FlexDx: A reconfigurable diagnosis framework

Mattias Krysander, Fredrik Heintz, Jacob Roll and Erik Frisk

FlexDx: A Reconfigurable Diagnosis Framework

Mattias Krysander[*], Fredrik Heintz[†], Jacob Roll[*], Erik Frisk[*]
[*] Dept. of Electrical Engineering,
Linköping University, SE-581 83 Linköping, Sweden
Email: {matkr,roll,frisk}@isy.liu.se
[†] Dept. of Computer and Information Science,
Linköping University, SE-581 83 Linköping, Sweden
Email: frehe@ida.liu.se

Abstract

Detecting and isolating multiple faults is a computationally expensive task. It typically consists of computing a set of tests and then computing the diagnoses based on the test results. This paper describes FlexDx, a reconfigurable diagnosis framework which reduces the computational burden while retaining the isolation performance by only running a subset of all tests that is sufficient to find new conflicts. Tests in FlexDx are thresholded residuals used to indicate conflicts in the monitored system. Special attention is given to the issues introduced by a reconfigurable diagnosis framework. For example, tests are added and removed dynamically, tests are partially performed on historic data, and synchronous and asynchronous processing are combined. To handle these issues FlexDx has been implemented using DyKnow, a stream-based knowledge processing middleware framework. Concrete methods for each component in the FlexDx framework are presented. The complete approach is exemplified on a dynamic system which clearly illustrates the complexity of the problem and the computational gain of the proposed approach.[1]

Keywords: Reconfigurable diagnosis framework, Diagnosing dynamical systems, Test reconfiguration, Test selection, Test initialization

[1]This work is partially supported by grants from the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII and the Swedish Research Council Linnaeus Center CADICS.

1. Introduction

Reliable real-time multiple fault diagnosis of dynamical systems in the presence of noise is of fundamental importance in many applications. The diagnosis problem typically consists of detecting and isolating faulty components given available sensor and actuator signals. Important challenges are real time issues, complexity issues when dealing with multiple faults, how to handle process dynamics, and noisy measurements. The main objective of this work is to describe FlexDx a diagnosis framework that performs multiple fault isolation for dynamic noisy systems using adaptive and reconfiguration techniques to lower the computational burden of the diagnostic system.

1.1. Problem background

In consistency-based diagnosis, sensor and actuator signals are compared to a formal description of the process, a model, to detect inconsistencies indicating faults. Inconsistency detection can for example be based on local propagation of observations, like solutions based on the well known General Diagnostic Engine (GDE) [1, 2]. However, such solutions have limitations when diagnosing dynamical systems in real-time [3, 4]. On the other hand, in the automatic control and signal processing communities there exists a large volume of research directed at detecting faults in noisy dynamic systems, see e.g. the books [5, 6, 7, 8]. These approaches are not based on local propagation, but typically on a set of pre-compiled tests, or residual generators, that are used together with a fault isolation module [5] to reach a diagnosis decision. A key topic in these approaches has been to study residual generator design for dynamical systems in noisy environments. However, efficient handling of multiple faults has not been a central topic in these works, as in the more AI-based literature [1, 9].

In several works, for example [3, 10, 11, 12], it is noted that the fault isolation techniques from the AI literature can be used together with the pre-compilation techniques from the automatic control community. These observations makes it possible to combine the techniques from the automatic control community to handle noise and system dynamics, with the fault isolation techniques from the AI-community to handle multiple fault isolation. However, due to the inherent computational complexity of the multiple fault isolation problem, there are still open problems that need to be addressed.

The computational complexity of a diagnostic system based on pre-compiled tests mainly originates from two sources: complexity of the process model and the number of behavioral modes considered. A high resolution capability of distinguishing between faults, especially when multiple faults are considered, requires a large number of diagnostic tests [13]. This also follows form the well known fact that the number of required minimal conflicts, which corresponds to triggered tests in a residual based approach, to solve a multiple fault diagnosis task grows exponentially in the number of faults in the system [14]. Also, the more complex the process model is, the more computationally expensive the execution of the diagnostic tests is. For example, if the model consists of non-linear dynamic equations, a typical test involves a non-linear observer which may be computationally expensive. Finally, since dynamical systems are considered here, tests must be recomputed sufficiently often to capture fault dynamics and to get a fast and accurate fault detection and isolation.

1.2. Solution outline

Our basic idea to mitigate the computational burden is to exploit the fact that all pre-compiled tests are not needed at all times. For example, only a subset of the available tests are needed to detect a fault. The remaining tests can, in case of an alarm, then be used to further isolate the faulty component but they are not needed when the system is in fault free operation. As will be shown, substantial reduction in the computational burden can be achieved by exploiting this observation.

This approach is similar to works coping with the complexity of multiple fault diagnosis from the AI-community. There the complexity issues related to having many tests have been avoided by applying propagation techniques directly on the process model. The propagation is initiated with the observed measurements and then only explore the part of the model that might be inconsistent. Here we propose a similar technique, using pre-compiled tests instead of local propagation.

The proposed approach, a reconfigurable diagnosis framework called FlexDx, chooses which tests to run at a particular instant based on the current set of diagnoses. If a test is violated, i.e. an alarm is generated, then an updated set of diagnoses is computed and the set of active tests is reconfigured. It is shown how such an approach requires controlled ways of initializing the dynamic diagnostic tests and algorithms how to select the new tests to be started when a set of diagnostic tests has generated an alarm. To facilitate a thorough analysis of the approach, linear process models are used in the paper but the framework is not based on this model assumption, but can be extended to non-linear models if non-linear test techniques are used, e.g. [15, 16, 17].

1.3. Paper Outline

The reconfigurable diagnosis framework is introduced in Section 2 and related work to the different components in the framework is discussed. To illustrate the properties of the approach, linear dynamical process models are used and the theoretical diagnosis background for such systems is presented in Section 3. Methods for how to determine, in a specific situation, which tests should be run are treated in Section 4. An appropriate initialization procedure for dynamic tests is described in Section 5. The complete approach is exemplified on a dynamic system in Section 6, which, in spite of its relatively small size, clearly illustrates the complexity of the problem and the computational gain of the proposed approach. The diagnosis framework is implemented using DyKnow, a stream-based knowledge processing middleware framework [18], which is briefly described in Section 7. Finally the paper is concluded with a summary in Section 8.

2. FlexDx: A Reconfigurable Diagnosis Framework

The main idea of this work is to reduce the overall computational burden of a diagnostic system by utilizing the observation that all tests are not needed at all times. For example, when starting a fault free system, there is no need to run tests that are designed with the sole purpose of distinguishing between faults. In such a case, only tests that are able to detect faults are needed, which may be significantly fewer compared to the complete set of tests. When a test triggers an alarm and a fault is detected, appropriate tests are started to refine the diagnosis.

FlexDx uses a consistency-based approach to diagnosis when determining if a supervised system is working correctly [1, 19, 20]. If a fault is detected, the diagnosis is incrementally refined by adding and removing tests in an iterative manner according to the following procedure:

1. Initiate the set of diagnoses.
2. Based on the set of diagnoses, compute the set of tests to be performed.
3. Compute the initial state of the selected tests.
4. Run the tests until an alarm is triggered.
5. Compute the new set of diagnoses based on the test results, then go to step 2.

FlexDx represents all diagnoses with the minimal diagnoses, which has been shown useful when dealing with multiple fault diagnosis [19]. When FlexDx is started, there are no conflicts and the only minimal diagnosis is the no-fault mode NF, i.e. the set of minimal diagnoses D is set to $\{\mathrm{NF}\}$ in step 1. Step 2 uses a function that given a set of diagnoses D returns the set of tests T to be performed to monitor whether a fault has occurred or to further explore the possible diagnoses. Step 3 initiates each of the tests in T. A test includes a residual generator given in state-space form. To properly initialize such a residual generator, it is necessary to estimate its initial condition. In step 4, the tests are performed until at least one is violated and a test result is generated in the form of a set of conflicts [19, 20]. Step 5 computes the new set of diagnoses D, given the previous set of diagnoses and the generated set of conflicts. This step can be performed by algorithms handling multiple fault diagnoses [1, 9].

