A flexible software framework for self-adapting algorithm-based fault detection and diagnosis in solar heating systems

Matthias Georgii, Christoph Schmelzer, Hagen Braas, Janybek Orozaliev and Klaus Vajen

Institute of Thermal Engineering, University of Kassel (Germany)

Abstract

A software framework for algorithm-based fault detection in solar heating systems is described. It automatically links modular algorithms in order to perform the best applicable fault detection methods considering the data available for each monitored system. It follows an object-oriented design, where algorithm objects communicate their needed data and parameter inputs, offered outputs as well as constraints via corresponding objects to a path finder. Special effort is made to enable the creation of flexible measurement data requests that do not need to know sensor names in advance. In this paper the basic concepts and implementation approaches to achieve this flexibility in the software framework are described. Finally, exemplary results are shown for a set of algorithms that were implemented in the software framework to determine gradual leakage in a monitored system and bring the capabilities of the framework to life.

Keywords: Solar heating, fault detection, function control, monitoring

1. Introduction

Solar heating systems for moderate climates are equipped with an auxiliary heater which is able to cover the whole heat demand on its own. Hence, faulty behavior of the solar part of the heating system often remains undetected by the end user, at least for a longer period. Higher fossil energy consumption and increased degradation of installed components may be the consequence. But the complexity of these systems can make it time-consuming (and thus expensive) even for an expert to assess the functioning or to detect faults of the solar heating system. Moreover, several faults may only occur temporarily under certain circumstances and remain hidden during an inspection. Therefore, automatic function control of solar heating systems is preferable.

However, the possibilities to automatically detect (and localize) faults depend on the system type, the underlying hydraulics, available measurement instrumentation and known system parameters. So, different approaches, additional assumptions or conditions may be necessary to detect a certain fault for different systems, implying different levels of accuracy that can be reached. Hence, it was the aim to design a FDD software package which can automatically adapt itself to such boundary conditions of each system (i.e. the available data basis) and subsequently perform the best applicable FDD algorithms.

To enhance automated fault detection for solar heating systems, ongoing research is important and necessary. This includes identifying relevant and observable symptoms of different faults for different systems and operation conditions, deriving algorithmic descriptions to check for these symptoms and faults, and -even more challenging- finding acceptable threshold values that differentiate between faulty and normal operation. Besides that, constant questioning and advancement of the best ways to implement and structure fault detection systems is important. One approach for a self-adapting fault detection system is presented here.

2. Basic concept

Since network-connectable data loggers and controllers get more common, a server-based approach for fault detection and diagnosis (FDD) of solar thermal systems (STS) is considered here. This means that measured data will be transferred from local data loggers to central servers which are tasked to monitor many solar thermal systems simultaneously. The local controller will only perform function control routines which either can be implemented easily without extra cost, or which check for conditions that require immediate response from the controller, e.g. to avoid damages in the STS. On the central server, the FDD analysis is performed and the results

doi:10.18086/swc.2017.19.05 Available at http://proceedings.ises.org

are passed to some "action module" which decides how to proceed (e.g. store to some database, write alarm messages, communicate with the controller, …). Such a centralized server-based solution allows for easy maintenance and continuous improvements of the FDD analysis routines that immediately benefit the whole user base. Furthermore, it does not require more powerful (and thus expensive) controllers in order to perform increasingly complex algorithms which are not part of the intrinsic control task. The software framework described here focuses on the part of the FDD analysis, i.e. how to extract faults and malfunctions in the STS from the available measurement data which is accessible in databases.

To achieve a FDD software package that can flexibly adapt itself to the available data basis, the whole procedure of fault detection and diagnosis is split into small subtasks which then will be managed automatically. This modular approach starts with the underlying FDD method which is described in Isermann (2006) and was also used by Küthe et. al (2011). It features several levels that are passed consecutively, as shown in fig. 1. Starting from measured values (e.g. the collector temperature), features are generated via appropriate feature algorithms. Features are (combined) properties of measured signals or some information derived thereof (e.g. nonzero flow rate in solar loop). Symptom algorithms then check related features for exceptional values to generate symptoms. Symptoms specify unusual or undesired states in the STS (e.g. unwanted cooling through collector during night). In the last step, fault algorithms usually combine several symptoms to identify the fault(s) which might cause the observed symptoms (e.g. malfunctioning check valve). Note that the outputs of symptom and fault algorithms are "statements about" symptoms and faults, since the result may also be their nonexistence. Hence, it is possible to differentiate whether a symptom or fault was checked for at all, and to make use of a "confirmed nonexistence".

