system sterujący, blokada liniowa,
analiza bezpieczeństwa, FMEA, metody formalne

Tadeusz CICHOCKI*
Janusz GÓRSKI**

ANALIZA FMEA DLA SYSTEMÓW Z WYMAGANIAMI BEZPIECZEŃSTWA ZAWIERAJĄCYCH KOMPONENTY PROGRAMOWE

Jednym ze sposobów osiągania wysokiego poziomu zaufania do systemu jest analiza własności modelu tego systemu. Jeśli model został poddany adekwatnej walidacji, to własności modelu mogą być rozumiane jako własności rzeczywistego systemu. Rezultaty analityczne są szczególnie wiarygodne w przypadku, gdy model oraz związane z nim metody i techniki są formalne. W pracy pokazano jak modelowanie zorientowane obiektowo, rozszerzone o notacje formalne Z i CSP może być zastosowane w celu wzmocnienia techniki FMEA (*Failure Mode and Effect Analysis*). Prezentacji dokonano odnosząc się do studium przypadku związanego z oprogramowaniem systemu sygnalizacji kolejowej.

1. WSTĘP

W wyniku postępu technologicznego rosnąca liczba systemów zawiera komponenty programowe, które mogą mieć bezpośredni wpływ na ryzyko związane z ich użytkowaniem. W niektórych zastosowaniach, np. w transporcie kolejowym i lotniczym, medycynie lub atomistyce, niezbędny jest bardzo wysoki poziom zaufania, że systemy te nie spowodują zniszczeń w swoim środowisku. Kryteria i wytyczne dotyczące sposobu osiągnięcia tego zaufania są definiowane w licznych normach i regulacjach międzynarodowych, krajowych i branżowych, np. [7], [8], [9].

Jednym z podstawowych sposobów osiągania wysokiego poziomu zaufania do systemu jest konstruowanie jego adekwatnych modeli, a następnie analiza własności tych modeli. Jeśli model jest poprawną reprezentacją rozważanego problemu, to własności modelu mogą być rozumiane jako własności rzeczywistego systemu. Rezultaty analityczne są szczególnie wiarygodne w przypadku, gdy model oraz związane z nim metody i techniki są formalne.

W niniejszej pracy przedstawiono zastosowanie modelowania obiektowego rozszerzonego o notacje formalne Z [17] i CSP [15] do modelowania systemu sygnalizacji kolejowej. Modele te zostały użyte w celu wzmocnienia techniki FMEA (*Failure Mode and Effect Analysis*) [3], [4], [5] zastosowanej do analizy bezpieczeństwa systemu.

* Adtranz Zwus, 40-142 Katowice, e-mail: tadeusz.cichocki@pl.adtranz.com
** Politechnika Gdańska, 80-952 Gdańsk, e-mail: jango@pg.gda.pl

2. PROBLEMY Z ZASTOSOWANIEM FMEA DO OPROGRAMOWANIA

Podstawowe działania techniki FMEA obejmują identyfikację architektury systemu, modelowanie zależności funkcjonalnych pomiędzy zidentyfikowanymi komponentami, utworzenie kompletnej listy ich potencjalnych defektów oraz identyfikację scenariuszy propagacji błędów inicjowanych przez te defekty [13]. Zastosowanie tej techniki (szeroko stosowanej do konstrukcji mechanicznych i/lub elektrycznych) napotyka na trudności w odniesieniu do modułów zrealizowanych przy pomocy oprogramowania (bliższe omówienie można znaleźć w [3]). W celu przezwyciężenia tych trudności w badaniach opisanych w niniejszej pracy przyjęto następujące założenia:

- należy wybrać model, który w wspiera reprezentację oprogramowania w kontekście jego środowiska, obejmując sprzęt i ludzi;
- model ten powinien reprezentować system na kolejnych poziomach szczegółowości;
- powinna istnieć możliwość formalizacji wybranego modelu w celu wzmocnienia precyzji analiz i uniknięcia niejednoznaczności;
- wybrany model i metoda jego wytworzenia nie powinny przeczyć praktykom inżynierów systemowych i oprogramowania oraz intuicjom analityków bezpieczeństwa;
- metoda FMEA zastosowana do wybranego modelu powinna być oparta na podejściu tradycyjnym (wspierając dotychczasowy sposób pracy analityków bezpieczeństwa), a ponadto (jeśli to będzie konieczne) powinna pozwolić na analizę precyzyjną i matematycznie poprawną;
- ’część formalna’ proponowanego podejścia powinna, tak daleko jak to jest możliwe, być wsparta przez narzędzia.

