Experiment planning using high-level component models at W7-X

Marc Lewerentz[a,*], Anett Spring[a],

Torsten Bluhm[a], Peter Heimann[b], Christine Hennig[a], Georg Kühner[a], Hugo Kroiss[b], Johannes G. Krom[a], Heike Laqua[a], Josef Maier[b], Heike Riemann[a], Jörg Schacht[a], Andreas Werner[a], Manfred Zilker[b]

[a]*Max-Planck-Institut für Plasmaphysik, EURATOM Association, Teilinstitut Greifswald, Wendelsteinstraße 1, D-17491 Greifswald, Germany*

[b]*Max-Planck-Institut für Plasmaphysik, EURATOM Association, Boltzmannstraße 2,D-85748 Garching, Germany*

[*]*Corresponding author; Tel.: +49-3834-88-2762; Fax: +49-3834-88-2509; E-mail address: marc.lewerentz@ipp.mpg.de*

The superconducting stellarator Wendelstein 7-X (W7-X) is a fusion device, which is capable of steady state operation. Furthermore W7-X is a very complex technical system. To cope with these requirements a modular and strongly hierarchical component-based control and data acquisition system has been designed.

The behavior of W7-X is characterized by thousands of technical parameters of the participating components. The intended sequential change of those parameters during an experiment is defined in an experiment program. Planning such an experiment program is a crucial and complex task. To reduce the complexity an abstract, more physics-oriented high-level layer has been introduced earlier. The so-called high-level (physics) parameters are used to encapsulate technical details.

This contribution will focus on the extension of this layer to a high-level component model. It completely describes the behavior of a component for a certain period of time. It allows not only defining simple value ranges but also complex dependencies between physics parameters. This can be: dependencies within components, dependencies between components or temporal dependencies.

Component models can now be analyzed to generate various views of an experiment. A first implementation of such an analyze process is already finished. A graphical preview of a planned discharge can be generated from a chronological sequence of component models. This allows physicists to survey complex planned experiment programs at a glance.

Keywords: experiment planning; high-level component behavior; component model; technical and temporal dependencies; segment control system

1. Introduction

The superconducting stellarator Wendelstein 7-X (W7-X) is a highly complex technical system. To cope with this complexity three basic design decisions have been made for the W7-X segment control system. A project – component hierarchy has been implemented [1].

The behavior of W7-X is characterized by thousands of technical parameters of the participating components. The intended sequential change of those parameters during an experiment is kept in so called (low-level) segments. Segments are the elementary temporal parts to define an experiment program from. A detailed description of the segment control concepts is given in [2].

Planning an experiment program is a crucial and complex task. To reduce the complexity an abstract, more physics-oriented high-level layer has been introduced earlier [3]. The so-called high-level (physics) parameters are used to encapsulate technical details. In high-level segments the high-level behavior (also called high-level task) of the components is described using high-level parameters for a certain period of time. Via transformation functions the high-level segments can be transformed to low-level segments, which can be executed by the participating components.

This contribution will focus on the so-called high-level model layer. It adds a component model instance on top of each high-level component task.

2. Motivation

author's email: marc.lewerentz@ipp.mpg.de

Experiments are executed using low-level segments and parameters. The executability of the low-level segments is checked short before and during the experiment [4].

Because it is very complex to plan an experiment using low-level segments and parameters this is done on the abstract high-level layer. But planning experiments is even on this layer a challenging task. During the planning process arises the question: Will the currently planned experiment be feasible?

This has to be checked already at planning time on the high-level layer. But how could one create a most likely feasible experiment program?

The feasibility is determined by certain constraints which have to be complied. These constraints can arise from (1) technical dependencies such as limits, resources, transitions, etc. or from (2) logical / temporal dependencies, which could be preparation phases, a calibration before use of a diagnostic, a post processing, etc.

To use those dependencies to assure the feasibility of an experiment program they (1) must be known, (2) must be possible to be formulated in mathematical terms and (3) must be possible to be automatically evaluated.