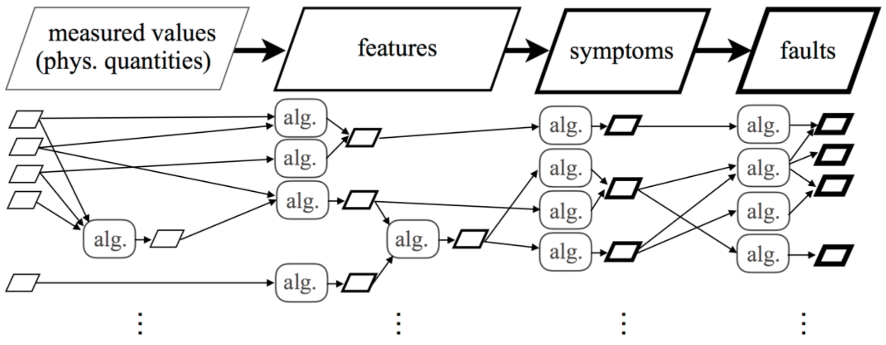

Fig. 1: Levels of the FDD method and exemplary algorithms of each level with their links. Each rhomboid denotes a specific quantity (i.e. a certain sensor/measuring point, feature, symptom, fault)

The FDD algorithms described in the previous section can be considered as "special knowledge" solving a small subtask under certain prerequisites and hence are part of a "knowledge base". Also, all information available about hydraulic schemes, each system's installed sensors as well as properties (called parameters here) of those sensors and of other installed hydraulic components contribute to the knowledge base. The software framework described here can access the knowledge base and then automatically link the knowledge found in order to apply it to a specific STS. The knowledge base can be extended easily over time, and every new piece of information will automatically be used wherever applicable. This distinction between knowledge base and automated knowledge linking is also indicated in fig. 2 where important software elements are depicted along with the interconnecting information flow.

3. Main software elements

In this section, first the structure of the knowledge base is described, on top of which the software framework will work. After that, the mechanism needed to communicate inputs, outputs and constraints between algorithms and the framework is presented. In the third subsection, the process of knowledge linking utilizing the communication mechanism is explained. The next subsection then sketches how the single parts are put together and which steps are performed during a particular investigation of a STS. The last subsection summarizes how data can be accessed and manipulated within an algorithm.

3.1 Knowledge base

As mentioned above, the knowledge base holds all FDD algorithms and known properties of STS, their components and measurement equipment. Two things are essential for every FDD analysis: what is the type/ hydraulic design of the STS, and what sensors are intended to measure which quantity. Also, the names for sensors and components have to be defined. For this purpose, each STS that is to be investigated by the FDD software must be assigned to both a hydraulic scheme and a sensor scheme. The hydraulic scheme is – in the easiest case – a name that uniquely identifies the hydraulic layout of a STS. The possibility to compose a hydraulic scheme from several hydraulic modules as described in (Dröscher et al., 2009) will also be implemented in future. Moreover, for a given hydraulic scheme, names can be defined for components (except sensors) in the STS layout (e.g. "primary solar pump"), along with their types (e.g. "pump"). Also default parameters for the components of the hydraulic