3. POSZUKIWANIE ROZWIĄZAŃ

Do reprezentacji architektury systemu wybrano metodykę modelowania obiektowego. Zaletami podejścia obiektowego w kontekście analizy FMEA są:

- udostępnienie języka, w którym różne komponenty systemu (oprogramowanie, sprzęt, ludzie) mogą być wspólnie reprezentowane,
- nacisk na jednoznaczną dekompozycję systemu oraz niezależność komponentów systemu (precyzyjnie zapisane ‘kontrakty’ dla interfejsów obiektów),
- objęcie aspektów statycznych, dynamicznych i funkcjonalnych systemu (np. [10], [16]),
- elastyczność i stabilność: modele mogą być stosunkowo łatwo modyfikowane, a modyfikacje dotyczą zwykle ograniczonej liczby obiektów,
- możliwość „gładkiego” przejścia od analizy do projektu i kodu: obiekty zidentyfikowane podczas analizy mają zwykle swoją bezpośrednią reprezentację w implementacji.

Rozszerzenie modelu obiektowego o specyfikację formalną zrealizowano w celu zapewnienia precyzji i jednoznaczności. Postępując za [11], [12] przedstawiono strukturę i stany obiektów, operacje obiektowe oraz niezmienniki stanów systemu w języku schematów notacji Z [17], a aspekty dynamiczne współpracy obiektów, przez zastosowanie struktur notacji CSP [15]. Na bazie tych środków wyrazu modelowane są *defekty* (ang. *faults*), a następnie analizowane są skutki tych defektów. Przez *defekt* rozumiemy dowolne

odchylenie od pożądanych własności obiektów lub ich interakcji, w tym zakłócenie zależności czasowych i wystąpień zdarzeń. Defekty mogą się ujawniać jako *uszkodzenia* komponentów, reprezentowane jako negacja pożądanych rezultatów ich funkcji (warunku końcowego, ang. *post-condition*) lub negacja dowolnego innego założenia przyjętego podczas specyfikacji pożądanego zachowania komponentów (np. *warunku wstępnego* funkcji, ang. *pre-condition*, lub niezmiennika stanu komponentu). W specyfikacji CSP defekty są reprezentowane jako dodatkowe *zdarzenia defektów*, ang. *fault events,* a specyfikacja jest rozszerzana o przypadki obsługi wystąpień tych zdarzeń.

Podobnie jak w tradycyjnej FMEA, tworzone są *kompletne* listy uszkodzeń komponentów. Analizowane są *dwie dziedziny uszkodzeń* oraz, odpowiednio, konstruowane są dwie tabele spójne z obecnymi ogólnymi klasyfikacjami defektów oprogramowania [1], [14]:

- *Tabela danych* (ang. *Data Table*), obejmująca uszkodzenia komunikacji oraz odchylenia danych wejściowych, używana do analizy zależności danych oraz błędów interfejsów programowych, i
- *Tabela zdarzeń* (ang. *Event Table*), obejmująca uszkodzenia procesów oraz ograniczenia dotyczące stanów oprogramowania, używana do analizy efektów uszkodzeń i odchyleń wyjść, być może spowodowanych przez błędnie funkcjonujące oprogramowanie.

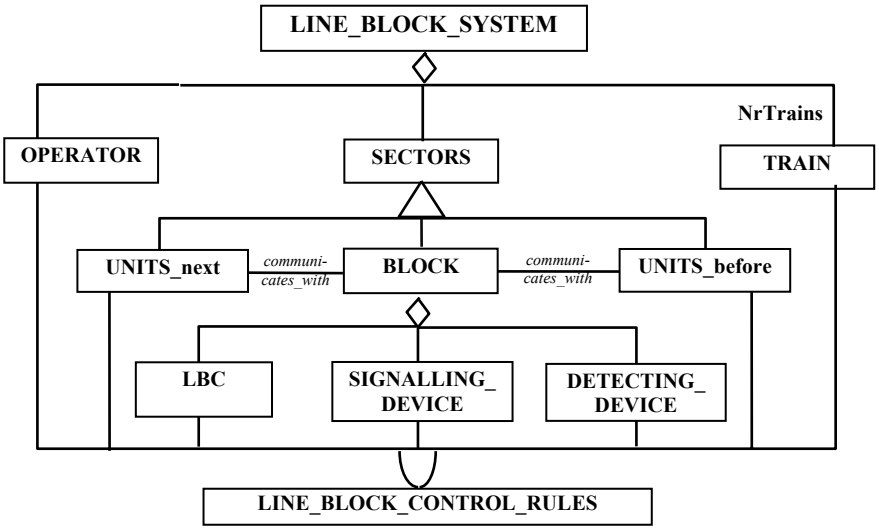
Ogólna procedura FMEA bazuje na specyfikacjach formalnych i obejmuje następujące działania:

