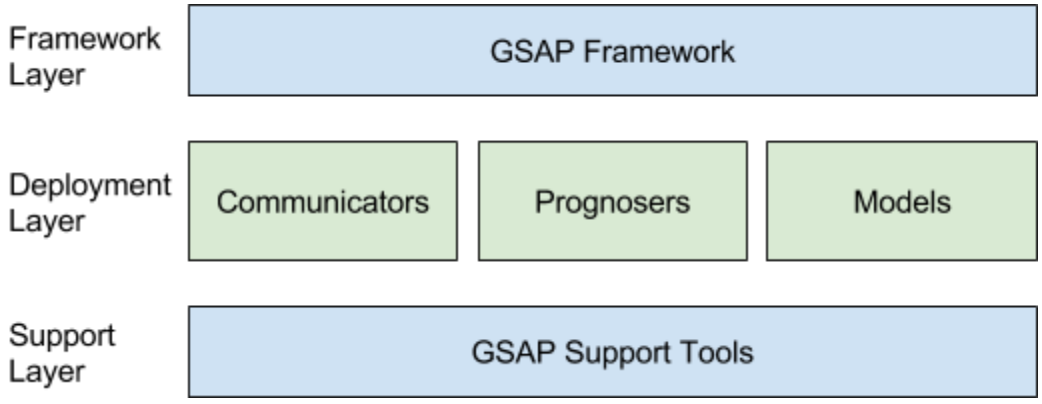
User Guide for Generic Software Architecture for Prognostics (GSAP) C++ V1.0

About

The Generic Software Architecture for Prognostics (GSAP) is a framework for applying prognostics. It makes applying prognostics easier by implementing many of the common elements across prognostic applications. The standard interface enables reuse of prognostic algorithms and models across systems using the GSAP framework.



The GSAP framework is used through the creation of communicators, prognosers, or models (the deployment layer). The elements of the deployment layer plugs into the framework and use the tools of the support layer. These elements are described further below:

- **Communicators:**
  Communicators are used to communicate data with the outside world. These function as interfaces with various data sources and sinks. Some examples could be a playback agent that reads from a file, a GUI for displaying prognostic results, an automated report generator, or a client that connects into a network messaging system (for example: SCADA). These systems can receive data which will be used by prognosers or communicate the results with operators.
- **Prognosers:**
  This is the core of the GSAP system. Inside the prognosers is the core logic for performing prognostics. A new prognoser is created to support a new method for performing prognostics. Many prognostics systems follow a common model-based structure. Those systems do not require the creation of a new prognoser, only the creation of a new model that will be used by the *modelBasedPrognoser*. For more information on this see the section *Extending*.
- **Models:**
  Models are a method of representing the behavior of a component. A common way of

performing prognostics is using a model that describes both the healthy and damaged behavior of the components. The *modelBasedPrognoser* uses the models to perform prognostics.

Each of these components is configured through the use of configuration files. This allows for a GSAP deployment to be configured to a new configuration or system without any software changes.

The GSAP system was tested with OS X 10.11, Red Hat Linux, Debian Linux, and Windows 7.

Compiling

Compile using the cmake list included and the cmake tool (https://cmake.org)

Running

Running a GSAP deployment requires an *entry point*. This entry point is a main function that registers any models, observers, predictors, prognosers, or communicators used with the relevant factory, and points the Prognostic Manager to the relevant prognostic configuration file.

Once the GSAP deployment is started up, a command prompt should appear. Here you can command the deployment between four modes:

- **Enabled**: This is the mode that that the system starts in. In this mode the communicators are initialized and running, and the prognosers are initialized, but not yet running. To start the prognosers use the *start* command.
- **Started:** In this mode both the communicators and prognosers are running. From here the prognosers can be paused with the *pause* command, and the system can be stopped with the *stop* command.
- **Paused:** At this point the communicators continue, but the prognosers have paused. From here the prognosers can be resumed with the *resume* command and the system can be stopped with the *stop* command
- **Stop:** This is the mode where everything is shut down, after execution. GSAP can be stopped using the *stop* command at any time.

The GSAP deployment keeps a log file (log.txt) of all operations in the same directory as the executable.

Extending

GSAP is designed to be easy to extend to fit your use. Extending GSAP is done by adding Prognosers, Models, or Communicators. When the behavior of the component being prognosed is represented by a model, users can create a new model and use the supplied modelBasedPrognoser for prognostics. This is doe instead of adding a new prognoser.

Adding New Models

New models are created with a new class that derives from either the Model class or the PrognosticsModel class. Models from the Model class implement only state and output equations. Models from the PrognosticsModel class implement also input equations (used to generate predicted inputs for a system being modeled) and a threshold equation (used to define events to be predicted, e.g., failure).

