system sterujący, usterki,
analiza bezpieczeństwa, FMEA, metody formalne

Tadeusz CICHOCKI*
Janusz GÓRSKI**

FORMALNE WSPOMAGANIE MODELOWANIA USTEREK I ANALIZY ICH KONSEKWENCJI [1]

W artykule przedstawiono jak notacja CSP i związane z nią narzędzie FDR są używane w celu wspomagania metody FMEA (ang. *Failure Mode and Effect Analysis*) stosowanej do systemów z komponentami programowymi. Artykuł wyjaśnia podstawowe kroki podejścia (formalna specyfikacja, systematyczna identyfikacja usterek, eksperymenty indukowania usterek i dokumentowanie) oraz podaje niektóre wyniki związane z zastosowaniem tej metody do przemysłowego studium przypadku - systemu sygnalizacji kolejowej.

1. WSTĘP

Sukces projektu dotyczącego systemu z wymaganiami bezpieczeństwa zależy w znacznym stopniu od zastosowania adekwatnych mechanizmów przeciwdziałania usterkom oraz dowodu poprawności konstrukcji systemu względem wymagań i zasad normatywnych dotyczących bezpieczeństwa. Cel ten jest wspierany przez metodę FMEA (ang. *Failure Mode and Effect Analysis*). W metodzie zakłada się, że system składa się z komponentów, uszkodzenia których mogą wpływać na własności całego systemu. FMEA zakłada, że usterki komponentów są identyfikowane w sposób systematyczny, wpływ tych usterek na własności systemu jest poddany analizie, a wyniki tych analiz są brane pod uwagę w kolejnych decyzjach projektowych. Uzasadnia się [1, 4, 5], że dla systemów z komponentami programowymi, skuteczność FMEA można zwiększyć poprzez formalizację metody – w ten sposób zwiększając precyzję i usuwając nieścisłości z analiz.

W tym artykule przedstawiono zastosowanie CSP [6] jako formalnej metody wspierającej proces FMEA. Prezentacja odwołuje się do przemysłowego studium przypadku Systemu Blokady Liniowej (SBL). Artykuł pokazuje jak CSP oraz związane z tą notacją narzędzie FDR (ang. *Failures Divergence Refinement*) [3], produkt firmy Formal Systems (Europe) Ltd., użyto do identyfikacji usterek komponentów i do analizy skutków

[1] Praca realizowana w ramach projektu badawczego KBN nr 8 T11C 037 19.

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

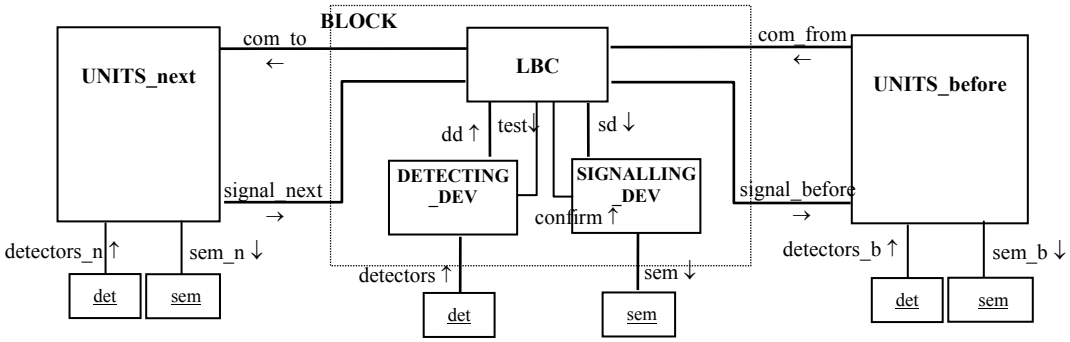
tych usterek w wyższych warstwach struktury komponentów systemu. Metoda składa się z następujących kroków:

- formalna specyfikacja systemu i jego komponentów,
- systematyczna identyfikacja usterek, na bazie specyfikacji formalnej,
- analiza skutków usterek,
- dokumentowanie: akceptacja usterki lub przeprojektowanie specyfikacji.