- konstruowanie modelu obiektowego systemu,
- opis formalny struktury i atrybutów komponentów modelu,
- inwentaryzację możliwych uszkodzeń komponentów,
- argumentację o kompletności list uszkodzeń,
- powiązanie efektów przewidywanych defektów niższego poziomu abstrakcji z założeniami i listami uszkodzeń na wyższym poziomie.

W kolejnych częściach zilustrujemy kilka z tych kroków w kontekście przypadku systemu blokady liniowej.

4. SYSTEM BLOKADY LINIOWEJ – MODEL OBIEKTOWY

Zadaniem systemu blokady liniowej, ang. *Line Block System* (LBS), jest sterowanie ruchem kolejowym na linii kolejowej pomiędzy dwiema stacjami. System używa sygnałów świetlnych semaforów, z których każdy osłania jeden segment linii. Segmenty są w sposób ciągły testowane w celu stwierdzenia ich zajętości lub dostępności dla pociągu przemieszczającego się w kierunku następnego segmentu. Głównym celem jest zachowanie separacji pociągów na linii, przy zachowaniu pożądanej przepustowości w pożądanym kierunku. Model obiektowy systemu LBS jest pokazany na Rys.1.



Rys.1. Struktura obiektów systemu LINE_BLOCK_SYSTEM (LBS).
Fig.1. LINE_BLOCK_SYSTEM (LBS) object structure.

Na najwyższym poziomie abstrakcji system LBS składa się z *operatora* (*OPERATOR*), sektorów linii kolejowej (*SECTORS*) i pociągów (*TRAIN*). Sektory są następnie specjalizowane z wyróżnieniem dowolnego sektora (zwanego *BLOCK*) oraz, ze względu na wyróżniony kierunek ruchu, sektory przed (*UNITS_before*) i po (*UNITS_next*) sektorze *BLOCK*. Segment *BLOCK* jest dalej dekomponowany na urządzenie detekcji pociągu *DETECTING_DEVICE,* urządzenie sygnalizacji do pociągu *SIGNALLING_DEVICE* oraz sterownik blokady LBC (*Line_Block_Controller*). Związek *Line_Block_Control_Rules* opisuje zależności pomiędzy obiektami, odzwierciedlając podstawowe zasady zdefiniowane w regulacjach Polskich Kolei Państwowych (PKP) [18]. Na przykład, jedna z nich wymaga, aby:

Wymaganie Bezpieczeństwa	Każdy zajęty segment linii musi być osłonięty sygnałem 'STOP' wyświetlonym na semaforze osłaniającym (SIGNALLING_DEVICE).

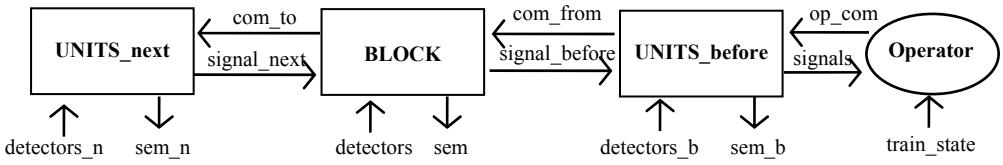
Zasady te, będące własnościami obiektów z Rys.1, są reprezentowane w notacji Z. Ze względu na ograniczenie miejsca przytaczamy tylko jeden przykład (pełna specyfikacja jest zawarta w [6]).

Wymaganie Bezpieczeństwa	(DETECTING_DEVICE.Detection=YES $\wedge$ LB_Traffic_direction = 1) $\Rightarrow$ SIGNALLING_DEVICE.Signalisation_1 $\in$ {S0, S1}

Komentarz. Atrybut *DETECTING_DEVICE.Detection* wskazuje, czy segment *BLOCK* jest zajęty przez pociąg, *LB_Traffic_direction = 1* oznacza, że kierunek ruchu na linii jest ustalony na '*z prawej strony na lewą*', a atrybut *SIGNALLING_DEVICE.Signalisation_1* ogranicza sygnały, które mogą być wyświetlone dla pociągów w takich sytuacjach.

