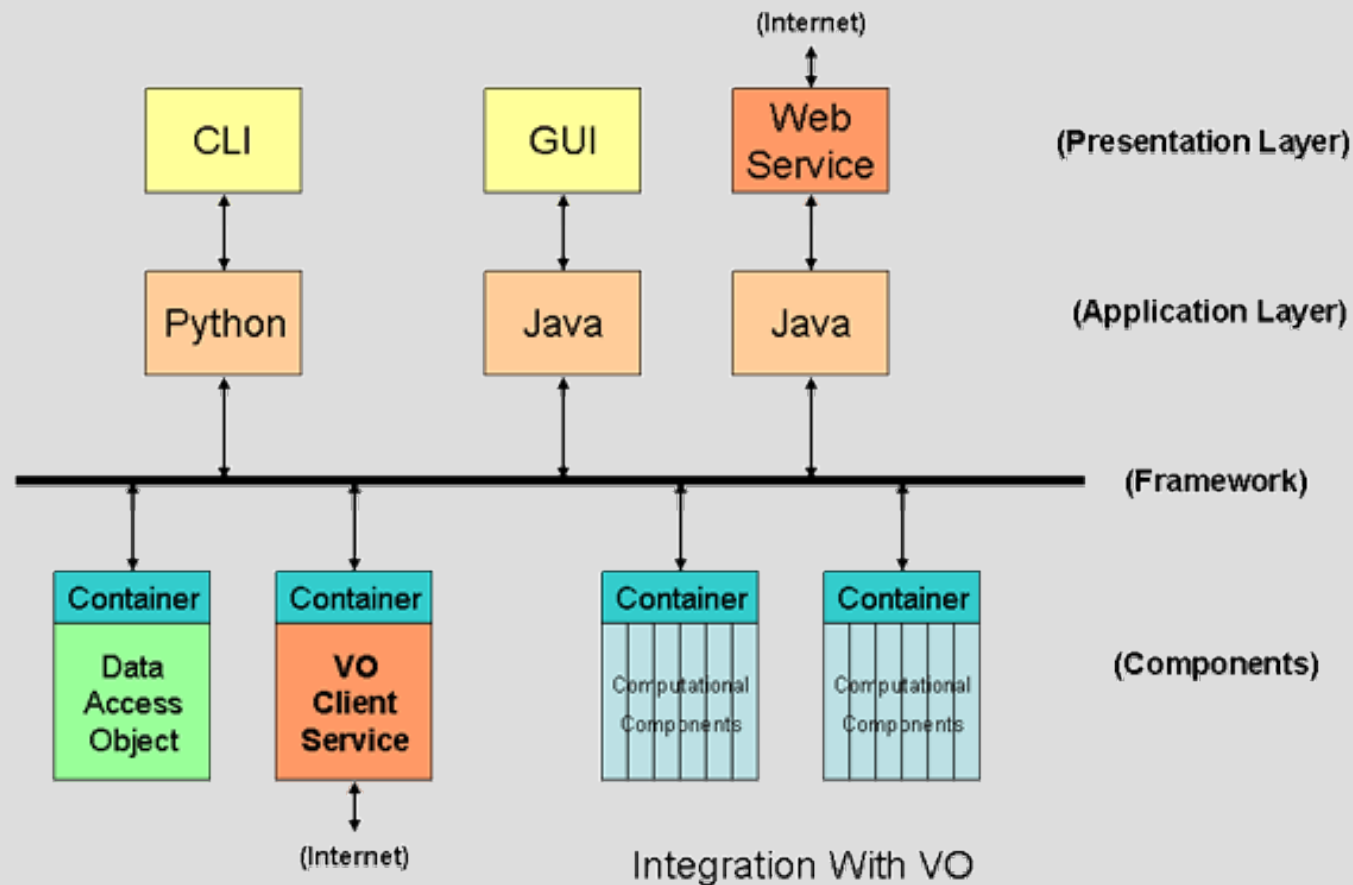


Introduction

- The Component/ Container model

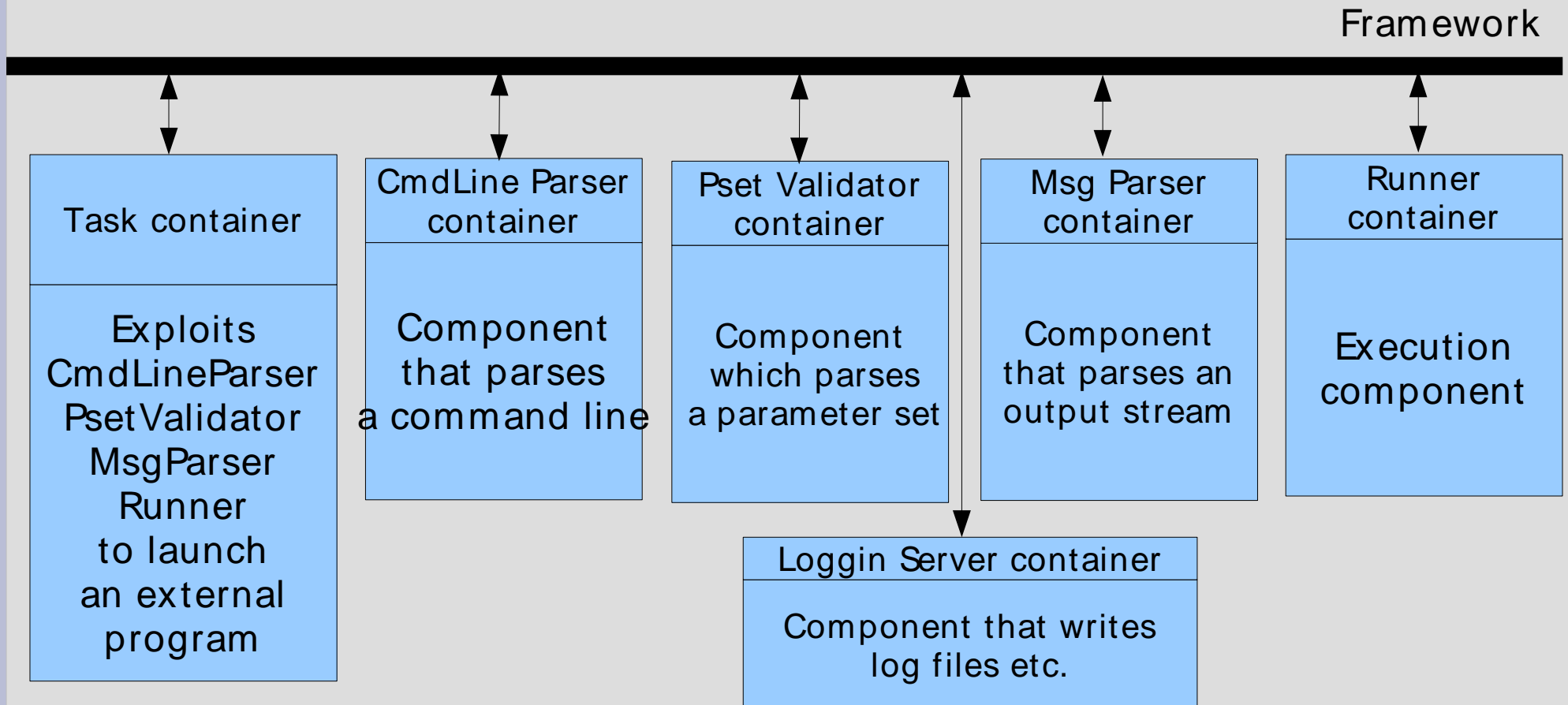


Introduction

- Our focus
 - Task component/ container
 - A task is a service of which the execution is driven by a parameter set
- Task container
 - Our implementation involves other general container services
 - Command- line parser
 - Parameter set validator
 - Messaging parser
 - Runner
 - Logging server
- Recent work
 - From practical implementation, definition of services interfaces has been derived

