GA–A26467

APPROACHES TO TOKAMAK OFF-NORMAL EVENT DETECTION AND RESPONSE AT DIII-D, KSTAR, AND EAST

by
B.S. SAMMULI, M.L. WALKER, D.A. HUMPHREYS, J.R. FERRON, R.D. JOHNSON,
B.G. PENAFLOR, D.A. PIGLOWSKI, S.H. HAHN, J. HONG, B. XIAO,
and Q. YUAN

JUNE 2009

GENERAL ATOMICS

DISCLAIMER

GA–A26467

APPROACHES TO TOKAMAK OFF-NORMAL EVENT DETECTION AND RESPONSE AT DIII-D, KSTAR, AND EAST

by
B.S. SAMMULI, M.L. WALKER, D.A. HUMPHREYS, J.R. FERRON, R.D. JOHNSON, B.G. PENAFLOR, D.A. PIGLOWSKI, S.H. HAHN,* J. HONG,* B. XIAO,† and Q. YUAN†

This is a preprint of an invited paper to be presented at the Seventh IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation in Fusion Research, June 15–19, 2009, Aix-en-Provence, France, and to be published in the *Proceedings.*

*National Fusion Research Institute, Daejon, Korea.
†Academica Sinica Institute of Plasma Physics, Hefei, China.

Work supported by
General Atomics Internal Research
and Development Funding

GENERAL ATOMICS PROJECT 40010
JUNE 2009

Abstract

The DIII-D digital plasma control system (PCS) has been adapted and used at a number of tokamaks around the world over the past several years. Implementations of the DIII-D PCS at KSTAR and EAST in particular have made use of, and improved upon, the methodology for off-normal event/fault detection and response that was originally incorporated into the DIII-D PCS. This work discusses the lessons learned implementing and installing codes for detecting and responding to off-normal events in both plasma discharges and tokamak systems. In particular, we discuss the implementation details for codes at DIII-D, EAST, and KSTAR, and discuss how those experiences are being incorporated into a proposed finite state machine architecture in support of ITER research at DIII-D.

1. Introduction

Off-normal event/fault detection and response (ONFD/R) on all installations of the plasma control system (PCS) [1] runs in parallel with the time-dependent control algorithms. At DIII-D, the PCS uses a limited number of responses to deal with a large number of possible events, in large part because DIII-D takes advantage of a significant hardware interlock infrastructure to address equipment protection concerns. Both EAST and KSTAR requested inclusion of additional methods for assisting in device protection, especially with regard to protections for superconducting coils.

While installations of ONFD/R have been successful in meeting the needs of present day devices, future devices such as ITER will demand a significantly more sophisticated collection of actions in response to a potentially much larger number of off-normal events. Currently, there exist proposed response actions for only a small subset of events, the most notable being plasma disruptions. For example, physics research underway at DIII-D is focused on characterizing plasma disruption events and their causes, as well as designing possible mitigation and recovery schemes. There are similar efforts underway by the DIII-D operations group to dynamically select the best possible response to faults in the device. However, the set of defined responses in either case is presently far from comprehensive. In addition, many types of off-normal events can have any number of causes and the possible responses are dependent on the device and plasma parameters.

The large number of anticipated sequences of complex state-dependent responses, along with the cumbersomeness of hard-coding each ONFD/R scenario in C, highlights the need for a more flexible and robust software approach to support of ONFD/R research. This need motivated a proposal for a finite state machine architecture to control asynchronous switching of control algorithms in the DIII-D PCS. The proposed framework incorporates elements similar to the ITER CODAC design [2], including the use of eXtensible Markup Language (XML) as a means of defining a finite state machine, but differs slightly in that its purpose is primarily as a support tool for development of ONFD/R scenarios.

2. Current Implementations

The DIII-D PCS incorporates a highly flexible architecture for fault detection and response, which finds a variety of experimental uses at devices using the PCS. In EAST operations, for example, the superconducting coils are protected from over-currents by directly detecting excursions close to a current limit or identifying a large error signal implying that the current control is in a fault condition. DIII-D experimental needs often use this fault response system in different ways. For example, in high performance discharges it is frequently necessary to change from one control phase to another if the physics goals of a discharge are not being met. Switching to a control phase with different programmed heating can produce profiles with different magnetohydrodynamic stability characteristics than the initial target, depending on modes and confinement quality produced initially.