Step 4 and 5 are standard steps used in diagnostic systems and will not be described in further detail. Step 2 and 3 are new steps, needed for dynamically changing the test set T, the details are given in Section 4 and 5 respectively.

To implement an instance of the FlexDx framework, a number of issues have to be managed besides implementing the algorithms for each step and integrating them in a system. Each test is implemented by a residual generator, computing the residual given the measurements of the system, and a monitor that checks if the residual has triggered a violation of the test. When a potential fault is detected, FlexDx computes the last known fault free time t_f and the new set of residual generators to be started at time t_f. To implement this, three issues have to be handled. First, the FlexDx instance must be reconfigured to replace the set of residual generators and their monitors. Second, the computation of the residuals must begin at time t_f which will be in the past. Third, at the same time as FlexDx is performing tests on historic data, system observations will keep coming at their normal rate. How these issues are solved is described in Section 7.

A key step in the reconfiguration procedure outlined above is the selection of which tests to run in a specific situation. Selection of tests is related to the selection of new measurement points. For example, in the General Diagnostic Engine [1] a solution is proposed where measurement points are selected in a sequential way such that, on average, the actual fault can be isolated using a minimum number of measurements. This is done by dynamically ranking the possible measurement points according to the expected information obtained from the measurement. A related approach is found in [21, 22] where tests are ranked and selected according to some information criterion. The approach adopted here for test selection is fundamentally different and the objective is to find a subset of tests, out of a predefined set of possible tests, such that running these tests provides all information available in the model and observations *given* the current set of diagnoses.

Several works [23, 5, 24] have considered test selection for achieving different objectives but contrary to this work no focus has been on on-line reconfiguration. Another related approach is presented in [25] although the models and diagnosis techniques are different. Recently, works on on-line reconfiguration of the diagnostic system have appeared. For a related work, see for example Benazera and Travé-Massuyès [26], where Kalman-filters are reconfigured based on diagnosis decisions.

3. Theoretical Background

The diagnostic systems considered in this paper include a set of precompiled tests. Each test consists of a residual $r(t)$ that is thresholded such that it triggers an alarm if $|r(t)| > 1$. Note that the threshold can be set to one without loss of generality. It is assumed that the residuals are normalized such that a given false alarm probability p_{FA} is obtained, i.e.

$$P(|r(t)| > 1|\text{NF}) = p_{\text{FA}} \tag{1}$$

The residuals are designed using a model of the process to be diagnosed.

3.1. The Model

The model class considered here is linear differential-algebraic models. Although the presentation in this paper relies on results for linear systems, the basic idea is equally applicable to non-linear model descriptions.

There are several ways to formulate differential-algebraic models. Here, a polynomial approach is adopted, but any model description is possible, e.g. standard state-space or descriptor models. The model is given by the expression

$$H(q)x + L(q)w + F(q)f = V(q)v \tag{2}$$

where $x(t) \in \mathbb{R}^{m_x}$, $w(t) \in \mathbb{R}^{m_w}$, $f(t) \in \mathbb{R}^{m_f}$, and $v(t) \in \mathbb{R}^{m_v}$. The matrices $H(q)$, $L(q)$, $F(q)$, and $V(q)$ are polynomial matrices in the time-shift operator q. The vector x contains all unknown signals, which include internal system states and unknown inputs. The vector w contains all known signals such as control signals and measured signals, the vector f contains the fault signals, and the vector v is white, possibly multidimensional, zero mean, unit covariance Gaussian distributed noise.

To guarantee that the model is well formed, it is assumed that the polynomial matrix $[H(z)\ L(z)]$ has full column rank for some $z \in \mathbb{C}$. This assumption assures, according to [27], that for any noise realization $v(t)$ and any fault signal $f(t)$ there exists a solution to the model equations (2).

3.2. Residual Generation

Residuals are used both to detect and isolate faults. This task can be formulated in a hypothesis testing setting. For this, let f_i denote both the fault signal and the corresponding behavioral mode [28] of a single fault. Let $\mathcal{F}$ be the set of all single faults in model (2).

A pair of hypotheses associated with a residual can then be stated as

$$\begin{aligned} H_0 &: f_i = 0 \text{ for all } f_i \in C \\ H_1 &: f_i \neq 0 \text{ for some } f_i \in C \end{aligned}$$

where $C \subseteq \mathcal{F}$ is the subset of faults the residual is designed to detect. This means that the residual is not supposed to detect all faults, only the faults in C. By generating a set of such residuals, each sensitive to different subsets C of faults, fault isolation is possible. This isolation procedure is briefly described in Section 3.3.

In the literature there exists several different ways to formally introduce residuals [5, 7]. In this paper an adapted version of the innovation filter defined in [29] is used. For this, it will be convenient to consider the nominal model under a specific hypothesis. The nominal model under hypothesis H_0 above is given by (2) with $V(q) = 0$ and $f_i = 0$ for all $f_i \in C$. With this notion, a nominal residual generator is a linear time-invariant filter $r = R(q)w$ where for all observations w, consistent with the nominal model (2) under hypothesis H_0, it holds that $\lim_{t\to\infty} r(t) = 0$.

Now, consider again the stochastic model (2) where it is clear that a residual generated with a nominal residual generator will be subject to a noise component from the process noise v. A nominal residual generator under H_0 is then said to be a residual generator for the stochastic model (2) if the noise component in the residual r is white Gaussian noise.

It can be shown [30] that all residual generators $R(q)$, as defined above, for the stochastic model (2) can be written as

$$R(q) = Q(q)L(q)$$

where the matrix operator $Q(q)$ satisfies the condition $Q(q)H(q) = 0$. This means that the residual is computed by $r = Q(q)L(q)w$ and the internal form of the residual is given by

$$r = Q(q)L(q)w = -Q(q)F(q)f + Q(q)V(q)v \tag{3}$$

Thus, the fault sensitivity is given by

$$r = -Q(q)F(q)f \tag{4}$$

and the statistical properties of the residual under H_0 by

$$r = Q(q)V(q)v \tag{5}$$

A complete design procedure is presented by Nikhoukhah [29] for state-space models and by Frisk [30] for models in the form (2). The objective here is not to describe a full design procedure, but it is worth mentioning that a design algorithm can be made fully automatic, that the main computational steps involve a null-space computation and a spectral factorization, and that the resulting residual generator is a basic dynamic linear filter.

3.3. Computing the Diagnoses

The fault sensitivity of the residual r in (3) is given by (4). Here, r is sensitive to the faults with non-zero transfer functions. Let C be the set of faults that a residual r is sensitive to. Then, if residual r triggers an alarm, at least one of the faults in C must have occurred and the conflict C is generated [20].

Now we can relate the test results to a diagnosis. Let a set of single faults $b \subseteq \mathcal{F}$ represent a system behavioral mode with the meaning that $f_i \neq 0$ for all $f_i \in b$ and $f_j = 0$ for all $f_j \notin b$. The set of faults b will with some abuse of notation sometimes be referred to as a behavioral mode. This is also in accordance with the notation used in for example [1, 20]. The set of all system behavioral modes is then represented with the power set of $\mathcal{F}$.

In short, the behavioral mode b is a diagnosis if it can explain all generated conflicts, i.e. if b has a non-empty intersection with each generated conflict, see [20] for details. A diagnosis b is considered a *minimal* diagnosis if no proper subset of b is a diagnosis [1, 20]. Algorithms to compute all minimal diagnoses for a given set of conflicts, which is equivalent to the so called minimal hitting set problem, can be found in for example [1, 20]. The following example illustrates the main principle.

Example 1. *Let an X in position (i, j) in the table below indicate that residual r_i is sensitive to fault f_j*

	f_1	f_2	f_3
r_1		X	X
r_2	X		X
r_3	X	X	

If residuals r_1 and r_2 trigger alarms, then conflicts $C_1 = \{f_2, f_3\}$ and $C_2 = \{f_1, f_3\}$ are generated. For C_1, for instance, this means that both f_2 and f_3 cannot be 0. Now, for a set of faults to be a diagnosis it must then explain both these conflicts. It is straightforward to verify that the minimal diagnoses in this case are $b_1 = \{f_3\}$ and $b_2 = \{f_1, f_2\}$. $\diamond$

4. Test Selection

There are many possible ways to select the set of tests T given a set D of minimal diagnoses. The method used by FlexDx relies on basic principles in consistency-based diagnosis based only on the deterministic properties of (2).