Introduction

- The Task container

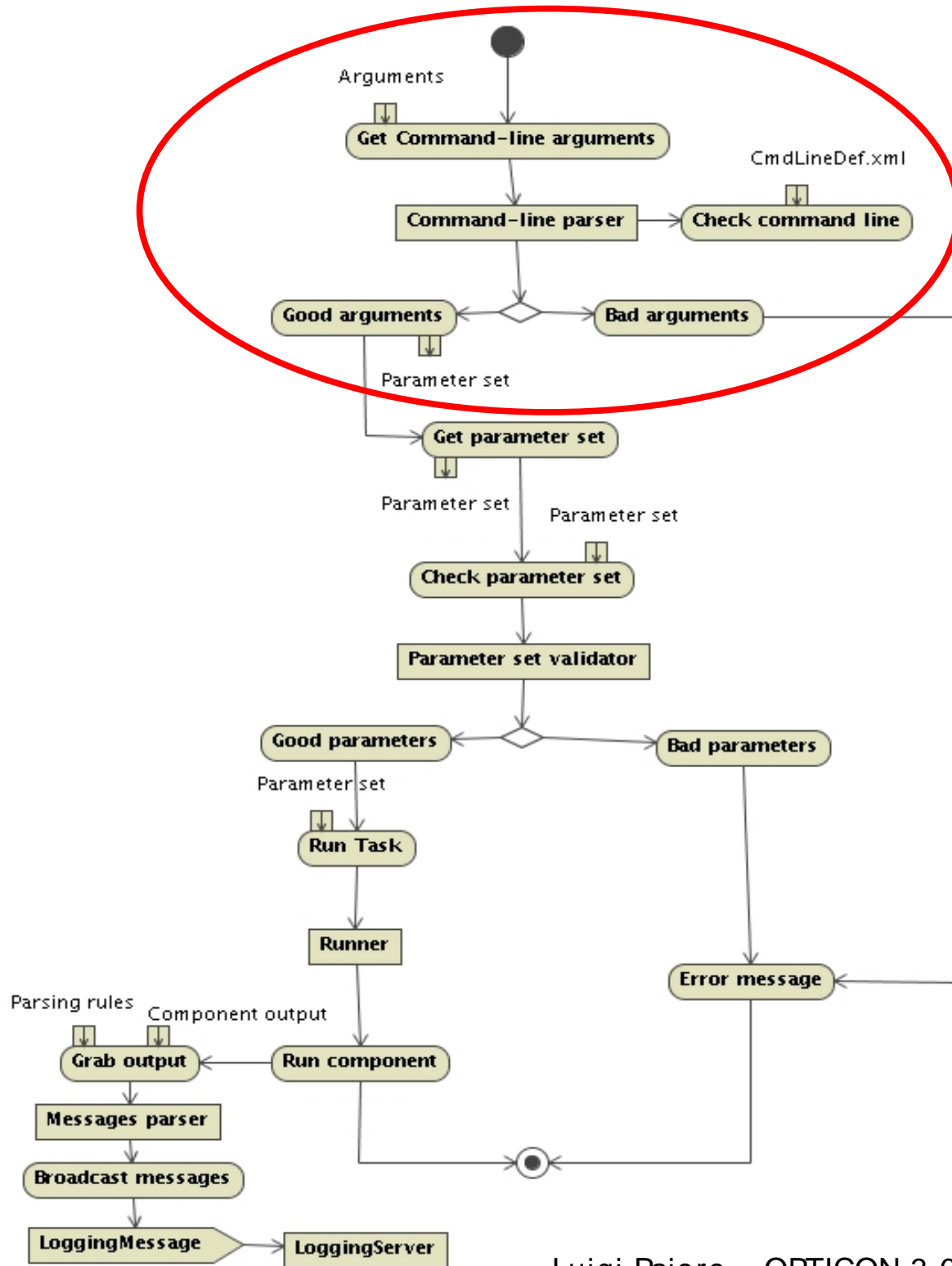


The task container

- The `fase.container.task.Task` interface
 - `+runCommandLine(args:String)`
 - `+run()`
- The `fase.common.Properties` interface
 - `+get(keyword:String):Variant`
 - `+set(keyword:String, value:Variant)`
 - `+getAll():Dictionary`

The command-line parser

- Assuming one launches the task from the command line, the arguments have to be parsed and translated as actual parameters