5. SYSTEM BLOKADY LINIOWEJ – MODEL DYNAMICZNY

Każdy obiekt jest specyfikowany jako proces CSP. Gdy pociąg przemieszcza się przez kolejne sektory linii kolejowej, stan systemu zmienia się w odpowiedzi na sygnały z czujników. Rys.2 pokazuje kanały zdarzeń pomiędzy sektorami linii oraz kanały z i do czujników pociągów oraz semaforów.



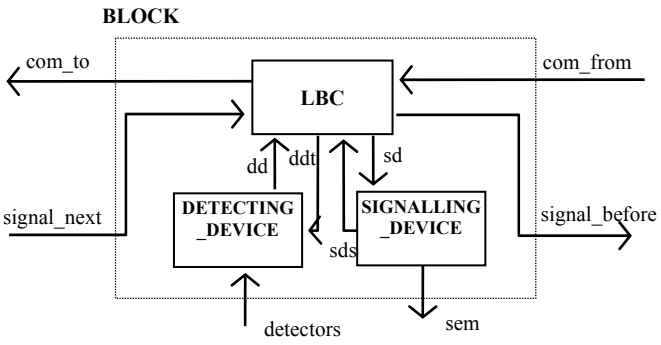
Rys. 2. Kanały komunikacyjne między obiektami.
Fig. 2. Communication channels between objects.

Formalną specyfikację zachowania obiektu *BLOCK* z Rys.2 przedstawiono poniżej. Tekst poprzedzony znakiem '--' jest komentarzem.

```
-- Specyfikacja procesu BLOCK
channel signal_before : {S0, S1, S2, S3, S4, S5}
channel signal_next : {S0, S1, S2, S3, S4, S5}
   -- Sygnały {S0, S1}, zwane sygnałami 'STOP', zabraniają
   -- wjazdu pociągu do rozważanego segmentu BLOCK
   -- ('S0' oznacza 'ciemny', a 'S1' oznacza 'czerwony' ),
   -- natomiast sygnały {S2, S3, S4, S5} zezwalają na wjazd.
channel sem : {S0, S1, S2, S3, S4, S5}
   -- sygnały na semaforze osłaniającym BLOCK.
channel com_from, com_to
   -- polecenia operatora (tutaj nie rozwijane).
channel detectors : {IN, OUT}
   -- sygnały z czujników pociągów

   -- Alfabet procesu BLOCK:
B = { |signal_before, signal_next, sem, detectors| }

BLOCK =
   detectors.IN →
       ( sem.S1 → signal_before.S1 → BLOCK
         [] sem.S0 → signal_before.S1 → BLOCK )
   [] detectors.OUT → BLOCK_not_occupied

BLOCK_not_occupied =
   signal_next.S0 → sem.S1 → signal_before.S1 → BLOCK
   [] signal_next.S1 → sem.S5 → signal_before.S5 → BLOCK
   [] signal_next.S2 → sem.S2 → signal_before.S2 → BLOCK
   [] signal_next.S3 → sem.S2 → signal_before.S2 → BLOCK
   [] signal_next.S4 → sem.S3 → signal_before.S3 → BLOCK
   [] signal_next.S5 → sem.S3 → signal_before.S3 → BLOCK
```

Jeśli czujniki wskazują na obecność pociągu w segmencie, to semafor osłaniający powinien wyświetlać sygnał S1 ('światło czerwone'), w przeciwnym przypadku rozważana jest

sytuacja w segmencie następnym (odczytana poprzez kanał *signal_next*). Sekwencja zdarzeń: *detectors.IN* → *sem.S1* reprezentuje poprawne (pożądane) zachowanie segmentu *BLOCK*. Odchylenia od tego zachowania modelujemy jako *widzialne zdarzenia* oferowane przez obiekt i występujące w wyniku inicjacji ze strony środowiska. Na przykład, specyfikacja obejmuje zdarzenie *sem.S0* (sygnał 'ciemny' zamiast sygnału 'czerwony'). Ten przypadek jest klasyfikowany przez regulacje kolejowe jako uszkodzenie niekrytyczne. Jak pokazano na Rys.1, *BLOCK* jest dekomponowany na obiekty: *LBC*, *SIGNALLING_DEVICE* i *DETECTING_DEVICE*. Na Rys.3 pokazano strukturę komunikacji związaną z tą dekompozycją.



Rys. 3. Struktura komunikacji komponentu BLOCK.
Fig. 3. Communication structure of the BLOCK component.

