Mariusz PELC
OPOLE UNIVERSITY OF TECHNOLOGY, FACULTY OF ELECTRICAL ENGINEERING, AUTOMATICS AND COMPUTER SCIENCE,
Sosnkowskiego 31, 45-272 OPOLE

Architecture for hierarchical policy-supervised fault detection component / middleware

PhD. eng. Mariusz PELC

His scientific activities are spread across various domains, which include automatic control and robotics, modern computer / embedded (control) systems, software systems as well as computer clusters and parallel processing. His most recent scientific activity is focused on autonomic and self-* systems and most of all - validation of verification of such systems, interoperability issues and certification of self-* systems. This recent work is carried on in close co-operation with the Computer Systems Department at the University of Greenwich in London.

e-mail: m.pelc@po.opole.pl

Abstract

In this paper an idea of a layered, hierarchical policy-supervised diagnostic / fault detection component and middleware supporting humans' decision-making will be presented. This system will be based on Open Decision Point architecture and as such the system will offer an ease of reconfiguration via policies replacement in the decision making component. For the diagnostic / decision making support purposes AGILE policies will be used. In order to present practical implementation of the subject architecture, a proof of concept application to the distillation column is described. Selected simulation results are used to show advantages of the proposed approach.

Keywords: computer control systems, autonomic systems, policy-based computing.

Architektura Hierarchicznego Zarządzanego Politykami Komponentu / Oprogramowania Pośredniczącego Systemu Detekcji Błędów

Streszczenie

W niniejszym artykule zaprezentowana zostanie architektura warstwowej, hierarchicznej zarządzanej politykami architektury komponentu i oprogramowania pośredniczącego dla systemu diagnostycznego. W ramach proponowanych rozwiązań wyróżnione zostały trzy funkcjonalne warstwy, które pozwalają na zwiększenie efektywności procesu decyzyjnego poprzez filtrowanie i preselekcję zdarzeń sygnalizowanych w systemie diagnostycznym / monitorującym. Oba zaproponowane rozwiązania (zarówno implementacja w obrębie jednego komponentu jak implementacja w systemie rozproszonym) wykorzystują architekturę Otwartego Punktu Decyzyjnego i jako takie oferują łatwość rekonfiguracji poprzez zamianę aktualnych polityk odpowiedzialnych za podejmowanie decyzji. Wsparcie dla systemu diagnostyki / podejmowania decyzji zapewniają polityki w języku AGILE. Aby zaprezentować praktyczną implementację tytułowych architektur opisane rozwiązania zostaną zastosowane w systemie sterowania kolumny destylacyjnej. Wybrane wyniki symulacji pokażą zalety zaproponowanego rozwiązania.

Słowa kluczowe: komputerowe systemy sterowania, systemy autonomiczne, przetwarzanie z wykorzystaniem polityk.

1. Introduction

Nowadays one can see very rapidly increasing interest in diagnostic systems of all kinds. On one side, this phenomenon is a result of versatility of today's application domains where the diagnostic system becomes very often the main component preventing some more or less complex systems from failure. On the other side, in the age of ubiquitous embedded / computerized control systems there is a need of existence of a coherent diagnostic mechanism in order to guarantee its optimal and fault-free behaviour.

The main requirements which can be set for the modern diagnostic systems include:

- flexibility; this requirement gives the systems ability to be easily adjustable to environmental / application domain changes (context-awareness / application-awareness),
- ease of re-configuration; this requirement allows these systems to alter their own behaviour / sensitivity / precision via post-deployment replacement (in the run-time) of the decision making logic only without the need to update the diagnostic system component as whole,
- autonomicity; this would allow these systems to operate and making decisions without or with limited human intervention.

The above requirements as such are non-trivial and greatly affect the diagnostic system complexity. For example, in order to equip the diagnostic system with a degree of flexibility, a level of fuzziness is required from its logic so that the diagnostic system would behave similarly for a certain range of input parameters. But at the same time, in order to support this kind of feature, the diagnostic system has to implement a mechanism which would rely on fuzzy diagnostic rules. As a result, the diagnostic system complexity increases, which ultimately negatively affects the system reliability as a whole.

Even more potential problems may be caused by embedding the run-time re-configuration feature. Normally each (diagnostic) system operates based on its core decision-making logic. This logic is usually hard-coded and does not change in run-time. The only way to change its behaviour is actually to re-compile the whole component code and re-deploy such a component into the target system. This approach, however, suffers from many inconveniences, because re-deployment is usually time consuming and very often requiring switching off the whole system or at least transient disabling of its selected features. Much better and much less invasive approach would be based on replacement the decision making logic. This approach, however, requires usually sophisticated component architecture and quite often it requires existence of some supporting services (the decision-making logic has to be somehow delivered to the right diagnostic component, some versioning or compatibility issues have to be also resolved). But the main advantage of the second approach is that the diagnostic component itself does not have to be re-deployed saving in this way many difficulties.