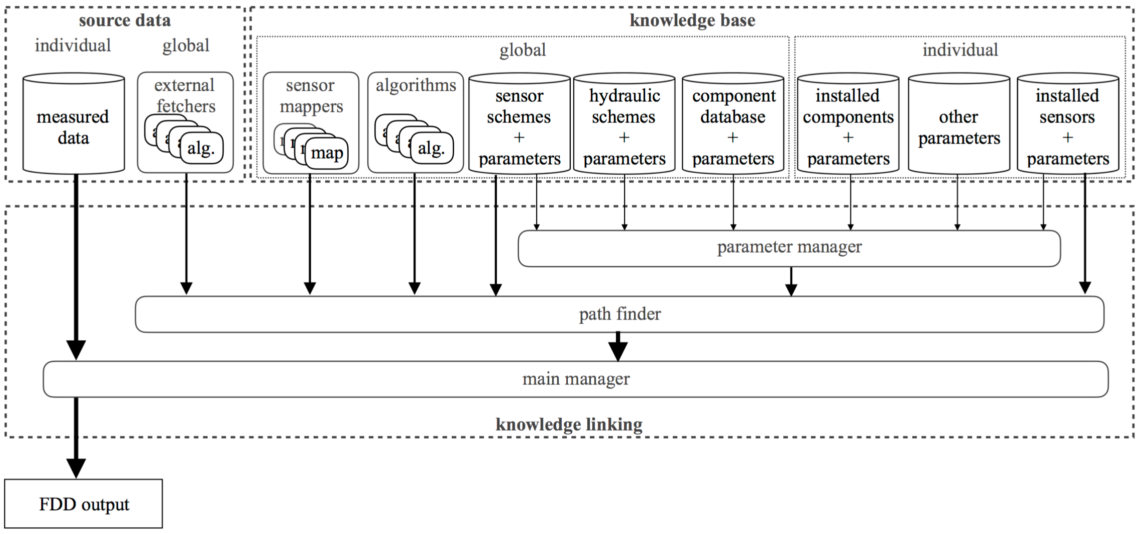

Fig. 2: Main software elements and the information flow

scheme may be provided if reasonable. All these component names and their default parameters are stored in a central hydraulic schemes database.

For each hydraulic scheme, one or several sensor schemes can be defined. The sensor scheme uniquely identifies the set of possible sensor names, where each sensor name is unambiguously linked to the measured quantity and a measuring location. Again, for each sensor also default parameters may be stored here (e.g. typical maximal flow rates or temperatures). By specifying the hydraulic scheme and the sensor scheme for each STS, the software framework knows the namespaces that are used to identify sensors and components in its data tables. A STS may also provide an *individual* installed sensors and installed components table, where individual parameters can be stored and override the default values in the scheme tables.

Moreover, a central component and sensor database is available. Here, for each component/sensor type, specific products can be registered along with known parameter values. Reversely, component or sensor parameters (e.g. collector efficiency, or measurement uncertainties) can be retrieved when type, manufacturer id, product id and parameter name are known.

Algorithms can be added by simply copying the corresponding python module file(s) into a specified directory. To be recognized by the framework, the module files must contain algorithm objects which are created by invoking a central factory function offered by the framework.

3.2 Communication mechanism

Each algorithm is confined to a small subtask and is valid under certain premises. For automatic linking, the surrounding management framework has to know the output of each algorithm as well as the required input quantities. In the design used by the framework, algorithms are objects which communicate their output quantity and input quantities via so-called "label" objects. Normally, each algorithm defines exactly one output label, only fault algorithms may have several fault labels as output. Besides these requested input labels, an algorithm can also request parameters and set up constraints that must be fulfilled in case of execution. Parameter requests and constraints are also objects. Parameters in this context are defined as fixed values that are looked up in the corresponding databases in the knowledge base (they might have been stored there by parameter identification algorithms, but the latter will not be integrated in regular FDD investigations). The most common constraints limit an algorithm to a set of valid hydraulic schemes. But also parameter constraints can be created, demanding that the parameter could be looked up and has the required value. Algorithm objects, label objects, parameter objects and constraint objects are generated in a module file by calling the corresponding central factory functions offered by the framework. The factory functions make sure that objects of the correct types are used.

The label objects describe a certain input or output quantity, but hold in their attributes much more information than a simple name: For instance, whether it is an offer or a request, what kind of data it is associated with (just a single value, or multiple values linked to points in time, etc.), the dimension and unit for the offered or requested data, and so on. Label objects provide methods to check whether they match each other to allow for comparisons. For the simplest class of labels, the matching rule looks for equality of the relevant attributes. Some attributes like "unit" are always irrelevant and never influence matching, since two data sets containing the same quantity in different units are equivalent in view of the information content. Such standard labels can have only one match or none. But also label classes with more sophisticated matching rules exist (see below). However, such labels can encounter multiple matches. In that case, the best or preferred match has to be selected.