Poniżej przedstawiono formalną specyfikację komponentów obiektu *BLOCK*.

```
-- Specyfikacja procesu LBC
channel signal_before : {S0, S1, S2, S3, S4, S5}
channel signal_next : {S0, S1, S2, S3, S4, S5}
channel com_from, com_to
   -- przekazuje polecenia operatora (tutaj nie rozwijane).
channel dd : {IN, OUT}
   -- wyjście czujników
channel ddt : {IN}
   -- test stanu czujników DETECTING_DEVICE
channel sd : {S0, S1, S2, S3, S4, S5}
   -- ustalenie obrazu na semaforze
channel sds : {S0, S1, S2, S3, S4, S5}
   -- potwierdzenie stanu semafora

   -- Alfabet procesu LBC jest następujący:
L = {|signal_before, signal_next, dd, ddt, sd, sds|}

LBC = ddt.IN →
   ( dd.IN → sd.S1 → sds.S1 → signal_before.S1 → LBC
   [] dd.OUT → LBC_not_occupied )

LBC_not_occupied =
signal_next.S0 → sd.S0 → sds.S0 → signal_before.S1 → LBC
[] signal_next.S1 → sd.S5 → sds.S5 → signal_before.S5 → LBC
[] signal_next.S2 → sd.S2 → sds.S2 → signal_before.S2 → LBC
[] signal_next.S3 → sd.S2 → sds.S2 → signal_before.S2 → LBC
```

```
[] signal_next.S4 → sd.S3 → sds.S3 → signal_before.S3 → LBC
[] signal_next.S5 → sd.S3 → sds.S3 → signal_before.S3 → LBC
```

```
-- Specyfikacja procesu DETECTING_DEVICE
channel detectors : {IN, OUT}
channel dd : {IN, OUT}
channel ddt : {IN}

   -- Alfabet procesu DETECTING_DEVICE:
D = { |detectors, dd, ddt| }

DETECTING_DEVICE =
   detectors.IN → ddt.IN → dd.IN → DETECTING_DEVICE
   [] detectors.OUT  → ddt.IN → dd.OUT → DETECTING_DEVICE
```

```
-- Specyfikacja procesu SIGNALLING_DEVICE
channel sem : {S0, S1, S2, S3, S4, S5}
   -- sygnały prezentowane przez semafor osłaniający BLOCK
channel sd : {S0, S1, S2, S3, S4, S5}
channel sds : {S0, S1, S2, S3, S4, S5}

   -- Alfabet procesu SIGNALLING_DEVICE:
S = { |sem, sd, sds| }

SIGNALLING_DEVICE =
   sd.S0 → sem.S0 → sds.S0 → SIGNALLING_DEVICE
   [] sd.S1 →
       ( sem.S1 → sds.S1 → SIGNALLING_DEVICE
       []
       sem.S0 → sds.S1 → SIGNALLING_DEVICE )
       -- uszkodzenie synchronizacji na zdarzeniu sem.S1
       -- na podstawie wyboru dokonanego przez środowisko
   [] sd.S2 → sem.S2 → sds.S2 → SIGNALLING_DEVICE
   [] sd.S3 → sem.S3 → sds.S3 → SIGNALLING_DEVICE
   [] sd.S4 → sem.S4 → sds.S4 → SIGNALLING_DEVICE
   [] sd.S5 → sem.S5 → sds.S5 → SIGNALLING_DEVICE
```

Proces BLOCK, będący kompozycją procesów LBC, DETECTING_DEVICE i SIGNALLING_DEVICE, jest modelowany następująco:

```
BLOCK' = DETECTING_DEVICE
                  | [D∩L] | LBC
                        | [L∩S] | SIGNALLING_DEVICE
```

Pożądana relacja pomiędzy bardziej abstrakcyjną specyfikacją procesu *BLOCK*, a jego specyfikacją bardziej szczegółową, wyrażona w semantyce śladów procesów CSP [15] wygląda następująco:

$$\texttt{BLOCK} \sqsubseteq \texttt{BLOCK'} \setminus \{\texttt{dd}, \texttt{ddt}, \texttt{sd}, \texttt{sds}\}$$

co oznacza, że specyfikacja bardziej szczegółowa powinna być spójna z bardziej abstrakcyjną. Ten krok weryfikacji jest realizowany przy pomocy narzędzi komputerowych.

6. REALIZACJA FMEA

6.1. Analiza uszkodzeń danych

Systematyczny przegląd uszkodzeń komponentów jest wspomagany przez listy kontrolne *generycznych* defektów komponentów (mechanicznych, elektrycznych, elektronicznych). Z punktu widzenia modelowania danych, rozróżniamy:

- zmiany wartości atrybutów poza zakres ich typu,
- zmiany niezmienników obiektu.