But the most desired feature of each sophisticated diagnostic system is the ability to operate with no or at least with limited human intervention during the decision making process. This feature is especially important in case of very complex systems where the amount of incoming diagnostic information may be beyond human's comprehension. In this situation any diagnostic system which is (at least to a level) autonomic may be of great help if not in making final decisions, then at least in filtering the most important information and simplifying in this way the decision making process. The key element of such an autonomic diagnostic system is a way of transforming (usually) expert knowledge about a diagnosed system into certain behaviours. Typically, the knowledge can be represented as a set of fuzzy rules or can be acquired during learning process (in case of Artificial Neural Networks - ANNs) or evolution process (genetic algorithms – GA). Both technologies support autonomic behaviour of systems.

However, if the diagnostic system is located / spread on / over an embedded platform, the additional requirement accompanying its flexibility, re-configurability and autonomicity should be compactness as embedded systems are typically the systems of constrained resources. In this situation fuzzy logic, ANNs or GAs may appear to be too heavyweight and some other technologies where an expert knowledge can be incorporated as to process the diagnostic information is sought. One of the technologies which support flexibility, re-configurability and autonomicity is policy-based computing. In case of this technology, an especially designed (diagnostic / control / decision-making) policy is used. This policy may (and mostly does) incorporate expert knowledge and describes usually in a programmatic form the way of using it in order to make decisions regarding further strategy (control, diagnostic, etc.).

2. Related work

There are numerous works where the diagnostic systems are of a hybrid structure where the core diagnostic / reasoning / decision-making logic is implemented with support of the artificial intelligence methods (these include mainly fuzzy logic / ANNs, but also GA). As a result, these systems become flexible and autonomic / adaptable.

In [22] the design and implementation of a fault diagnostic system is presented. The diagnostic system uses the heuristic knowledge base for a fuzzy logic-based technical diagnostic problem solving strategy. The dynamic fuzzy reasoning system uses expert knowledge in a form of a fuzzy model reflecting the mapping symptoms-faults. This diagnostic method was verified experimentally during simulations. Another application of fuzzy systems can be found in [21] where a method called fuzzy possibilistic diagnostic is presented. This method is based on certain associations between *disorders* and *symptoms* (called alternatively *manifestations* based on expert knowledge. Uncertainty of knowledge is incorporated by adopting fuzzy logic methodology. A set of use cases and examples presents how this method can be implemented in practice.

In [19] an adaptive approach to designing diagnostic systems is proposed. For this purpose a special neuro-fuzzy network was constructed and trained. As such it allows solving fuzzy logic equations and designing and adjusting fuzzy relations using either, expert and / or experimental information. The proposed system architecture was verified using formal apparatus as well as by some computational experiments. Similarly, in [20]. In this system the diagnosis method is based on inverse logical reference by using consequences to restore causes (based on some fuzzy relational matrices). The diagnostic decisions are obtained as a result of solution of fuzzy logical equations. The genetic algorithm using typical operations (crossover, mutation and selection) is core of the logical equations solving system. To verify the proposed solution, a computer-based simulations were carried out. The same problem is addressed in [19].

Applications of fuzzy logic techniques to diagnostic of industrial processes can also be found [10]. The same technique was used in the industrial application to a sugar factory evaporator (more to be found in [9]).

Artificial Neural Networks were used [6] as the core methodology implemented for diagnostic purposes of a technical system. For the diagnostics purpose functional and diagnostic models were developed. This model constituted the basis for initial diagnostic information provided by the sets of information concerning the elements of the basic modules and their output signals. The specific task set for Artificial Neural Networks was to recognise the states of reparable technical objects. A proof of concept installation was built using a car engine (as the example of the technical system) and a specialized diagnostic software.

An application of expert systems and Artificial Neural Networks to solve a problem of fault diagnosis in the aircraft fuel system is described in [13]. For the diagnosis purposes a multilayer neural network model for fault detection was built. The fault diagnostic results are presented accordingly to the fault deterministic rules reflecting an expert knowledge linking typical symptoms with possible causes. Another application of ANNs is presented in [15]. In this solution a diagnostic system with Artificial Neural Networks trained to detect, isolate and assess faults in some of the components of a single spool gas turbine is used. The diagnostic system is of hierarchical structure where a number of decentralized and specifically trained networks is used to handle specialised diagnostic tasks. The information from the monitored system is represented as measurement patterns. Depending on whether the pattern contains a fault detected, an appropriate action is taken.

Another application of Artificial Neural Networks in the task of process monitoring in fault detection and isolation systems was described in [8]. This solution is based on a design bank of neural networks with dynamic neurons that model a system diagnosed at normal and faulty operating points. An approach to the diagnosis of non-linear dynamic systems was also presented in [14]. In this paper a modified genetic programming approach for model structure selection combined with a classical technique for parameter estimation is presented. A possibility of employing such a model in a fault scheme was indicated.

