Dynamic Test Selection for Reconfigurable Diagnosis

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 intense task which typically consists of computing a set of tests, and then computing the diagnoses based on the test results. This paper proposes a method to reduce the computational burden by only running the tests that are currently needed, and dynamically starting new tests when the need changes. A main 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 the test selection algorithm, the test initialization procedures, and a knowledge processing framework that supports the functionality needed. The approach is exemplified on a relatively small dynamical system, which still illustrates the complexity and possible computational gain with the proposed approach.** [1]

I. 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 model complexity. A real-time, model-based diagnosis system often consists of a set of pre-compiled 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.

The computational complexity of such a diagnosis system mainly originates from two sources: complexity of the process model and the number of fault 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. This paper develops a reconfiguration scheme to handle computational issues while still being able to handle multiple faults. Recently, works on on-line reconfiguration of the diagnosis system has 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 stopped and restarted 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, it will be shown how such an approach requires controlled ways of initializing the dynamic diagnostic tests, and how to select which new tests that need to be started when a certain set of diagnostic tests has generated an alarm.

The reconfigurable diagnosis framework proposed in this paper is introduced in Section II, and the theoretical diagnosis background needed is presented in Section III. Methods how to determine, in a specific situation, which tests that should be started next are treated in Section IV. A proper initialization procedure for dynamic tests is described in Section V. The complete approach is exemplified on a small dynamic system in Section VI, 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 computational framework used to implement the approach, DyKnow, is briefly described in Section VII, and finally some conclusions are given in Section VIII.

II. FlexDx: A Reconfigurable Diagnosis Framework

This section gives an overview of the proposed reconfigurable diagnosis framework, named FlexDx. As mentioned in the introduction, the approach must be capable of switching on and off 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.

[1]This work is partially supported by grants from the Swedish Aeronautics Research Council (NFFP4-S4203) and the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII.

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 $\{\text{NF}\}$ in step 1. Step 2 uses a function that given a set of diagnoses D returns the set of tests T to be performed next. Step 3 initiates each of the tests in T. A test might include a residual generator given in state-space form. This means that the start-up of such a test involves the estimation of the initial condition of the residual generator. In step 4, the tests are performed until at least one of them triggers an alarm and a test result is generated in the form of a set of conflicts [4], [16]. Step 5 is a function that computes the current set of diagnoses D, given the previous set of diagnoses and the generated set of conflicts. This step can be performed with 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, while step 2 and 3 are new steps needed for dynamically changing the test set T. The details of these steps are given in Section IV and V respectively. But before that, a brief theoretical background on test construction for dynamic systems.

III. 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.

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

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

A. The Model

The class of models considered here are linear differential-algebraic models. It is worth noting that even if the developed approach 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 similar results can be adopted for other model formulations, e.g. 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 includes 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).

B. 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 single fault mode and let $\mathcal{F}$ be the set of faults.

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

$$\begin{aligned} H_0 &: f_i = 0,\ f_i \in \mathcal{F}_0 \\ H_1 &: f_i \neq 0 \text{ for some } f_i \in \mathcal{F}_0 \end{aligned}$$

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 III-C.

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

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

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 is given by

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

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 and that the main computational steps involve a null-space computation and a spectral factorization.

C. 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, then at least one of the faults in C must have occurred and the conflict [16] C is generated.

Let a set $b \subseteq \mathcal{F}$ represent a system behavioral mode, which means that $f_i \neq 0$ for all $f_i \in b \subseteq \mathcal{F}$ and $f_j = 0$ for all $f_j \notin b$. The behavioral mode b is then a diagnosis if it can explain all generated conflicts, i.e. b has a non-empty intersection with each generated conflict. A diagnosis b is considered a *minimal* diagnosis if no proper subset of b is a diagnosis [4], [16]. 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 [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 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 $b_2 = \{f_1, f_2\}$. $\diamond$

IV. Test Selection

This section describes step 2 in the procedure given in Section II, 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, and 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.

A. 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

$$O(b) = \{w | \exists x, \exists_{f_i \in b} f_i : H(q)x + L(q)w + \sum_{f_i \in b} F_i(q) f_i = 0\} \tag{6}$$

This means that a mode b is consistent with the deterministic part of model (2) and an observation w if $w \in O(b)$. 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 II, 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.