2. STUDIUM PRZYPADKU SYSTEMU BLOKADY LINIOWEJ

System Samoczynnej Blokady Liniowej (SBL) jest systemem sygnalizacji kolejowej na szlaku torowym pomiędzy dwiema stacjami, obecnie wytwarzanym w Adtranz Zwus w Katowicach. W [1] pokazano jak podejście obiektowe było zastosowane do modelowania tego systemu. Model ten reprezentuje architekturę systemu na kolejnych poziomach o wzrastającej szczegółowości. Rys.1 reprezentuje diagram współpracy [7] systemu SBL. Obiekt BLOCK nadzoruje pojedynczy sektor szlaku torowego. Komunikuje się on z czujnikami pojazdów torowych (det) i semaforami (sem) oraz z innymi sektorami przed i po nim, ze względu na ustalony kierunek jazdy po szlaku (reprezentowanymi przez obiekty UNIT_next i UNIT_before).

Na Rys.1 pokazano również wewnętrzną strukturę obiektu BLOCK (zawartego w prostokącie o wykropkowanych bokach). Jest to poziom większego uszczegółowienia modelu gdzie obiekty wyższego poziomu są reprezentowane poprzez ich komponenty. BLOCK jest reprezentowany jako kompozycja obiektów LBC, DETECTING_DEV i SIGNALLING_DEV. Kanały wejściowe i wyjściowe obiektu BLOCK są kanałami wejściowymi i wyjściowymi jego komponenów. Ponadto, komponenty mogą wymieniać sygnały, które nie są widziane spoza obiektu BLOCK (są sygnałami wewnętrznymi dla BLOCK). *Test*, *confirm*, *dd* i *sd* są przykładami tych sygnałów.



Rys. 1. Diagram współpracy Systemu Blokady Liniowej.

Fig. 1. Line Block System Collaboration diagram.

Zaletą modelu systemu zorientowanego obiektowo jest to, że odzwierciedla on rzeczywistą strukturę systemu i intuicje projektantów systemu. Hierarchiczne modelowanie wspiera rozróżnienie pomiędzy projektem a implementacją systemu. Na podstawie tego schematu możliwa jest analiza wpływu jaki komponenty niższej warstwy mogą mieć na ich kompozycje, komponenty warstwy wyższej, co stanowi istotę podejścia FMEA.

3. FORMALNA SPECYFIKACJA I WERYFIKACJA

W celu osiągnięcia precyzji i jednoznaczności zastosowano specyfikacje formalne. Wybrano CSP [6], gdyż ta notacja umożliwia specyfikowanie obiektów i ich interakcji poprzez kanały komunikacyjne. Predykaty CSP opisują wzorce przyczynowości zdarzeń i sposobu synchronizacji współpracujących procesów. Synchronizacja jest osiągana na specyficznych zdarzeniach i może obejmować wymianę danych pomiędzy procesami.

Proces CSP jest obserwowany poprzez ślady (skończone ciągi) związanych z nim zdarzeń. Dla procesu P, *alpha*(P) oznacza zbiór wszystkich zdarzeń związanych z P, a *traces*(P) jest zbiorem wszystkich (skończonych) śladów P. *Uszkodzenie* (ang. *failure*) procesu to (skończony) ślad wraz ze *zbiorem odmowy* (ang. *refusal set*), przez który rozumiemy zbiór zdarzeń w które, po wykonaniu danego śladu, proces na pewno się nie zaangażuje (zauważmy, że jeśli zbiór ten jest równy *alpha*(P), to proces P zakleszcza się – nie zaakceptuje żadnego zdarzenia). Dla procesu P, *failures*(P) oznacza zbiór wszystkich uszkodzeń P. *Rozbieżności* (ang. *divergences*) procesu P, oznaczane *divergences*(P), to zbiór śladów, po których P może być rozbieżny, to jest realizować nieskończony ciąg zdarzeń wewnętrznych (niezauważalnych dla zewnętrznego obserwatora).