A fundamental task in consistency-based diagnosis is to compute the set of consistent modes given a model, a set of possible behavioral modes, and observations [1]. The design goal of the test selection algorithm is to perform tests such that the set of consistent modes is equal to the set of diagnoses computed by the diagnostic system.

4.1. Consistent Behavioral Modes

The deterministic behavior in a behavioral mode b is described by (2) when $v = 0$, $f_i \neq 0$ for all $f_i \in b$, and $f_j = 0$ for all $f_j \notin b$. A set of observations consistent with b is consequently given by

$$\begin{aligned} O(b) = \{w | \exists x \exists f \quad & (\forall j : f_j \notin b \rightarrow f_j = 0) \wedge \\ & H(q)x + L(q)w + F(q)f = 0\} \end{aligned} \tag{6}$$

This means that a mode b is consistent with the deterministic part of model (2) and an observation w if $w \in O(b)$. Note that $f_i \in b$ is required to be nonzero in mode b, but this is not required in the definition of $O(b)$. In noisy environments, it will be impossible to distinguish infinitely small faults $f_i \neq 0$ from the case when $f_i = 0$. To capture this property in the deterministic analysis here we include also the case when $f_i = 0$ in $O(b)$.

The design goal can now be formulated in terms of the sets $O(b)$ as follows. The set of diagnoses should, given an observation w, be equal to $\{b \in B | w \in O(b)\}$ where B denotes the set of all behavioral modes. As mentioned in Section 2, we will use minimal diagnoses to represent all diagnoses. This is possible since (6) implies that $O(b') \subseteq O(b)$ if $b' \subseteq b$. Hence, if b' is consistent it follows that b is consistent and therefore it is sufficient to check if the minimal consistent modes remain consistent when new observations are processed.

4.2. Tests for Checking Model Consistency

Next, we will describe how tests can be used to detect if $w \notin O(b)$. Let $\mathcal{T}$ be the set of all available tests and let $r_i = Q_i(q)L(q)w$ be the residual corresponding to test t_i.

A residual generator checks the consistency of a part of the complete model. To determine which part, only the deterministic model needs to be considered. It can be shown that residual r_i checks the consistency of $\xi_i(q)w = 0$ where $\xi_i(q)$ is a polynomial in the time-shift operator q [27]. By defining the set of consistent observations for tests in a similar way as for models, we define

$$O(t_i) = \{w|\xi_i(q)w = 0\} \tag{7}$$

Now, we can characterize all test sets T that are capable of detecting any inconsistency between an observation w and the assumption that $w \in O(b)$. For this purpose, only tests t_i with the property that $O(b) \subseteq O(t_i)$ can be used. For such a test, an alarm implies that $w \notin O(t_i)$ which further implies that $w \notin O(b)$. This means that a test set T is capable of detecting any inconsistency of $w \in O(b)$ if and only if

$$O(b) = \bigcap_{\forall t \in \{t_i \in T | O(b) \subseteq O(t_i)\}} O(t) \tag{8}$$

A trivial solution to (8) is $T = \{t\}$ where $O(t) = O(b)$.

4.3. The Set of All Available Tests

If $\mathcal{T}$ is not capable of checking the consistency of b, then no subset of tests will be capable of doing this either. Hence, this approach puts requirements on the entire set of tests $\mathcal{T}$. By applying the approach to a model consisting of the considered set of tests, a diagnostic system with the same diagnosis capability as the considered set of tests will be the result. In this paper, we will use two different types of test sets $\mathcal{T}$ fulfilling (8) for all modes $b \in B$. These are introduced by the following example.

Example 2. *Consider the model*

$$\begin{array}{lcl} x_1(t+1) & = & \alpha x_1(t) + w_1(t) + f_1(t) \\ x_2(t) & = & x_1(t) + f_2(t) \\ w_2(t) & = & x_1(t) + f_3(t) \\ w_3(t) & = & x_2(t) + f_4(t) \end{array} \tag{9}$$

where x_i are unknowns, w_i known variables, α a known parameter, and f_i the faults. There are 2^4 modes and the set of observations consistent with each mode is

$$
\begin{aligned}
O(\emptyset) &= \{w| \begin{bmatrix} w_1(t) + \alpha w_2(t) - w_2(t+1) \\ -w_2(t) + w_3(t) \end{bmatrix} = 0\} \\
O(\{f_1\}) &= \{w| - w_2(t) + w_3(t) = 0\} \\
O(\{f_2\}) &= O(\{f_4\}) = O(\{f_2, f_4\}) = \\
&= \{w|w_1(t) + \alpha w_2(t) - w_2(t+1) = 0\} \\
O(\{f_3\}) &= \{w|w_1(t) + \alpha w_3(t) - w_3(t+1) = 0\} \\
O(b) &= \mathbb{R}^3 \text{ for the remaining modes.}
\end{aligned}
$$

The behavioral models for the 10 last modes b do not contain any redundancy and the observations are therefore not restricted, i.e. $O(b) = \mathbb{R}^3$. In contrast to (6), the sets of consistent observations are here expressed in the same form as for tests, that is with linear differential equations in the known variables only. Any set described as in (6) can be written in this form [31]. ◇

The first type of test set $\mathcal{T}_1$ will include one test for each distinct behavioral model containing redundancy. For the example, $\mathcal{T}_1$ consists of four tests t_i such that $O(t_1) = O(\emptyset)$, $O(t_2) = O(\{f_1\})$, $O(t_3) = O(\{f_2\}) = O(\{f_4\})$, and $O(t_4) = O(\{f_3\})$. To check the consistency of $w \in O(\emptyset)$, two linear residuals are needed, which is the degree of redundancy of the model. These two residuals can be combined in a positive definite quadratic form to obtain a scalar test quantity. When stochastic properties are considered, the quadratic form is chosen such that the test quantity conforms to a χ^2-distribution.

Tests for models with a high degree of redundancy can be complex, and the second type of test set $\mathcal{T}_2$ includes only the tests for the behavioral models with degree of redundancy 1. For the example, $\mathcal{T}_2 = \{t_2, t_3, t_4\}$ and by noting that $O(\emptyset) = O(t_i) \cap O(t_j)$ for any $i \neq j$ where $i, j \in \{2, 3, 4\}$, any two tests can be used to check the consistency of $w \in O(\emptyset)$. In [13] it has been shown under some general conditions that $\mathcal{T}_2$ fulfills (8) for all modes $b \in B$.

4.4. Test Selection Methods

We will exemplify methods that given a set of minimal diagnoses D select a test set $T \subseteq \mathcal{T}$ such that (8) is fulfilled for all $b \in D$. An optional requirement that might be desirable is to select such a test set T with minimum cardinality. The reason for not requiring minimum cardinality is that the computational complexity of computing a minimum cardinality solution is generally much higher than to find any solution.

A straightforward method is to use the first type of tests and not require minimum cardinality solutions. Since this type of test set includes a trivial test $O(t_i) = O(b)$ for all modes b with model redundancy, it follows that a strategy is to start the tests corresponding to the minimal diagnoses in D.

Example 3. *Consider Example 2 and assume that the set of minimal diagnoses is $D = \{\emptyset\}$. Then it is sufficient to perform test t_1, i.e. $T = \{t_1\}$. If the set of minimal diagnoses are $D = \{\{f_2\}, \{f_3\}, \{f_4\}\}$, then t_3 is used to check the consistency of both $\{f_2\}$ and $\{f_4\}$ and the total set of tests is $T = \{t_3, t_4\}$. For this example, this strategy produces the minimum cardinality solutions, but this is not true in general.*

A second method is to use the second type of tests and for example require a minimum cardinality solution. The discussion of the method will be given in Section 6 where this method has been applied to a larger example.