While features, symptoms and faults can simply be labeled with meaningful unique names to link offers and requests, the situation is somewhat more difficult for measurement data, as will be explained in the following. In order to request measured data, an algorithm needs to specify it (or the corresponding sensor) in a well-defined way. An obvious approach would be to set up a universal naming scheme, according to which each sensor of each STS gets an unambiguous name, in the sense that equivalent sensors in different STS get the same name. Such a naming scheme should include all aspects that might be needed for differentiation (e.g. quantity and some hierarchical description of position, like "T_solarloop_primary_return"). But this results in sensor names that are cumbersome in simple setups, while the naming scheme still may fail in cases where an unforeseen degree of distinction is needed. It also doesn't account for the fact that differently named sensors of different systems may serve the same purpose in some context, thus necessitating several algorithms, with their only difference being the sensor name requested. For instance, an algorithm needing the flow temperature coming to a heat storage from the solar loop might have to request "T_solarloop_secondary_supply" (supposing an according naming scheme) in case of the solar loop being connected via an external heat exchanger, but "T_solarloop_primary_supply" in case of an internal heat exchanger. Therefore, we take a more flexible approach by including an additional layer: the "PHYSICALQTY" label objects that describe a measurement data request in general terms, i.e. without knowing hydraulic details and from the perspective of measurement purpose. While PHYSICALQTY labels describe sensor requests, offered sensors are still identified via sensor names which are contained in the sensor label objects. But in contrast to a universal naming scheme, these sensor names can be chosen freely within the sensor schemes that were introduced in the previous subsection. Hence, the sensor names can be adapted to the needs of the hydraulic layout. To link PHYSICALQTY requests and offered sensor labels, sensor mappers (compare fig. 2) must be provided for each sensor scheme.

In the context of PHYSICALQTY labels, also the advanced label matching methods come into play. In principle, PHYSICALQTY labels have several attributes that hierarchically narrow down the requested measuring purpose. Here matching rules can be implemented that only check the equality up to the level of detail that is demanded by the request label. For instance, it is possible to generate labels that request the flow temperature in the solar loop, but do not care whether it is measured close to the collector or close to the heat sink. Due to the advanced matching rule, there is no need to create a mapping link just for this request variant. It is sufficient if the PHYSICALQTY label with highest level of detail was linked, easing the mapping effort.

Hence, to request measured data for an algorithm, one still can use the corresponding sensor name via a sensor label object. This leads to the implicit constraint, that the algorithm can only be applicable if the investigated STS also refers to the hydraulic scheme and sensor scheme where the sensor name is defined. This is welcome for algorithms which are valid for only one hydraulic scheme. For algorithms applicable to several hydraulic schemes resp. sensor schemes, the requested measured quantity is specified in general terms via corresponding PHYSICALQTY labels.

To summarize, we want to emphasize four advantages of PHYSICALQTY labels when compared to the utilization of a universal sensor naming scheme: Firstly, these are objects and not strings that are composed in a fixed pattern, making it easier to access the contained information and to alter the constituent parts in future. Secondly, the description of a measuring point does not need to include every aspect that might play a role in some special contexts. Instead, only measuring points which are quite common on many hydraulics need to be addressable with sufficient precision by PHYSICALQTY labels, since only those might be used in universal algorithms which do not depend on the peculiarities of the hydraulic scheme. Thirdly, a universal naming scheme starts from possible measurement points in hydraulic layouts and makes sure to assign each one a unique name, meaning each “offer” is identified unambiguously. The PHYSICALQTY labels reversely describe sensors from the perspective of requests (which are unaware of hydraulic details), so not each offered, but each requested quantity is uniquely identified. This means that different PHYSICALQTY labels may point to to two different sensors in one hydraulic scheme, but to just a single sensor in another hydraulic scheme. To illustrate this, let one request be the sensor that measures the temperature of the heat carrier coming from the solar loop to the heat storage, and the other request the sensor measuring the flow temperature in the main (primary) solar loop before the heat sink. These are two different measuring points when an external heat exchanger is present (primary and secondary supply temperature), but a single one for an internal heat exchanger. Fourthly, with PHYSICALQTY labels, also requests of sensor collections can be defined, like “all available storage temperature sensors” or “all available output temperature sensors for parallel collector fields”. To translate PHYSICALQTY labels to the delivering sensor labels, corresponding sensor maps are provided in the knowledge base for each sensor scheme (compare fig. 2).