Rozważane są trzy modele zachowania procesu: *śladów* (ang. *traces*, T), *uszkodzeń* (ang. *failures*, F) i *uszkodzeń-rozbieżności* (ang. *failures-divergences*, FD). W modelu *śladów*, P jest charakteryzowany przez zbiór *traces*(P) i, z definicji, pomiędzy dwoma procesami P i Q zachodzi *relacja uszczegółowienia śladów* (ang. *traces refinement relation)*, oznaczana P [T= Q) wtedy i tylko wtedy gdy *alpha*(P) = *alpha*(Q) i *traces*(Q) $\subseteq$ *traces*(P).

W modelu *uszkodzeń*, P jest charakteryzowany przez zbiór *failures*(P) i pomiędzy dwoma procesami P i Q zachodzi *relacja uszczegółowienia uszkodzeń* (ang. *failures refinement relation*, oznaczana P [F= Q), wtedy i tylko wtedy gdy *alpha*(P) = *alpha*(Q) i *failures*(Q) $\subseteq$ *failures*(P).

W modelu *uszkodzeń-rozbieżności*, P jest charakteryzowany przez parę zbiorów *failures*(P) i *divergences*(P) i pomiędzy dwoma procesami P i Q zachodzi *relacja uszczegółowienia uszkodzeń-rozbieżności* (ang. *failures-divergences refinement relation)*, oznaczana P [FD= Q), wtedy i tylko wtedy gdy *alpha*(P) = *alpha*(Q), *failures*(Q) $\subseteq$ *failures*(P) i *divergences*(Q) $\subseteq$ *divergences*(P).

Zauważmy, że dla procesów bez rozbieżności, relacje [FD= i [F= są równoważne.

Do specyfikacji naszych procesów używamy dialektu CSP związanego z narzędziem analitycznym FDR (ang. *Failures Divergence Refinement*) [3]. FDR umożliwia sprawdzenie różnych własności specyfikacji procesów CSP, a w tym:

- istnienie zakleszczeń – proces nigdy nie dochodzi do stanu, w którym nie istnieje możliwość kontynuacji (wykonania następnego zdarzenia),
- istnienie rozbieżności – proces nigdy nie dochodzi do stanu, w którym możliwy jest nieskończony ciąg zdarzeń bez wystąpienia zdarzenia zewnętrznego,
- relacje semantyczne procesów – weryfikacja relacji uszczegółowienia śladów, uszczegółowienia uszkodzeń, uszczegółowienia uszkodzeń-rozbieżności pomiędzy procesami CSP.

Dla analizowanego systemu SBL wykonano (w odniesieniu do jego obiektowego modelu prezentowanego na Rys.1) specyfikacje formalne zawartych tam obiektów, traktując je jako procesy CSP (więcej szczegółów można znaleźć w [2]). Specyfikacja ta opisuje wszystkie możliwe ślady zdarzeń, które mogą być obserwowane na interfejsie obiektu BLOCK. Specyfikacja ta jest następnie przetwarzana przy pomocy narzędzia FDR, w celu sprawdzenia niektórych z jej własności. Przykładami walidowanych w ten sposób twierdzeń są:

```
assert BLOCK :[deadlock free]
assert BLOCK :[divergence free]
```

Pozytywna walidacja powyższych twierdzeń oznacza, że specyfikacja obiektu BLOCK jest wolna od zakleszczeń i rozbieżności. Na poziomie większego uszczegółowienia model wyjaśnia wewnętrzną strukturę obiektu BLOCK (jak pokazano wewnątrz wykropkowanego prostokąta na Rys.1). Wyróżnione komponenty obiektu BLOCK są również specyfikowane w notacji CSP. Następnie, mając ich formalne specyfikacje, deklarujemy ich kompozycję jako obiekt BLOCK_IMP – implementację obiektu BLOCK.

```
-- BLOCK_IMP
BLOCK_IMP = DETECTING_DEV
      [|{|dd,test|}|] (LBC [|{|sd,confirm|}|] SIGNALLING_DEV)
```

