FlexDx: A Reconfigurable Diagnosis Framework

Fredrik Heintz[†]**, Mattias Krysander**[∗]**, Jacob Roll**[∗]**, Erik Frisk**[∗]

† Dept. of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden
Email: `frehe@ida.liu.se`
∗ Dept. of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden
Email: {`matkr,roll,frisk`}`@isy.liu.se`

Abstract

Detecting and isolating multiple faults is a computationally intense task which 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 by only running the tests that are currently needed. The method selects tests such that the isolation performance of the diagnostic system is maintained. Special attention is given to the practical issues introduced by a reconfigurable diagnosis framework such as FlexDx. 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 uses DyKnow, a stream-based knowledge processing middleware framework. The approach is exemplified on a relatively small dynamical system, which still illustrates the computational gain with the proposed approach.[1]

1 Introduction

Detection and isolation of multiple faults in a dynamic process is a computationally expensive task, and the cost increases rapidly with the number of faults and the model complexity. A real-time, model-based diagnosis system that supervises a dynamic system with non-linear behavior often consists of a set of precompiled diagnostic tests together with a fault isolation module [3; 14]. The diagnostic tests are based on a formal description of the process, often in the form of differential or difference equations. For this type of system, pre-compiled test is an attractive solution compared to e.g. solutions based on propagating values like GDE.

The computational complexity of such a diagnosis system mainly originates from two sources: complexity of the process model and the number of behavioral modes that are considered. A high capability of distinguishing between faults, especially when multiple faults are considered, requires a large number of diagnostic tests [9]. Also, the more complex the process model is, the more computationally intense is the task of executing the diagnostic tests. In this paper we develop a reconfiguration scheme to handle computational issues while still being able to handle multiple faults. A related approach is presented in [17] although the models and diagnosis techniques are different. Recently, works on on-line reconfiguration of the diagnosis system have appeared. For a related work, see e.g. [2], where Kalman-filters are reconfigured based on diagnosis decisions.

The main idea of this work is to utilize the observation that all tests are not needed at all times, which can be used to reduce the overall computational burden. 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 make it possible to compute a refined diagnosis decision. Such an approach requires a flexible and reconfigurable framework where tests can be added and removed on-line in a controlled fashion, and also be run on historical data.

The objective of this paper is to illustrate how such a dynamic approach to diagnosis can be designed and implemented using linear dynamical process models. In particular, the implementation issues introduced by a reconfigurable diagnosis framework are discussed and a solution using DyKnow [7; 8], a stream-based knowledge processing middleware framework, is described. It will also be shown how such an approach requires controlled ways of initializing the dynamic diagnostic tests, and how to select the new tests to be started when a set of diagnostic tests has generated an alarm.

The reconfigurable diagnosis framework proposed in this paper, named FlexDx, is introduced in Section 2, and the theoretical diagnosis background needed is presented in Section 3. Methods how to determine, in a specific situation, which tests should be started next are treated in Section 4. A proper initialization procedure for dynamic tests is described in Section 5. The complete approach is exemplified on a small dynamic system in Section 6, which, in spite of the relatively small size of the example, clearly illustrates the complexity of the problem and the possible computational gain with the proposed approach. The software framework which facilitates the implementation of FlexDx, DyKnow, is briefly described in Section 7, and finally a summary is given in Section 8.

---

2 FlexDx: A Reconfigurable Diagnosis Framework

As mentioned in the introduction, a framework like FlexDx must be capable of adding and removing tests dynamically while refining the set of diagnoses. This is done in an iterative manner by 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 current set of diagnoses based on the test results, then go to step 2.

When dealing with multiple fault diagnosis, it has been shown useful to represent all diagnoses with the minimal diagnoses [5]. This representation will also be used here. 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 $\{NF\}$ 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. This means that the start-up of such a residual generator involves the estimation of its initial condition. In step 4, the tests are performed until at least one triggers an alarm and a test result is generated in the form of a set of conflicts [4; 16]. 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 [4; 11].

Step 4 and 5 are standard steps used in diagnosis 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 and integrating them to 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 started 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 which will be 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. How these issues are solved is described in Section 7.

3 Theoretical Background

The diagnosis systems considered in this paper consist of a set of 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 of loss of generality. It is assumed that the residuals are normalized such that a given false alarm probability p_{FA} is obtained, i.e.

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

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. It is worth noting that even if the presentation here in the paper relies on results for linear systems, the basic idea is equally applicable also 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

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

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 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 of a single fault. Let $\mathcal{F}$ be the set of faults.