3.3 Linking process (path finder)

To make use of the gradually growing knowledge base for a particular investigation, you need to check the applicability of all known algorithms, find linking possibilities and select the best ones. This is the task of the surrounding management framework, specifically the “path finder” (see fig. 2).

The path finder will search for known algorithms starting in a configurable root directory, but can be told to import just certain types of algorithms. For instance, for each algorithm a maximum automated execution frequency/minimum time interval can be specified, since some algorithms may need to look at larger timescales to work properly, meaning it also makes no sense to call them too often. The path finder is also given a threshold time interval, making it ignore all algorithms that have higher minimum time intervals. Hence, a path finder for daily evaluations will omit algorithms that do not add value if called more often than once a month, but monthly path finder will import them as well as daily algorithms.

To start the path search, the path finder fetches the set of installed sensors at the STS. It uses the sensor mappers from the knowledge base to determine the set of deliverable PHYSICALQTY labels. For every algorithm imported, it is checked if required input data can be supplied, if requested parameters could be looked up successfully by the parameter manager (compare fig. 2) and if all constraints are fulfilled. The parameter manager knows how to access all parameter tables and will look for a parameter at all relevant places. For instance, an equipment-dependent sensor parameter will first be searched in the individual installed sensors table. If not successful, it will try to retrieve manufacturer id and model id (first individual, then default) and use them to search for the parameter in the central component database. A specially treated kind of algorithms are external fetchers which may deliver data from external weather databases. These external fetchers normally are only activated by the path finder, if the data cannot be provided by sensors installed at the heating system.

As illustrated in fig. 1, several applicable algorithms providing the same output quantity may exist. For instance, the simple feature “it is night” could be calculated more accurately, if the zip code of the STS is known, but can also be given, if only the country is known. Thus, the algorithms are branded with a ranking score which rates the reliability and accuracy of an algorithm under the prerequisite of accurate input data. If several algorithms delivering the same output are applicable, the situation is resolved depending on the configuration of the path finder. It can be configured to choose the best algorithm a priori, based on the ranking score of the algorithm. Alternatively, each of those algorithms is added to the execution list to calculate the output quantity. Then the result data set with the highest reliability score will be chosen for further use. The reliability score of the output data will be calculated via the reliability scores of the inputs and the ranking of the algorithm.

Algorithms can not only have inputs of the lower level, but also of the same level (compare fig. 1). This allows for “virtual” sensors (e.g. calculating heat sums) as well as sophisticated features derived from other features. To make sure to detect all linking possibilities, the path finder iteratively checks the applicability of algorithms with inputs of the same level. It will adjust the execution order automatically, but avoid creating circular references.

At the end of the path search, when no algorithms can be added any more, the path finder removes all algorithms from the execution list that are placed on dead end execution paths. Dead ends are defined by giving a threshold target level to the path finder, usually being the symptom level. In that case, only algorithms that contribute to at least one symptom generation would be executed. However, also symptoms that are not used by any operable fault algorithm would be generated, as they might be of interest when post-processing the FDD results.

Because many comparisons of label objects have to be made during a path search, the path finder uses data structures for its internal input/output registries that on the one hand utilize fast membership testing via hashing methods, on the other hand also minimize the number of unavoidable matching comparisons. This means the comparison effort for the path search shows rather linear than quadratic growth with increasing number of algorithms.