4.5. Relaxing the Design Goal

The design goal to perform tests such that the set of consistent modes is equal to the set of diagnoses is an ambitious goal that may require many tests. Three different principles for relaxing the approach will be discussed next.

First, the test selection methods are not limited to take the set of all minimal diagnoses as input. If the number of minimal diagnoses is large, a focusing strategy [32] can be incorporated to reduce the number of diagnosis and thereby also the number of selected tests. Minimal cardinality diagnosis or most probable diagnosis given some a priori probabilities are examples of possible focusing strategies.

The second type of relaxation reduces the total number of tests in $\mathcal{T}$. It may require a large number of tests to fulfill (8) for all modes $b \in \mathcal{B}$ and sometimes it is not interesting to consider unlikely modes including a large number faulty components. By considering a subset of modes $\mathcal{B}' \subset \mathcal{B}$, the goal can be modified as follows. Tests should be performed such that among the modes in $\mathcal{B}'$ exactly the consistent ones are diagnoses. This means that $\mathcal{T}$ is selected such that (8) is fulfilled for all behavioral modes in $\mathcal{B}'$. When running the diagnostic system, the test selection $T \subseteq \mathcal{T}$ has to fulfill (8) for all minimal diagnoses in $\mathcal{B}'$.

A third option is to use a given set of tests $\mathcal{T}'$, that is not required to fulfill (8) for any modes. The goal can then be formulated as follows. Given the current minimal diagnoses, perform the tests in $T \subseteq \mathcal{T}'$ such that the diagnoses computed with the selected subset of tests would be equal to the diagnoses computed using all tests. This means that, given a set of minimal diagnoses D, the test selection $T \subseteq \mathcal{T}'$ has to fulfill

$$\bigcap_{\forall t \in \{t_i \in \mathcal{T}' | O(b) \subseteq O(t_i)\}} O(t) = \bigcap_{\forall t \in \{t_i \in T | O(b) \subseteq O(t_i)\}} O(t) \tag{10}$$

for each $b \in D$.

The three relaxations can be used individually or mixed. As an example of how the three relaxations can be combined, we can consider the tests in a given set $\mathcal{T}'$, include only the fault free mode, the single fault modes, and the double fault modes in $\mathcal{B}'$, and use the minimal cardinality focusing strategy. Then, given a set of minimal diagnoses D, the test selection $T \subseteq \mathcal{T}'$ has to fulfill (10) for all minimal cardinality diagnoses b that belong to $\mathcal{B}'$. Which relaxation, or which combination of relaxations, that are appropriate for a specific application must be determined case by case.

5. Initialization

When a new test selection has been made, the new tests have to be initialized. Since information about faults sometimes are only visible in the residuals for a short time-period after a fault occurrence, we would like a new test to start before the currently considered fault occurred, otherwise valuable information could be missed. It is also important that the state of the new test gets properly initialized, such that the fault sensitivity is appropriate already from the start, and the residuals can deliver test results immediately. Therefore, the initialization following a new test selection consists of:

1. Estimate the time of the fault from the alarming test(s).
2. Estimate the initial condition for each new test.

Both these steps require the use of historical data, which therefore have to be stored. The fault time estimation will use the historical residuals from the triggered test, while the initial condition estimation uses the measured data from the process before the fault occurred. In case not enough historical data is available, it is reasonable to use all available data. In such a case, one may expect some degradation in detection performance compared to running all tests at all times.

5.1. Estimating the Fault Time

There are many possibilities to estimate the fault time. See for example [33, 8] for standard approaches based on likelihood ratios. Here, a window-based test has been chosen. It should be noted, however, that for the given framework, what is important is not really to find the exact fault time, but rather to find a time-point before the fault has occurred. The estimated time-point will be denoted by t_f.

Given a number of residuals from an alarming test, $r(1), \ldots, r(n)$, let us compute the sum of the squared residuals over a sliding window, i.e.,

$$S(t) = \frac{1}{\sigma^2} \sum_{j=1}^{\ell} r^2(t+j), \qquad t = 0, \ldots, n-\ell \tag{11}$$

If the residual generator is designed such that under the null hypothesis no fault has occurred, $(r(j))_{j=1}^{n}$ are white and Gaussian with variance σ^2, then $S(t) \sim \chi^2(\ell)$ in the fault free case. Hence, $S(t)$ can be used to test whether this null hypothesis has been rejected at different time-points by a simple χ^2-test. The length ℓ of the sliding window is selected according to the expected fault responses. A slowly increasing fault response requires a long time-window while short time-windows are preferable for impulses.

Since it is preferable to get an estimated time-point that occurs before the actual fault time, rather than after, the threshold of the χ^2-test should be chosen such that the null hypothesis is fairly easily rejected. The estimate t_f is then set to the time-point of the last non-rejected test. Also, in order not to risk a too late estimate, the time-point at the beginning of the sliding window is used.

5.2. Estimating the Initial Condition

Having found t_f, the next step is to initialize the state of the new residual generator. The method used here considers a time-window of samples of $w(t_f - k + 1), \ldots, w(t_f)$ as input to find a good initial state $x(t_f)$ of the filter at the last time point of the window.

Consider the following residual generator:

$$\begin{aligned} x(t+1) &= Ax(t) + Bw(t) \\ r(t) &= Cx(t) + Dw(t) \end{aligned} \tag{12}$$

Assume that $w(t) = w_0(t) + Nv(t)$ where $w_0(t)$ is the noise-free data (inputs and outputs) from the process model and $v(t)$ is Gaussian noise. In fault free operation, there is a state sequence $x_0(t)$, such that the output $r(t) = 0$ if $v(t) = 0$,

$$\begin{aligned} x_0(t+1) &= Ax_0(t) + Bw_0(t) \\ 0 &= Cx_0(t) + Dw_0(t) \end{aligned} \tag{13}$$

Given $w(t)$, $t = t_f - k + 1, \ldots, t_f$, we would now like to estimate $x_0(t_f)$, which, using (13) and $w(t) = w_0(t) + Nv(t)$, can be expressed as

$$\begin{aligned} x_0(t_f) &= A^{k-1}x_0(t_f - k + 1) + F_w W_0 \\ &= A^{k-1}x_0(t_f - k + 1) + F_w(W - D_V V) \end{aligned} \tag{14}$$

where

$$\begin{aligned} F_w &= \begin{bmatrix} A^{k-2}B & A^{k-3}B & \ldots & AB & B & 0 \end{bmatrix} \\ W_0 &= \begin{bmatrix} w_0(t_f - k + 1) \\ \vdots \\ w_0(t_f) \end{bmatrix} \quad W = \begin{bmatrix} w(t_f - k + 1) \\ \vdots \\ w(t_f) \end{bmatrix} \\ V &= \begin{bmatrix} v(t_f - k + 1) \\ \vdots \\ v(t_f) \end{bmatrix} \quad D_V = \begin{bmatrix} N & 0 & \ldots & 0 \\ 0 & N & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & N \end{bmatrix} \end{aligned}$$

In a similar manner, the second line of (13) gives

$$0 = R_x x_0(t_f - k + 1) + R_w(W - D_V V) \tag{15}$$

where

$$R_x = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{k-1} \end{bmatrix} \quad R_w = \begin{bmatrix} D & 0 & 0 & \ldots \\ CB & D & 0 & \ldots \\ CAB & CB & D & \ldots \\ \ldots & & & \\ CA^{k-2}B & \ldots & & D \end{bmatrix}$$

The value of k will be greater or equal to the number of states in (12) and since the residual generator is observable, it follows that R_x has full column rank. Let N_{R_x} be a matrix whose rows form a basis of the left null space of R_x, so that $N_{R_x} R_x = 0$. By multiplying (15) from the left with the full rank matrix

$$\begin{bmatrix} (R_x^T R_x)^{-1} R_x^T \\ N_{R_x} \end{bmatrix}$$

we get

$$x_0(t_f - k + 1) = -(R_x^T R_x)^{-1} R_x^T R_w (W - D_V V) \tag{16}$$

$$0 = N_{R_x} R_w (W - D_V V) \tag{17}$$

which is equivalent to (15). Elimination of $x_0(t_f - k + 1)$ in (14) implies that

$$x_0(t_f) = (-A^{k-1}(R_x^T R_x)^{-1} R_x^T R_w + F_w)(W - D_V V) \tag{18}$$

Now, to get an estimate of $x_0(t_f)$, first $E[V|W]$ can be computed using (17), and then $E[x_0(t_f)|W]$ can be computed from this result and (18).