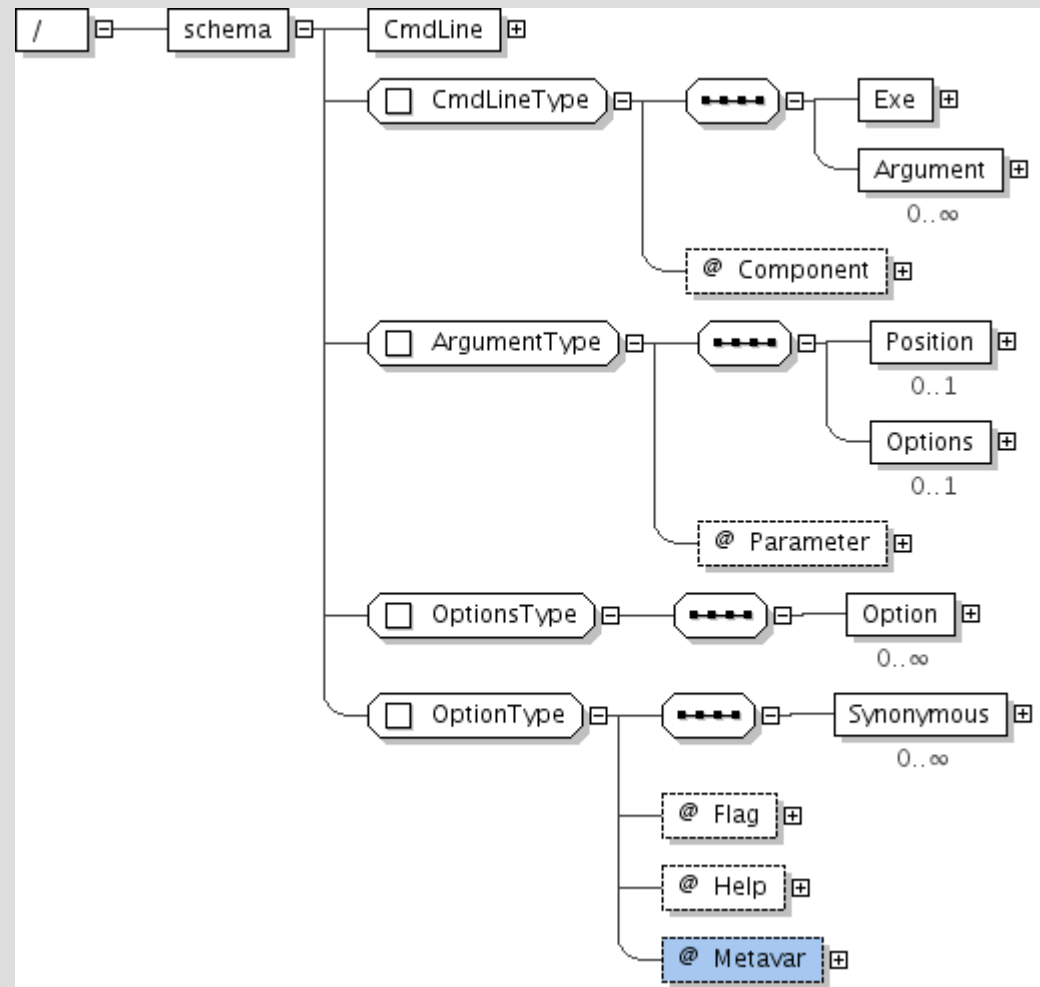
The command- line parsing

- The `fase.container.task.CmdLineParser` interface

<code>+getExeName() : String</code>	Return the executable name of the current component, as defined within the <code><Exe></code> tag of the <code>CmdLineDef.xsd</code> XML instance file
<code>+getXMLFile() : String</code>	Get the current <code>CmdLineDef.xsd</code> XML instance file of the related task component.
<code>+parse(args : String) : Dictionary</code>	Parse a command-line basing on the current <code>CmdLineDef.xsd</code> instance.
<code>+setXMLFile(xmlFile : String)</code>	Set the <code>CmdLineDef.xsd</code> XML instance file of the related task component.

The command-line parsing

- The command line parsing rules
- Based on XML



The parameters validation



- Get the parameter set (e.g. from the command-line parser)
- Validate them using a parameter set definition file

