Toolbox for real time control task design using Matlab/Simulink.

Roberto M. Linares Zamora[1], Dr. Pedro Mejía Álvarez[1], Dr. Alberto Soria López[2]

[1]CINVESTAV-IPN Sección de Computación Av. IPN No. 2508, México D.F
[2]CINVESTAV-IPN Departamento de Control Automático, Av. IPN No. 2508, México D.F
rlinares@computacion.cs.cinvestav.mx pmejia@cs.cinvestav.mx soria@ctrl.cinvestav.mx .

Abstract. Traditional control systems design often disregards the temporal constraints arising from the implementation of real-time control applications. Nowadays, real-time control process design does not include the timing effects into its design process. For this purpose, we have developed a tool for the design of control task in Simulink which are implemented into a real-time operating system. The aim of our approach is to generate real-time control applications without coding a single line of code, instead using the visual programming capabilities of Simulink. The basic idea is to generate a C-API Simulink block capable of handling the C code produced by the MATLAB tool as a standalone application within a Real Time kernel as control tasks. The resulting code can be executed on off-the-shelf standard personal computers, without requiring complex or expensive hardware architectures (which is often associated with specialized real time systems) and without causing any performance loss. Performance and functionality tests were carried out by using a real time microkernel developed on MS-DOS.

Keywords: C-API, toolbox, Kernel, ERT, TLC

1. Introduction.

Most computer control systems are embedded systems where the computer is a component within a larger engineering system. The controllers are often implemented as one or several tasks on a microprocessor using a real time operating system. In most cases the microprocessor also contains other concurrent tasks performing different functions, e.g., communication and user interfaces that require processor time [1]. The Operating System (OS) typically uses multiprogramming, context switch, scheduling policies, etc. to multiplex the execution of the different tasks on a single CPU.

Nowadays, the design of this type of applications begins to experiment new approaches, some of them focusing new tools and others like the proposed approach, uses a combination of different tools, to include real-time constraints into control system design. The primary motivation of this paper lies in the generation of C code using the

© A. Gelbukh, C. Yáñez Márquez, O. Camacho Nieto (Eds.)
Advances in Artificial Intelligence and Computer Science
Research on Computing Science 14, 2005, pp. 369-380

developed API (Application programming interface) to execute it in a real time microkernel as control process.

It is common practice in many control applications the use of design tools like MATLAB/Simulink. Simulink is a useful tool for the development of control systems that simulates complex control systems. However, these tools only focus on the control problem, without considering how the concurrency and scheduling affects control performance. In the design of any control application it is necessary to consider the effects produced by the concurrency and the scheduling policy used along with the following real-time constraints: the period and the computing time of the control algorithm [2].

The design of real-time control systems must use efficient and flexible software alternatives to complement the overall design, where aspects such as the sampling period, the priority as well as the scheduling policies are considered when these processes are implemented as control tasks in a real time kernel.

MATLAB/Simulink/Embedded Real TimeWorkshop (ERT) suite [4, 5, 6], an automatic C language code generator for Simulink. When using Simulink, it is possible to create, simulate and analyze complex dynamic systems by simply connecting functional blocks, mostly available from various preconfigured libraries, within a friendly graphical user interface (GUI). Furthermore, being Simulink a package of the MATLAB problem solving environment, it shares the same straightforward integration of computation, monitoring and visualization, thus allowing an easy display of any result of a running simulation. By using Simulink the user can concentrate on modelling and control issues, as opposed to programming issues. The ERT tool, controls the translation of the blocks in a series of C functions than can be compiled and linked to obtain executable code that can be used as a standalone application.

In this work, we propose the use of a real time microkernel to test the generated code by Simulink. The main function of the real time microkernel, is to allow the concurrent execution of several processes, besides allowing the creation, elimination, synchronization and communication between processes. The generation of processes (control tasks) is carried out by ERT tool of MATLAB/Simulink.

2. Related Work.

Some recent works [1] have pointed out the need to integrate control systems with real time systems, creating an interest for the development of new applications this has helped in the design of control systems with time restrictions using different scheduling policies. One of the first works where the control and the real time are discussed is a

prototype tool for the co-design of control system and real time developed under Matlab [2]. The basic idea is to simulate a real time kernel in parallel with continuous plant dynamics. This Matlab toolbox allows the user to explore the time behaviour of the control algorithm, and to study the interaction between the control tasks and the scheduler. This tool is useful for the simulation of control system in real time; nevertheless, the tool cannot simulate concurrent processes because Matlab does not allow it, causing a poor support for the handling interruptions. In [3], a hard real time Linux environment for control applications using MATLAB/Simulink is discussed. In this application is used the RTW tool to shape the characteristics of the RTAI. This work is not complete yet, however, a high reliability application for simple control systems design is proposed.