Dla ilustracji pierwszego przypadku, rozważamy następującą specyfikację Z obiektu *BLOCK*.

BLOCK

Traffic_direction : {-1, 0, 1}	-- '1' odpowiada kierunkowi ruchu
Occupation : {YES, NO}	-- '*z prawej na lewą*', '-1' odpowiada kierunkowi
Signalisation_1 : {S0, S1, S2, S3, S4, S5}	-- przeciwnemu, a '0' oznacza '*nie zdefiniowano kierunku*
Signalisation_-1 : {S0, S1, S2, S3, S4, S5}	-- *ruchu*'

Traffic_direction = 1 $\Rightarrow$
((Occupation = YES $\Rightarrow$ Signalisation_1 $\in$ {S0, S1})
& (Occupation = NO $\Rightarrow$ Signalisation_1 $\in$ {S2, S3, S4, S5}))
& Signalisation_-1 $\in$ {S2, S3}

Analizując możliwe odchylenia wartości atrybutów odwołujemy się do interpretacji atrybutów w świecie rzeczywistym. Na przykład, dla atrybutu *Occupation,* oczekuje się wartości w zakresie {YES, NO}. Jednak doświadczenie pokazuje, że powinniśmy także rozważyć wartość Y/N odpowiadającą trudnościom w detekcji obecności pociągu. Podobnie wiedząc, że atrybut *Signalisation_1* reprezentuje możliwe 'legalne' kombinacje świateł semafora, na podstawie fizycznej realizacji możemy wnioskować, że także inne, 'nielegalne' kombinacje są możliwe, np. 'czerwony+zielony'. Tak więc również te sytuacje powinny być uwzględnione i objęte analizą.

Drugi przypadek dotyczy niezmienników stanów. Poniżej wskazujemy na warunek uszkodzenia wynikający z zaprzeczenia rozważanego wyżej *Wymagania Bezpieczeństwa* (niezmiennika stanu). Warunek ten mówi, że pociąg zajmuje dany segment linii (*DETECTING_DEVICE.Detection = YES*), ale jednocześnie stan semafora osłaniającego jest inny niż 'STOP', co oczywiście jest stanem zagrożenia.

Tabela 1. Warunek uszkodzenia niezmiennika stanu.
Table 1. State invariant failure condition.

Warunek	**Konsekwencja**
DETECTING_DEVICE.Detection = YES & LB_Traffic_direction = 1 & SIGNALLING_DEVICE.Signalisation_1 ∉ {S0, S1}	**Zagrożenie:** sygnały inne niż S0 lub S1 mogą być interpretowane jako sygnały *zezwalające*.

6.2. Analiza uszkodzeń komunikacji

Komunikacja pomiędzy dwoma obiektami (np. jeden obiekt wywołuje operację drugiego) jest wynikiem jednoczesnego wykonania procesów CSP reprezentujących te obiekty. Z punktu widzenia obiektu BLOCK, czujniki pociągów odczytują stany segmentów linii i wywołują właściwą operację obiektu *BLOCK* związaną z kanałem *detectors*. Ta operacja, nazwijmy ją *Monitor* (ponieważ monitoruje stan segmentu) zakłada, że sygnały otrzymane ze środowiska reprezentują w sposób spójny sytuację na linii kolejowej (*warunek wstępny* operacji). Zakłada także, że zdarzenia sygnalizowane przez kanał należą do zdefiniowanej wcześniej dziedziny (warunek *guard* operacji). W wyniku wykonania operacji na interfejsie obiektu *BLOCK,* obserwowany jest pewien widzialny efekt (specyfikowany jako *warunek końcowy* związany z tą operacją). Specyfikacja operacji *Monitor* jest przedstawiona poniżej. Operacja ta jest wywołana przez kanał *detectors,* a jej efekty są widzialne przez kanał *sem*.

```
pre Monitor =
      ('zajęty segment generuje zdarzenie detectors.IN a
      nie-zajęty, zdarzenie detectors.OUT')
-- Warunek ten nie jest tutaj formalizowany.
guard Monitor = ( detectors : {IN, OUT} )
post Monitor = ( ( detectors.IN ⇒ sem.[S0, S1] )
            & ( detectors.OUT  ⇒ sem.[S2, S3, S4, S5] ) )
```

Poprawnie zrealizowana komunikacja wymaga koniunkcji trzech warunków:

post *oper_wysyłająca* & **guard** *oper_odbierająca* & **pre** *oper_odbierająca.*

Lista uszkodzeń komponentów (wraz z ich skutkami) jest reprezentowana w następującej tabeli FMEA:

Tabela 2. Struktura tabeli FMEA.
Table 2. The structure of FMEA table.