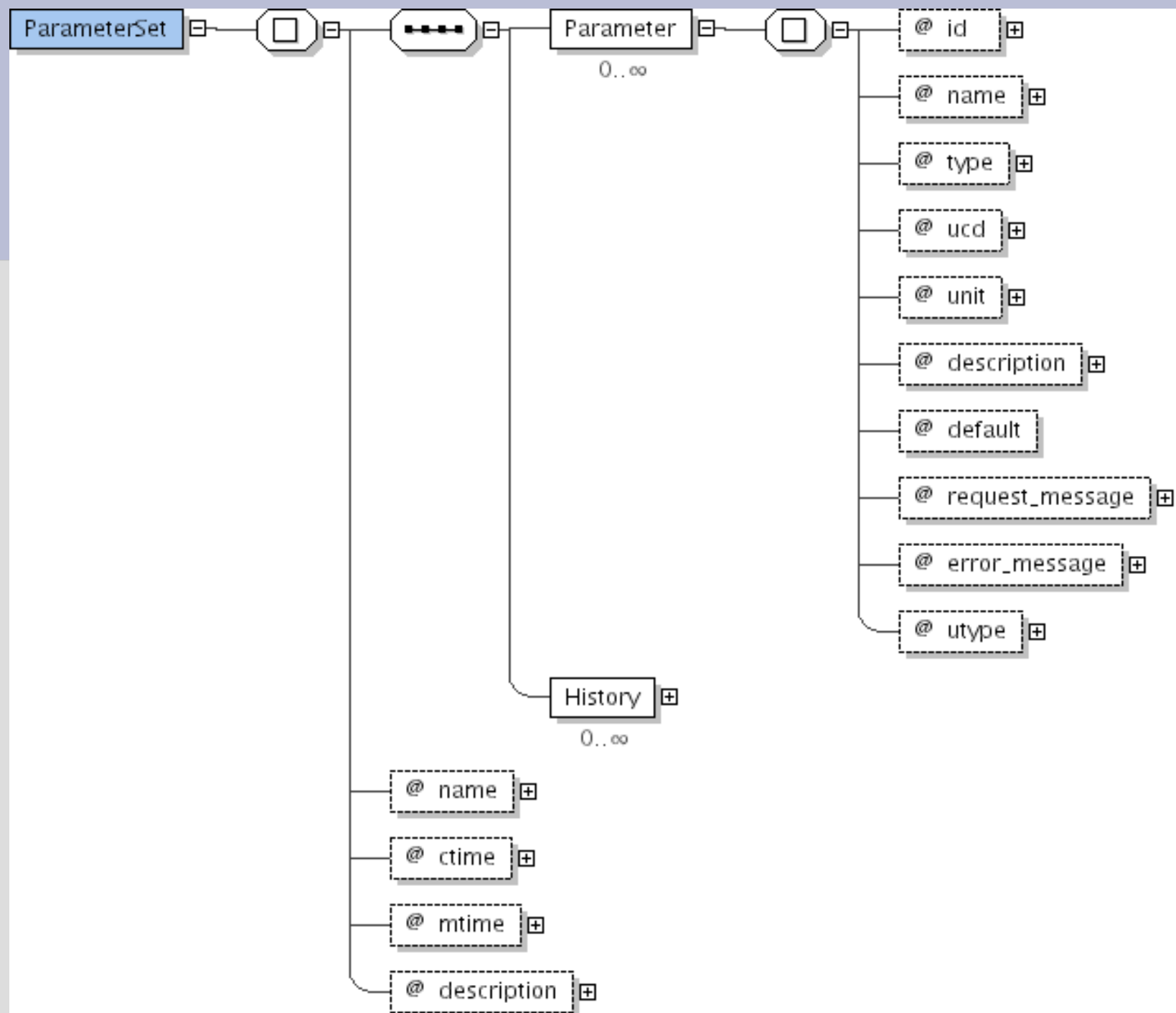
The parameter set validation

- The fase.container.task.ParameterSetValidator interface

+getDefinitionFile() : String	Get the PsetDef.xml file of the related task component.
+getParametersSetDefinitionAsDictionary() : Dictionary	Return the parameter set definition a dictionary.
+getParametersSetDefinitionAsXML() : String	Return the parameter set definition as an XML string.
+getParametersValuesAsDictionary(default : boolean) : Dictionary	Returns a string with the current parameter set values in a dictionary.
+getParametersValuesAsXML(default : boolean) : String	Returns a string with the current parameter set values in a suitable XML format.
+setDefinitionFile(defFile : String)	Set the PsetDef.xml file of the related task component.
+setParametersValuesFromDictionary(values : Dictionary)	Loads the pset values from the passed dictionary.
+setParametersValuesFromXML(xml : String)	Loads the pset values from an XML string.
+setParametersValuesFromXMLFile(file_path : String)	Loads the pset values stored in an XML file.