The method used in the DIII-D PCS and its derivatives is to execute a set of fault algorithms, whose only purpose is to detect and respond to faults in device operation, in parallel with the feedback control algorithms. Such parallel execution is trivially implemented in the highly parallel PCS. When the detection algorithms detect an off-normal condition, several different response scenarios can be switched to, depending on the severity of the fault. The PCS architecture supports a layered response: to try to continue the discharge, if it makes sense to do so; to perform a controlled shutdown, e.g. ramp the coil currents to zero, if it is safe to take the time to do this; or to perform a fast shutdown if danger to the device does not allow for a controlled shutdown.

Additionally, a set of bits in the PCS digital I/O circuitry can be used by external systems to signal an externally detected fault, which may require a response by the PCS. At EAST, the poloidal field (PF) power supply systems can change the state of one of these bits if their internal controls detect a fault. At DIII-D, a bit in a “watchdog timer” circuit is toggled on every control cycle by the PCS. If the PCS fails to toggle this bit, a timeout will occur and trigger a fault condition that is handled by a fault detection algorithm.

At DIII-D, most, but not all, detection logic is contained within a central fault detection process, outside of the parallel feedback control loop. Table 1 lists the DIII-D fault tests and corresponding responses. One major enhancement of EAST and KSTAR implementations of ONFD/R is centralization of the fault detection logic. At EAST and KSTAR all detection logic is strictly excluded from the feedback control loops, although data computed within these loops is sometimes passed to the centralized fault algorithms. This centralization of fault detection and response makes it much easier to coordinate

appropriate responses and has the added benefit of reducing code complexity and maintenance. Installations of ONFD/R at EAST and KSTAR also extend the capabilities of the DIII-D PCS by increasing the flexibility of the ONFD/R algorithm. In the DIII-D central detection process, all off-normal events trigger a single response. At both EAST and KSTAR, each distinct off-normal event maps to a dedicated response.

Table 1 DIII-D ONFD/R. thr = threshold level for the signal.

Test	Description	Condition	Response
Non-central tests contained in feedback control loops.			
I^2t test	Protect ohmic heating coil (E-coil) from overheating. Check if integrated squared current is too large.	$\int_{-\infty}^{t} I_E^2 d\tau > thr$	Ramp down the plasma current
Volt-second limit test	Check max E-coil current swing	| E-coil current | > *thr*	Switch to ramp down control of I_p
Poloidal field coil (F-coil) over-current test	Prevent F-coils from over-current fault	F-coil current > *max*	Switch to alternate phase sequence for I_p control
Plasma facing components (PFC) protection test	Protect PFC in outer part of vessel	(Plasma to PFC gap < *thr1*) AND (Radiated pwr<*thr2*) AND (ECH power < *thr3*)	Switch the neutral beam control category to the "rad interlock" control phase sequence
Gyrotron trigger watch	Switch from open loop to closed loop control	Test if feedback value has exceeded the trigger level	Switch to the gyrotron/beam "Feedback" phase sequence
Beam trigger watch			
Central fault detection process tests (external to feedback control loops).			
Toroidal field time derivative ($\dot{B}$) amplitude test	These tests check various parameters against thresholds in order to determine if the device/plasma conforms to the PCS control setup	$\dot{B}$ Ampl > *thr*	The response to any one of these conditions is to switch the control phase of all control algorithms based on pre-defined operator setup
$\dot{B}$ frequency test		$\dot{B}$ freq > *thr*	
Beam error test		Beam power error > *thr*	
Beam τ_E test		Filtered τ_E > *thr*	
I_p error test		I_p error > *thr*	
ECH power test		ECH power < *thr*	
MHD ampl. check		MHD Ampl > *thr*	
MHD freq test		MHD Freq > *thr*	

The EAST PCS fault detection process checks plasma current (I_p) control error, PF current control error, and PF current against time-dependent thresholds. If any of these values crosses a threshold for a specified amount of time, the fault detection process switches the control phase of four different algorithms: Plasma shape control, isoflux control, gas control, and density control. The changes in control phases are tailored to the specific event detected.

At KSTAR the fault detection process checks a similar list: I_p error, n_e error, PF error, PF current, power supply (PS) voltage error, PS voltage, and I_p minimum (tripped when I_p falls too low). The response to any of these tests is to change the phase of the gas control algorithm and the shape control algorithm.