2.1 Case study

The subsequently described concepts will be illustrated using a case study. The behavior of the component gas inlet at the CoDaC prototype WEGA is determined by several high-level parameters. The gas inlet has 2 valves with different maximal gas flows. They could be opened or closed. For each opened valve a gas type must be chosen. The maximal gas flow depends on the chosen gas type. The valves allow a higher maximal gas flow for certain gas types. Furthermore the desired actual gas flow has to be set.

3. Component Model Framework

Component Models are an abstract, high-level description of a component of a project, such as Wendelstein 7-X or our CoDaC prototype WEGA. They add the high-level knowledge of dependencies and constraints. This knowledge is successively gained by experimenting or is already known, e.g. from technical properties.

The only information available from the high-level parameters of the component task is their value and static predefined limits. The component models of course allow keeping the values for the high-level parameters also. Those values can be constrained, not only by simple limits, but by complex value ranges. Those value ranges can dynamically change, because they could be affected by a dependency.

Furthermore stateful parameters are introduced. They allow defining replacement values for certain states. Case study: If a valve of the gas inlet is closed the corresponding parameters gas flow and gas type aren't set. The actual values, which are of course 0 and undefined, used in the low-level task, are hitherto hidden in the transformation function (see Figure 1). To allow a generic analysis of the high-level behavior of a component this implicit knowledge must be made visible.

The component model layer is built on top of the already existing high-level layer (Figure 1).

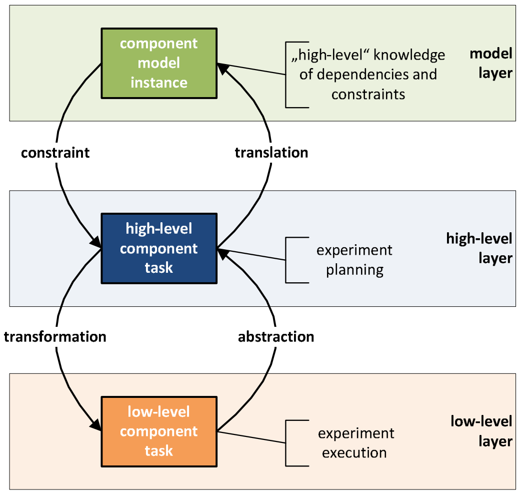

Figure 1 Different layers at experiment planning time.

4. Ingredients

The high-level layer consists of a high-level descriptor for each component, high-level tasks and transformation functions. The descriptor holds the information of the used parameter types and their structuring. The corresponding tasks represent various sets of parameters with their actual values.

The ingredients of the component model framework and activities between them are shown schematically in Figure 2.

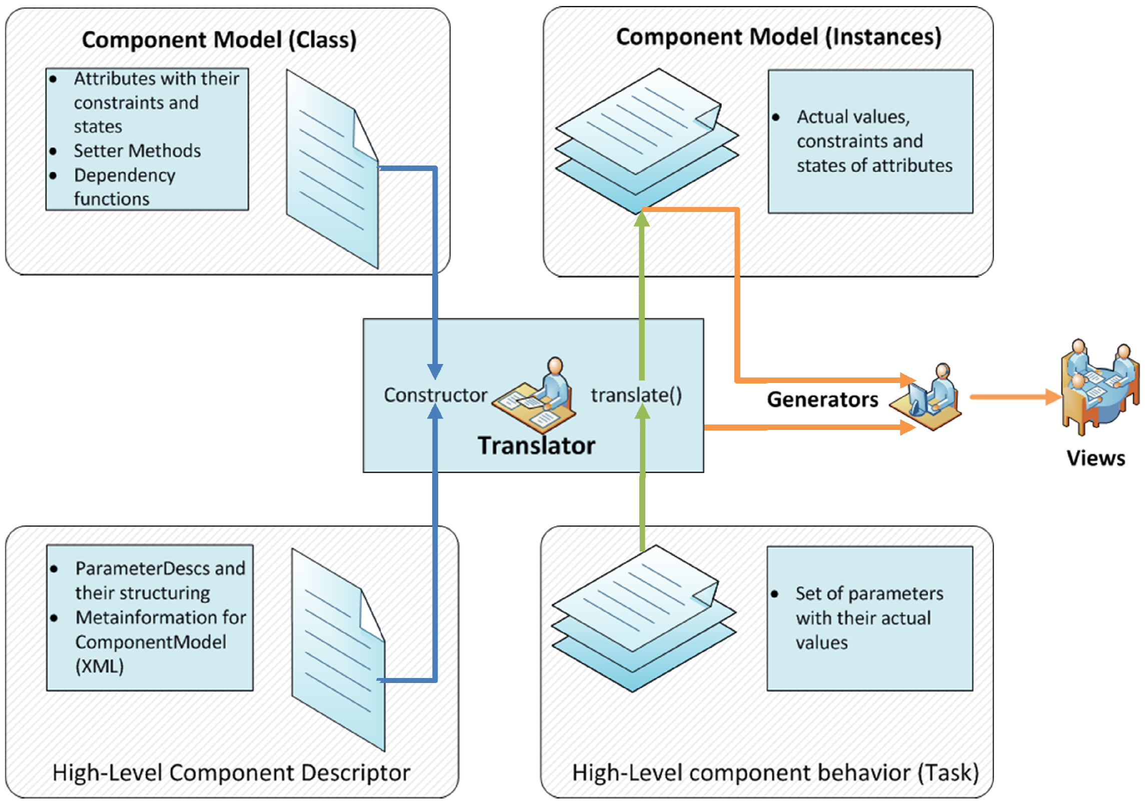
Component Model (Class)
Attributes with their constraints and states
Setter Methods
Dependency functions
Component Model (Instances)
Actual values, constraints and states of attributes
Constructor
translate()
Translator
Generators
Views
ParameterDescs and their structuring
Metainformation for ComponentModel (XML)
High-Level Component Descriptor
Set of parameters with their actual values
High-Level component behavior (Task)