+validate(defFile : String)	Validates the current parameter set values over the definition file of the Component.
+validateDictionary(dictionary : Dictionary, defFile : String)	Validates the dictionary of parameter set values over the definition file of the Component.
+validateSingleValue(keyword : String, value : Variant, defFile)	Validates the single parameter value over the definition file of the Component.
+validateXML(xml : String, defFile : String)	Validates the XML parameter set values over the definition file of the Component.
+validateXMLFile(file_path : String, defFile : String)	Validates the XML file parameter set values over the definition file of the Component.

The parameter set

- The parameter set data model
 - A collection of single parameters
 - With a set of properties/ attributes
- Serialization
 - Simple
 - Dictionary
 - parameter name
 - value
 - Complex
 - VOTable (1.1)
 - Others?
 - ALMA prototype



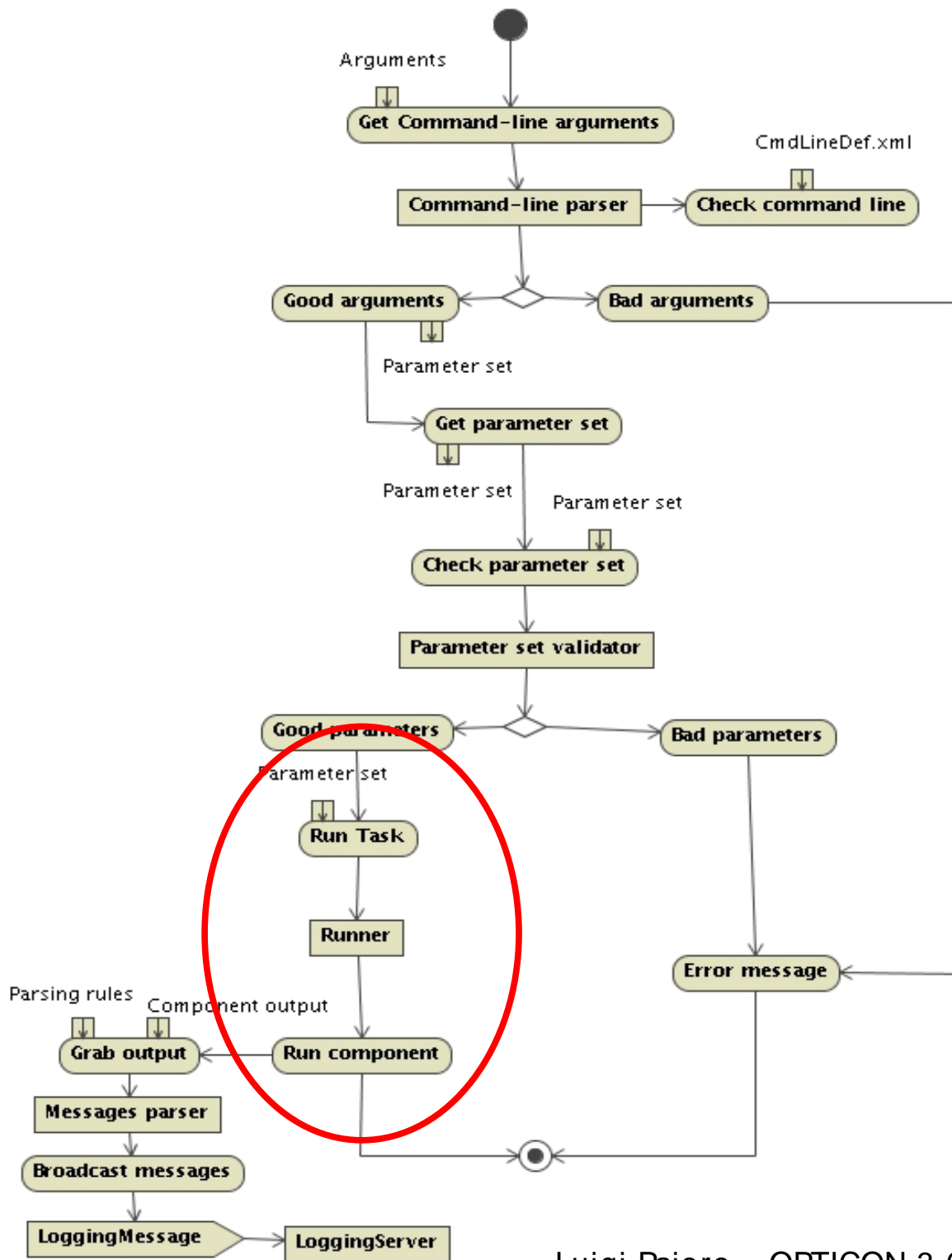
The parameter set

- The parameter set UTYPE
 - Used by the framework to know how deal with them
 - UCD+ style
 - Defined so far:

