Real-time framework for ITER control systems

W. Lee¹, A. Zagar¹, B. Bauvir¹, T. Tak¹, A. Winter², S. Lee³, M. Knap⁴, P. Karlovsek⁴, P. Perek⁵, D. Makowski⁵

¹ ITER Organization, St. Paul Lez Durance Cedex, France.
² Max Planck Institut für Plasmaphysik, Greifswald, Germany
³ Korea Institute of Fusion Energy, Daejeon, Republic of Korea
⁴ Cosylab d.d., Ljubljana, Slovenia
⁵ Lodz University of Technology, Lodz, Poland

woongryol.lee@iter.org
Outline

- Introduction RTF (what it is and how it works)
- Evaluation of RTF from a use case study
- Conclusion

- The views and opinions expressed herein do not necessarily reflect those of the ITER Organization.
Introduction

- **EPICS is an application framework** to build the applications for controllers and servers (records databases and state machines).

- **Real-Time Framework (RTF)** is a middleware providing common services and capabilities to build real-time control applications in ITER, such as the Plasma Control System (PCS) and plasma diagnostics.

- The **Plasma Control System** is a dominant factor for the ITER pulsed operation. It controls all aspects of the plasma discharge from powering the superconducting magnets up to the plasma termination.

- **Diagnostics** adopted by plasma physicists for measuring plasma properties requires a lengthy process due to complex algorithms.
**Introduction**

- Approximately 190 control functions must be harmonized and coordinated as a whole, and the control scheme can be different for each pulse depending on the goal in a relatively short time interval.

- Thus a flexible high-performance software suite was needed to facilitate the development and deployment of complex real-time applications. Initially aimed to the control algorithms, the RTF can also be the basis for real-time data processing applications in diagnostics.


- The architecture design fully considered the modularity and portability of the software, and is applicable and extendable even in none-ITER environments.

---

[1] SEQA-45 - Software Engineering and Quality Assurance for CODAC (2NRS2K)
Overview of Real-Time Framework

- **Managers** for resource, service and factories.
- **Services** provide site-specific facilities orthogonal to function blocks. Services include mechanisms for transferring data between nodes, logging, archiving, monitoring and control.
- **Real-time threads** that execute all processable objects
- **Function blocks** are chained by connecting output ports to input ports of other function blocks.

Function Block

- An atomic component to build an application.
- Influenced by parameters, event triggers or event handlers, each function block accepts inputs and produces outputs whenever it is processed.
- Factory design pattern for configuration-driven instantiation.
- Encapsulation of other function blocks giving the RT application an apparent hierarchical structure.

```
<FunctionBlock Name="evGen" Type="EventGenerator<int32>">
  <InputPort Name="In" Signal="Lameule:out"/>
  <EventTrigger Name="GenerateEvent" Id="event::LHtransition"/>
</FunctionBlock>

<FunctionBlock Name="hpId" Type="KpId<int32>">
  <Parameter Name="Kp" Value="1.0"/>
  <Parameter Name="Ki" Value="0.0"/>
  <Parameter Name="Kd" Value="0.0"/>
  <Parameter Name="Oh" Value="0.0"/>
  <Parameter Name="Ov" Value="0"/>
  <InputPort Name="Setpoint" Signal="ref_H:out"/>
  <InputPort Name="Feedback" Signal="density:out"/>
  <InputPort Name="errorTh" Signal="errThrd_H:out"/>
  <OutputPort Name="Out" Signal="hpId:cmd"/>
  <OutputPort Name="Out" Signal="hpId:err"/>
  <OutputPort Name="Out" Signal="hpId:Eval"/>
  <OutputPort Name="Out" Signal="hpId:Dval"/>
</FunctionBlock>
```

```
<!-- Connects reset handler to a given event id -->
<EventHandler Name="ResetHandler" To Id="event::LHtransition"/>
</FunctionBlock>
```

"my name" in <Type>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>output</td>
<td></td>
</tr>
</tbody>
</table>

Supports various types of signals e.g. scalar, none-scalar, nested data inside framework.
Real-Time application

- The RT application defines the processing logic that executes the desired behaviour according to the designer’s intention.
- The framework handles the dependency-based execution of FBs in either single-threaded or multi-threaded environments as specified in the deployment configuration.
- Implicit insertion inserts necessary gateway function blocks to implement inter-thread or inter-process (or node) communication.
Real-Time application

- The RT application defines the processing logic that executes the desired behaviour according to the designer’s intention.