B. Tests for Checking Model Consistency

Next, we will describe how tests can be used to detect if $w \notin O(b)$. Let $\mathcal{T} = \{t_i | i \in \{1, 2, \ldots\}\}$ 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. 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 are ready to characterize all test sets T that are capable of detecting any inconsistency of $w \in O(b)$. 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

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

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

C. 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 $\mathcal{T}$. 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

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

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

$$\begin{aligned} O(\emptyset) &= \{w | \begin{bmatrix} w_1(t) + \alpha w_2(t) - w_2(t+1) \\ -w_2(t) + w_3(t) \end{bmatrix} = 0\} \\ O(\{f_1\}) &= \{w | -w_2(t) + w_3(t) = 0\} \\ O(\{f_2\}) &= O(\{f_4\}) = O(\{f_2, f_4\}) = \\ &= \{w | w_1(t) + \alpha w_2(t) - w_2(t+1) = 0\} \\ O(\{f_3\}) &= \{w | w_1(t) + \alpha w_3(t) - w_3(t+1) = 0\} \end{aligned}$$

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]. $\diamond$

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 and this number is the 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 stochastics 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 $b \in B$.

D. 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 sometimes 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.

The most straightforward method is to use the first type of tests and not require minimum cardinality solutions. Since the first 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 VI where this method has been applied to a larger example.

V. 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 tests results immediately. Therefore, the initialization of a new test consists of two steps:

1) Estimate the time of the fault.
2) Estimate the initial condition.

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.

A. 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.

B. 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:

$$x(t+1) = Ax(t) + Bw(t) \tag{11}$$

$$r(t) = Cx(t) + Dw(t) \tag{12}$$

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 $v(t) = 0$, i.e.,

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

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

$$\begin{aligned} & 0 = R_x x_0(t_f - k) + R_w W_0 \\ \Leftrightarrow \quad & R_x x_0(t_f - k) + R_w W = R_w D_V V \end{aligned} \tag{15}$$

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,

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

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

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

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 based on a maximum probability of false alarms during the initial time steps.

VI. 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 the following equations:

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

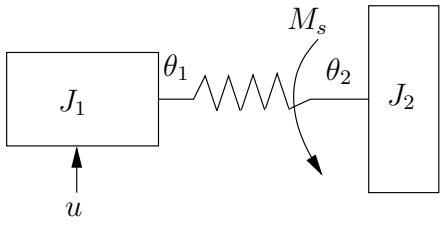

Fig. 1. Illustration of the example process; a DC-servo connected to an inertia with a spring.

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

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

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

Since the diagnosis framework will work on sampled data, we discretize the model before designing the tests, using a zero-order hold assumption. The noise is implemented as i.i.d. Gaussian noise with variance 10^{-3}. Here, the second type of tests described in Section IV-C for the discretized system is used. Tests for all behavioral models with degree of redundancy 1 result in a set of 13 tests. Their corresponding fault sensitivities are obtained directly from the model equations, using expression (4), and are shown in Table I.

A. Reduction of the Computational Burden

To quantify the reduction in computational burden in this example, a simple measure is used, where the number of residual values computed in the FlexDx framework is compared to the number of residual values computed for the case where no dynamic reconfiguration of tests is used. This is of course a coarse measure: for instance, it is not taken into consideration that different tests may have drastically different computational requirements. However, since the objective here is to illustrate general principles rather than to quantify an exact reduction for a particular example, the simple approach is deemed sufficient.

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 $t = 200$,

TABLE I

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		

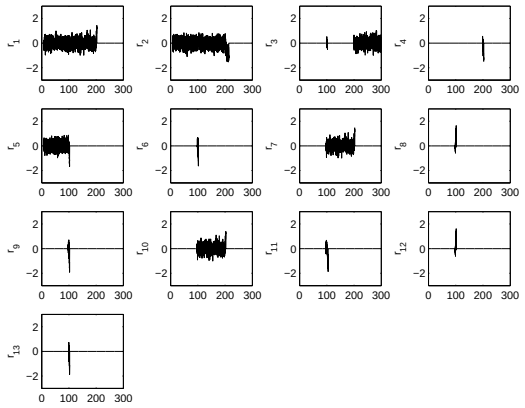

Fig. 2. Residuals computed by FlexDx.

f_5 is set to 0.1. The residuals computed by the diagnosis system are shown in Figure 2. It is important to note that during time intervals where the residual is plotted as being exactly 0, no residual computations are performed, and thus residuals are not computed for all the time-points. Here the second test selection method in Section IV-D has been used. By comparing the number of residual values computed for a diagnosis system running all tests at all times with the number of residuals computed with the proposed system, a 78% reduction of the number of computed residuals is obtained for the simulated scenario. Further, the figure shows that the largest number of tests is performed during the fault transitions for a short period of time. The reduction obtained in the example is significant and for systems with a low failure rate or high degree of redundancy, a larger reduction can be expected.