<code>in</code>	<code>Input</code>
<code>out</code>	<code>Output</code>
<code>inout</code>	<code>Input and output</code>
<code>file</code>	<code>File</code>
<code>param</code>	<code>General parameter</code>
<code>mandatory</code>	<code>Mandatory</code>
<code>optional</code>	<code>Optional</code>
<code>default</code>	<code>Default value</code>
<code>prompt</code>	<code>Prompt message</code>

The Runner

Execution engine
Our
implementation
- sh shell



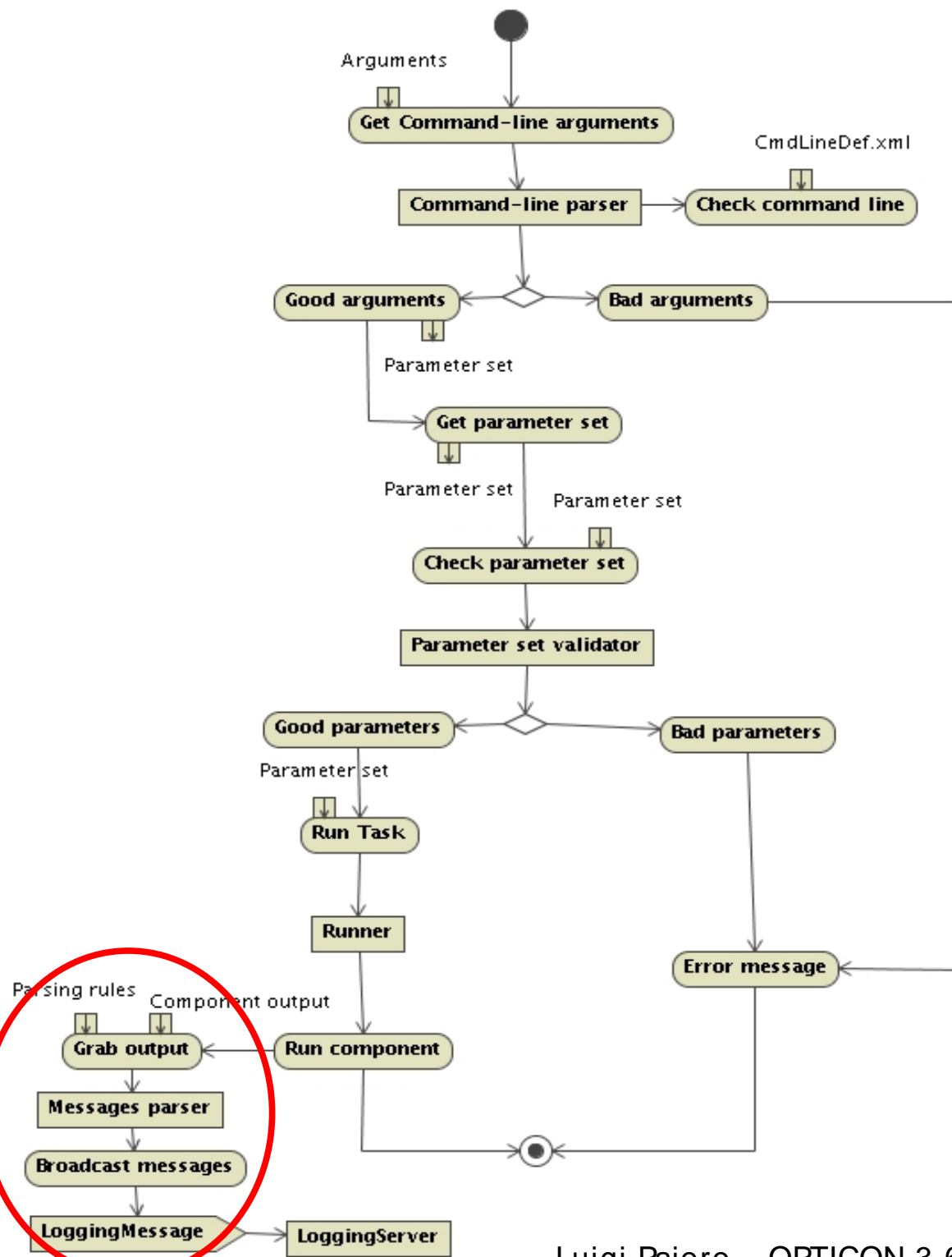
The Runner service

- The `fase.container.task.Runner` operations

<code>+getPid() : int</code>	The process ID of the child process.
<code>+poll() : int</code>	Returns -1 if child process hasn't completed yet, or its return code otherwise.
<code>+read(buffer_size : int)</code>	Read at most <code>buffer_size</code> bytes from the child process output like a file (less if the read hits EOF before obtaining size bytes).
<code>+readline(buffer_size : int)</code>	Read one entire line from the child process output like a file.
<code>+run(cmdline : String)</code>	Run the command line in a separated thread.
<code>+wait()</code>	Waits for and returns the status code of the child process.