In diagnostic systems, state reconstruction techniques are used for system failure detection. State observers are systems which perform the task of task reconstruction for diagnostic / monitoring / control purposes. They typically use a system model in order to reconstruct (usually inaccessible) state variables. These state variables in case of control systems enable implementation of optimal state-based control algorithms (for example LQR / LQG control) and in case of diagnostic systems they enable tracking of system state variables in order to take an appropriate action in case of any deviation detected. This kind of approach together with application examples was described in [11] and [12].

3. AGILE policy definition language

As the solution which can be an alternative for typical approaches exploring Fuzzy Logic, Artificial Neural Networks and Genetic Algorithms for the purpose of widely understood diagnosis, in this paper the AGILE Policy Definition Language is used for defining various fault detection / monitoring policies. This language specification was widely described in numerous works [1, 2, 3]. However, for this paper purposes the most important elements of AGILE have to be recalled, so that it was entirely clear how the AGILE PDL may fit the typical diagnosis-related problems. This language is based on XML structure and defines the following policy objects:

- *PolicySuite* contains all other policy objects. *Policy* defines decision making policy (one source policy file may contain zero or more policies). Policies can load *Templates* or execute *Actions*.
- *Template* is used to configure policies (for example, to assign some values to *InternalVariabies*).
- *Action* gathers policy decision-makind logic (it evaluates *Rules*, *TRCs*, *UFs*, it can also set local variables or yield *Policies*) and finaly returns a policy decision.
- *Rule* is used to compare variables (*ExternalVariables* or *InternalVariables*), values, etc. Depending on the comparison result an appropriate *Action* is executed.
- *ToleranceRangeCheck* (TRC) is an implementation of deadzone which is especially useful in case of tracking of a dynamic goal; the deadzone is specified as one of the TRC parameters. Depending on the value of the tracked parameter an appropriate action is executed.
- *UtilityFunction* (UF) is used to indicate goal-attainment or want-satisfaction. Depending on the utility level an appropriate *Action* is taken.

4. Architecture of the policy-supervised diagnostic system

As mentioned before, there are numerous applications of diagnostic systems implementing artificial intelligence, evolutionary algorithms and formal methods. The novelty of the solution presented in this paper will be purely based on the features of AGILE policies. AGILE policies can be used for diagnosis task by using:

- structural features of AGILE policies,
- dynamic decision making feature of AGILE policies.

The structural features are meant by existence of policy objects which would be the most convenient for some specific diagnostic task. One of these objects are *ToleranceRangeChecks* described in Section 3. TRCs are ideal for tracing whether some selected diagnostic parameters are sufficiently close (with respect to *Tolerance*) to the reference value. Depending on relation between them a specific action may be taken.

The dynamic decision making features of AGILE policies are their most powerful feature. Usually the policy logic is defined by a system who can incorporate decision making rules based on his experience and expertise. Based on the logic policies can make an appropriate decisions in the given situation.

For the diagnostic purposes a policy-supervised diagnostic / fault detection component was designed. This component comprises of key elements which make it structure very flexible whilst remaining very robust. The system architecture is shown in Fig. 1.

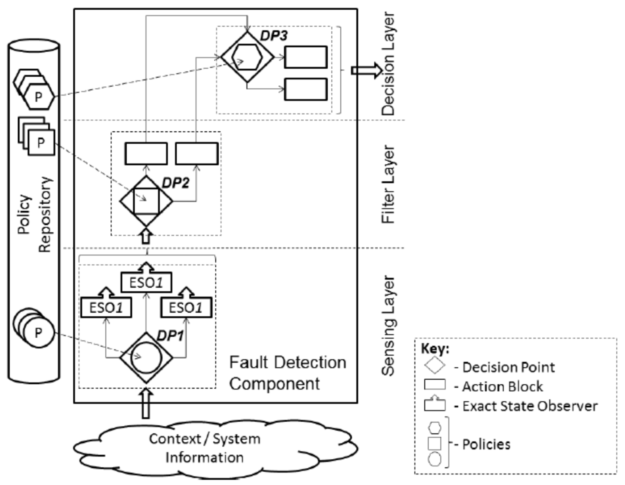

Fig. 1. Fault detection component architecture
Rys. 1. Architektura komponentu dla systemu detekcji błędów

4.1. Sensing Layer

The *Sensing Layer* is responsible for delivery of information about the system. This information can be delivered directly via set of sensors or it can be delivered from state observers, which will use a fractional information about a complex system provided by sensors and will estimate the other, missing parameters. This task will be performed by state observers.

However, usually the observers parameters are chosen so that they should provide the best possible state estimate. For example, in case of Kalman State Estimators (KSE) [7] the filter parameters are selected with respect to the assumed / known disturbances characteristics. In case of Exact State Observers (ESOs) [5] the observers parameters (filtration functionals) are selected for the assumed worst disturbances from unit ball (the norm of disturbances is assumed to be equal 1). In both cases though the optimal filtration / estimation characteristic is possible only if the real disturbances characteristics match those assumed. Otherwise, the state estimate may contain an estimation error.