3. Matlab ERT

The Real-Time Workshop Embedded (ERT) tool is a separate, add-on product for use with Real-Time Workshop. It is intended for use in embedded systems development. The ERT provides a framework for the development of production code that is optimized for speed, memory usage, and simplicity. The most important characteristic of ERT is that can work as a *host* or a *target*. A *target* is an environment—hardware and operating system—on which the generated code will run. The process of specifying this environment is called *targeting*. The process of generating target-specific code is controlled by a *system target file*, a *template makefile*, and a *make command*. To select the target for our work, we specify these items individually. The *host* is the system you use to run MATLAB, Simulink, and Embedded Real-Time Workshop. Using the build tools on the host, we create code that runs on our target system microkernel.

The ERT generates optimized ANSI-C or ISO-C code for fixed-point and floating-point microprocessors. It extends the capabilities provided by the Real-Time workshop to support specification, integration, deployment, and testing of production applications on embedded targets. During the generating code from a Simulink model, the ERT creates several files that contain the complete description of the model. These files contain data structures implemented by the ERT tool allowing the data handling of the Simulink model parameters.

4. Framework Structure.

The general structure of the application is depicted in the Fig. 1. From a Simulink diagram the C-API toolbox will generate C code. The process focuses mainly in the generation of source C code from a Simulink control diagram using the C-API toolbox, using a modified TLC (Target Language Compiler) file that simplifies the code.

Once the C-API toolbox is added to the Simulink model, the script associated with the block automatically sets all parameters of the active configuration set that are relevant to code generation, including selection of the appropriate target.

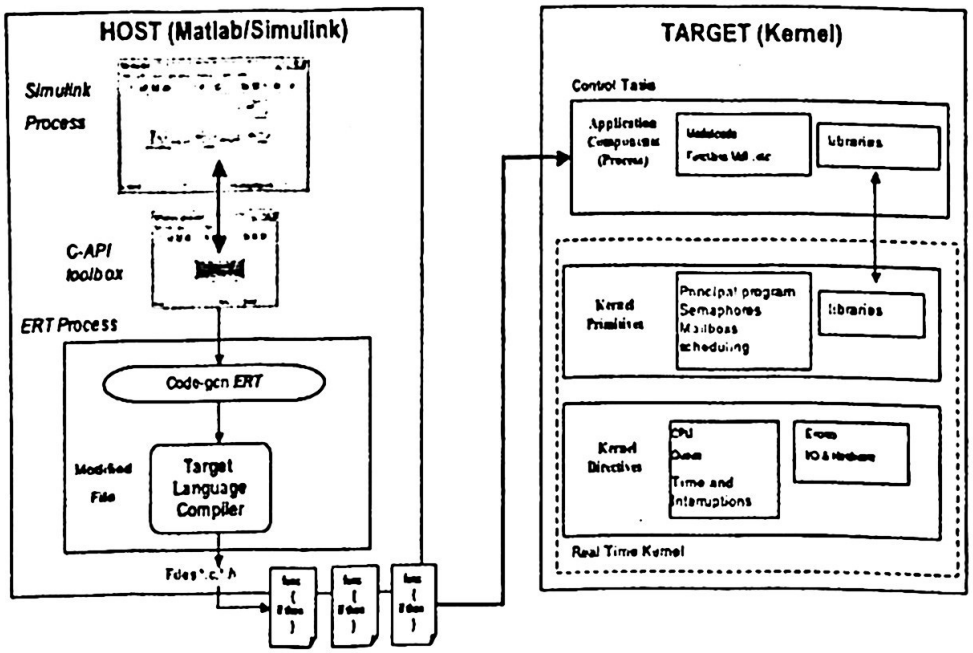

Fig. 1. Application Structure.

5. RTW Structure.

The process of generating source code from Simulink models is depicted in Fig. 2. A high-level M-file command controls the Real-Time Workshop build process. The default command, used with most targets, is *make_rtw*. When the build process is initiated, the Real-Time Workshop invokes *make_rtw*. This command invokes the Target Language Compiler TLC as wel as other utilities such as *make* command The build process consists of the following steps invoked by the *make_rtw* command :

1.- Compilation of the block diagram and generation of a model description file, *model*.rtw.