3. New Architecture

Although the KSTAR and EAST installations are more flexible in terms of mapping off-normal events to appropriate responses, off-normal event detection algorithms are still hard coded in C and programmers well versed in the internals of the PCS codes are needed to make modifications. Another limitation of the current ONFD/R architecture is that the PCS does not provide a means of easily visualizing the various ONFD/R setups. Rather, the user performs setup by navigating a series a user interface menus. In light of these limitations, implementation of a new PCS ONFD/R architecture has begun, with the following design goals:

1. Allow rapid development and testing of ONFD/R by non-programmers and PCS operators. This is of particular importance for ongoing ONFD/R physics research for ITER at DIII-D.
2. Provide an easy to understand graphical representation of ONFD/R.

This off-normal event and fault system (ONFS) performs high-level decision-making in response to off-normal fault detection using finite state machine logic in parallel with the time-dependent control of the plasma. The PCS user sets up the state machine logic via a new user interface that presents ONFD/R configurations in a finite state machine diagram.

Figure 1 presents a conceptual representation of this state machine approach. The figure identifies five categories of states, *Normal, Alternate, Recovery, Controlled Shutdown,* and *Fast Shutdown*. The last three sets of these states will also be referred to as *Response States*. The *Normal States* correspond to nominal operation without off-normal fault (ONF) conditions. When an ONF condition (e.g. C1) is detected within a state, there is a transition to another state, which is designed to deal with this condition. The most desirable response is to enter into a *Recovery State*, in which an attempt is made to recover from the ONF condition, then return to nominal operation by returning to a *Normal State*. If a more severe ONF condition (C5) is detected while in a recovery state, the preferred response is to enter into a *Controlled Shutdown State*, where a controlled shutdown of the plasma discharge is attempted. If a still more severe ONF condition (C7) is detected while in that state, a final transition to a *Fast Shutdown State* is made, where the control system attempts to shut down the discharge as quickly as possible and simultaneously mitigate any sources of potential damage to the device.

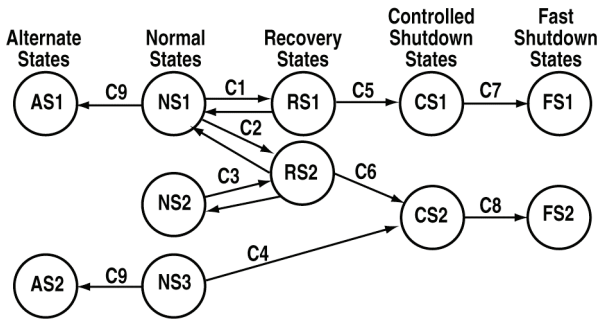

Fig. 1. Conceptual representation of the finite state machine architecture. Each arrow represents a transition caused by the discovery of a condition that is off-normal for that state. C* = condition that causes state transition.

Within the ONFS finite state machine architecture, a state is defined as a collection of the following:

- A set of the plasma and device parameters with the corresponding allowable ranges of the parameters in this state.
- An entry action. In most cases the entry action taken on initial transition to a state will be to initiate a new set of PCS phase sequences to execute. In the PCS, a phase sequence is a time-dependent prescription for what control will be applied to the device and plasma. It includes the algorithms to execute (possibly changing in time), the parameters that customize how each algorithm executes, and the time-dependent programmed target signals, which specify the desired behavior of the plasma.
- An exit action. Optionally, on exit a state may execute an algorithm for performing any cleanup.

Each transition between states has associated event detection logic. This logic outputs a single Boolean value that dictates that, when true, the system should transition from the current state to the target state.

4. Design Overview

As in previous PCS implementations, the real-time portion of ONFS monitors the tokamak in parallel with the execution of the various real-time control algorithms. Unlike in previous PCS implementations in which tests were implemented as compiled code, the ONFS is a data-driven architecture that employs XML as a means of specifying the finite state machine.

The ONFS user interface (a finite state machine editor) generates an XML data file that specifies the structure of a finite state machine. The PCS loads the XML file and validates it. The PCS then pre-processes the file to determine the size of memory that needs allocation for the state machine data structure. The PCS passes the sizing information on to a real-time computer, which, as part of an initialization routine, performs the necessary memory allocation. The ONFS real-time initialization routine then parses the file and places C++ objects into memory based on element specifications within the XML. At this point the state machine in the real-time computer is fully initialized and can begin executing.