One of the solutions which can be applied in the situation where an observer is to be used in the system of unpredictable or wide spectrum of disturbances characteristics is to use a set of observers and design a system which would select the observer of the parameters guaranteeing the best state estimation quality in the given circumstances. This solution (partially described in [16]) was chosen for the Fault Detection Component where a set of three ESOs was designed with different observation windows (2s, 4s and 6s). Well known feature of these observers is that the wider the observation window is the better state estimation quality is and the less sensitive the observer becomes on the disturbances. The negative side is that widening the observation window results in lengthening the state reconstruction process. In Fig. 2 one can see in details how the solution works.

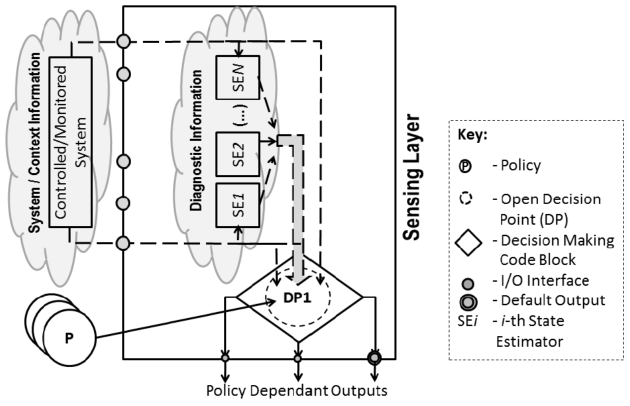
System / Context Information
Controlled/Monitored System
Diagnostic Information
SE*N*
(...)
SE2
SE1
DP1
P
Sensing Layer
Policy Dependant Outputs
Key:
- Policy
- Open Decision Point (DP)
- Decision Making Code Block
- I/O Interface
- Default Output
SE*i* - *i*-th State Estimator

Fig. 2. Sensing Layer
Rys. 2. Warstwa wykrywania

The policy selects the most appropriate observer in order to deliver the most exact information about the monitored system. The information provided by the state observers together with the information from sensors constitute the whole diagnostic information used for making decisions regarding whether there is any failure at all and if yes, then how to handle this situation at best.

4.2. Filter layer

The existence of the *Filter Layer* is a consequence of tendency to keep policies as simple as possible. Obviously, there is no physical limitation regarding what is the maximum level of policy complexity, but the fact is that the more complex the policy is the longer it takes to evaluate it and thus make a decision. Following this principle, in case of the proposed layered hierarchical structure / architecture of the Fault Detection Component it was decided that the direct analysis of the diagnostic information (signals) will be structurally decoupled from the higher level logic analysing the system state as whole. Thus, the policies used within the Filter Layer can be of relatively simple logic processing some selected diagnostic information in order to detect some deviations from expected / normal situation. If any failure will be detected, the Filter Layer will provide the *Diagnostic Layer* with ready information about where some problems were detected.

4.3. Decision Layer

Thanks to decoupling the Filter Layer from the *Decision Layer*, the Decision Layer would contain very sophisticated and specialised policies dealing with higher level handling of the monitoring information. These policies should contains a very specialised descriptions regarding what to do when certain system parameters will indicate a failure of a part of the system. The logic cannot be oversensitive in order to avoid false alarms and on the other side it shouldn't miss any failure in order react respectively and in this way limit potential damage done to the whole system.

The Decision Layer in the proposed solution is very lightweight in comparison to the other artificial intelligence methods. On the other side, if it is needed, the variety of AGILE policies blocks enables the policy designers to optimise policies so that they guaranteed a balance between policy complexity and its efficiency. Besides, integration of policies with the Open Decision

Point architecture (more to be found in [23]) enables the fault detection system easy and fast reconfiguration via policies updates. The updates can be done in run-time so that the Fault Detection Component would not be re-deployed to the target monitored system.

5. Middleware approach

The fault detection component architecture shown in Fig. 1. can also have a middleware implementation. In this case the sensing, filter and diagnostic functions would be spread across different components, that would logically and functionally constitute the three layers. In order to reflect this logical and functional division of the components roles, they would need to be appropriately tied via provider-subscriber dependencies, where providers would provide their policies outcomes as context information and the subscribers would subscribe only to the appropriate context items. The idea of fault detection middleware is shown in Fig. 3.