2.- Generation by the Target Language Compiler of target-specific code *model*.rtw as specified by the selected system target file.

3.- Creation of a makefile, *model*.mk, from the selected template makefile.

4.- Compilation and linking by the *make* command. ,as instructed in the generated *make* file creating an *.exe* file. In our application we modified the TLC to create a standalone application.

6. Target Language Compiler.

ERT generates source code for models and blocks through the Target Language Compiler, which reads script files (or *TLC files)* that specify the format and content of the generated source files. The Target Language Compiler (TLC) is designed for one purpose — to convert the model description file, *model*.rtw, (or similar files) into target-specific code or text. As an integral component of Real-Time Workshop, the Target Language Compiler transforms an intermediate form of a Simulink block diagram, called *model*.rtw, into C code. The *model*.rtw file contains a "compiled" representation of the model describing the execution semantics of the block diagram in a very high level language.

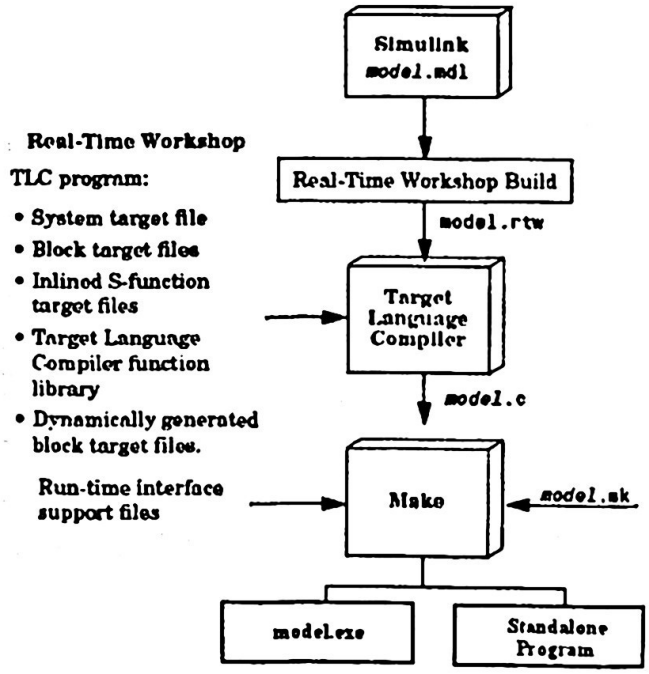

Fig. 2. Real-Time Workshop Code Generation Process.

After reading the *model*.rtw file, the Target Language Compiler generates the code based on *target files*, which specify particular code for each block, and *model-wide files*, which specify the overall code style. The TLC works like a text processor, using the target files and the *model*.rtw file to generate ANSI C code. In order to create a target-specific application, ERT also requires a template makefile that specifies the appropriate C compiler and compiler options for the build process. The template makefile is transformed into a makefile (*model*.mk) by performing token expansion specific to a given model. The Target Language Compiler uses *block target files* to transform each

block in the *model*.rtw file and a *model-wide target file* for global customization of the code.

Fig. 3, shows how the Target Language Compiler Process works with the asociated target files and the Real-Time Workshop output to produce the code. When generating code from a Simulink model using Real-Time Workshop, the first step in the automated process is to generate a *model*.rtw file. The *model*.rtw file includes all of the model-specific information required for generating code from the Simulink model. *model*.rtw is passed to the Target Language Compiler, which uses it in combination with a set of included system target files and block target files to generate the code.