As a means of introducing some of the design aspects of the ONFS software, consider a trivial finite state machine, as shown in Fig. 2. This state machine performs a simple test during nominal operation to check if the error in control of plasma current exceeds a specified threshold, and transitions to a shutdown scenario if the threshold is exceeded. Figure 3 is a simplified version of how the ONFS user interface would export the state machine in Fig. 2 as XML. This markup describes the state machine as a set of nested XML elements. Within the formal XML syntax, an element consists of an opening label (optionally with attributes), text, and a closing label, in the form <label attribute=”some attribute”>Some text</label>. An element may also contain sub-elements (also referred to as children or child elements), creating a hierarchical data structure. The *Device* element in this markup contains a *DeviceParameter* sub-element that describes the name of a parameter, iperr (the IP error value). The two states in the diagram similarly map to the two state elements in the XML (i.e. those with name attributes “normal” and “shutdown”). The normal state element has two child XML elements: 1) *ParamRange*, which specifies that the *DeviceParam* iperr has a range of 0 to 100 in the normal state and 2) *Transition*. By convention, transitions between states are specified in the markup for the origin state. The target state is defined as an attribute of the transition element. Additionally, the transition element has children of type *ThresholdEventDetector*, which specifies the parameters that are to be input into a threshold comparison algorithm. In this case, the transition between normal and shutdown states is triggered when the iperr parameter falls outside the range defined in the normal state

ParamRange sub-element. The shutdown state element has an *EntryAction* child, which specifies a new phase sequence for the PCS, controlled shutdown.

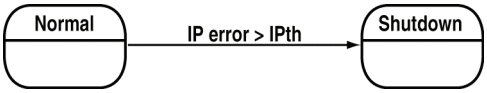

Fig. 2. Example finite state machine.

```
<?xml version="1.0" ?>
<FSM>
  <Device name="D" initial_state="normal">
    <DeviceParameter name="iperr" />
  </Device>
  <State name="normal">
    <ParamRange low="0" high="100" param="iperr" />
    <Transition target_state="off_normal">
      <ThresholdEventDetector>
        <DeviceParameter name="iperr" />
      </ThresholdEventDetector>
    </Transition>
  </State>
  <State name="shutdown">
    <EntryAction  phase_seq="controlled shutdown" />
  </State>
</FSM>
```

Fig. 3. Simplified version of how the ONFS user interface would export the state machine in Fig. 2 as XML.

5. Real-time Code

C++ was chosen for the implementation to take advantage of object-oriented data encapsulation and inheritance. A C++ class of the same name represents each element in the XML shown above. Because each installation of the PCS varies in implementation and hardware details, it is expected that some of the classes will need to be extended or overridden through inheritance. For instance, the *DeviceParameter* class defines a method for retrieving signal data from some source (which may vary from device to device). The DIII-D PCS will include a child class of *DeviceParameter* called *DiiidRtParameter* that implements the data retrieval method by accessing real-time shared memory.

Due to constraints imposed by the use of the real-time heap shared memory in the PCS [1], all ONFS classes are separated into data-only components and data-handling classes. The data-handling classes contain member functions that can be overridden by child classes in order to implement device-specific functionality, as in the example above. This type of inheritance is referred to as virtual inheritance. As a general rule, C++ objects using virtual inheritance that are instantiated into shared memory cannot be accessed by multiple processes. Therefore, we restrict usage of shared memory in the ONFS architecture to data-only classes that do not use virtual inheritance. However, we achieve virtual inheritance in the real-time process by embedding the data-only objects with self-describing metadata used to instantiate appropriate data-handling objects, which do use virtual inheritance, in the real-time process address space. For example, consider again the *DiiidRtParameter* class discussed above. As part of the state machine initialization process, if the input XML data file calls for a *DiiidRtParameter* object, then a corresponding data-only object of a class called *DataDiiidRtParameter* is placed in a linked list of data-only objects in shared memory. The *DataDiiidRtParameter* object contains a fixed-size string specifying that its handler is *DiiidRtParameter*. When the real-time process iterates over the list of data-only objects during subsequent initializations, it reads the data-handler specification attribute of the *DataDiiidRtParameter* object and instantiates a corresponding *DiiidRtParameter* object in the real-time process memory space.

Because the PCS real-time processes cannot recover from memory allocation errors during a shot [1], special care must be taken to ensure that no dynamic allocation of memory is done in code running on the real-time computer. Therefore, some standard C++ classes are off limits. For example, these include classes that perform automatic resizing, such as strings, maps, and vectors.