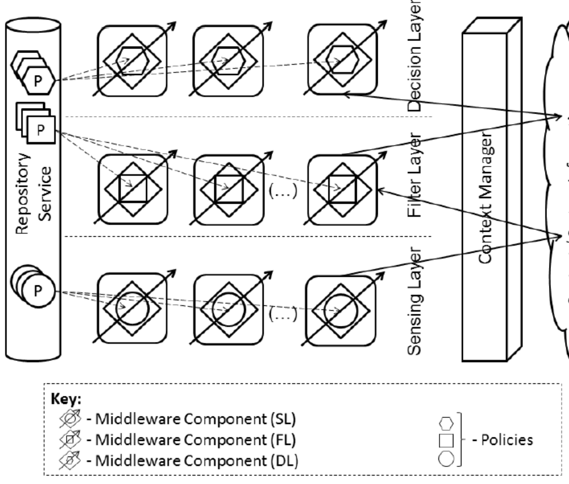
Repository Service
P
P
P
(...)
(...)
Decision Layer
Filter Layer
Sensing Layer
Context Manager
Context / System Information
Key:
- Middleware Component (SL)
- Middleware Component (FL)
- Middleware Component (DL)
- Policies

Fig. 3. Fault detection middleware architecture
Rys. 3. Architektura oprogramowania pośredniczącego systemu detekcji błędów

As shown in Fig. 3., the components from Sensing Layer (SL) provide context for the components from Filter Layer (FL) and these components from FL provide context for the components from Decision Layer (DL). In this way the logical and functional separation between functionalities of these components is kept. All the provide-subscribe dependencies are handled by specialised service called Context Manager whereas the Repository Manager Service (RM) is responsible for policies management (an example implementation of these services was described in [17]).

In Fig. 3 one can also see the advantage of the middleware-based approach over the single component one which is its higher robustness. This is due to the fact that this encapsulation of one functionality per component protects the diagnostic system as whole from the failure in case of one of the components failure (if a single component breaks, it may be re-initialised, whereas the rest of the diagnostic system will be still operational). The component-based approach integrates all the functionalities within one component which means that the whole Fault Detection Component would need to be re-initialised in case of any of its elements broke. However, the component-based approach is much less resource hungry and can be applied in the constrained systems.

6. Simplified model of a distillation column

The architecture of Fault Detection Component described in Section 4 was then applied to the task of monitoring the distillation column parameters. The distillation column installation is a physical installation existing at the AGH University of Science and Technology. A mathematical model of the column (see Fig. 4) will be used to prove concept simulations.

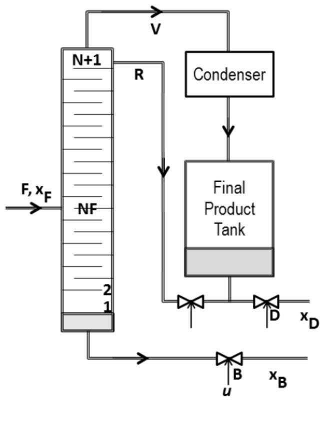

Fig. 4. Technological scheme of the distillation installation
Rys. 4. Schemat technologiczny kolumny destylacyjnej

A binary mixture F of composition x_F is delivered to a distillation column of N shelves. The control system task is to guarantee the requested composition of the distillation process remainder x_B via calculation of optimal control signals u based on the distillate x_D condensation. The product of the distillation process is the alcohol condensate which is stored in a tank [5].

To the distillation column control system a diagnostic / monitoring system was deployed. One of its parts is the Exact State Observer (ESO). However, as the ESOs theory is not a core part of this paper, the Reader is referred to [4, 5] for more information.

For the state estimation purposes a model of the distillation column is required. This is obtained using a linearization process, resulting in the following state-space model [5].

$$\dot{x}(t) = Ax(t) + Bu(t) + D\begin{bmatrix} x_F \\ F \end{bmatrix} \tag{1}$$

where A, B, C are system matrices of an LTI system. Here D is an additional system-specific matrix that provides fine-tuning for the specific physical configuration. The form of these matrices for the distillation column are shown below and originate from [5].

$$A = \begin{bmatrix} a_1 & h_1 & 0 & \dots & 0 \\ l_2 & a_2 & h_2 & \dots & 0 \\ 0 & l_3 & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & h_{N-1} \\ 0 & \dots & 0 & l_N & a_N \end{bmatrix} \tag{2}$$

To define matrix A the following notation is introduced for the upper diagonal ($\overline{H}$), diagonal ($\overline{A}$) and lower diagonal ($\overline{L}$).

$$\overline{A} = \begin{bmatrix} a_1 \\ a_2 \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ a_N \end{bmatrix} = \begin{bmatrix} -0.1514 \\ -1.6421 \\ -1.5210 \\ -1.4236 \\ -1.3543 \\ -1.3088 \\ -0.8540 \\ -0.7961 \\ -0.7432 \\ -0.7007 \\ -0.1403 \end{bmatrix}, \overline{H} = \begin{bmatrix} h_1 \\ h_2 \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ h_{N-1} \end{bmatrix} = \begin{bmatrix} 0.0985 \\ 0.7884 \\ 0.7884 \\ 0.7884 \\ 0.7884 \\ 0.3884 \\ 0.3884 \\ 0.3884 \\ 0.3884 \\ 0.3884 \end{bmatrix}, , \overline{L} = \begin{bmatrix} l_1 \\ l_2 \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ l_{N-1} \end{bmatrix} = \begin{bmatrix} 0.9838 \\ 0.8537 \\ 0.7326 \\ 0.6352 \\ 0.5659 \\ 0.5204 \\ 0.4656 \\ 0.4077 \\ 0.3548 \\ 0.0781 \end{bmatrix}$$