Figure 2 Ingredients of the component model framework

4.1 Component Model

The equivalents to the descriptors on the high-level layer are the component model classes on the model layer. They are implemented as Java classes and follow the JavaBeans specification [5]. Each class contains a set of attributes specific for that component. The attributes are the equivalent to the parameters from the descriptor, but could be constrained or stateful, as mentioned in chapter 3. Furthermore there are setter methods to change the value or state of the attributes.

If the value, state or constraint of an attribute has changed an event is fired.

Case study: the component model class of the gas inlet has the attributes: *magneticValveGI1*, *magneticValveGI2, gasTypeGI1* (stateful), *gasTypeGI2* (stateful), *gasFlowGI1* (stateful and constrained) and *gasFlowGI2* (stateful and constrained). GI1 or GI2 determines the chosen valve of the gas inlet component.

4.2 Translator

For each component a (component) translator instance is created using the static information from the corresponding component model class and the parameter types and the structuring from the high-level component descriptor.

Additionally some meta-information is stored in the descriptor which is read by the translator. This comprises the corresponding component model, the available attributes and dependencies. Furthermore settings for the creation of views, e.g. a graphical preview for experiment programs (see [6]), by a generator (see following chapter) is found here.

Case study: a list that keeps all attributes defined in the component model class of the gas inlet is generated.

The main purpose of the component translator is to translate high-level component tasks to instances of the corresponding component model class. These instances contain the actual values, constraints and states of the attributes.

Furthermore the translator creates dependency objects which are generated from the meta-information of the high-level component descriptor. They listen to change events of their input variables and recalculate the output variable using the defined dependency function. Thus dependent attribute values, states and constraints are automatically updated.

Case study: the translator for the gas inlet iterates over all high-level parameters of a given high-level task and calls the corresponding setter methods to set the actual values and states (if available) for all model attributes. If the value of an attribute with a defined dependency is changed the output variable is updated.