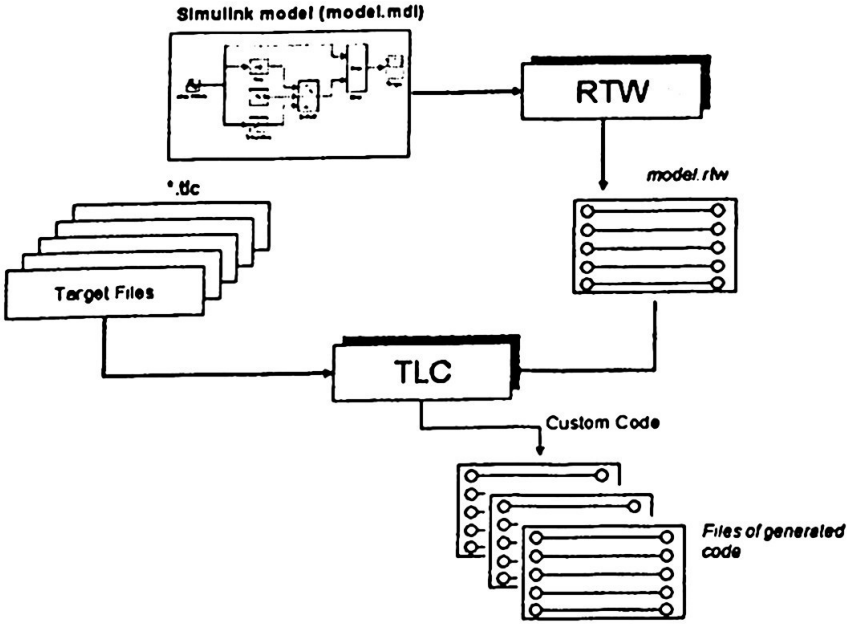

Fig. 3. The Target Language Compiler Process

7. ERT Data Structures.

The ERT generates the following functions to describe the Simulink model behaviour:

*model*_intialize() — Performs all model initialization and should be called once at the execution of the Simulink model.

*model*_step() — Contains the output and update code for all blocks in the Simulink model.

*model*_output() — Contains the output code for all blocks in the Simulink model.

*model*_update() —This contains the update code for all blocks in the Simulink model.

*model*_terminate() — This contains all model shutdown code and should be called as part of system shutdown.

When a model is constructed in Simulink, the ERT encapsulates all the information about the model in a data structure, called "rtModel". This structure contains only information required for the model build process. rtModel structure may also contain model-specific information related to timing, solvers, and model data such as inputs, outputs, states, and parameters.

The symbol definitions for rtModel structure in the generated code are given by:

- Structure definition (in *model*.h):

```
struct _RT_MODEL_model_Tag {
...
};
```

- Forward declaration typedef (in *model*_types.h):

```
typedef struct _RT_MODEL_model_Tag  RT_MODEL_model;
```

- Variable and pointer declarations (in *model*.c):

```
RT_MODEL_model model_M_;
RT_MODEL_model *model_M = &model_M_;
```

- Variable export declaration (in *model*.h):

```
extern RT_MODEL_model *model_M;
```

In order to interface the user code to rtModel, the ERT provides accessor macros. User code can use the macros, and access the fields they reference in the *model*.h file. If the code interfacing involved only a single model, the user it will be refer to its rtModel generically as *model*_M, that is used to access its rtModel as in the following code fragment.

```
#include "model.h"
const char *errStatus = rtmGetErrorStatus(model_M);
```

To interface the code to the rtModel structures when more than one model is used, the user must include *model*.h headers for each model, as in the following code fragment:

```
#include "modelA.h" /* Make model A entry points visible */
#include "modelB.h" /* Make model B entry points visible */

void myHandWrittenFunction(void)
    {
     const char_T *errStatus;
     modelA_initialize(1); /* Call model A initializer */
     modelB_initialize(1); /* Call model B initializer */
     /* Refer to model A's rtModel */
     errStatus = rtmGetErrorStatus(modelA_M);
     /* Refer to model B's rtModel */
     errStatus = rtmGetErrorStatus(modelB_M);
    }
```

8. C-API Code Generator.

The C-API toolbox consists of a block library from Simulink designed for the generation of C code, as shown in Fig.2. The block is a fixed-step MATLAB Script function written in C. The C-API block executes and configures all parameters of the model's active configuration set without manual intervention of the user and optimizes the code generation from the Simulink models quickly and easily. This block can execute a configuration script independently from the code generation process and the changes are then visible in the GUI and can be saved with the model.

8.1. Block Function

In many MATLAB Real-Time Workshop applications, the user may want to interact with a model's signals or parameters in the generated code. For example, you may want to monitor and modify parameters. Or, in a signal monitoring or data logging application, you may want to interface with signals. The developed C-API block can generate code to interface with signals and parameters within other applications and without the MATLAB intervention working as stand-alone program. This is a target-based Real-Time Workshop feature that provides access to global block outputs and global parameters in the generated code.

8.2. The C-API Function.

The C-API script implements a single function without a return value. The function takes the single argument *cs*:

```
function gen_tar(cs);
```

The argument *cs* is a handle to a proprietary object that contains information about the model's active configuration set. Simulink obtains this handle and passes it in to the configuration function when the user double-clicks a C-API block. The code use *cs* as a "black box" object that transmits information to and from the active configuration set, using the accessor functions described below.

