ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV006

AUTOMATED OPERATION OF ITER USING BEHAVIOR TREE SEMANTICS

W. Van Herck[†], B. Bauvir, G. Ferro, ITER Organization, St. Paul lez Durance, France

Abstract

The inherent complexity of the ITER machine and the diversity of the ways it will be operated in different phases, like commissioning or engineering operation, poses a great challenge for striking the right balance between operability, integration and automation.

To facilitate the creation and execution of operational procedures in a robust and repeatable way, a software framework was designed and developed: the Sequencer. As a supporting framework for tasks that are mostly goal-oriented, the Sequencer's semantics are based on a behavior tree model that also supports concurrent flows of execution [1].

In view of its intended use in very diverse situations, from small scale tests to full integrated operation, the architecture was designed to be composable and extensible from the start. User interactions with the Sequencer are fully decoupled and can be linked through dependency injection.

The Sequencer library is currently feature-complete and comes with a command line interface for the encapsulation of procedures as system daemons or simple interactive use. It is highly maintainable due to its small and low complexity code base and dependencies to third party libraries are properly encapsulated.

Forecasted activities for this year include its use for the commissioning of plant systems, its incorporation as a foundation for ITER CODAC central monitoring and automation functions and the development of a graphical user interface.

INTRODUCTION

During the different phases of the ITER machine, many operational procedures will need to be defined, verified and executed in a traceable and maintainable way.

During regular operation, these procedures are mainly associated with Operational Tasks, e.g. venting, baking, etc. These procedures will very likely need to evolve in the lifetime of ITER, either by:

- changing the content and order of the different steps,
- changing parameters that influence the execution of the steps, or by
- increasing the automation of the execution by removing unnecessary user interactions as the procedure becomes more mature and trusted.

During the testing, calibration and commissioning of plant systems, local procedures will be defined to carry out tasks in a repeatable and traceable way. These procedures will likely be even more often adapted.

The Sequencer is a software tool meant to facilitate the creation, adaptation, approval and execution of those procedures. It also defines the format of these procedures, allowing version control and easy traceability. It provides a means to replace procedural documents with instructions and integrate automatic verification.

It will rely on the configuration, monitoring and control functions of the Supervision and Automation System (SUP) to carry out certain actions defined in the procedure. The Sequencer will thus need to adhere to the protocols defined by SUP.

This tool also has to provide the flexibility to be deployed and used in different environments:

- SUP interfacing with the Sequencer to execute a procedure,
- local activities using a standalone GUI,
- creation and editing can be done in an offline environment.

This paper describes the design and implementation choices and provides an outlook to future enhancements or extensions of the framework.

DESIGN

The language of operational procedures, whether for engineering operations or commissioning tasks, formulates certain goals that need to be achieved at each step. The Sequencer framework was based on behavior tree semantics since, as a formal language, it is generally better adapted to express such goal-oriented procedures than for example finite state machines. Goals can be described by a variety of rules that apply to sub-goals, such as: a Sequence succeeds if all its sub-goals succeed. Another advantage of behavior tree semantics is that it can express parallelism in a natural way. This is often required in operational procedures, where certain goals need to be achieved while maintaining conditions on the machine.

A procedure in the Sequencer library consists of tree structures of instruction objects and a workspace, providing access to variables that need to be shared or communicated between instructions. A procedure is executed by sending 'Execute' commands to a root instruction in the tree until it indicates failure or success. The root instruction is responsible for propagating this 'Execute' command further down the tree.

Figure 1 shows a simplified example of how the goal of starting up a plant system consists of either successfully activating the system (left branch) or, if that fails, executing steps to recover the system to a known and safe state (right branch). Each of those goals can in turn be expressed as a number of actions that need to be carried out sequentially.