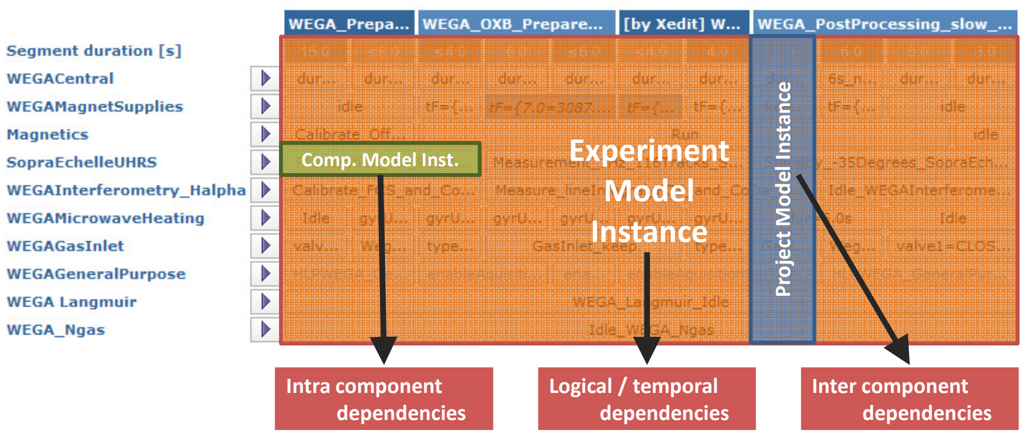

Figure 3 Model types and handled dependencies (Overlay on experiment program in Xedit)

Fehler! Verweisquelle konnte nicht gefunden werden. shows instances of the three different model types which all have special translators. They are depicted as overlay on an experiment program structure in the experiment program editor Xedit. Each box in the background represents a task for one of the components on left side. A vertical aggregation of tasks is a segment.

A component model instance reflects the task of a single component. The corresponding component translator handles all intra component dependencies. A project model instance is an aggregation of component model instances of a project, which is for example W7-X or WEGA. It can additionally define attributes on project level, e.g. which components must be mandatory available during this phase of the experiment and which are optional. The corresponding project translator handles inter component dependencies. The experiment model instances contain a matrix of component model instances inclusive the segment switches defined in an experiment program. The experiment translator handles logical / temporal dependencies.

4.3 Generator / View

Generators allow analyzing information from the component model instances and the corresponding translators generically to produce different views. Here is where the benefit from the component model framework emerges. At WEGA a graphical preview generator (see Figure 4), an automatic segment and task name generator and a constraints checker are already in use. Because of their generic nature these generators can be used at W7-X without modification.

Case study: for the view depicted in Figure 4 the attributes *gasFlowGI1* and *gasFlowGI2* have been selected in the meta-information defined in the descriptor. The valve 1 is closed during the complete experiment. This means that no value is set for *gasFlowGI1*. It is in state *NOT_SET*. Therefore it is automatically removed from the experiment preview to maximize the clarity of the graphical preview. Only the time course (green curve) of *gasFlowGI2* is shown (beside other characteristic values for the planned experiment). The course of the values is determined by analyzing the values from all component model instances over the complete span of the experiment. An in-depth description of the graphical preview can be found in [6].

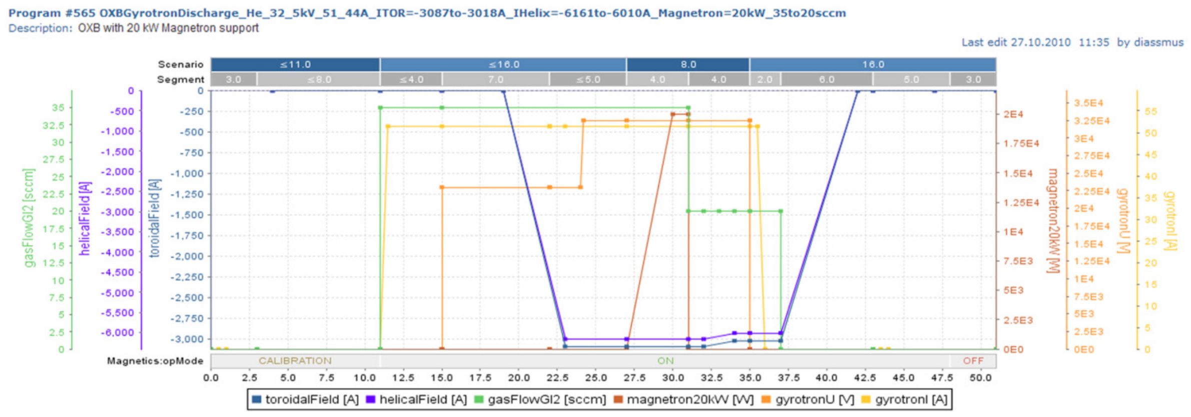

Figure 4 Rendering of a graphical preview in Xedit (more detailed information in [6]): characteristic values over time for a planned experiment is shown;

5. Dependencies

Dependencies are defined by a set of input variables, one output variable and a dependency function. Input variables could be any values, states or constraints of model attributes. They are used as input parameters of the dependency function, which is currently implemented in Java. But using a script language here would allow changing a dependency function without the need to re-deploy it. The result of the dependency function is written to the output variable, which again could be any attribute value, state or constraint.