Assuming that the distribution of V is known, say, $V \sim N(0, \Sigma_V)$, (17) gives

$$E[V|W] = \Sigma_V D_V^T R_w^T N_{R_x}^T \left(N_{R_x} R_w D_V \Sigma_V D_V^T R_w^T N_{R_x}^T\right)^{-1} N_{R_x} R_w W \tag{19}$$

and this together with (18) gives the estimate

$$\begin{aligned} \hat{x}_0(t_f) &= E[x_0(t_f)|W] \\ &= (-A^{k-1}(R_x^T R_x)^{-1} R_x^T R_w + F_w)(W - D_V \cdot E[V|W]) \end{aligned} \tag{20}$$

5.3. Determining the Length k of the Time-window

The choice of k is made in advance, based on the computed standard deviation $\sigma_r(t), t \geq t_f$ of the initial residuals given $\hat{x}_0(t_f)$. Figure 1 exemplifies how the standard deviation of an initialized residual may vary. The thick line in the bottom indicates the stationary standard deviation that the standard deviation of the initialized residual will converge to. The four curves above show the standard deviations obtained for indicated values of k. The larger k is, the more accurate initial state estimation $\hat{x}_0(t_f)$ and the closer the standard deviation $\sigma_r(t)$ comes to the stationary case. Hence, k can be chosen via a trade-off between minimizing the additional overhead that the computations in Section 5.2 represent and minimizing the maximum probability of false alarms during the initial time steps as follows.

Let $\bar{\sigma}_r$ be the maximum acceptable residual standard deviation which corresponds to a maximum acceptable probability of false alarms during the initial time steps. Further, given a k let K be the matrix such that $\hat{x}_0(t_f)$ is equal to KW according to (19) and (20). Note that k implicitly defines the matrices. Since $W = W_0 + D_V V$, the initial covariance of the estimated state $\hat{x}_0(t_f)$ is given by

$$\Sigma_{\hat{x}_0}(t_f) = K D_V \Sigma_V D_V^T K^T$$

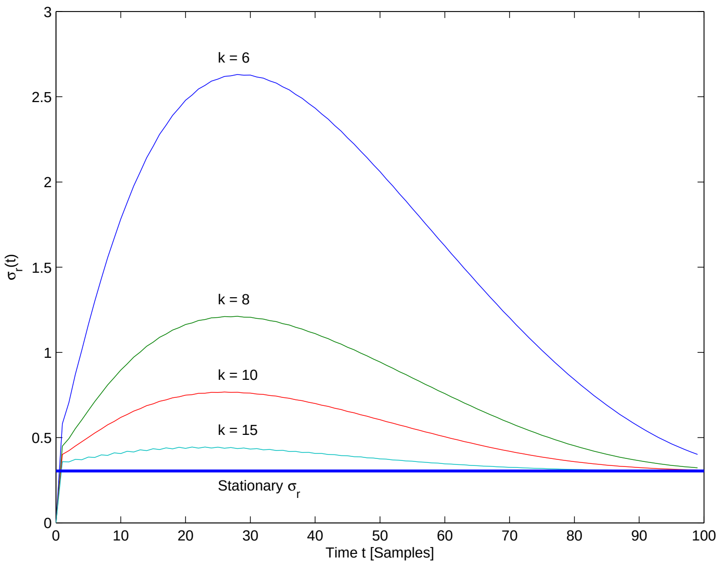

Figure 1: An example of how the standard deviation of a residual varies directly after initiation for different values of k. The line represents the standard deviation of the residual in the stationary case.

and the standard deviation of $r(t_f)$ by

$$\sigma_r(t) = \sqrt{C\Sigma_{\hat{x}_0}(t)C^T + DN\Sigma_v N^T D^T} \tag{21}$$

for $t = t_f$.

Figure 1 shows that the maximum standard deviation $\sigma_r(t)$ for $t \geq t_f$ is typically not obtained at $t = t_f$ and therefore both the state covariance and residual standard deviation have to be computed for $t \geq t_f$ based on

$$\Sigma_{\hat{x}_0}(t+1) = A\Sigma_{\hat{x}_0}(t)A^T + BN\Sigma_v N^T B^T$$

and (21). The k is selected as the smallest k such that

$$\sigma_r(t) \leq \bar{\sigma}_r \tag{22}$$

for all $t \geq t_f$.

6. Example

To illustrate the FlexDx framework, let us consider the simulated example system shown in Figure 2, where a DC-servo is connected to a flywheel through a

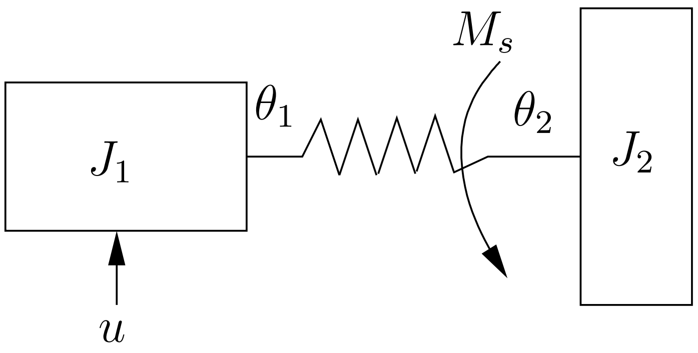

Figure 2: Illustration of the example process; a DC-servo connected to an inertia with a spring.

rotational (damped) spring. The system dynamics can be described by:

$$\begin{aligned}
J_1\ddot{\theta}_1(t) &= ku(t) - \alpha_1\dot{\theta}_1(t) - M_s(t)\\
M_s(t) &= \alpha_2(\theta_1(t) - \theta_2(t)) + \alpha_3(\dot{\theta}_1(t) - \dot{\theta}_2(t))\\
J_2\ddot{\theta}_2(t) &= -\alpha_4\dot{\theta}_2(t) + M_s(t)
\end{aligned}$$

where $u(t)$ is an input signal controlling the torque from the motor (with a scaling coefficient $k = 1.1$), $\theta_1(t)$ and $\theta_2(t)$ are the angles of the motor axis and the flywheel, respectively, and $M_s(t)$ is the torque of the spring. The moments of inertia in the motor is $J_1 = 1$ and for the flywheel $J_2 = 0.5$. The parameters $\alpha_1 = 1$ and $\alpha_4 = 0.1$ determine the viscous friction at the motor and flywheel respectively, while $\alpha_2 = 0.05$ is the spring constant and $\alpha_3 = 0.1$ the viscous damping coefficient of the spring.

As outputs, the motor axis angle and velocity, and the angle of the flywheel are measured. We will design the diagnostic system for six possible single faults $f_1(t), \ldots, f_6(t)$; one for each equation. The augmented system model becomes

$$\begin{aligned}
J_1\ddot{\theta}_1(t) &= k(u(t) + f_1(t)) - \alpha_1\dot{\theta}_1(t) - M_s(t)\\
M_s(t) &= \alpha_2(\theta_1(t) - \theta_2(t)) + \alpha_3(\dot{\theta}_1(t) - \dot{\theta}_2(t)) + f_2(t)\\
J_2\ddot{\theta}_2(t) &= -\alpha_4\dot{\theta}_2(t) + M_s(t) + f_3(t)\\
y_1(t) &= \theta_1(t) + f_4(t) + v_1(t)\\
y_2(t) &= \dot{\theta}_1(t) + f_5(t) + v_2(t)\\
y_3(t) &= \theta_2(t) + f_6(t) + v_3(t)
\end{aligned}$$

Here, $v_i(t)$, for $i = 1, 2, 3$, are measurement noise terms.