The remaining system matrices are defined as follows:

$$\overline{B} == \begin{bmatrix} -0.00048 & 0.00034 \\ -0.00048 & 0.00319 \\ -0.00048 & 0.00320 \\ -0.00389 & 0.00277 \\ -0.00296 & 0.00210 \\ -0.00204 & 0.00294 \\ -0.00258 & 0.00373 \\ -0.00289 & 0.00418 \\ -0.00283 & 0.00409 \\ -0.00243 & 0.00351 \\ 0.00000 & 0.00000 \end{bmatrix}, D = \begin{bmatrix} 0.00000 & 0.00034 \\ 0.00000 & 0.00319 \\ 0.00000 & 0.00320 \\ 0.00000 & 0.00277 \\ 0.00000 & 0.00210 \\ 0.00000 & 0.00000 \\ 0.00000 & 0.00000 \\ 0.00000 & 0.00000 \\ 0.00000 & 0.00000 \\ 0.00000 & 0.00000 \\ 0.00000 & 0.00000 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$K = \begin{bmatrix} -143.5 & 5.723 \\ -11.38 & 0.00744 \\ -7.302 & -0.193 \\ -4.719 & -0.210 \\ -3.0399 & -0.0099 \\ -1.942 & 0.4583 \\ -1.287 & 1.355 \\ -1.654 & 2.35 \\ -3.181 & 3.647 \\ -6.558 & 5.501 \\ -49.8075 & 3.1879 \end{bmatrix}.$$

If one denotes by N the number of shelves in the distillation column, the state $x(t)$ is given by

$$x(t) = [x_1, \quad ..., \quad x_{N+1}]; x_1 = x_B, x_{N+1} = x_D.$$

The control signal is of structure

$$u(t) = [V \quad D]$$

where V is a flow of vapour heating the column and V is flow of reflux. The control signal is generated by the LQR of form of the matrix K shown above. The system output is

$$y(t) = [x_B, \quad x_D].$$

7. Simulations

In this section some proof of concept simulations will be carried on. The main purpose of these simulation is to show applicability of policy-based computing to the task of a system diagnostic.

7.1. Simulation scenario

For the simulations purposes the Fault Detection Component was practically implemented in *C/C++* programming language. AGILE_Lite library (for more details see [17]) was used to process policies. In order to show how the three layers of Fault Detection Component work together, the following simulation scenario was designed:

- within the Sensing Layer there was designed a policy monitoring current level of disturbances (noise) in the system. The policy decision was to select the most appropriate Exact State Observer out of three pre-designed (one with observation window 2s, second with the observation window 4s and the last with the observation window 6s) in order to guarantee the best possible sensing quality. Utility Functions are used for the purpose of observer selection.
- within the Filter Layer a policy was designed in order to trace one of the system state variables (state variable x_1) and in case of the state variable value *Check*) was out of the reference value (*Compare* - set to be equal 0.1) with respect to the safety margin (*Tolerance* set to be 0.1), the policy decision was to report an error. Otherwise the policy was taking no action.
- within the Decision Layer there was a policy designed as to signal the error occurrence to the operator in case of any filter reported such a situation. This policy could be of much more complex structure, however, the main reason here was to show the interactions between the three layers of Fault Detection Component rather than prove how sophisticated AGILE policies can be.

7.2. Simulation results

In each simulation step the three policies described in Subsection 7.1. were evaluated sequentially. In Fig.5. there is presented a simulation trace recorded at the time instant t=0.04s, showing evaluation of the observers selection policy.

```
noise norm=[0.001029] : simulation time=[0.040000]
~~~~~ DP:<"dp1"> evaluation ~~~~~
~~~~~ DP:<"dp1"> context range check ~~~~~
~~~~~ DP:<"dp1"> policy evaluation ~~~~~
          -->Evaluate Policy: "Policy1"<--
          -->Execute Action: "Start"<--
          -->Evaluate UtilityFunction: "UF1"<--
          -->Execute Action: "ChooseWindow2"<--
          -->Evaluate ReturnValue: "Window2"<--
DP:<dp1> return value -->1
```

Fig. 5. Simulation trace for the Observer Selection Policy
Rys. 5. Przebieg symulacji dla polityki wybierającej obserwator

As one can see in Fig. 5, this policy chooses the Exact State Observer with the observation window T=4s. Once the best observer was selected for the current noise norm (norm value was 0.001029 whereas the reference norm was 0.01), as the next the state variable x_1 tracking policy was evaluated.

In Fig. 5 one can see two succeeding evaluations of the policy. Between the time instants t=0.16s and t=0.20 the value of x_1 state variable entered the allowed margin and the policy returned *OK* instead of *ERROR*. This means that the fault was no longer existing and the policy notified the Decision Layer about this fact.