3.4 Main manager

The main manager is responsible for performing all steps that are necessary to perform an FDD analysis for a given STS and time range. Based on provided configuration parameters, the main manager creates instances of the pathfinder and the parameter manager. Also, all necessary fetchers are initialized. Each fetcher knows how to look up certain kind of information in a certain database table. Now the path finder imports the algorithms and sensor mappers and determines the set and order of algorithms to execute. After that, the set of sensors needed as inputs is determined. Their measurement data are fetched from the database and packed into a data collection object, where they can be preprocessed. The data series can be checked for boundary values and/or plausibility, where the preprocessing parameters are gathered from individual installed sensor or sensor scheme table. Invalid values are registered and considered as data gap. Small data gaps may be interpolated (depending on the sensor's preprocessing parameter), larger data gaps are retained and registered. Also, a reliability score may be computed. Optionally, a sensor may be totally removed from the list of available sensors, necessitating a new path search. Of course, it is also possible to skip the preprocessing procedure here and perform it already before inserting new measurement data into a database holding sanitized data. In any case, it is ensured that in the end all measured data arrays are synchronized with one equidistant gapless timeline determining the fundamental time resolution of all subsequent calculations. If necessary, data points are mapped or interpolated accordingly.

When the data collection of all needed preprocessed measurement data is created, it is passed from one algorithm to the next. Each algorithm retrieves the requested input data from the collection via its input label objects. Thereby, the label request is first mapped to the right offer in the data collection (e.g. a PHYSICALQTY to the real sensor data), if necessary. Furthermore, the data is automatically converted to the requested unit as needed. Finally, the result data is added to the data collection, where it gets labeled automatically. After the execution of the last algorithm, the data collection can be passed to some "action" module responsible for final processing, which is not in the focus here. Commonly the results would be stored in some database where they can be accessed later, but also immediate triggering of alarm messages could be implemented here. For the algorithm development and testing tasks, the data collection is passed to our result visualization and evaluation tool.

3.5 Data sets

The core of each algorithm object is its method where real processing takes place. When an algorithm retrieves input data from the data collection via the request label, the data is returned as data set object. Different kinds of data set types exist, with continuous data sets being the most common one. They contain the data as array and a corresponding timeline object. The timeline object provides many convenient methods e.g. to locate days, weeks, months and so on, or to determine the day of the week for each point. Also, sub-timelines can be created easily. Likewise, especially the continuous data sets offer convenient methods of often needed operations. For instance, time averages over a specified time interval, daily/weekly/monthly sums, shifting data by a specified time range, determining start or end points in mask arrays, and other signal processing methods or filters like smoothing will be added. This makes it easier to write the processing steps in a few lines of codes for many algorithms.

4. Validation

The software framework is developed and employed in our academic environment where a few STS are monitored. In particular, two STS for domestic hot water preparation in multi-family houses (MFH) are available that serve as regular heat sources under real use conditions, but may be used for experiments as well. Since they are equipped with an extensive measurement instrumentation, these systems allow to generate real life data of the STS when a deliberately induced fault is present. More information regarding the MFH STS can be found in (Schmelzer et al., 2015).

Dozens of algorithms for features, symptoms and faults are already implemented in the knowledge base of the software framework. They were checked against measurement data from the MFH STS as far as possible, or adopted from previous projects. Of course, the correct operation of the software framework including path finder and all implemented mapping and matching functionalities were checked with dedicated test algorithms and test cases, but also confirmed when used for the algorithm development and validation.

To bring the capabilities of the framework to life, exemplary results of a set of advanced algorithms are shown. They were implemented in the software framework, aiming to detect gradual leakage in the solar loop when a pressure sensor is available. Based on a validated physical model of the temperature-dependent pressure in the solar loop and related sensitivity analysis studies, the expected pressure signal and uncertainty margins are calculated as features with help of several temperature signals from the STS. The model only considers states when the pump is off and no stagnation occurs. Hence, according features provided by other algorithms are used as input, too. From a first version of algorithms which used several sensors that are only available at the MFH STS, a more

