schema of the code generation from DARTSVIEW like the one shown on Figure 10. A new standard or third-party tool is then targeted by LabVIEW using an XML file to describe the code generation for the tests, the calculation of the temporal parameters of each task, and the feed-back of these results to the DARTSVIEW model. Consequently, thanks to DARTSVIEW the time-to-market of the development of system will be better than the one using the traditional approach.

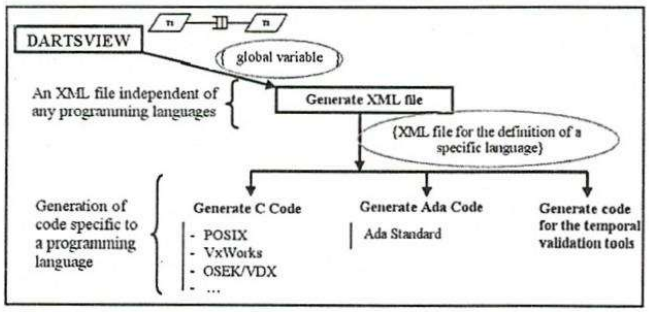

Figure 10. Schema of code generation from DARTSVIEW

7. CONCLUSION

DARTSVIEW Toolkit is a helpful tool for the DARTS development of control-command applications in LabVIEW, as well as a helpful tool for the development of embedded applications using a target language based on a specific standard or proprietary library. The use of XML-DTD facilitates the generation of code from the DARTSVIEW model, and allows the designer to choose a third-party tool for the validation of the timing requirements.

DARTSVIEW Toolkit is already used as a first multitasking environment for students in two French schools.

DARTSVIEW, CÔNG CỤ HỖ TRỢ THIẾT KẾ DARTS TRÊN LABVIEW

Ngô Khánh Hiếu[1], Grolleau Emmanuel[2]
(1) Trường Đại học Bách Khoa, ĐHQG-HCM
(2) Phòng Thí nghiệm Khoa học Máy tính ứng dụng, LISI-ENSMA, Pháp

TÓM TẮT: *DARTS (Design Approach for Real Time Systems) [4] là một phương pháp thiết kế cấu trúc phần mềm cho các hệ thống thời gian thực. LabVIEW (Laboratory Virtual Instrument Engineering Workbench) là một ứng dụng lập trình đồ họa dựa trên cấu trúc dòng dữ liệu (dataflow) của ngôn ngữ G được phát triển bởi tập đoàn National Instrument [6][2]. LabVIEW là ngôn ngữ lập trình đa nhiệm (multithreaded) và hỗ trợ các phương thức giúp thiết lập dễ dàng các giao tiếp "task", cho phép sử dụng nó như là một ngôn ngữ lập trình cho hệ điều khiển cũng như các ứng dụng thời gian thực có rang buộc lỏng (soft real-time applications). Để có thể hỗ trợ người thiết kế trong việc phát triển các ứng dụng thời gian thực, chúng tôi đưa ra một thư viện các công cụ thiết kế DARTS dựa trên nền tản ngôn ngữ*

LabVIEW, được gọi là thư viện DARTSVIEW, nhờ vào công cụ này việc chuyển đổi từ các khái niệm ứng dụng đa nhiệm (multitasking) của thiết kế DARTS sang LabVIEW có thể được triển khai dễ dàng [8]. Hơn thế nữa, DARTSVIEW có thể được dùng vào các giai đoạn khác nhau trong vòng phát triển phần mềm của hệ thời gian thực. Phiên bản gần đây nhất của DARTSVIEW còn cho phép xuất code XML của các ứng dụng thời gian thực đã được xây dựng từ LabVIEW bằng DARTSVIEW, nhằm phục vụ cho mục đích chuyển đổi sang code của các ngôn ngữ lập trình vi xử lý hỗ trợ lập trình đa nhiệm như Ada, POSIX 1003.1, VxWorks, OSEK/VDX, vv. Và nhờ đó, việc kiểm tra trực tiếp trên vi xử lý về đáp ứng các ràng buộc thời gian có thể được tiến hành nhanh chóng, thuận lợi. Mục tiêu của bài viết này là tổng quan về công cụ DARTSVIEW, vai trò của DARTSVIEW trong chu trình phát triển cấu trúc phần mềm của hệ thời gian thực, và cuối cùng là một vài hướng mở rộng cho công cụ này trong thời gian sắp tới.

Từ khóa: *công cụ DARTSVIEW, công cụ thiết kế DARTS.*

REFERENCE

[1]. F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri, *Scheduling in Real-Time Systems*, J. W. & Son, (2002).

[2]. F. Cottet, E. Grolleau, *Systèmes temps réel de contrôle-commande, Conception et Implémentation*, Dunod, 561 (2005).

[3]. Emmanuel Gerveaux, *Conception d'un environnement de développement des applications de contrôle de procédé basé sur le modèle formel GRAFCET et fondé sur un langage graphique flot de données*, rapport de Thèse, LISI-ENSMA (1998).

[4]. Hassan Gomaa, *Software Design Methods for Concurrent and Real-Time Systems*, Addison Wesley, SEI Series in Software Engineering (1993).

[5]. Emmanuel Grolleau, *Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de Petri en environnement monoprocesseur et multiprocesseur*, rapport de Thèse, LISI-ENSMA (1999).

[6]. National Instruments, *LabVIEW™ 5 Software Reference and User Manual* (1998).

[7]. M. G. Harbour, J.J. G. Garcia, J.C. P. Gutierrez, J.M. D. Moyano, *MAST: Modeling and Analysis Suite for Real-Time Applications, Proc. Of the 13th IEEE Euromicro Conference in Real-Time Systems* (2001).

[8]. K.H. Ngo, E. Grolleau, *La Méthode DARTS et La Programmation Multitâche en LabVIEW, FuturVIEW'2003*, ENSMA (2003).