```
x1=[0.002796] <ref=-0.1(+/-)0.1> :
simulation time=[0.160000]
~~~~~ DP:<"dp2"> evaluation ~~~~~
~~~~~ DP:<"dp2"> context range check ~~~~~
~~~~~ DP:<"dp2"> policy evaluation ~~~~~
          -->Evaluate Policy: "Policy1"<--
          -->Execute Action: "Start"<--
          -->Evaluate TRC: "CheckStateVariable"<--
          -->Execute Action: "ReturnERROR"<--
          -->Evaluate ReturnValue: "ERROR"<--
DP:<dp2> return value -->1

x1=[-0.026878] <ref=-0.1(+/-)0.1> :
simulation time=[0.200000]
~~~~~ DP:<"dp2"> evaluation ~~~~~
~~~~~ DP:<"dp2"> context range check ~~~~~
~~~~~ DP:<"dp2"> policy evaluation ~~~~~
          -->Evaluate Policy: "Policy1"<--
          -->Execute Action: "Start"<--
          -->Evaluate TRC: "CheckStateVariable"<--
          -->Execute Action: "ReturnOK"<--
          -->Evaluate ReturnValue: "OK"<--
DP:<dp2> return value -->0
```

Fig. 6. Simulation trace for the State Variable Tracking Policy
Rys. 6. Przebieg symulacji dla polityki śledzenia zmiennej stanu

In the same time the policy evaluated in the Decision Layer returned the results shown in Fig. 7.