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

$$H_0 : f_i = 0 \text{ for all } f_i \in \mathcal{F}_0$$
$$H_1 : f_i \neq 0 \text{ for some } f_i \in \mathcal{F}_0$$

where $\mathcal{F}_0 \subseteq \mathcal{F}$ is the set of faults the residual is designed to detect. This means that the residual is not supposed to detect all faults, only the faults in $\mathcal{F}_0$. By generating a set of such residuals, each sensitive to different subsets $\mathcal{F}_0$ 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. In this paper an adapted version of the innovation filter defined in [10] 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 $f_i \in \mathcal{F}_0$. 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 [6] 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 it is immediate that the internal form of the residual is given by

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

Thus, the fault sensitivity is given by

$$r = -Q(q)F(q)f \quad (4)$$

and the statistical properties of the residual under H_0 by

$$r = Q(q)V(q)v \quad (5)$$

A complete design procedure is given in e.g. [10] for state-space models and in [6] for models on 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 [16] C is generated.

Now we can relate the test results to a diagnosis. Let a set $b \subseteq \mathcal{F}$ represent a system behavioral mode with the meaning that $f_i \neq 0$ for all $f_i \in b \subseteq \mathcal{F}$ and $f_j = 0$ for all $f_j \notin b$. In short, see [16] for details, the behavioral mode b is then a diagnosis if it can explain all generated conflicts, i.e. if b has a non-empty intersection with each generated conflict. Algorithms to compute all minimal diagnoses for a given set of conflicts, which is equivalent to the so called hitting set problem, can be found in e.g. [4; 16]. 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 this means that both f_2 and f_3 can not be 0. Thus, 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 $b_2 = \{f_1, f_2\}$. ◇

4 Test Selection

This section describes step 2 in the FlexDx procedure given in Section 2, i.e. how the set of tests T is selected given a set

D of minimal diagnoses. There are many possible ways how this can be done. The method that will be described here is based on the deterministic properties of (2) only and relies on basic principles in consistency-based diagnosis.

A fundamental task in consistency-based diagnosis is to compute the set of consistent modes [4] given a model, a set of possible behavioral modes, and observations. The design goal of the test selection algorithm will be to perform tests such that the set of consistent modes is equal to the set of diagnoses computed by the diagnosis system.

4.1 Consistent Behavioral Modes

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

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

This means that a mode b is consistent with the deterministic part of model (2) and an observation w if $w \in O(b)$. Hence, to achieve the goal 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 [12] 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. By defining the set of consistent observations for tests in a similar way as for models, we define

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

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

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

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 sets requirements on the entire set of tests $\mathcal{T}$. If such set of tests is difficult to obtain for a particular model,

any set of tests will do. By applying the approach to a model consisting of the considered set of tests, a diagnosis 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*

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

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

$$
O(\emptyset) = \{w| \begin{bmatrix} w_1(t) + \alpha w_2(t) - w_2(t+1) \\ -w_2(t) + w_3(t) \end{bmatrix} = 0\}
$$

$$
O(\{f_1\}) = \{w| - w_2(t) + w_3(t) = 0\}
$$

$$
O(\{f_2\}) = O(\{f_4\}) = O(\{f_2, f_4\}) =
$$
$$
= \{w|w_1(t) + \alpha w_2(t) - w_2(t+1) = 0\}
$$

$$
O(\{f_3\}) = \{w|w_1(t) + \alpha w_3(t) - w_3(t+1) = 0\}
$$

The behavioral models for the 10 remaining 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 [15]. ◇

The first type of test set $\mathcal{T}_1$ will be to design one test for each distinct behavioral model containing redundancy, i.e., for the example $\mathcal{T}_1$ consists of four tests t_i such that $O(t_1) = O(\emptyset)$, $O(t_2) = O(\{f_1\})$, $O(t_3) = O(\{f_2\})$, 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 a 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 [9] 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.

5 Initialization

When a new test selection has been made, 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 running before the currently considered fault occurred; otherwise valuable information would 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.

5.1 Estimating the Fault Time

There are many possibilities to estimate the fault time. See for example [13; 1] 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, $r(1), \ldots, r(k)$, let us compute the sum of the squared residuals over a sliding window, i.e.,

$$
S(t) = \frac{1}{\sigma^2} \sum_{j=1}^{\ell} r^2(t+j), \qquad t = 0, \ldots, k - \ell
\tag{10}
$$

If the residual generator is designed such that, under the null hypothesis that no fault has occurred, $(r(j))_{j=1}^k$ 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. 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 $w(t_f - k), \ldots, w(t_f)$ 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:

$$x(t+1) = Ax(t) + Bw(t) \qquad (11)$$
$$r(t) = Cx(t) + Dw(t) \qquad (12)$$

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 $v(t) = 0$,

$$x_0(t+1) = Ax_0(t) + Bw_0(t) \qquad (13)$$
$$0 = Cx_0(t) + Dw_0(t) \qquad (14)$$

Given $w(t)$, $t = t_f - k, \ldots, t_f$, we would like to estimate $x_0(t_f)$. This will be done by first estimating $x_0(t_f - k)$.