The messages parser

- Grab the component output
- Format standard logging messages



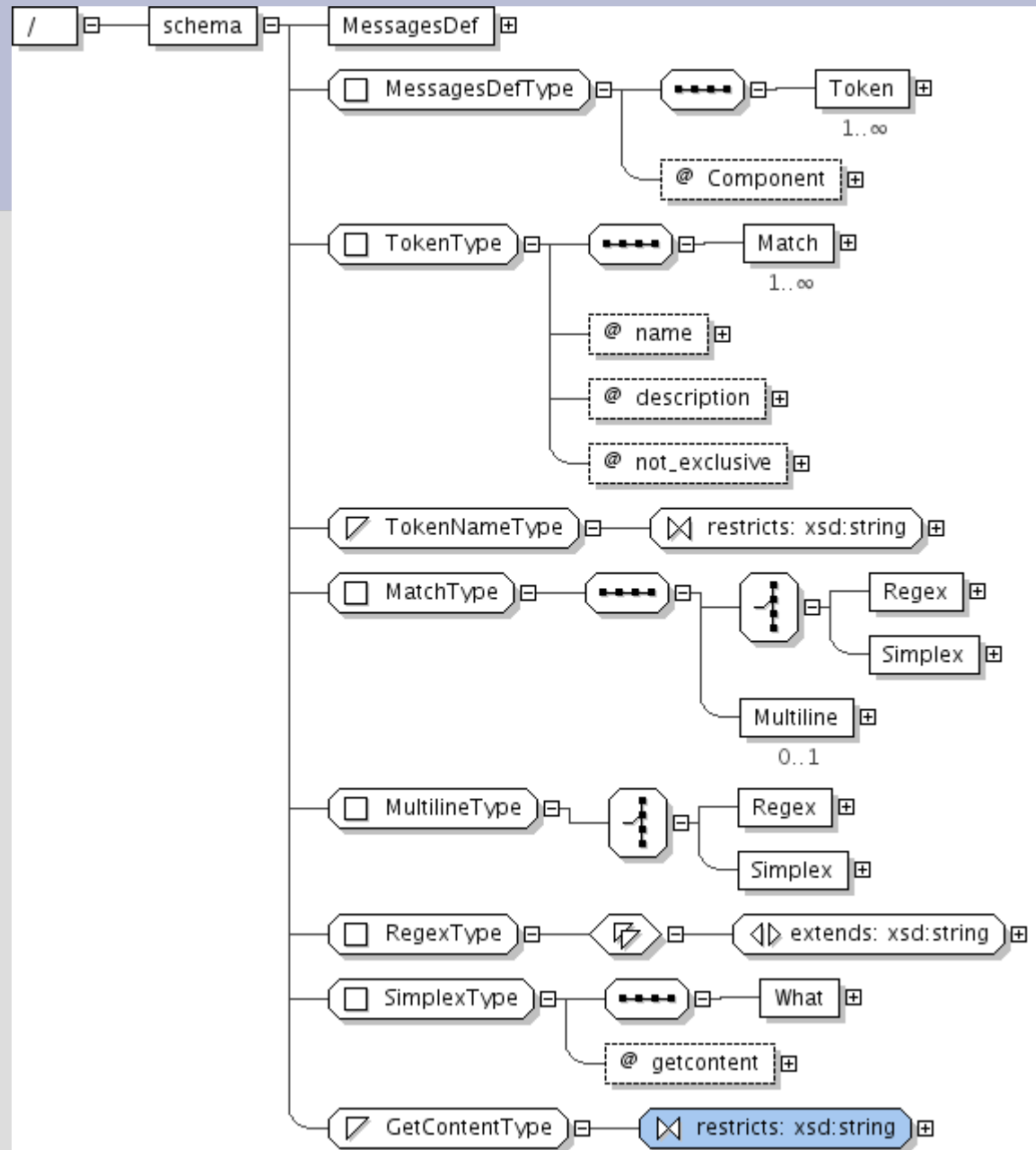
The messaging parsing

- The `fase.container.task.MessagesParser` interface

<code>+getFileDefinitionPath() : String</code>	Return the MessagesDef.xsd instance file path.
<code>+grab(cmdline : String)</code>	Execute a command line a grab the output basing on the related MessagesDef.xsd instance file.
<code>+load(file : String)</code>	Load a MessagesDef.xsd instance file.