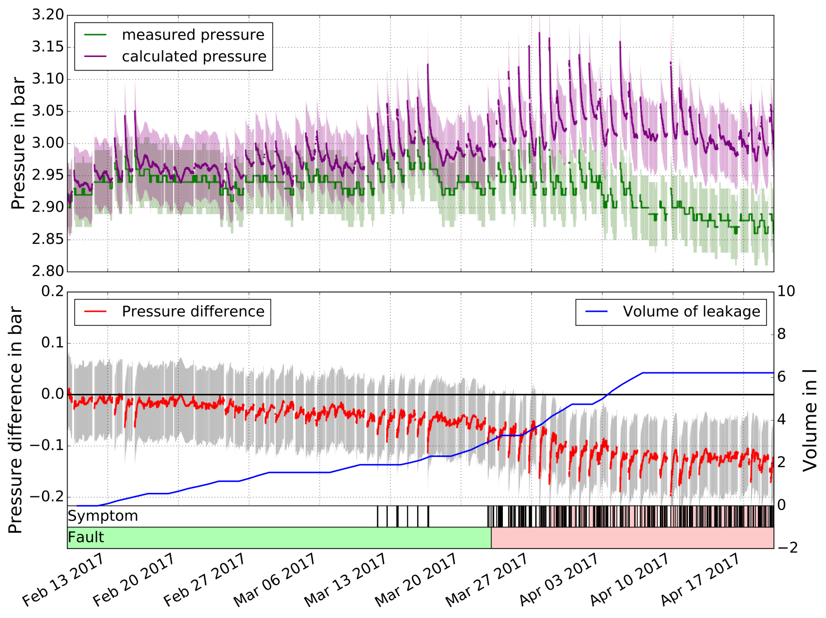

Fig. 3: Measured (green) and calculated (violet) pressure signals incl. their uncertainties, their difference (red) incl. uncertainty (gray), the cumulated leakage volume (blue) and the times where the corresponding symptom and fault are recognized.

simplified version using more common sensors was developed. They both performed similar well and were able to detect the gradual leakage quite soon. The results of the simplified set of algorithms are shown in fig. 3. On the MFH STS, a gradual leakage was initiated on purpose and the total leakage volume measured (blue line fig. 3). Fig. 3 shows the calculated and measured pressure signals and their difference including the corresponding calculated uncertainties. At the bottom it is indicated where symptoms are recognized and for which days faults are reported. Although only 3 liter (<5% of fluid in collectors and supply/return pipes, <2% of expansion vessel volume) were lost in total, and although the measured pressure stayed on the same level, the leakage was detected correctly by the algorithms. While a detection approach looking only at absolute pressure drops would have failed, the implemented algorithms could derive that the pressure should have increased in normal operation. Reversely, no leakage was detected in the leakage-free operation times. However, in order to be applicable, a pressure sensor as well as several parameters like size of the expansion vessel and the (rough) fluid volume must be available. Hence, it is useful for medium to large STS. Algorithms that require fewer parameters, yielding in higher uncertainties, will be derived, too. Thanks to the path finder of the software framework, the best applicable algorithms of these variants will be chosen automatically.

5. Summary

A flexible software framework for fault detection and diagnosis was presented. It automatically links modular algorithms and other information from the knowledge base to flexibly adapt itself to the boundary conditions of each solar thermal system. Hence, it can perform the best faults detection methods in dependence of the information available. In this paper, the underlying concepts and implementation approaches to achieve this flexibility were introduced. The usefulness and interaction of offer and request label objects, data set and algorithm objects and the functioning of the path finder and the overall procedure were described. Exemplary results for algorithms detecting a gradual leakage were also shown. The FDD software framework and the corresponding algorithms currently are developed and tested in an academic environment where a few well known systems are monitored.

6. References

Dröscher, A., Ohnewein, P., Haller, M. Y., Heimrath, R., 2009. Modular specification of large-scale solar thermal systems for the implementation of an intelligent monitoring system, Proceedings of the ISES Solar World Congress 2009, pp. 683-688

Isermann, R., 2006. Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance, Springer, Berlin.

Küthe, S., de Keizer, C., Shahbazfar, R., Vajen, K., 2011. Implementation of Data Processing and Automated Algorithm Based Fault Detection for Solar Thermal Systems, Proceedings of the ISES Solar World Congress 2011, doi: 10.18086/swc.2011.28.16

Schmelzer C., Georgii M., Vajen K., 2015. Entwicklung, Untersuchung und Anwendung von Methoden zur Langzeitüberwachung und automatisierter Fehlerdetektion großer, solarunterstützter Wärmeversorgungs-systeme, final report of the FeDet project FKZ-0325975A, funded by the German Federal Ministry for Economic Affairs and Energy

7. Acknowledgements

The software development was supported by the German Federal Ministry for Economic Affairs and Energy within the project FKZ-0325975A.