From (13) and $w(t) = w_0(t) + Nv(t)$ we get

$$0 = R_x x_0(t_f - k) + R_w W_0$$
$$\Leftrightarrow \quad R_x x_0(t_f - k) + R_w W = R_w D_V V \qquad (15)$$

where

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

$$W = \begin{bmatrix} w(t_f - k) \\ \vdots \\ w(t_f) \end{bmatrix} \qquad W_0 = \begin{bmatrix} w_0(t_f - k) \\ \vdots \\ w_0(t_f) \end{bmatrix}$$

$$V = \begin{bmatrix} v(t_f - k) \\ \vdots \\ v(t_f) \end{bmatrix} \qquad D_V = \begin{bmatrix} N & 0 & \ldots & 0 \\ 0 & N & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & N \end{bmatrix}$$

Assuming that the distribution of V is known, say, $V \sim N(0, \Sigma_V)$, (15) means that $R_x x_0(t_f - k) + R_w W$ is a zero-mean stochastic vector with covariance matrix $R_w D_V \Sigma_V D_V^T R_W^T$. Note that the expression above corresponds to the actual residuals obtained when starting in $x_0(t_f - k)$. Due to the design of the residual generator giving white residuals, this means that $R_w D_V \Sigma_V D_V^T R_w^T \approx \sigma^2 I$. Hence, a reasonable estimate of $x_0(t_f - k)$ is given by the regular least-squares estimate,

$$\hat{x}_0(t_f - k) = -(R_x^T R_x)^{-1} R_x^T R_w W \qquad (16)$$

From this, $\hat{x}_0(t_f)$ can be computed as

$$\hat{x}_0(t_f) = A^k \hat{x}_0(t_f - k) + \\ \begin{bmatrix} A^{k-1}B & A^{k-2}B & \ldots & AB & B & 0 \end{bmatrix} W$$
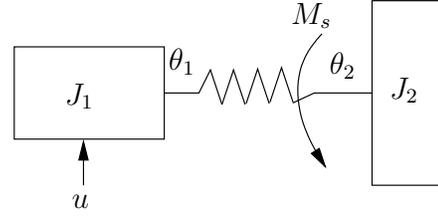

Figure 1: Illustration of the example process; a DC-servo connected to an inertia with a spring.

The choice of k is made in advance, based on the computed variance of the initial residuals given $\hat{x}_0(t_f)$. The larger k is, the closer this variance comes to the stationary case. Hence, k can be chosen via a trade-off between the minimizing the additional overhead that the above computations represent, and minimizing the maximum probability of false alarms during the initial time steps.

6 Example

To illustrate the FlexDx framework, let us consider the simulated example system shown in Figure 1, where a DC-servo is connected to a flywheel through a rotational (damped) spring. The system dynamics can be described by:

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

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 diagnosis system for six possible single faults $f_1(t), \ldots, f_6(t)$; one for each equation. The augmented system model becomes

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

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

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 and their fault sensitivity

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		

Figure 2: Residuals computed by FlexDx.

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	–	$1, 23, 26, 36, 245, 345, 456$	$1, 2, 7, 10$

is shown in Table 1. These 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 diagnosis 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 diagnoses shows a simplified representation of the minimal diagnoses during the corresponding phase. 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, are shown in the second row. Returning to step 2, the degree of redundancy for each of the behavioral models corresponding to minimal diagnoses are 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.
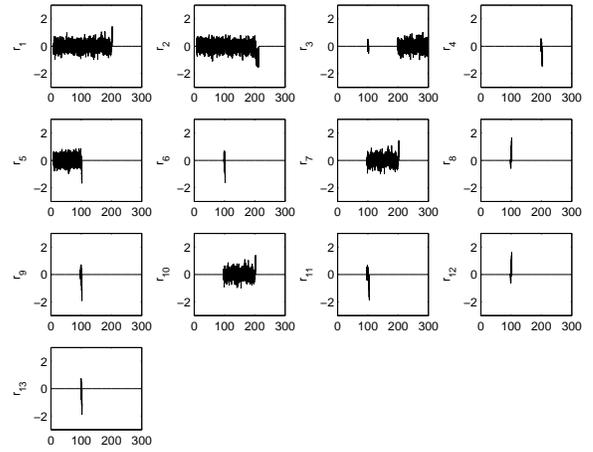
6.2 Reduction of the Computational Burden