[†] Walter.Vanherck@iter.org

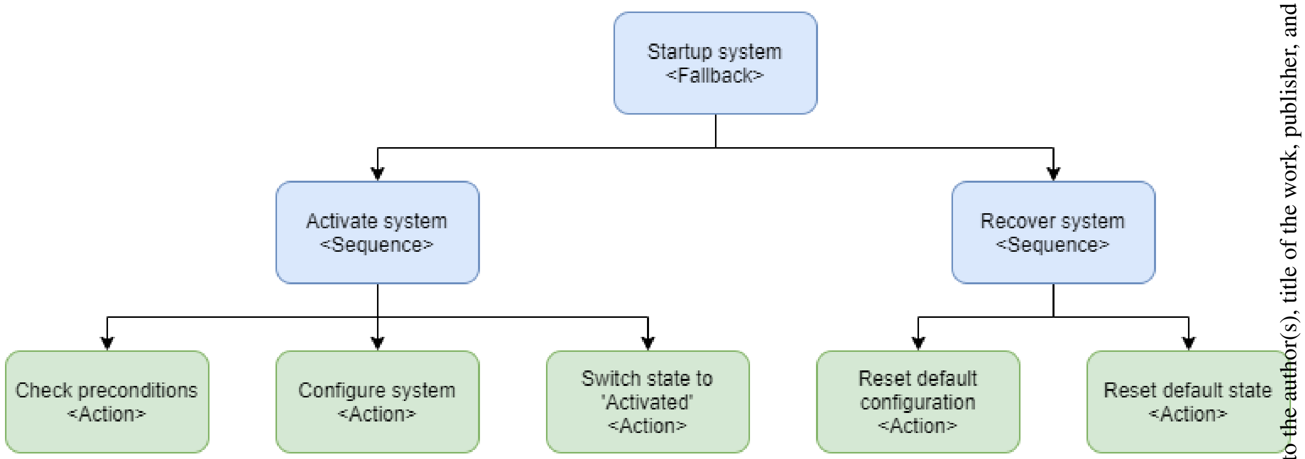

Figure 1: Example procedure for starting up a plant system.

The 'Action' instructions typically contain the domain specific commands or goals, while the compound instructions, such as 'Sequence' or 'Fallback', are domain agnostic.

The core library of the Sequencer framework contains generic instructions and variable types. Domain specific behavior or general extensions can be provided by a plugin mechanism, where extra instruction or variable types can be defined. By loading different plugins, the Sequencer framework can be used in different environments.

The framework also supports serialization of procedures in a human readable format (currently only XML is supported) to allow for version control and traceability.

IMPLEMENTATION AND QUALITY ASSURANCE

To support usage of the Sequencer framework for varied operational tasks and in different environments, composability and extensibility are key. Composability is provided by the basic building blocks of behavior trees: actions/goals are aggregated into compound goals. The Sequencer framework also allows to include a whole instruction tree from either the same or another procedure file. This allows designers of operational procedures to build complex procedures from ever simpler building blocks.

Extensibility is provided by a plugin mechanism and a generic API for instructions and variables. Behavior can then be customized by implementing instructions and variables that comply with this API and then exposing them as a plugin library.

The framework is implemented in C++11 and is highly maintainable due to its small codebase (currently ca. 5k lines of code) and the amount of decoupling between different components. The source code passes the current criteria for CODAC Software Integrity Level 1 (>95% unit test coverage, no major/critical/blocker issues).

User interaction with procedure execution, e.g. to allow for step-by-step execution or get visual feedback on the progress, is provided by a pure interface and concrete implementations of this user interface are injected into the framework (see Fig. 2). The core framework currently provides two such implementations:

- sequencer-cli: a command line interface to run procedure files with configurable amount of verbosity,
- sequencer-daemon: a non-interactive executable targeted to be run as a daemon process.

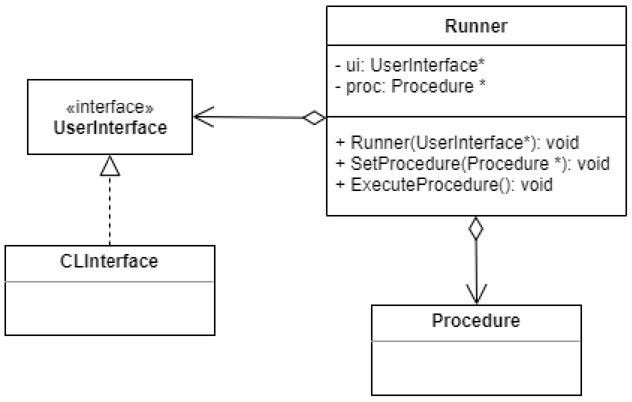
Runner
- ui: UserInterface*
- proc: Procedure *
+ Runner(UserInterface*): void
+ SetProcedure(Procedure *): void
+ ExecuteProcedure(): void
«interface»
UserInterface
CLInterface
Procedure

Figure 2: Constructor injection of UserInterface into the execution engine (Runner). Here, a command line interface (CLInterface) is shown as an example.

FORWARD LOOK

It is expected that the framework, but also its possible use cases, will continue to evolve due to changing and extending requirements. For reasons of compatibility and stability, such changes need to minimize the impact on the core library and should therefore be implemented in plugins whenever possible.

For the coming year, a number of use cases and further development activities have been identified:

- The framework will be used during the commissioning of plant systems: procedures will be developed to commission and automate routine tasks, such as starting up or shutting down the plant system. Concretely, it will be used for the commissioning of the Reactive Power Compensation and Harmonic Filter system and for the Site Acceptance Test and commissioning of the Magnetics Diagnostics.
- The Sequencer will also be evaluated for use in monitoring the ITER plant and automating central control

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV006

system activities. As part of these activities, a plugin is being developed for SUP specific instructions (e.g. for plant system configuration and verification).

- Lastly, a graphical user interface (GUI) is being developed that will enable users to easily create and edit Sequencer procedures. This GUI will also allow to interactively execute such procedures in multiple ways: step-by-step execution, breakpoints at given instructions, etc.

REFERENCES

[1] R. G. Dromey, “Formalizing the Transition from Requirements to Design”, in *Mathematical Frameworks for Component Software Models for Analysis and Synthesis*, Nov. 2006, pp. 173-205. doi:10.1142/9789812772831_0006