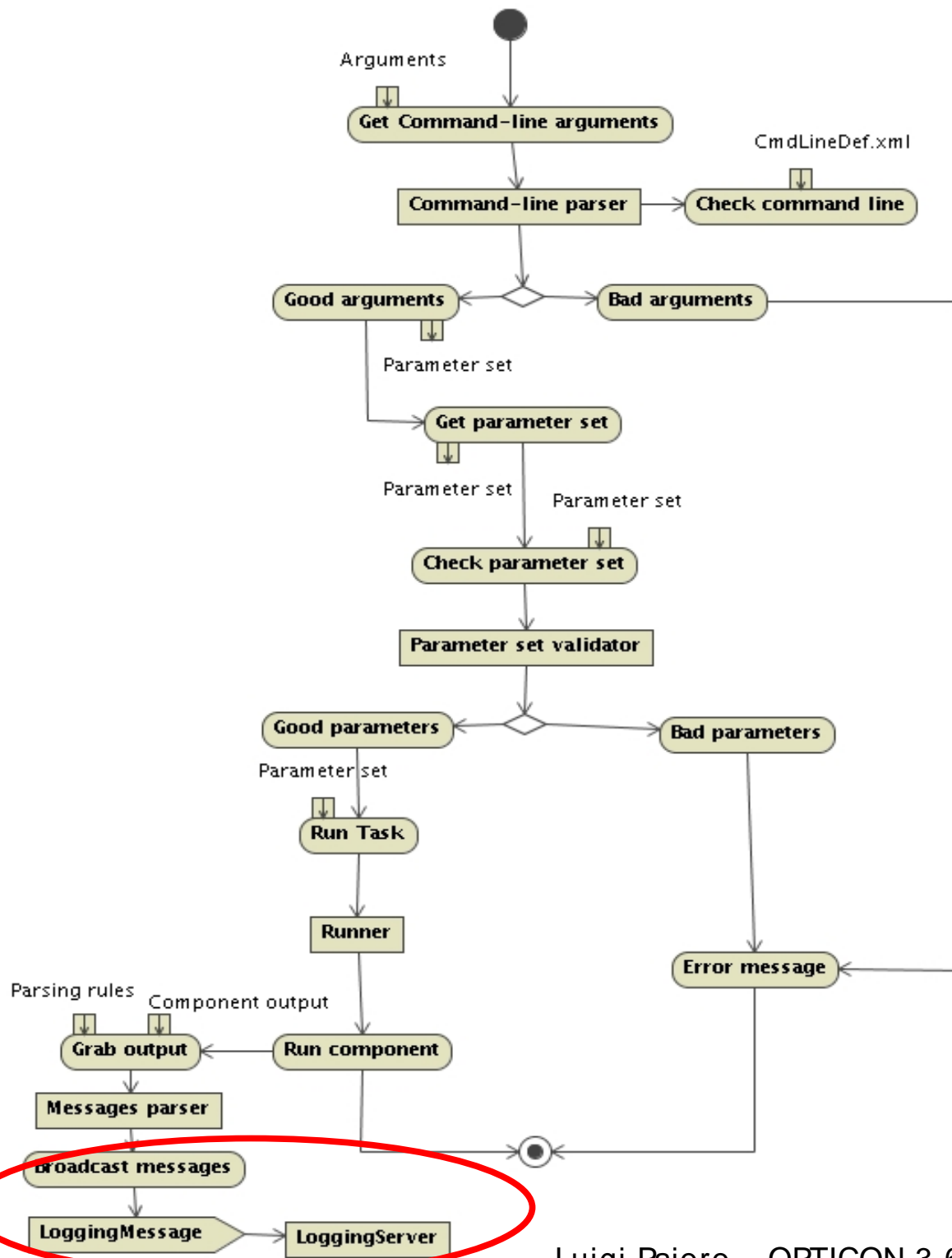
The messaging parsing

- The component output messaging parsing rules
- Based on XML
- Tokens
 - INFO
 - WARNING
 - ERROR
 - DEBUG
 - CRITICAL
 - OUTPUT



The Logging Server

- Stores logging messages



The logging server

- All the containers share a connection to a central logging server
 - `fase.logging.LoggingServer` interface
 - `+log(msg : Dictionary)`
- The standard logging message is serialized as a dictionary:
 - `type`: has one of the following values:
 - INFO: general messages (default);
 - WARNING: important messages;
 - ERROR: errors;
 - DEBUG: verbose messages (useful for debugging);
 - CRITICAL: unexpected errors or messages.
 - `time`: the message time (seconds since the epoch, in UTC);
 - `origin`: the message origin (usually the component name);
 - `description`: a general description;
 - `message`: the message content;
 - `host`: the process host name;
 - `user`: the process owner.

Portability to other RPC systems

- D- Bus Variant type
 - Used by ParameterValidator interface
 - Possible solution
 - dictionary
 - type
 - value
- fase.basics.ObjectFactory
 - System persistent object which creates new “instances” of whatever object needed
 - `+get_object(object_path : String, owner : String) : String`
 - Investigation needed

Next step

- Plug-in our applications for spectra reduction and analysis
 - Support for spectra exchange
 - VO Spectra DM and SSA 1.0 (Tody et al.)
- Implementations
 - C libraries wrapped in Python
 - SSA 1.0 Web Service (Java)