In a simulated scenario, 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 diagnosis system are shown in Figure 2. It is noteworthy that the residuals have not been computed for all time-points. By comparing the number of residuals computed for a diagnosis system running all tests at all times with the number of residuals computed with the proposed system, a 78.3% reduction in the number of computed residuals is obtained for the simulated scenario. This number is in itself not an indication of expected computational gain in a typical application. For systems with low failure rate, more redundancy, or more complex system model the reduction will typically be much larger. 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. The largest number of tests is performed during the fault transitions which lasts only a short period of time.

7 DyKnow

To implement an instance of the FlexDx framework, a number of issues have to be managed besides implementing the algorithms and integrating them to 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 implementing applications processing asynchronous streams of information [7; 8].

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 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 sample 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 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 3.

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
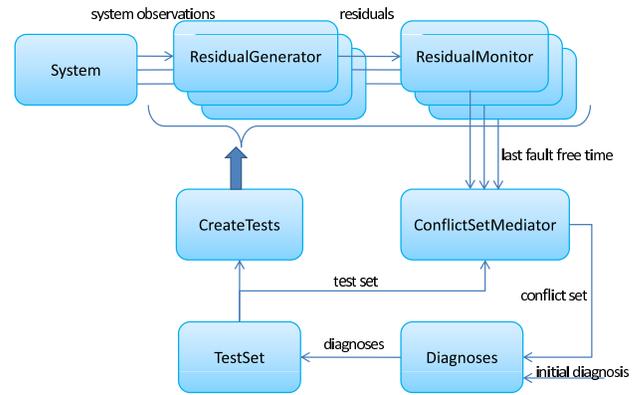

Figure 3: An overview of the components of the FlexDx implementation. The boxes are knowledge processes and the arrows are streams.

data in a multitude of external representations to the stream-based framework. For example, in FlexDx the initial diagnosis and the stream of observations of the system being diagnosed are seen as a primitive processes 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 3:

- 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; this is the output of the process.

- Diagnosis – 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 issues, 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 other streams 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 cost. 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.

FlexDx will continue to add new tests until there is exactly one consistent single fault or all tests have been added.

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 1, 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. If there are no consistent single faults FlexDx will continue to add tests until all have been evaluated.

8   Summary

An implemented reconfigurable diagnosis framework FlexDx 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. Specific algorithms for diagnosing linear dynamical systems have been developed to illustrate the diagnosis framework, but the framework itself is general.

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, and for larger systems the reduction is expected to be higher.

References

[1] M. Basseville and I.V. Nikiforov. *Detection of Abrupt Changes*. PTR Prentice-Hall, Inc, 1993.

[2] E. Benazera and L. Travé-Massuyès. A diagnosis driven self-reconfigurable filter. In *Proc. DX'07*, 2007.

[3] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki. *Diagnosis and Fault-Tolerant Control*. Springer, 2003.

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

[5] J. de Kleer, A. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56, 1992.

[6] Erik Frisk. Residual generation in linear stochastic systems - a polynomial approach. In *Proc. of the 40th IEEE Conference on Decision and Control*, 2001.

[7] Fredrik Heintz and Patrick Doherty. DyKnow: An approach to middleware for knowledge processing. *Journal of Intelligent and Fuzzy Systems*, 15(1):3–13, November 2004.

[8] Fredrik Heintz and Patrick Doherty. A knowledge processing middleware framework and its relation to the JDL data fusion model. *Journal of Intelligent and Fuzzy Systems*, 17(4):335–351, 2006.

[9] Mattias Krysander. *Design and Analysis of Diagnosis Systems Using Structural Methods*. PhD thesis, Linköpings universitet, June 2006.

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

[11] Mattias Nyberg. A fault isolation algorithm for the case of multiple faults and multiple fault types. In *Proceedings of IFAC Safeprocess'06*, 2006.

[12] Mattias Nyberg and Erik Frisk. Residual generation for fault diagnosis of systems described by linear differential-algebraic equations. *IEEE Transactions on Automatic Control*, 51(12), 2006.

[13] E.S. Page. Continuous inspection schemes. *Biometrika*, 41:100–115, 1954.

[14] R. J. Patton, P. M. Frank, and R. N. Clark, editors. *Issues of Fault Diagnosis for Dynamic Systems*. Springer, 2000.

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

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

[17] Peter Struss. Testing for discrimination of diagnoses. In *Proc. of DX'94*, 1994.