Następnie dokonywane jest porównanie specyfikacji obiektów BLOCK i BLOCK_IMP w celu sprawdzenia ich wzajemnej spójności. Może to być łatwo wykonane przy pomocy narzędzia FDR. Walidowane twierdzenia podajemy poniżej:

```
assert BLOCK [T= BLOCK_IMP \ {|dd, test, sd, confirm|}
assert BLOCK [FD= BLOCK_IMP \ {|dd, test, sd, confirm|}

assert BLOCK_IMP \ {|dd, test, sd, confirm|} [T= BLOCK
assert BLOCK_IMP \ {|dd, test, sd, confirm|} [FD= BLOCK
```

4. SYSTEMATYCZNA IDENTYFIKACJA USTEREK

Rozważmy specyfikację CSP obiektu A i jego otoczenia. Niech `channel s:stype` będzie zewnętrznym kanałem komunikacyjnym deklarowanym dla A. Rozważamy odchylenia od specyfikacji zdarzeń pojawiających się na interfejsach zewnętrznych obiektu, które uniemożliwiają synchronizację tego obiektu z jego otoczeniem, w tym modyfikacje zbioru możliwych zdarzeń obserwowanych (zewnętrznych) i modyfikacje typów kanałów. Każde postulowane odchylenie oceniamy ze względu na możliwość jego wystąpienia w rzeczywistym systemie. Odchylenia zwalidowane pozytywnie (to jest te, których wystąpienie uznano za wystarczająco prawdopodobne), zostają włączone do Tabeli Usterek obiektu A. Nazywamy je *usterkami syntaktycznymi*[2].

Drugą rozważaną grupę odchyleń stanowią te, które wpływają na wzorzec przyczynowości zachowania obiektu A i jego otoczenia. Rozważamy możliwe wystąpienia zdarzeń, które nie są spójne ze stanami wewnętrznymi obiektu (nie wynikają z nich, są z nimi sprzeczne), a jednocześnie prowadzą, według aktualnej specyfikacji zachowania, do synchronizacji A i jego otoczenia. Rozważany zakres odchyleń obejmuje (zgodne z typem kanału `s`) zdarzenia, których nieprawidłowość nie będzie mogła być ujawniona przez otoczenie synchronizujące się z obiektem A. Jak poprzednio rozważamy możliwą częstotliwość ich wystąpienia w rzeczywistym systemie. Te zwalidowane pozytywnie są włączane do Tabeli Usterek obiektu A. Nazywamy je *usterkami semantycznymi*[3].

Analiza obiektu BLOCK jest prowadzona następująco. Na podstawie ich specyfikacji komponentów identyfikowane są ich możliwe usterki syntaktyczne i semantyczne. Następnie usterki są poddane analizie walidacyjnej (oceniającej możliwość ich wystąpienia) i dokumentowane w odpowiedniej Tabeli Usterek. Przytoczone niżej tabele, stanowią przykładowe fragmenty Tabel Usterek dla komponentów obiektu BLOCK.

Tabela 1. Tabela Usterek obiektu DETECTING_DEV (fragment).

Table 1. Fault Table of DETECTING_DEV (extract).

Nazwa	**Opis usterki**	
	Normalna synchronizacja	**Odchylenie synchronizacji**
DV1	`detectors.IN -> dd.IN`	`detectors.IN -> dd.OUT`
DV2	`detectors.INO -> dd.INO`	`detectors.INO -> dd.OUT`
DV3	`detectors.INO -> dd.INO`	`detectors.INO -> dd.IN`
DV4	`detectors.OUT -> dd.OUT`	`detectors.OUT -> dd.IN`

Tabela 2. Tabela Usterek obiektu SIGNALLING_DEV (fragment).

Table 2. Fault Table of SIGNALLING_DEV (extract).