	PROCES: *nazwa*		
Kod	**Kanał**	**Założony defekt**	**Odpowiadające mu odchylenie ...**
(1)	(2)	(3)	(4)
	Kanały Wejściowe	¬ **guard** *op* ∨ ¬ **pre** *op* (*wsparcie Tabelą danych*)	*Analiza dostępności i poprawności danych wejściowych operacji*
	Kanały Wyjściowe	¬ **post** *op* (*wsparcie Tabelą zdarzeń*)	*Analiza poprawności wykonania operacji oraz transformacji danych*

W przypadku operacji *Monitor* warunki wejściowe, które należy rozważyć obejmują:

```
¬ pre Monitor =   ('nie-zajęty segment                (przypadek 1a)
                  generuje zdarzenie detectors.IN lub
                  zajęty, zdarzenie detectors.OUT')   (przypadek 1b)
¬ guard Monitor = ¬ ( detectors : {IN, OUT} )
                  = detectors.I/O                     (przypadek 1c)
```

Wyniki analiz pokazano w Tabeli 3.

Mając na celu prewencję zagrożeń lub redukcję związanego z nimi ryzyka, projektanci mogą dokonać modyfikacji specyfikacji obiektu BLOCK. Przykład modyfikacji podajemy poniżej (rozszerzenia są wytłuszczone).

```
channel detectors : { IN, OUT, I/O }
BLOCK = detectors.IN →
            ( sem.S1 → signal_before.S1 → BLOCK
            []
            sem.S0 → signal_before.S1 → BLOCK )
        []
        detectors.OUT → BLOCK_not_occupied
        []
        detectors.I/O → sem.S1 → signal_before.S1 → BLOCK
```

Tabela 3. Tabela FMEA dla obiektu BLOCK (fragment).
Table 3. FMEA table for the BLOCK object (an extract).

	PROCES: **BLOCK**		
Kod	**Kanał**	**Założony defekt**	**Odpowiadające mu odchylenie …**
(1)	(2)	(3)	(4)
1a	detectors	detectors.IN – *błędna interpretacja stanu nie-zajętości segmentu*	sem.[S0, S1] – sygnały zabraniające 'STOP', zgodnie ze specyfikacją *poprawnego* zachowania - **utrata dostępności systemu.**
1b		detectors.OUT - *błędna interpretacja stanu zajętości segmentu*	sem.[S2, S3, S4, S5] – jeden z sygnałów *zezwalających* – **stan zagrożenia.**
1c		detectors.I/O – *niezidenty-fikowany stan segmentu* (*nowa oferta środowiska*)	**decyzja projektowa**: rozszerzyć specyfikację o wystąpienie i konsekwencje zdarzenia *detectors.I/O*.
2a	sem	*poprawne zdarzenie* detectors.IN *i błędne wyjście na **sem***	sem.[S2, S3, S4, S5] – jeden z sygnałów *zezwalających* – **stan zagrożenia.**

Rozwiązanie problemu akceptacji zdarzenia *detectors.I/O* w DETECTING_DEVICE oraz pokazanie jak LBC obsługuje zdarzenie *dd.I/O* wymaga również zmian w specyfikacjach tych komponentów obiektu BLOCK.

7. WNIOSKI

Artykuł pokazuje zastosowanie FMEA do systemu z związanego z bezpieczeństwem. Jednym z naszych celów było rozszerzenie FMEA w taki sposób, aby możliwe było zastosowanie tej techniki do systemów z oprogramowaniem bez wprowadzania istotnych zmian w ogólnych zasadach stosowania techniki FMEA. Istotą proponowanego rozszerzenia jest oparcie procesu FMEA o sformalizowane, zorientowane obiektowo specyfikacje. W ten sposób możliwe jest rozwiązanie niektórych problemów specyficznych dla systemów z oprogramowaniem, utrudniających bezpośrednie stosowanie FMEA. Innym celem proponowanej metody było jej zdefiniowanie w ramach procesów projektowych praktykowanych w firmie wytwarzającej systemy sygnalizacji kolejowej. Zadanie to jest wciąż kontynuowane i obecnie jest realizowane równolegle do tradycyjnie stosowanych w firmie analiz i dokumentowania. Dotychczasowe konkluzje dotyczące proponowanego podejścia są następujące:

- na pierwszych etapach projektu LBS osiągnięto znaczną koncentrację na istotnych zagadnieniach oraz uzyskano znaczne wsparcie w identyfikacji problemów i ich rozwiązań na drodze projektowania,
- notacja formalna zmusiła projektantów do koncentracji na strukturze i własnościach systemu, a jednocześnie umożliwiała zachowanie właściwego poziomu abstrakcji,
- specyfikacje formalne wspierały systematyczne poszukiwania potencjalnych odchyleń i niespójności oraz analizę ich skutków,
- proponowana notacja znacznie poprawia precyzję, kompletność i zwartość specyfikacji; pomimo tego, że stosowanie jej wymaga przeszkolenia i wprawy, może być stosowana w codziennej praktyce inżynierskiej.