- The framework handles the dependency-based execution of FBs in either single-threaded or multi-threaded environments as specified in the deployment configuration.

- Implicit insertion inserts necessary gateway function blocks to implement inter-thread or inter-process (or node) communication.
Real-Time application

- The RT application defines the processing logic that executes the desired behaviour according to the designer’s intention.
- The framework handles the dependency-based execution of FBs in either single-threaded or multi-threaded environments as specified in the deployment configuration.
- Implicit insertion inserts necessary gateway function blocks to implement inter-thread or inter-process (or node) communication.
Processing of FB in a thread

- FBs are instantiated and serialized in the loading phase. The ordering is determined by the relation between function blocks once the configuration is parsed.

- Executing the FBs is implemented with busy-wait rather than through interrupts or callbacks to avoid context switching; they effectively minimize jitter and response times.

- The RT process should be constant over all the cycles. The underneath rule of execution of FB is to execute the process method periodically under rt-thread, and thus ensure predictability of the execution times.
Framework Life Cycle Management

- Since the framework is part of a distributed control system, it needed to centrally control multiple instances in an organized manner.
- Life Cycle Management Service (LCMS) allows centrally orchestrate the state transition in conjunction with loading configuration.
- Provision of external interface can be customized to site-specific requirements. Two basic interfaces were supported: `pvAccess` protocol from EPICS v7 and native **TCP/IP**.

```
leew2 @ diag-fcl.codac.iter.org :~ $ pvlist localhost
CTRL_PCS:N1-LOAD-APP
CTRL_PCS:N1-LOAD-SERVICE
CTRL_PCS:N1-RTF-OPREQ
CTRL_PCS:N1-RTF-OPSTATE
CTRL_PCS:N1-RTF-RESET
[ 12:26:17 ]
leew2 @ diag-fcl.codac.iter.org :~ $
```

Operational PVs for interworking with central supervision system.
Framework Life Cycle Management

- Since the framework is part of a distributed control system, it needed to centrally control multiple instances in an organized manner.
- Life Cycle Management Service (LCMS) allows for the central orchestration of state transitions in conjunction with loading configurations.
- Provision of external interfaces can be customized to site-specific requirements. Two basic interfaces were supported: pvAccess protocol from EPICS v7 and native TCP/IP.

Operational PVs for interworking:

PVs are dynamically created after receiving configuration.
Framework Life Cycle Management

- Since the framework is part of a distributed control system, it needed to centrally control multiple instances in an organized manner.
- Life Cycle Management Service (LCMS) allows centrally orchestrating the state transition in conjunction with loading configuration.
- Provision of external interface can be customized to site-specific requirements. Two basic interfaces were supported: pvAccess protocol from EPICS v7 and native TCP/IP.

Operational PVs for interworking with central supervision system.

PVs are dynamically created after receiving configuration.
Interface support

- Foundational networking infrastructure is implemented as a *transport layer service*, and supports communication on all levels of RTF
  - Synchronous Data-bus Network (SDN) for feedback control
  - Data Archiving Network (DAN) for experimental data archiving
  - Plant Operation Network (PON) in EPICS pvAccess protocol
  - Nominal Device Support (NDS) for physical hardware interfacing is under development
  - Any necessary in the future

- Simulink interface uses generated code from Simulink Coder™
  - The wrapper FB loads the compiled library, and RTF performs a validation process to verify the interface data structure.
Introduction of a use case study
PCS Prototyping: Basic sketch of the PCS architecture

[1] PCS FDR presentation: PCS architecture design (3BX5LU)
PCS Prototyping: Basic sketch of the PCS architecture

[1] PCS FDR presentation: PCS architecture design (3B5LU)
PCS Prototyping: Basic sketch of the PCS architecture

[1] PCS FDR presentation: PCS architecture design (3BX5LU)
PCS Prototyping: Basic sketch of the PCS architecture

[1] PCS FDR presentation: PCS architecture design (3BX5LU)
PCS Prototyping : Basic sketch of the PCS architecture

[1] PCS FDR presentation: PCS architecture design (3BX5LU)
PCS Prototyping: Basic sketch of the PCS architecture

[1] PCS FDR presentation: PCS architecture design (3BX5LU)
Controller model from the Simulink platform can be converted to the code under Simulink constraints.