Table 1: The fault sensitivity of the residuals.

	f_1	f_2	f_3	f_4	f_5	f_6
r_1				X	X	X
r_2		X	X		X	X
r_3		X	X	X		X
r_4		X	X	X	X	
r_5	X		X		X	X
r_6	X		X	X		X
r_7	X		X	X	X	
r_8	X	X			X	X
r_9	X	X		X		X
r_{10}	X	X		X	X	
r_{11}	X	X	X			X
r_{12}	X	X	X		X	
r_{13}	X	X	X	X		

Since the diagnosis framework will work on sampled data, the model is discretized before designing the tests using a zero-order hold assumption. The noise is implemented as i.i.d. Gaussian noise with variance 10^{-3}. By using the second type of tests described in Section 4.3 for the discretized system, a set of 13 tests were needed. Their fault sensitivity is shown in Table 1. The false alarm probability is set to 10^{-3} and the maximum false alarm probability during test initiation $1.1 \cdot 10^{-3}$. To achieve this performance, the number of samples needed for initiating the 13 tests are, according to the method proposed in Section 5.3, 38, 86, 65, 92, 23, 40, 52, 69, 41, 42, 113, 82, and 108 respectively. The tests will in the following simulations be combined with the second test selection method described in Section 4.4.

6.1. Test Reconfiguration

To show how the diagnostic system is reconfigured during a fault transient, we will describe what happens when the fault f_1 occurs at $t = 100$ in a simulated scenario. The course of events is described in Table 2.

Each row in the table gives the most important properties of one iteration in the FlexDx procedure given in Section 2. In one such iteration, the set of active tests are executed on observations collected from time t_f to t_a. The column Minimal

Table 2: Diagnosis events

	t_f	t_a	Minimal Diagnoses	Active Tests
1	0	102.6	NF	$1, 2, \mathbf{5}$
2	98.9	102.7	$1, 3, 5, 6$	$1, 3, 10, \mathbf{13}$
3	98.9	102.2	$1, 3, 25, 26, 45, 46$	$1, 2, 6, 7, \mathbf{8}, 11, \mathbf{12}$
4	98.9	102.3	$1, 23, 25, 26, 35, 36, 45$	$1, 2, \mathbf{6}, 7, 9, 10, 11$
5	98.9	102.6	$1, 23, 26, 35, 36, 45$	$1, 2, 7, \mathbf{9}, 10, 11$
6	98.9	105.2	$1, 23, 26, 36, 45$	$1, 2, 7, 10, \mathbf{11}$
7	100.6	$-$	$1, 23, 26, 36, 245, 345, 456$	$1, 2, 7, 10$

Diagnoses shows a simplified representation of the minimal diagnoses during the corresponding phase. For example 25 represents the mode $\{f_2, f_5\}$. Each iteration ends when one or several of the active tests trigger an alarm. These are shown in bold type.

Let us take a closer look at the steps of the FlexDx procedure. Step 1 initiates the set of minimal diagnoses to $D = \{\text{NF}\}$, which is shown in row 1. The degree of redundancy of the behavioral model for NF is 3, and therefore 3 tests are needed to check if $w \in O(\text{NF})$ is consistent. Step 2 computes the first, in lexicographical ordering, minimum cardinality solution to (8), which is the test set $T = \{1, 2, 5\}$ given in row 1. Step 3 initiates the tests T and test 5 triggers an alarm at time $t_a = 102.6$. From the fault sensitivity of residual r_5 given in Table 1, $C = \{f_1, f_3, f_5, f_6\}$ becomes a conflict which is the output of step 4. The new set of minimal diagnoses, computed in step 5, is shown in the second row. Returning to step 2, the degree of redundancy for each of the behavioral models corresponding to minimal diagnoses is 2, and therefore at least two tests are needed to check the consistency of each of them. The minimum cardinality test set computed in step 2 is $T = \{1, 3, 10, 13\}$. This set is shown in row 2. Tests 1 and 3 check the consistency of $\{f_1\}$, 1 and 10 the consistency of $\{f_3\}$, 3 and 13 the consistency of $\{f_5\}$, and 10 and 13 the consistency of $\{f_6\}$. In step 3, the last fault free time is estimated to $t_f = 98.9$ by using the alarming residual r_5. The initial states of the residuals used in the tests T are estimated using observations sampled in a time interval ending at t_f. Proceeding in this way, FlexDx finds in row 4 that $\{f_1\}$ is the only consistent single fault and then the multiple fault diagnoses are further refined. In row 7 the last fault free time is estimated to be 100.6 due to slow fault response in test 11. A correct estimation can be obtained by tuning the length of the time-window and the threshold of the fault time estimator (11).

One straightforward option to further decrease the computational load is to modify the approach to focus on single faults, as indicated in Section 4.5. In such a case, the first two iterations in Table 2 are the same since the minimal diagnosis only include single faults. However, after the triggering of test 13, also multiple faults appear in the minimal diagnoses. If only single fault diagnoses are considered, only the diagnoses $\{f_1\}$ and $\{f_3\}$ are left. Then, by disregarding the multiple faults diagnoses, it is concluded that tests $T = \{1, 2, 8\}$ should be started instead of the set $T = \{1, 2, 6, 7, 8, 11, 12\}$ which is needed when also considering the multiple fault diagnoses. The next test to trigger is test 8 and then the only single fault diagnosis is $\{f_1\}$ and we have finalized the isolation procedure. From this it is clear that the number of tests needed can be further reduced by focusing on single faults. If the situation arises that no single fault diagnoses are left, focus is then shifted to double faults, and so on.

6.2. Reduction of the Computational Burden

Let us consider a second simulated scenario, where the system is started in the fault-free mode. At $t = 100$, f_1 is set to 0.2, and at $t = 200$, f_5 is set to 0.1. The residuals computed by the diagnostic system are shown in Figure 3. It is noteworthy that the residuals have not been computed for all time-points and thereby the expected cost reduction has been achieved. It is difficult to quantify the reduction in computational cost. In this case, where all residual generators are linear filters, one possibility is to evaluate the computational cost by examining the number of multiplication and addition operations used to compute the residuals. Using that approach in this simulation, by comparing the computational cost for a diagnostic system running all tests at all times with the computational cost with the proposed system without using focusing, a 98% reduction of the computational cost is obtained for the simulated scenario. This number is in itself not an indication of expected computational gain in a typical application. The reduction strongly depends on for example failure rates, degree of redundancy, complexity of the system model, and fault isolation requirements. The key point is that not all tests are run at all times, and during fault free operation, typically only a few tests are needed which typically results in a significant reduction in the computational load.

The largest number of tests is performed during the fault transitions which last only a short period of time. Although the computational load decreases with the approach, during fault transients, i.e. when faults are isolated, the number of tests

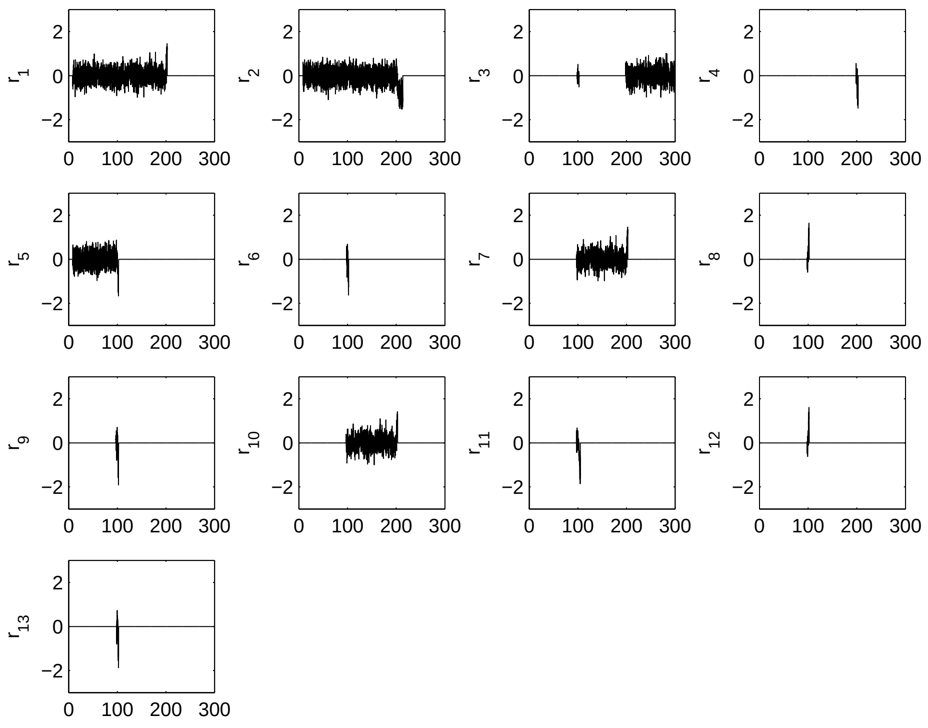
Figure 3: Residuals computed by FlexDx.

to run may still be too large with a too high computational load. Since the FlexDx framework includes the possibility to store observational data and run the tests in an asynchronous way, it is straightforward to serialize the test computations. Thereby, the need for high computational power can be traded against increased memory usage and increased time for detection and isolation.