B. Test Reconfiguration

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

Each row in the table gives the most important properties of one iteration in the procedure given in Section II. 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 and these are in bold type.

TABLE II

DIAGNOSIS EVENTS

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

A step by step description of the procedure given in Section II will be given next. 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 I, $C = \{f_1, f_3, f_5, f_6\}$ 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 fault 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 before time t_f. Proceeding in this way, the diagnosis system concludes in row 4 that $\{f_1\}$ is the only consistent single fault.

VII. IMPLEMENTATION ISSUES

To implement the FlexDx framework, a number of issues have to be managed besides implementing the algorithms and connecting them into a system. When a potential fault is detected, FlexDx computes the last known fault free time t_f and the new set of residuals to be tested starting at time t_f. To implement this, three issues have to be solved. First, FlexDx must be reconfigured to compute the new set of residuals and their tests. Second, these computations 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.

To manage these issues, FlexDx is implemented using DyKnow, a knowledge processing middleware framework for describing, implementing and interacting with applications

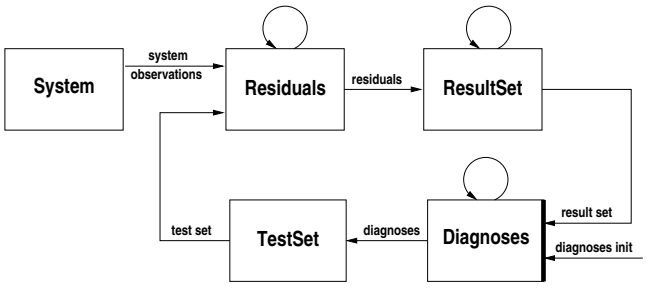

Fig. 3. An overview of the components of the FlexDx implementation. The boxes are computational units and the arrows are streams. A self referring arrow means that a computational unit has an internal state which is fed back to itself. The thick black line means that the inputs are synchronized in time.

processing asynchronous streams of information [7], [8]. DyKnow processes streams on many levels of abstractions generated by different components in a distributed system mediating information between sensing and deliberating processes. These streams can be viewed as time-series and can start as continuous streams of sensor readings. Processes combine such streams by computing, synchronizing, filtering and approximating to derive higher level abstractions.

A DyKnow application consists of a set of sources representing processes providing external streams (e.g., sensor inputs), a set of computational units representing processes on streams, and a set of labeled streams generated from the sources and computational units. A computational unit can encapsulate any computation on one or more streams. Examples of computational units are filters and other signal processing algorithms, but also more complex procedures such as the test selection algorithm presented in Section IV. Each stream is described by a declarative policy which defines both which source 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.

An overview of the FlexDx implementation is shown in Figure 3. It consists of four computational units: Residuals to compute the residuals, ResultSet to perform the tests on the residuals, Diagnoses to compute the current diagnoses based on the test results, and TestSet which computes the set of tests to be performed based on the current diagnoses. The computational units are connected by streams. The input to FlexDx is a stream of system observations coming from a source System and the initial diagnoses.

The three features of DyKnow which provides the necessary support for FlexDx is the ability to buffer streams, to create streams starting from a time in the past and to replace computational units at run-time. When the set of tests changes, TestSet will replace Residuals and ResultSet with new instances computing the current residuals and tests. It will also replace the stream of system observations by a new stream of system observations, but starting from the last known fault free time. The system will then resume operation as before, until the next fault is detected.

VIII. Discussion and Conclusions

The diagnosis framework proposed here 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, test initialization, and the knowledge processing middleware framework DyKnow that supports the needed functionality. Specific algorithms for diagnosing linear dynamical systems have been developed to illustrate the diagnosis framework, but the framework itself is more general. 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.

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 *18th International Workshop on Principles of Diagnosis (DX-07)*, pages 21–28, Nashville, USA, 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 *Proceedings of the 40th IEEE Conference on Decision and Control*, Orlando, Florida, 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*, Beijing, China, 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):1995–2000, 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.