To set options or obtain option values, we use the Simulink **set_param** and **get_param** functions. The option names are passed in to **set_param** and **get_param** as strings specifying an *internal option name*. The internal option name is not always the same as the corresponding option label on the GUI (for example, the ***Configuration Parameters*** dialog).The following code excerpt turns off the ***Support absolute time*** option:

```
set_param(cs,'SupportAbsoluteTime','off');
```

8.3 Selecting a Target.

The C-API defines a target configuration. The script uses the ERT target as default for generate the C code. The script first stores string variables that correspond to the **System target file**, **Template makefile**, and **Make command** settings:

```
stf = 'ert.tlc';
tmf = 'ert_gen.tmf';
mc = 'make_rtw';
```

The system target file is selected by passing the cs object and the *stf* string to the switch Target function:

```
switchTarget(cs,stf,[]);
```

9. Model Example.

This section describes a simple example using de C-API toolbox to generate C code and use it like a standalone applications in a real time microkernel. This example simulates a control system, named **model.mdl** (Fig. 4). The system response is showed in the Fig. 5.The next step is use the C-API to generate the C code.

9.1. Adding C-API toolbox to the model.

1 Select C-API block from the library. Drag and drop the block into the Simulink model (**model.mdl**).

2 In the model, double-click your C-API block.

3 In the MATLAB window, you should see the test message of the C-API script: *Custom Configuration Wizard Script completed.* This indicates that the C-API block successfully executed the script.

4 Open the **Configuration Parameters** dialog and view the Real-Time Workshop options. You should now see that the model is configured for the ERT target.

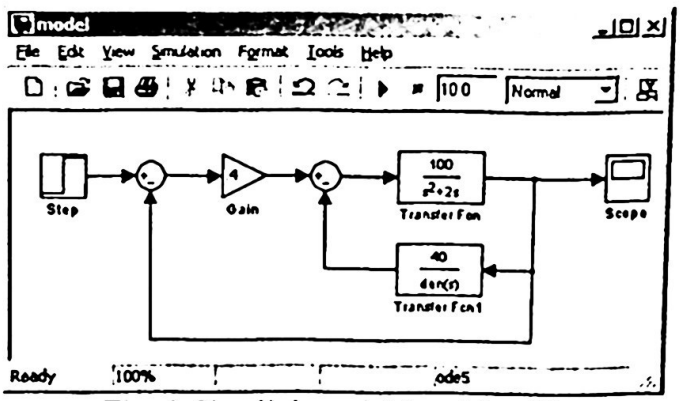

Fig. 4. Simulink model "model.mdl".

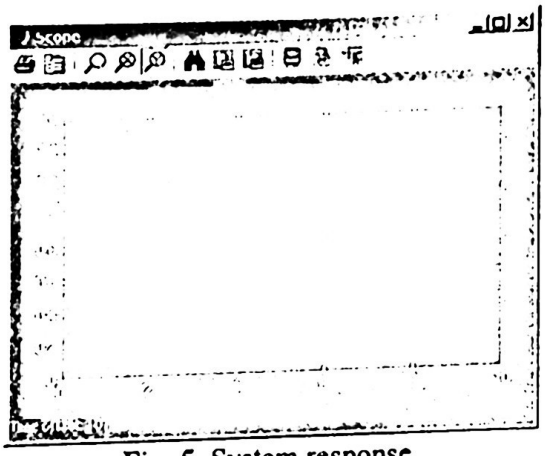

Fig. 5. System response.

Table 1. Real Time Workshop embedded coder file packing file description.

File	Description
model.c	Contains entry points for all code implementing the model algorithm (for example, *model*_step, *model*_initialize, *model*_terminate, *model*_SetEventsForThisBaseStep).
*model*_private.h	Contains local macros and local data that are required by the model and subsystems.
model.h	Declares model data structures and a public interface to the model entry points and data structures.
*model*_data.c	*model*_data.c is conditionally generated. It contains the declarations for the parameters data structure, the constant block I/O data structure, and any zero representations used for the model's data structure types.
*model*_types.h	Provides forward declarations for the real-time model data structure and the parameters data structure. Also provides type definitions for user-defined types used by the model.
rtwtypes.h	Defines data types, structures and macros required by Real-Time Workshop Embedded Coder generated code. Most other generated code modules require these definitions.