Nazwa	**Opis usterki**	
	Normalna przyczynowość	**Odchylenie przyczynowości**
SV1	`Sd.SB4 -> SIGNALLING_not_occupied(S4)`	`sd.SB4 -> SIGNALLING_not_occupied(S3)`
SV2	`Sd.SB5 -> SIGNALLING_not_occupied(S5)`	`sd.SB5 -> SIGNALLING_not_occupied(S4)`

2 Mówimy również o wpływie *poziomym* komponentu.
3 Mówimy również o wpływie *pionowym* komponentu.

Tabela 3. Tabela Usterek obiektu LBC (fragment).

Table 3. Fault Table of LBC (extract).

Nazwa	Opis usterki	
	Normalna synchronizacja lub przyczynowość	Odchylenie synchronizacji lub przyczynowości
LV1	`dd.IN`	`Rozszerzenie deklaracji typu kanału dd o zdarzenie INOX i symulacja zdarzenia dd.INOX`
LV2	`signal_next.S5 -> sd.SB3`	`signal_next.S5 -> sd.SB4`

5. EKSPERYMENTY WSTRZYKIWANIA USTEREK

Każda usterka udokumentowana w Tabeli Usterek jest przedmiotem eksperymentu, który nazywamy *eksperymentem wstrzykiwania usterek* (ang. *fault injection experiment*). Eksperyment taki składa się z dwóch kroków: (1) wstrzyknięcie usterki i (2) analiza skutków usterki. Krok wstrzykiwania usterki obejmuje wprowadzenie zmiany do specyfikacji komponentu. Dla usterek syntaktycznych zmiany mogą obejmować deklaracje typów kanałów i przeprojektowanie interfejsu komponentu. Krok analizy skutków usterki jest realizowany z zastosowanie narzędzia FDR. Celem tej analizy jest sprawdzenie czy (oraz jak) dana usterka może naruszyć specyfikację na wyższym poziomie struktury (specyfikację obiektu BLOCK w naszym przypadku). Listę weryfikowanych warunków dla obiektu BLOCK podano w Tabeli 4.

Tabela 4. Warunki podlegające weryfikacji.

Table 4. Verification conditions.

Nazwa	Warunek
R1	`BLOCK [T= BLOCK_IMP \ {|dd, test, sd, confirm|}`
R2	`BLOCK [FD= BLOCK_IMP \ {|dd, test, sd, confirm|}`
R3	`BLOCK_IMP \ {|dd, test, sd, confirm|} [T= BLOCK`
R4	`BLOCK_IMP \ {|dd, test, sd, confirm|} [FD= BLOCK`

Wyniki eksperymentów wstrzykiwania usterek udokumentowano w Tabelach 5, 6 i 7. Znak *krzyżyka i kropki* ×• (według konwencji FDR) oznacza, że sprawdzenie zostało zakończone i został znaleziony (co najmniej) jeden kontr-przykład dla badanego warunku. Kropka w tym znaku wskazuje, że kontrprzykład jest dostępny poprzez opcję *debug* narzędzia FDR.

Tabela 5. Wyniki sprawdzenia dla usterek obiektu DETECTING_DEV.

Table 5. The results for the DETECTING_DEV faults.

Specyfikacja odniesienia: **BLOCK** Komponent: **DETECTING_DEV**		
Nazwa usterki	**Rezultat sprawdzenia FDR**	
DV1	R1: ×• i R2: ×• BLOCK_IMP realizuje: _tau test.interlocking detectors.IN dd.OUT signal_next.S2	R3: ×• i R4: ×• BLOCK realizuje: _tau detectors.IN _tau sem.S1
DV2	R1: ×• i R2: ×• BLOCK_IMP realizuje: _tau test.interlocking detectors.INO dd.OUT signal_next.S2	R3: ×• i R4: ×• BLOCK realizuje: _tau detectors.INO _tau sem.S1
DV3	R1: ✓ and R2: ✓	R3: ✓ i R4: ✓
DV4	R1: ×• i R2: ×• BLOCK_IMP realizuje: _tau test.interlocking detectors.OUT dd.IN sd.SB1 _tau sem.S1	R3: ×• i R4: ×• BLOCK realizuje: _tau detectors.OUT signal_next.S2

Tabela 6. Wyniki sprawdzenia dla usterek obiektu SIGNALLING_DEV.