7. DyKnow

To implement an instance of the FlexDx framework, a number of issues have to be managed besides implementing the described algorithms and integrating them in a system. When a potential fault is detected, FlexDx computes the last known fault free time t_f and the new set of residual generators to be monitored starting at time t_f. To implement this, three issues have to be solved. First, the FlexDx instance must be reconfigured to replace the set of residual generators and their monitors. Second, the computation of the residuals must begin at time t_f in the past. Third, at the same time as FlexDx is computing residuals and performing tests on the historic data, system observations will keep coming at their normal rate.

To manage these issues, FlexDx is implemented using DyKnow, a stream-based knowledge processing middleware framework for processing asynchronous streams of information [18]. Even though FlexDx could have been implemented with a dedicated solution with less overhead there are a number of benefits of using DyKnow. First, it provides solutions to the mentioned issues within an existing framework. Second, DyKnow provides a distributed infrastructure which allows sensors and other components to be hosted on many different computers in a network. This can be used both to collect data from distributed sensors and to distribute computations in the case were no computer is powerful enough.

DyKnow provides both a conceptual framework and an implementation infrastructure for integrating a wide variety of components and managing the information that needs to flow between them. It allows a system to incrementally process low-level sensor data and generate a coherent view of the environment at increasing levels of abstraction. Due to the need for incremental refinement of information at different levels of abstraction, we model computations and processes within the knowledge processing framework as active and sustained *knowledge processes*. The complexity of such processes may vary greatly, ranging from simple adaptation of raw sensor data to controllers to diagnosis algorithms.

The system being diagnosed by FlexDx is assumed to be synchronous. At the same time the diagnosis procedure is asynchronous, jumping back and forth in time trying to figure out which fault has occurred. This requires knowledge processes to be decoupled and asynchronous to a certain degree. In DyKnow, this is achieved by allowing a knowledge process to declare a set of *stream generators*, each of which has a *label* and can be *subscribed* to by an arbitrary number of processes. A subscription can be viewed as a continuous query, which creates a distinct asynchronous *stream* onto which new data is pushed as it is generated. Each stream is described by a declarative *policy* which defines both which generator it comes from and the constraints on the stream. These constraints can for example specify the maximum delay, how to approximate missing values or that the stream should contain samples added with a regular sampling period. Each stream created by a stream generator can have different properties and a stream generator only has to process data if it produces any streams. The contents of a stream may be seen by the receiver as data, information, or knowledge.

A stream-based system pushing information easily lends itself to “on-availability” processing, i.e. processing data as soon as it is available. This minimizes the processing delays, compared to a query-based system where polling introduces

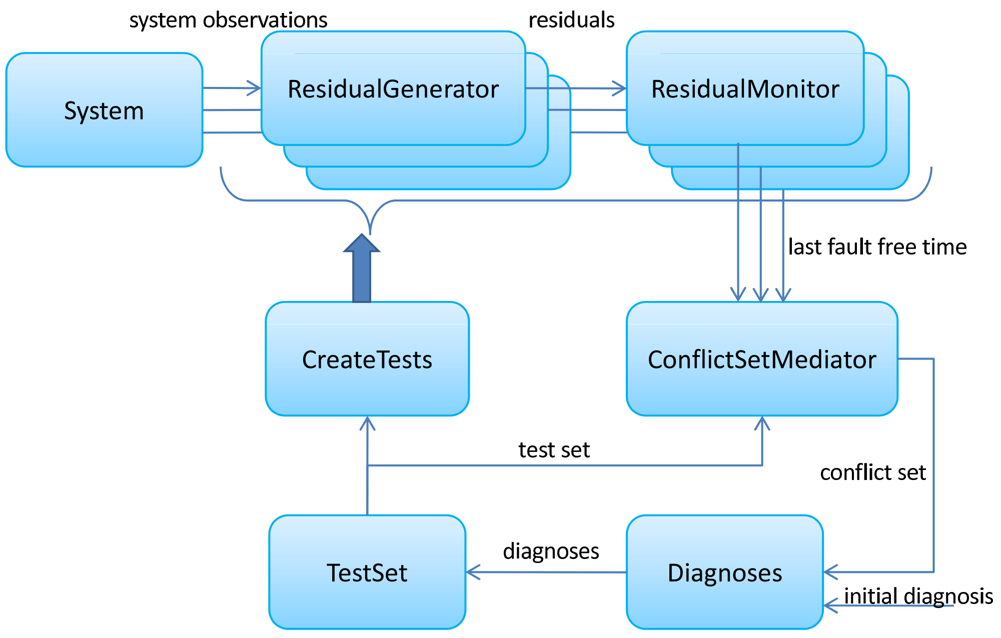

Figure 4: An overview of the components of the FlexDx implementation. The boxes are knowledge processes and the arrows are streams.

unnecessary delays in processing and the risk of missing potentially essential updates as well as wastes resources. This is a highly desired feature in a diagnostic system where faults should be detected as soon as possible.

For the purpose of modeling, DyKnow provides four distinct types of knowledge processes: primitive processes, refinement processes, configuration processes, and mediation processes. To introduce these processes and to describe how the three issues introduced by FlexDx are solved, we will use a concrete FlexDx instance as an example. An overview of the processes and streams is shown in Figure 4.

Primitive processes serve as an interface to the outside world, connecting to sensors, databases, or other information sources that in themselves have no explicit support for stream-based knowledge processing. Such processes have no stream inputs but provide a non-empty set of stream generators. In general, they tend to be quite simple, mainly adapting data in a multitude of external representations to the stream-based framework. For example, in FlexDx the stream of observations of the system being diagnosed is provided by a primitive process System.

The second process type to be considered is the *refinement process*, which takes a set of streams as input and provides one or more stream generators producing refined, abstracted, or otherwise processed values. In FlexDx there are four refinement processes, as seen in Figure 4:

- ResidualGenerator – Computes the residual for a particular test from system observations. The residual is initialized as described in Section 5.
- ResidualMonitor – Monitors a residual and checks whether it has triggered a test. This can either be a simple threshold check or a more elaborate test which checks properties of the residual over time, such as if it has been above or below the threshold for more than five consecutive samples. If a test has been triggered the process computes the last known fault free time, which is the output of the process.
- Diagnoses – Computes the new set of diagnoses each time a test has been triggered.
- TestSet – Computes the new set of residual generators to be monitored when the set of diagnoses changes.

The third type of process, the *configuration process*, takes a set of streams as input but produces no new streams. Instead, it enables dynamic reconfiguration by adding or removing streams and processes. In FlexDx a configuration process is required to handle the first issue, to be able to reconfigure the set of residuals and tests that are computed.

- CreateTests – Updates the set of residual generators and monitors as the set of tests changes. Each test consists of two refinement processes, one to compute the residual and one to monitor the test on the residual. In order to manage the second issue, that residuals are computed starting at the last known fault free time. The input to a residual is a stream which begins at this time-point. This is part of the policy the configuration process uses to set up the new residual generator process. Creating streams partially consisting of historic data is a DyKnow feature.

Finally, a *mediation process* generates streams by selecting or collecting information from other streams. Here, one or more of the inputs can be a stream of labels identifying stream generators to which the mediation process may subscribe. This allows a different type of dynamic reconfiguration in the case where not all potential inputs to a process are known in advance or where one does not want to simultaneously subscribe to all potential inputs due to processing costs. FlexDx uses a mediation process to collect the detected conflicts.