9.2 Generated Code Modules

The ERTW creates a build directory in the working directory to store the generated source code. The build directory also contains object files, a makefile, and other files created during the code generation process. The table 1 summarizes the structure of source code generated by the ERT. The generated C code is executed as an independent application inside the real time microkernel. In this example, 4 concurrent processes are executed with a Round Robin (RR) scheduling policy:

Process 1 => Basic clock
Process 2 => Basic Text Processor
Process 3 => Ball

Process 4 => Irregular polygon
Process 5 => p_simulink (additional process)

An additional process was created, called "*p_Simulink*" that invokes the generated code. In the microkernel, 4 processes are executed: the clock, the process *p_Simulink*, the ball and the irregular polygon. The *p_Simulink* process plots the response of the system. The Fig. 6, shows the response of simulated control system executed within the microkernel. It can be noted that the response is very close to the one obtained in Simulink.

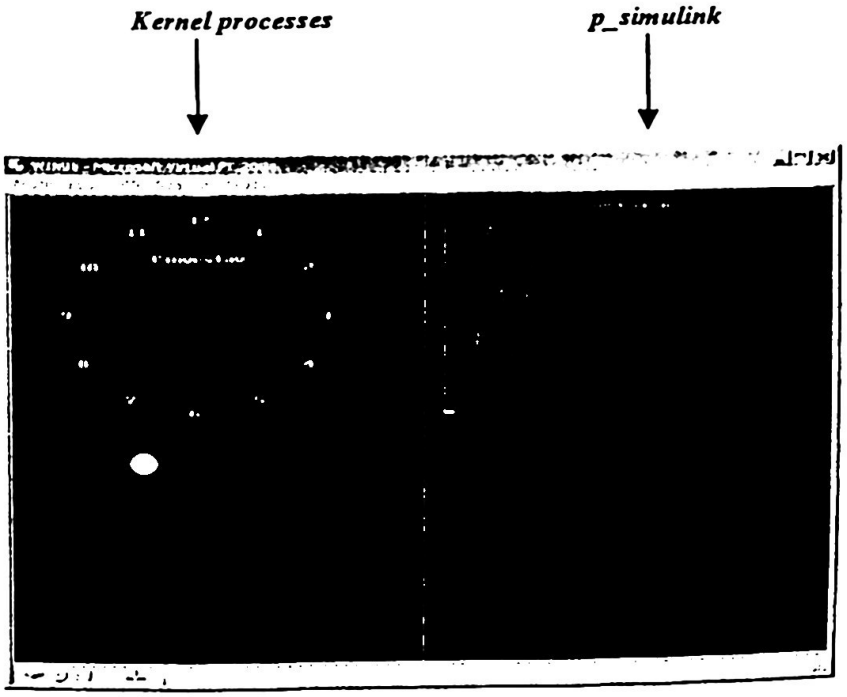

Fig. 6. Microkernel processes.

The generated code is one more process inside the microkernel, opening the possibility of creating more processes from Simulink models and test several control processes to obtain a more detailed study of the behaviour of complex control systems.

10. Conclusion.

Real-yime control design is difficult and is very sensitive to errors. The desing of control code usually disregards timing constraints making difficult the integration of control and real-time applications in an integrated framework. In this paper, our aim is to design real-time control applications without programming a single line of code. Instead to generated C code. The C generated is then used as a task into a real-time kernel.

An example of code generation was developed to simulate the execution of a control system and its execution use the Round Robin scheduling process. Due to the characteristics of our design tool it is possible to obtain code for different platforms (UNIX, DOS and WINDOWS) and to handle it like an independent application. Future works will be dedicated to use the generated C-API to handle several concurrent processes within microkernel to develop an application that allows systematically to design of real time control systems, using the developed interface.

References

1. Johan Eker: "A Matllab Toolbox for Real-Time and Control Systems Co-Desing". *Procedings of 6ht International Conference on Real-Time Computing Systems and Applications*, Hong Kong, , December 1999.
2. Anton Cervin, Dan Henriksson, Bo Lincoln.: "Jitterbug and True Time: Analysis Tools for Real-Time Control Systems ". *Proceeding of the 2nd Workshop on Real-Time Tools*, Copenhagen, Denmark, August, 2002.
3. Quaranta, P. Mantegazza. : *Using Matlab/Simulink RTW to build Real Time Control Applications in user space with RTIA-LXRT* . Dipartimento di Ingegneria Aerospaziale et Control, Politecnico di Milano, Italy, 2003.
4. Matlab User's Guide. The Matworks Inc., Third printing Revised for Version 6.1 (Release 14SP1), October 2004.
5. Simulink User's Guide. The Matworks Inc., Third printing Revised for Version 6.1 (Release 14SP1), October 2004.
6. Real-Time Workshop User's Guide. The Matworks Inc., Third printing Revised for Version 6.1 (Release 14SP1), October 2004.