A designer can devise a desired function only by changing parameters, while maintaining the same external interface to the other FB.

Need an appropriate granularity in the controller model for conversion.
PCS Prototyping: Controller model from the generated code

- How to interface in between Simulink and RTF

- Header file generated from KCURR model that defines the data structure for the interface to RTF

- Simulink wrapper FB loads the compiled library.
- Interface with structured data e.g., matrix and array is supported in the parameter attribute from the release of v2.2.4.

Example for converting to RTF configuration:
PCS Prototyping: Controller model from the generated code

- How to interface in between Simulink and RTF

- Header file generated from KCURR model that defines the data structure for the interface to RTF

- Simulink wrapper FB loads the compiled library.

- Interface with structured data e.g., matrix and array is supported in the parameter attribute from the release of v2.2.4.
Plasma diagnostics: Edge Thomson Scattering diagnostics

- The Thomson Scattering diagnostics gives a reliable electron temperature and density profiles in magnetically confined plasma.

- The customized DAQ FB archives raw data through RTF transport layer

- The output links to the fitting FB to eliminate back scattered signal.

- Electron temperature is measured using lookup table where calibrated data is stored as per the wavelength from the polychromator signal.
Plasma diagnostics: Edge Thomson Scattering diagnostics

- The Thomson Scattering diagnostics gives a reliable electron temperature and density profiles in magnetically confined plasma.

- The customized DAQ FB archives raw data through RTF transport layer.
- The output links to the fitting FB to eliminate back scattered signal.
- Electron temperature is measured using lookup table where calibrated data is stored as per the wavelength from the polychromator signal.
Plasma diagnostics: **Edge Thomson Scattering diagnostics**

- The Thomson Scattering diagnostics gives a reliable electron temperature and density profiles in magnetically confined plasma.
- The customized DAQ FB archives raw data through RTF transport layer.
- The output links to the fitting FB to eliminate back scattered signal.
- Electron temperature is measured using lookup table where calibrated data is stored as per the wavelength from the polychromator signal.
Plasma diagnostics: Edge Thomson Scattering diagnostics

<table>
<thead>
<tr>
<th>Function block</th>
<th>min</th>
<th>avg</th>
<th>max</th>
<th>Std.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pulse fitting</td>
<td>0.52 ms</td>
<td>1.31 ms</td>
<td>2.10 ms</td>
<td>0.18 ms</td>
</tr>
<tr>
<td>Pulse generation</td>
<td>0.31 ms</td>
<td>0.32 ms</td>
<td>0.54 ms</td>
<td>0.01 ms</td>
</tr>
<tr>
<td>$T_e$ calculation</td>
<td>0.01 ms</td>
<td>0.01 ms</td>
<td>0.02 ms</td>
<td>0.00 ms</td>
</tr>
</tbody>
</table>
Actuator control: Poloidal Field coil control

- ITER started implementing a real-life controller in order to evaluate both functional and non-functional behaviour of the PCS.
- 11 PF controllers were devised by complying with KSTAR native function model. Verified 20kHz control cycle in consecutive process pipeline such as exception handler, waveform generator, and PID function.
Conclusions

- The RTF is a flexible high-performance software platform that facilitates the development and deployment of complex real-time applications.
- It was designed to be portable and modular, enabling high reusability and maintainability of components constituting the real-time applications.
- Factory design pattern and rich function for multi-threaded program enables building application through configuration-driven process.
- Prototyping activities on some of the operating Tokamaks have demonstrated its applicability for the implementation of ITER real-time control systems.
Thank you for your attention!