```
Status=[1] : simulation time=[0.160000]
~~~~~ DP:<"dp3"> evaluation ~~~~~
~~~~~ DP:<"dp3"> context range check ~~~~~
~~~~~ DP:<"dp3"> policy evaluation ~~~~~
            -->Evaluate Policy: "Policy1"<--
            -->Execute Action: "Start"<--
            -->Evaluate Rule: "CheckIfStatusOK"<--
            -->Evaluate Rule: "CheckIfStatusERROR"<--
            -->Execute Action: "ReturnSignal"<--
            -->Evaluate ReturnValue: "Signal"<--
DP:<dp3> return value -->1

Status=[0] : simulation time=[0.200000]
~~~~~ DP:<"dp3"> evaluation ~~~~~
~~~~~ DP:<"dp3"> context range check ~~~~~
~~~~~ DP:<"dp3"> policy evaluation ~~~~~
            -->Evaluate Policy: "Policy1"<--
            -->Execute Action: "Start"<--
            -->Evaluate Rule: "CheckIfStatusOK"<--
            -->Execute Action: "ReturnDoNothing"<--
            -->Evaluate ReturnValue: "DoNothing"<--
DP:<dp3> return value -->0
```

Fig. 7. Simulation trace for the Decision Policy
Rys. 7. Przebieg symulacji dla polityki decyzyjnej

As one can see, the *Status* environment variable changes value from 0 (this corresponds to the *ERROR* value returned by the State Variable Tracking Policy) to 1 (and this corresponds to the *OK* value returned by the State Variable Tracking Policy). As a result the decision is changed from signalling the fault to the operator to indicating that there is no failure in the system.

8. Conclusions

In this paper a new approach to the problem of diagnostic of systems is presented. The use of Open Decision Point architecture together with the policy-based computing is described. Some of the solutions shown in this paper were part of other works carried out by the author in cooperation with the Autonomic Research Group at the University of Greenwich. However, for this paper purposes a completely new architecture of software component for diagnostic systems was designed.

The proposed architecture for the Fault Detection Component was proven to satisfy all the requirements set for the ideal diagnostic systems: it is flexible and context-aware, is easily re-configurable (via policy replacement) and enables the components to operate with no or with limited human intervention (autonomicity). Besides, as the AGILE PDL is XML-kind language, it actually allows the policy designer to design policies of any complexity and usability. It is worth stressing that thought the example AGILE policies were used here in the application to the task of fault detection, in general AGILE policies are not application specific and can be used for any purpose.

An alternative middleware-based approach with using specialised components performing respectively, sensing, filtering and diagnostic tasks is also presented. This solution may be applied to more complex, multi-agent diagnostic systems. This approach is discussed against the component-based approach and some pros and cons of both solutions are given.

9. References

[1] Anthony R.J. (2006): A policy-definition language and prototype implementation library for policy-based autonomic systems, Proceedings of 3rd International Conference on Autonomic Computing (ICAC2006) pp. 265–276.

[2] Anthony R.J., Pelc M., Ward P. and Hawthorne J. (2008): Flexible and robust run-time configuration for selfmanaging systems, SASO ’08: Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems pp. 491–492.

[3] Anthony R., Pelc M. and Byrski W. (2010): Context-aware reconfiguration of autonomic managers in real-time control applicaitons, Proc. of 7th IEEE Conference on Autonomic Computing pp. 73–74.

[4] Byrski W. (2003): The survey for the exact and optimal state observers in hilbert spaces, Proceedings of 7th European Control Conference pp. 6.

[5] Byrski W. and Fuksa S. (1984): Optimal finite parameter observer in an application to synthesis of stabilizing feedback for a linear system, Control and Cybernetics 13(1).

[6] Duer S. (2010): Diagnostic system with an artificial neural network in diagnostics of an analogue technical object, Neural Comput. Appl. 19: pp. 55–60.

[7] Kalman R.E. (1960): A new approach to linear filtering and prediction problems, Trans. ASME-Jnl. Basic Engineering 82: pp. 35–45.

[8] Korbicz J., Patan K. and Obuchowicz A. (1999): Dynamic neural networks for process modelling in fault detection and isolation systems, Int. J. Appl. Math. and Comp. Sci, .9: pp. 519–546.

[9] Koscielny J. M., Ostasz A. and Wasiewicz P. (2000): Fault detecition based fuzy neural networks - application to sugar factory evaporator, Proc. IFAC Symp. Fault Detection, Supervision and Safety for Technical Process, Vol. 1, IFAC, Budapest, Hungary, pp. 337–342.

[10] Koscielny J.M. and Syfert M. (2000): Current diagnostics of power boiler system with use of fuzzy logic, Proc. IFAC Symp. Fault Detection, Supervision and Safety for Technical Process, Vol. 2, IFAC, Budapest, Hungary, pp. 681– 686.

[11] Kowalczuk Z., Suchomski P. and Bialaszewski T. (1999a): Evolutionary multi-objective pareto optimisation of diagnostic state observers, Int. J. Appl. Math. Comp. Sci. 9: pp. 689–709.

[12] Kowalczuk Z., Suchomski P. and Bialaszewski T. (1999b): Genetic multi-objective pareto optimisation of state observers for fdi, Proceedings of European Control Conference ECC09, ECC ’09, ECC, Kalrsrhue, pp. 1–6.

[13] Long H. and Wang X. (2009): Application of aircraft fuel fault diagnostic expert system based on fuzzy neural network, Proceedings of the 2009 WASE International Conference on Information Engineering - Volume 02, IEEE Computer Society, Washington, DC, USA, pp. 202–205.

[14] Metenidis M. F., Witczak M. and Korbicz J. (2004): A novel genetic programming approach to nonlinear system modelling: application to the damadics benchmark problem, Engineering Applications of Artificial Intelligence 17: pp. 363–370.

[15] Ogaji S.O.T. and Singh R. (2002): Advanced engine diagnostics using artificial neural networks, Proceedings of the 2002 IEEE International Conference on Artificial Intelligence Systems (ICAIS’02), ICAIS ’02, IEEE Computer Society, Washington, DC, USA, pp. 236.

[16] Pelc M., Anthony R. and Byrski W. (2009): Policy supervised exact state reconstruction in real-time embedded control systems, Proceedings of 7th Workshop on Advanced Control and Diagnostics ACD2009.

[17] Pelc M., Anthony R. and Hawthorne J. (2009): Practical implementation of a middleware and software component architecture supporting reconfigurability of real-time embedded systems, Proceedings of 2009 International Conference on Computational Science and Engineering, ICSE ’09, ICSE, Vancouver, Canada, pp. 394–401.

[18] Rotshtein A.P. and Rakityanskaya A.B. (2001): Solution of a diagnostics problem on the basis of fuzzy relations and a genetic algorithm, Cybernetics and Sys. Anal. 37: 918– 925.

[19] Rotshtein A.P. and Rakytyanska H.B. (2009): Adaptive diagnostic system based on fuzzy relations, Cybernetics and Sys. Anal. 45: pp. 623–637.

[20] Rotshtein A. and Rakytyanska H. (2001): Genetic algorithm for fuzzy logical equations solving in diagnostic expert systems, Proceedings of the 14th International conference on Industrial and engineering applications of artificial intelligence and expert systems: engineering of intelligent systems, IEA/AIE ’01, Springer-Verlag, London, UK, UK, pp. 349–358.

[21] Sala A. (2008): Encoding fuzzy possibilistic diagnostics as a constrained optimization problem, Inf. Sci. 178: 4246–4263.

[22] Ulieru M. and Isermann R. (1993): Design of a fuzzy-logic based diagnostic model for technical processes, Fuzzy Set Syst. 58: pp. 249–271.

[23] Ward P., Pelc M., Hawthorne J. and Anthony R. J. (2008): Embedding dynamic behaviour into a self-configuring software system, Proceedings of 5th International Conference on Autonomic and Trusted Computing (Springer LNCS), pp. 373–387.

[24] Pelc M. and Anthony R.J. (2010): Policy-based selfmanagement in embedded systems, SystemsInternational Transactions on Systems Science and Applications (ITSSA), 6: pp. 43–59.

otrzymano / received: 03.07.2011
przyjęto do druku / accepted: 05.08.2011 *artykuł recenzowany*