5.1 Intra component dependencies

Dependencies between attribute values, states or constraints of a single component are called intra component dependencies. They are already routinely used at the CoDaC prototype WEGA.

Case study: A simple example is the upper limit of the gas flow which depends on a certain calibration factor of the chosen gas type and the maximal flow of the chosen valve.

5.2 Inter component dependencies

Dependencies between attribute values, states or constraints of different components of a project are called inter component dependencies. First implementations for some examples at WEGA are currently in process.

Case study: A simple example is a dependency between the wavelength of the component SOPRA Echelle Spectrometer and the gas type at the gas inlet. A more complex one is the requirement of a minimal gas flow if the plasma heating is active. It depends on the heating scenarios such as Magnetron only, Gyrotron X2 and OXB and the chosen gas type, typically Helium, Argon or Deuterium.

5.3 logical / temporal dependencies

Dependencies that require a certain order of tasks or segments are called logical / temporal dependencies. Their expected quantity for long-term discharges is very high. For multi-experiment discharges it will increase even further, because a certain state of the components might be required before the start of the next experiment within the discharge. The concepts for this type of dependencies are still in development.

Case study: a very simple, but very important experiment wide dependency is that it is not allowed to set different gas types for the same valve. It is simply technically impossible to do this at the WEGA. Running such an experiment will always fail. Consistently the attributes *gasTypeGI1* and *gasTypeGI2* should be attributes of the experiment model. This will be implemented as soon as the conceptional design for experiment models and logical / temporal dependencies is finished.

One could easily find other examples for logical / temporal dependencies. E.g. a laser that has to be calibrated before it is ready for use. The maximal duration of use could be a few minutes before it has to get calibrated again.

6. Conclusion and Outlook

Component models support the creation of feasible experiment programs already at experiment planning time. They are easily upgradeable with the growing knowledge about the physical and technical constraints and dependencies. By analyzing component models by generators various views of an experiment can be generated. Thus it is possible to compile and present crucial information to the experiment planner at a glance.

The implementation of inter component dependencies is currently under development and will be available for use soon.

The next step will be the differentiation between hard and soft constraints. At the moment all defined constraints are hard constraints. They prevent the creation of illegal experiment programs, which would fail or would knowingly damage the experimental machine. On the other hand there are a lot of soft constraints. They should warn the experiment planner that the usual value ranges are left. But soft constraints don’t prevent experimenters from experimenting. They only mark extraordinary value ranges.

Acknowledgments

The developers would like to thank the WEGA team for the close co-operation, intense testing and many fruitful discussions.

References

[1] J. Schacht, H. Laqua, M. Lewerentz, I. Müller, S. Pingel, A. Spring, A. Wölk, Overview and status of the control system of WENDELSTEIN 7-X, Proceedings of the 24th Symposium on Fusion Technology, October 2007, Fusion Engineering and Design, 82 (988-994)

[2] H. Laqua, H. Niedermeyer, J. Schacht, A. Spring, Real-time software for the fusion experiment WENDELSTEIN 7-X, 5th IAEA TM on Control, Data Acquisition and Remote Participation for Fusion Research, July 2006, Fusion Engineering and Design, 81 (15-17)

[3] H. Riemann et al., From a physics discharge program to device control – linking the scientific and technical world at Wendelstein 7-X, Proceedings of the 25th Symposium on Fusion Technology, June 2009, Fusion Engineering and Design, 84 (1598-1601)

[4] H. Laqua et al., Resource Checking and Event Handling within the W7-X Segment Control Framework, this issue

[5] G. Hamilton (Editor), JavaBeans specification, http://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html

[6] A. Spring, Marc Lewerentz et al., A W7-X experiment program editor – a usage driven development, this issue