Gdy rozmiar specyfikacji rośnie, niezbędne jest wsparcie narzędziowe. Obecnie rozważane są w tym zakresie różne możliwości, np. użycie oprogramowania analitycznego FDR firmy Formal Systems (Europe) Ltd. [15], [19].

LITERATURA

[1] Barbacci, B. R., Klein, M. H., Weinstock, C. B.: Principles for Evaluating the Quality Attributes of a Software Architecture, Software Engineering Institute, Carnegie Mellon University, Technical Report, CMU/SEI-96-TR-036, March 1997.

[2] Bowen, J., Stavridou, V.: Safety-Critical Systems, Formal Methods and Standards. *Oxford University Computing Laboratory Technical Report*, PRG-TR-5-92, 1992.

[3] Cichocki, T., Górski, J.: Safety assessment of computerized railway signalling equipment, Proc. of CENELEC SC9XA/WGA10 Workshop, Münich (Germany), May 11, 1999.

[4] Cichocki, T., Górski, J.: Safety assessment of computerized railway signalling equipment supported by formal techniques, Proc. of FMERail Workshop #5, Toulouse (France), September, 22-24, 1999.

[5] Cichocki, T., Górski J.: Failure Mode and Effect Analysis for Safety-Critical Systems with Software Components, Safecomp'2000, Rotterdam, The Netherlands, 25-27 October, 2000 (accepted).

[6] Cichocki, T., Górski, J.: Towards software Failure Mode and Effect Analysis, ETI Technical Report, Technical University of Gdansk (*to appear*).

[7] EN 50128: Railway applications. Software for Railway Control and Protection Systems, CENELEC, Final Draft version, July 1998.

[8] ENV 50129: Railway applications. Safety Related Electronic Systems for Signalling, CENELEC, May 1998.

[9] Defence Standard 00-55: Requirements for Safety Related Software in Defence Equipment (Part 1&2), Issue 1, UK Ministry of Defence, 1997.

[10] D'Souza, D., Wills, A. C.: Objects, Components, and Frameworks with UML. The Catalysis Approach, Addison Wesley Longman, Inc. 1998.
[11] Fischer, C.: Combining CSP and Z, Univ. of Oldenburg, Technical Report, TRCF-97-1.
[12] Heisel, M.: Methodology and Machine Support for the Application of Formal Techniques in Software Engineering, Habilitation Thesis, Technische Universität Berlin, Berlin, 1997.
[13] IEC 812 (1985): Procedure for failure mode and effects analysis (FMEA), TC56.
[14] Lutz, R. R., Woodhouse, R. M.: Requirements Analysis Using Forward and Backward Search, (Annals of Software Engineering, 1997) JPL California Institute of Technology Technical Report, May 2, 1997.
[15] Roscoe, A. W.: The Theory and Practice of Concurrency, Prentice Hall, 1998, ISBN 0-13-674409-5, pp. xv + 565.
[16] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: *Object-Oriented Modelling and Design*, Prentice-Hall Int., 1991.
[17] Spivey, J. M.: The Z Notation: A Reference Manual, First published by Prentice Hall International (UK) Ltd., 1992 (Second edition), ISBN 0-13-629312-3.
[18] WTB-E10: Wytyczne Techniczne Budowy Urządzeń Sterowania Ruchem Kolejowym w Przedsiębiorstwie PKP, Polskie Koleje Państwowe, Warszawa 1996.
[19] Formal Systems (Europe) Ltd.: *Failures-Divergence Refinement, FDR2 User Manual*, 24 October 1997.

FMEA ANALYSIS FOR SAFETY-CRITICAL SYSTEMS WITH SOFTWARE COMPONENTS

One of possible ways to achieve a very high level of confidence in a system is to develop its adequate model and then to analyse the properties of this model. The paper presents how object oriented modelling extended with formal specifications is used to support FMEA of software intensive systems. The case study the paper refers to is a computerised railway signalling system.