- ConflictSetMediator – Subscribes to the output of each of the tests and aggregates these to a single stream. When tests are added or removed the current set of subscriptions is updated accordingly. The output of this process is a stream of pairs, each pair containing the identifier of the test that was triggered and the last known fault free time for the corresponding residual.

Without any specific termination condition FlexDx will run as long as the system produces output and there are tests that have not been violated yet. It is possible to stop earlier, for example when there is a unique consistent single fault.

To give a concrete example of a run of the system, consider the example from Section 6 as described in Table 2. When the system is started, tests 1, 2, and 5 are created by CreateTests. These are computing the residuals and performing tests from time 0 to 102.6, when test 5 is triggered. Then the refinement process for test 5 computes the last known fault free time to 98.9. Using this information Diagnosis computes the set of minimal diagnosis to $\{1, 3, 5, 6\}$ and TestSet the new set of tests to $\{1, 3, 10, 13\}$. The old tests 2 and 5 are removed and the new tests are added by CreateTests. All of the tests are computed from time 98.9 until time 102.7 when test 13 is triggered, which means that they are computed from historic data until time 102.6. In this manner the set of tests is updated one more time before concluding that f_1 is the only consistent single fault.

The FlexDx algorithms are implemented in Matlab and integrated through code generation into DyKnow which is implemented in C++ using CORBA as a communication infrastructure.

8. Conclusions

FlexDx, an implemented reconfigurable diagnosis framework is proposed. It reduces the computational burden of performing multiple fault diagnosis by only running the tests that are currently needed. This involves a method for dynamically starting new tests. An important contribution is a method to select tests such that the computational burden is reduced while maintaining the isolation performance of the diagnostic system. Key components in the approach are test selection and test initialization. To illustrate that the general framework can be instantiated, specific algorithms for diagnosing linear dynamical systems have been developed for each component.

Implementing a reconfigurable diagnosis framework such as FlexDx introduces a number of interesting issues. First, FlexDx must be reconfigured to compute the new set of tests each time the set changes. Second, these computations must begin at the last known fault free time, which will be in the past. Third, at the same time as FlexDx is performing tests on historic data, system observations will keep coming at their normal rate. To handle these issues FlexDx is implemented using DyKnow, a stream-based knowledge processing middleware framework.

In the given example, the proposed approach has shown a significant reduction of the computational burden for a relatively small dynamical system. For systems with a high degree of redundancy, i.e. systems for which there exists many possible tests, the reduction can be expected to be even higher. Systems with low failure rate are also a class of systems where the approach can be expected to be advantageous, since then typically only a small subset of the tests are required to run continuously, rendering a significant reduction in computational burden.

[1] J. de Kleer, Diagnosing multiple faults, Artificial Intelligence 32 (1) (1987) 97–130.

[2] W. Hamscher, L. Console, J. de Kleer (Eds.), Readings in model-based diagnosis, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[3] M.-O. Cordier, P. Dague, F. L. J. Montmain, M. Staroswiecki, L. Trave-Massuyes, Conflicts versus analytical redundancy relations: A comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives, IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics 34 (5) (2004) 2163– 2177.

[4] B. Pulido, C. Gonzalez, Possible conflicts: a compilation technique for consistency-based diagnosis, IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics 34 (5) (2004) 2192– 2206.

[5] J. Gertler, Fault Detection and Diagnosis in Enginerering Systems, Marcel Dekker, Inc., 1998.

[6] R. J. Patton, P. M. Frank, R. N. Clark (Eds.), Issues of Fault Diagnosis for Dynamic Systems, Springer, 2000.

[7] M. Blanke, M. Kinnaert, J. Lunze, M. Staroswiecki, J. Schröder, Diagnosis and Fault-Tolerant Control, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[8] M. Basseville, I. Nikiforov, Detection of Abrupt Changes, PTR Prentice-Hall, Inc, 1993.

[9] M. Nyberg, A fault isolation algorithm for the case of multiple faults and multiple fault types, in: Proceedings of Safeprocess, 2006.

[10] M. Nyberg, M. Krysander, Combining AI, FDI, and statistical hypothesis-testing in a framework for diagnosis, in: Proceedings of IFAC Safeprocess'03, Washington, USA, 2003.

[11] S. Ploix, S. Touaf, J. M. Flaus, A logical framework for isolation in fault diagnosis, in: Proceedings of IFAC Safeprocess'03, Washington, USA, 2003.

[12] P. M. Frank, B. Köppen-Seliger, New developments using AI in fault diagnosis, Engineering Applications of Artificial Intelligence 10 (1) (1997) 3 – 14.

[13] M. Krysander, Design and analysis of diagnosis systems using structural methods, Ph.D. thesis, Linköpings universitet (Jun. 2006).

[14] J. de Kleer, J. Kurien, Fundamentals of model-based diagnosis, in: Proceedings of IFAC Safeprocess'03, Washington, USA, 2003, pp. 25–36.

[15] P. Frank, On-line fault detection in uncertain nonlinear systems using diagnostic observers: a survey, International Journal of Systems Science 25 (12) (1994) 2129–2154.

[16] M. Staroswiecki, G. Comtet-Varga, Analytical redundancy relations for fault detection and isolation in algebraic dynamic systems, Automatica 37 (5) (2001) 687–699.

[17] C. D. Persis, A. Isidori, A geometric approach to nonlinear fault detection and isolation, IEEE Trans. on Automatic Control 46 (6) (2001) 853–865.

[18] F. Heintz, DyKnow: A stream-based knowledge processing middleware framework, Ph.D. thesis, Linköpings universitet (Mar. 2009).

[19] J. de Kleer, A. Mackworth, R. Reiter, Characterizing diagnoses and systems, Artificial Intelligence 56 (2-3) (1992) 197–222.

[20] R. Reiter, A theory of diagnosis from first principles, Artificial Intelligence 32 (1) (1987) 57–95.

[21] J. Biteus, M. Nyberg, E. Frisk, J. Åslund, Determining the fault status of a component and its readiness, with a distributed automotive application, Engineering Applications of Artificial Intelligence 22 (3) (2009) 363–373.

[22] J. Biteus, Fault isolation in distributed embedded systems, Ph.D. thesis, Linköpings universitet (April 2007).

[23] H. Efendic, Model-on-Demand MATLAB Toolbox for Fault Diagnosis, in: Proceedings of the 5th International Conference on Circuits, Systems, Electronics, Control & Signal Processing, 2006.

[24] A. Korbicz, J. M. Koscielny, W. Cholewa, Z. Kowalczuk, Fault Diagnosis: Models, Artificial Intelligence, Applications, Springer, 2004.

[25] P. Struss, Testing for discrimination of diagnoses, in: Proceedings of DX, 1994.

[26] E. Benazera, L. Travé-Massuyès, A diagnosis driven self-reconfigurable filter, in: Proceedings of DX, 2007.

[27] M. Nyberg, E. Frisk, Residual generation for fault diagnosis of systems described by linear differential-algebraic equations, IEEE Transactions on Automatic Control 51 (12) (2006) 1995–2000.

[28] J. de Kleer, B. Williams, Diagnosis with behavioral modes, in: Readings in model-based diagnosis, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992, pp. 124–130.

[29] R. Nikoukhah, Innovations generation in the presence of unknown inputs: Application to robust failure detection, Automatica 30 (12) (1994) 1851–1867.

[30] E. Frisk, Residual generation in linear stochastic systems - a polynomial approach, in: Proc. of the 40th IEEE Conference on Decision and Control, 2001.

[31] J. W. Polderman, J. C. Willems, Introduction to Mathematical Systems Theory: A Behavioral Approach, Springer-Verlag, 1998.

[32] S. Tuhrim, J. Reggia, S. Goodall, An experimental study of criteria for hypothesis plausibility, Journal of Experimental & Theoretical Artificial Intelligence 3 (2) (1991) 129–144.

[33] E. Page, Continuous inspection schemes, Biometrika 41 (1954) 100–115.