Table 6. The results for the SIGNALLING_DEV faults.

Specyfikacja odniesienia: **BLOCK** Komponent: **SIGNALLING_DEV**		
Nazwa usterki	**Rezultat sprawdzenia FDR**	
SV1	R1: ✓ i R2: ✓	R3: ✓ i R4: ✓
SV2	R1: ×• i R2: ×• BLOCK_IMP realizuje: _tau test.interlocking detectors.OUT dd.OUT signal_next.S0 sd.SB5 sem.S4	R3: ×• i R4: ×• BLOCK realizuje: _tau detectors.OUT signal_next.S0 sem.S5

Tabela 7. Wyniki sprawdzenia dla usterek obiektu LBC.

Table 7. The results for the LBC faults.

Specyfikacja odniesienia: **BLOCK** Komponent: **LBC**		
Nazwa usterki	**Rezultat sprawdzenia FDR**	
LV1	R1: ✓ i R2: ✓	R3: ✓ i R4: ✓

LV2	R1: ×• i R2: ×• BLOCK_IMP realizuje: _tau test.interlocking detectors.OUT dd.OUT signal_next.S5 sd.SB4 sem.S1	R3: ×• i R4: ×• BLOCK realizuje: _tau detectors.OUT signal_next.S5 sem.S3

Każdy eksperyment wstrzykiwania usterki, który nie został pozytywnie zweryfikowany przez narzędzie FDR jest następnie analizowany w celu identyfikacji natury ujawnionej niespójności. Pomagają w tym w decydujący sposób kontrprzykłady udostępniane przez narzędzie, które pokazują scenariusze zdarzeń prowadzące do uszkodzeń. Na przykład, dla usterki LV2, naruszona usterką implementacja BLOCK_IMP realizuje następujący ciąg zdarzeń zewnętrznych (jak pokazuje ślad dla warunków R1 i R2 w Tabeli 7): `detectors.OUT -> signal_next.S5 -> sem.S1`, podczas gdy dozwolony ciąg zdarzeń dla obiektu BLOCK jest (jak pokazuje ślad dla warunków R3 i R4 w Tabeli 7) następujący: `detectors.OUT -> signal_next.S5 -> sem.S3`. Analiza takich sytuacji prowadzi, w ogólnym przypadku, do następujących decyzji:

- Akceptacja: akceptujemy (negatywne) skutki usterki. Jednakże projektanci otrzymują wskazówkę dotyczącą potrzeby obniżenia możliwości wystąpienia usterki (np. poprzez wybór bardziej niezawodnej technologii),
- Przeprojektowanie: aktualna architektura systemu jest zmieniana w celu eliminacji negatywnych konsekwencji możliwego wystąpienia usterki.

Analizy wykonano przy pomocy narzędzia FDR ver. 2.66 realizowanego w systemie operacyjnym Red Hat LINUX 6.2 na procesorze INTEL Pentium III 600 MHz. Pełna specyfikacja obiektu BLOCK składała się z 35 linii kodu CSP/FDR. Specyfikację komponentów DETECTING_DEV, SIGNALLING_DEV i LBC stanowiło 60 linii kodu CSP/FDR. Na całą analizę poświęcono 12 godzin (13 minut na jedną usterkę). Całkowita liczba rozważonych usterek dla komponentów obiektu BLOCK jest podana w Tabeli 8.

Tabela 8. Liczby usterek rozważonych podczas analizy.

Table 8. The numbers of faults considered during analyses.