Adding New Prognosers

This is done for Prognosers that do not follow the modelBasedPrognoser pattern. If you are using the modelBasedPrognoser pattern, you do not need to create a new Prognoser. In that case you use the modelBasedPrognoser and add a new Model.
If you are not using modelBasedPrognoser follow the following instructions to create a new Prognoser:

1. Copy EmptyPrognoser.cpp and EmptyPrognoser.h and rename to the name of your prognoser
2. Follow the instructions inside EmptyPrognoser.cpp to create your prognoser

Adding New Communicators

1. Copy EmptyCommunicator.cpp and EmptyCommunicator.h and rename to the name of your communicator
2. Follow the instructions inside EmptyCommunicator.cpp to create your communicator

Configuring

A GSAP distribution is tuned using configuration files. There are three types of configuration files: the Primary Configuration File, Prognoser Configuration Files, and Communicator Configuration Files. These are described further below.

Primary Configuration File

This is the configuration file identified to the ProgManager. It identifies where all the other configuration files are, and specifies top-level configuration A list of the accepted parameters can be seen below:

Parameter	**Description**
Prognosers	A list of the prognoser configuration files to use. A prognoser will be made for each configuration file in the list

Communicators	A list of the communicator configuration files to use. A communicator will be made for each configuration file in the list
CommManager.step_size [optional]	Wait time between iterations of the comm manager in milliseconds

Prognoser Configuration Files

This is for configuring individual prognosers. A list of the accepted configuration parameters can be seen below:

Prognoser	Parameter	Description
All	type [Required]	The type of prognoser (ex: modelBasedPrognoser)
All	name [Required]	The name of the component being prognosed (ex: battery1)
All	id [Required]	A unique identifier for the piece of hardware being prognosed (ex: Serial Number)
All	histPath [Optional]	A path for the history files
All	inTags [Optional]	A list of tags expected from communicators
All	resetHist [Optional]	A flag to reset the recorded history for the component. Will archive the current history file and start a new one
modelBasedPrognoser	model [Required]	The model to be used by the modelBasedPrognoser
modelBasedPrognoser	observer [Required]	The observer to be used by the modelBasedPrognoser
modelBasedPrognoser	predictor [Required]	The predictor to be used by the modelBasedPrognoser
modelBasedPrognoser	Model.event [Required]	The name of the event to predict.
modelBasedPrognoser	Predictor.numSamples [Required]	The number of samples used by a predictor.

modelBasedPrognoser	Predictor.horizon [Required]	The time horizon for prediction (in seconds), which is relative to the current time.
modelBasedPrognoser	Model.predictedOutputs [Required]	The names of the system variables that are to be predicted and written to the SystemTrajectoies. These must be specified in the order that is consistent with the model.
modelBasedPrognoser	inputs [Required]	The names of the input variables for the model. These must be specified in the order that is consistent with the model.
modelBasedPrognoser	outputs [Required]	The names of the (measured) output variables for the model, associated with the system sensors. These must be specified in the order that is consistent with the model.
modelBasedPrognoser with Battery Model	Battery.qMobile [Optional]	The amount of mobile ions in the battery. This is representative of battery capacity (a larger value means a larger capacity).
modelBasedPrognoser with Battery Model	Battery.Ro [Optional]	Internal Ohmic resistance of the battery.
modelBasedPrognoser with Battery Model	Battery.VEOD [Optional]	The voltage level that defines end-of-discharge (EOD).
modelBasedPrognoser with UnscentedKalmanFilter	Observer.Q [Required]	The process noise covariance matrix for the model. These must be specified in the order that is consistent with the model. It is represented as a comma-separated list of values, going row by row.
modelBasedPrognoser with UnscentedKalmanFilter	Observer.R [Required]	The sensor noise covariance matrix for the model. These must be specified in the order that is consistent with the model. It is represented as a comma-separated list of values, going row by row.
MonteCarloPredictor	Model.processNoise [Required]	This is a vector consisting of the variances of each process noise term.

		There exists a process noise term corresponding to every state in the model, and is assumed to follow a Gaussian distribution with zero mean and variance specified in the configuration file.
MonteCarloPredictor	Predictor.inputUncertanty [Required]	A specification of the uncertainty in the input parameters specifying the future inputs to the system. For the battery model, it expects quadruples of numbers, one for each loading segment: the mean of the load (in W), the standard deviation of the load, the mean of the segment duration (relative to the initial time) and the standard deviation of the segment duration.

Communicator Configuration Files

This is for configuring individual communicators.

Communicator	**Parameter**	**Description**
Playback	file [Optional]	The file to be used for playback
Playback	Delim [Optional]	The delimiter used in the playback file
Playback	timestampFromFile [Optional]	If set to ‘true’ the playback data will be assigned the timestamp in the file. Otherwise, it will be assigned the current time
Recorder	saveFile [Optional]	The file to be recorded to
Recorder	recordProbOccur [Optional]	If ‘true’ the probability of occurrence will be written to the file
Recorder	recordOccurance [Optional]	If ‘true’ the occurrence matrix will be written to the file
Recorder	recordPredictions [Optional]	If ‘true’ the predictions will be written to the file, otherwise only the curent timestep will be recorded
Recorder	recordSystemTrajectories	If ‘true’ the system trajectories will be

	[Optional]	written to the file
Random	step [Optional]	The minimum step between adjacent random numbers
Random	max [Optional]	The maximum random number used (minimum is 0)

Contact

If you have questions, please contact Chris Teubert (christopher.a.teubert@nasa.gov)

Change Log

V0.1 [5/27/16]: Initial draft for software release V0.1