6. User Interface

A Java-based tool called ArgoUML [3] is the basis for the ONFS editor. ArgoUML is a universal modeling language (UML) [4] editor that includes UML state diagram editing capabilities. ArgoUML was chosen because it is open source, so relevant codes can be modified and embedded into a user interface that can be launched from the PCS. Also, ArgoUML natively exports state diagrams in XML format. Other editors typically export a state machine by generating source code that must be compiled, rather than loaded at runtime. Parsing an XML file during shot initialization provides an advantage over code generation since a key goal is rapid deployment of finite state machine scenarios. Additionally, by embedding the state machine information into XML, freely available open source XML parsers and validation tools can be leveraged.

7. XML Handling

A number of other groups in the fusion community, such as the Integrated Tokamak Modeling Task Force [5] and MAST [6], have made use of XML. Current work implementing the ONFS uses a simplified XML schema for proof of principle rather than State Chart XML [7], as in the ITER CODAC Conceptual Design [2]. However, it is expected that transformation between this simple schema and State Chart XML could be accomplished using Extensible Stylesheet Language Transformations (XSLT) [8] tools.

Because the use of XML in the ONFS provides great flexibility in defining ONFD/R scenarios that can be deployed rapidly, there exists a need to ensure that a particular state machine definition is valid. During initial development of a scenario in the ONFS, it is expected that the developer will test the scenario using a hardware-in-the-loop simulation [9]. For simulation purposes it will suffice to validate that the generated XML file uses proper syntax. However, for a state machine to be deployed in an operational system, additional restrictions would apply. A minimum set of ONFD/R states and triggers must be present in a deployed state machine in order to ensure device protection. The ONFS enforces such restrictions by specifying inclusion of specific elements within an XML schema description (XSD) [10] and validating XML input files against the schema description before executing it in the PCS. The schema description may, for instance, mandate the presence of a set of over-current detection algorithms in the state machine. At devices such as DIII-D, physicists will be able to modify a state machine specification during operation as long as the modified XML conforms to the schema description. The XML data will be archived with the rest of the shot setup. Use of such a state machine architecture at ITER would differ in that only a fully validated and tested state machine would be run, so that XML data files would need to be put under source control and would not be modifiable during operation.

8. Status and Future Work

Work is currently underway in implementing a prototype version of the ONFS. The major technical issues involved in loading an XML description of a finite state machine into the PCS real-time infrastructure have been resolved and a fully data-driven finite state machine design appears feasible. The ONFS will initially run in a hardware-in-the-loop simulation on the DIII-D PCS, and, once fully validated, will be tested on actual DIII-D hardware. This new design continues the trend toward greater flexibility in defining ONFD/R algorithms in the DIII-D PCS, and is expected to provide advantages in development time and visualization when developing ONFD/R scenarios for ITER.

References

[1] J.R. Ferron, et al., A flexible software architecture for tokamak discharge control systems, 16th IEEE/NPSS Symp. on Fusion Engineering, 1996, vol. 2, p. 870

[2] ITER CODAC Conceptual Design, February 2008, www.iter.org

[3] ArgoUML Project Home, http://argouml.tigris.org

[4] International Organization for Standardization, Unified Modeling Language (UML) Version 1.4.2, ISO/IEC 19501:2005

[5] G. Manduchi, et al., A universal access layer for the Integrated Tokamak Modelling Task Force, Fusion Eng. Design **83** (2008) 462.

[6] J. Storrs, G. McArdle, An XML-based configuration system for MAST PCS, Fusion Eng. Design **83** (2008) 429–433.

[7] World Wide Web Consortium, State Chart XML (SCXML): State Machine Notation for Control Abstraction Working Draft, 7 May 2009, http://www.w3.org/TR/scxml

[8] World Wide Web Consortium, XSL Transformations (XSLT) Version 1.0 Recommendation, 16 Nov 1999, http://www.w3.org/TR/xslt

[9] J.A. Leuer, et al., Development of a closed loop simulator for poloidal field control in DIII-D, 18th IEEE/NPSS Symp. on Fusion Engineering,1999, p. 531.

[10] World Wide Web Consortium, XML Schema Part 0: Primer Second Edition, 28 Oct 2004, http://www.w3.org/TR/xmlschema-0/

Acknowledgment

This work was supported by General Atomics Internal Research and Development funding.