Nazwa komponentu	**Liczba usterek**
LBC	21
DETECTING_DEV	10
SIGNALLING_DEV	25

6. PODSUMOWANIE

Potencjalne możliwości metod formalnych w zakresie wspomagania analiz bezpieczeństwa są dobrze zrozumiane [4]. Jednakże ich zastosowanie w środowisku przemysłowym wciąż napotyka na trudności. Jedną z nich jest mało znana w środowisku

inżynierskim specyfika podejścia formalnego oraz złożoność formalnej analizy, jeśli metoda ta nie jest wspomagana przez dostatecznie mocne narzędzie. Pojawienie się dojrzałych narzędzi, którymi mogą posługiwać się inżynierowie, otwiera nowe możliwości w stosowaniu metod formalnych w analizie bezpieczeństwa. Wymaga to badań wskazujących na możliwe wzorce postępowania i poszukiwania sposobów wspierania przez metody formalne tradycyjnych technik i metod stosowanych przez inżynierów bezpieczeństwa.

W artykule pokazano studium przypadku zastosowania notacji CSP i związanego z nią narzędzia, do wspomagania techniki FMEA dla systemu z wymaganiami bezpieczeństwa. Analizy formalne koncentrują się tu głównie na badaniu relacji pomiędzy kolejnymi poziomami specyfikacji systemu. Dzięki temu złożoność analizy formalnej może być ograniczona nawet w przypadku, gdy projektowany system jest stosunkowo duży.

Modele proponowanego podejścia tworzą hierarchię zstępującą, zaczynając od wymagań względem systemu i zstępując do komponentów implementowanych w postaci sprzętowej lub programowej. Analiza jest również prowadzona zstępująco, dla kolejnych poziomów hierarchi modeli, w każdym przypadku w odniesieniu do poziomu bezpośrednio wyższego. Identyfikowane są możliwe usterki, a następnie analizowane ich skutki obserwowane na wyższym poziomie struktury systemu. Aby dokonać wstępnej selekcji usterek (i skoncentrować się tylko na istotnych dla danej aplikacji) trzeba ocenić elementy potencjalne usterki ze względu na możliwość ich wystąpienia w systemie rzeczywistym. W ocenie tej wykorzystywana jest wiedza zewnętrzna, np. profile uszkodzeń wykorzystywanych elementów, ocena technologii zastosowanej do implementacji komponentu, itd. Dokumentacja procesu analizy formalnej stowarzyszonej z FMEA jest istotnym elementem pakietu bezpieczeństwa, przygotowywanego według wymagań „kolejowej" normy europejskiej EN 50129. Jednoznaczność list usterek, wynikający z zastosowania narzędzia obiektywizm oceny ich skutków oraz przejrzystość związanych z tymi usterkami decyzji projektowych stanowią o wadze tej argumentacji.

LITERATURA

[1] T. CICHOCKI, J. GÓRSKI, Failure Mode and Effect Analysis for Safety-Critical Systems with Software Components, in: Springer Lecture Notes in Computer Science, vol. 1943, 2000, pp. 382-394

[2] T. CICHOCKI, J. GÓRSKI, Formal Support for Fault Modelling and Analysis, accepted for SAFECOMP'2001, Budapest, Hungary, 26-28 September, 2001.

[3] *Failures-Divergence Refinement, FDR2 User Manual*, Formal Systems (Europe) Ltd., 24 October 1997.

[4] N. G. LEVESON, Safeware: System Safety and Computers, Addison-Wesley Publishing Company (680 pp), 1995, ISBN 0-201-11972-2.

[5] J. D. REESE, Software Deviation Analysis, University of California, Irvine, PhD Thesis, 1996.

[6] A. W. ROSCOE, The Theory and Practice of Concurrency, Prentice-Hall, International Series in Computer Science (580 pp), 1998, ISBN 0-13-674409-5.

[7] *OMG Unified Modeling Language Specification*, Version 1.3, June 1999.

FORMAL SUPPORT FOR FAULT MODELLING AND ANALYSIS OF ITS CONSEQUENCES

The paper presents how CSP and the associated tool FDR are used to support FMEA of a software intensive system. The paper explains the basic steps of our approach (formal specification, systematic fault identification, fault injection experiments and follow-up) and gives some results related to the application of this method to the industrial case study, a railway signalling system which is presently under development.