Model-based Diagnosis of Embedded Systems

DWFTT 2007
September 13, 2007
Jurryt Pietersma
Contents

1. Fault diagnosis
2. Model-based Diagnosis (MBD)
3. Spectrum-based Fault Localization (SFL) (*brief intro*)
4. Tangible results and outlook

The work presented is based mostly on the ESI/ASML Tangram research project.
Personal Introduction

• PhD student, Computer Science Delft University of Technology (DUT)
• Aerospace Engineering MSc DUT
• Member of a research group dedicated to Model-Based Diagnosis
• Part of the ESI/ASML Tangram project
• Thesis subject:

“Modeling Systems for Efficient Quality-controlled Fault Diagnosis”
Fault Diagnosis

- Problems
- Description
- Terminology
- Methods
Fault Diagnosis Problems

it’s actually a lot more than $25^2 = 625$:

$$\sum_{i=1}^{25} \binom{25}{i} = 2^{25} - 1 = 33554431$$
Fault Diagnosis Problems

- Fault diagnosis of complex systems is difficult and computationally hard.

- Some examples of complex systems and related industries:
  - wafer scanners (ASML)
  - copiers (OCE)
  - advanced medical equipment (Philips)
  - consumer electronics (NXP)

- System dependability degrades due to:
  - loss of functionality
  - long diagnosis time, up to 60 % of down-time
  - catastrophic failures (no recovery)
What is Fault Diagnosis?

Definition of Fault Diagnosis:

- Identify and localize the faults that are the root cause of non nominal system behavior.

Note:
An important first step in fault diagnosis is to divide the system into components and to pinpoint to the faulty component.
What is Fault Diagnosis?

Fault Diagnosis is a well-known topic in many disciplines. Compare for example with medical diagnosis.
Terminology

- **fault**: Delivered service ≠ correct service (e.g. program crash)
- **error**: System state that may cause a failure (e.g. index out of bounds)
- **failure**: The cause of an error in the system (bug: array index un-initialized)

  - Faults do not automatically lead to errors
  - Errors do not automatically lead to failures
For our purposes, the distinction between errors and failures is less relevant: failures are errors that affect the user; i.e. that are externally observable.
# Example fault diagnoses

<table>
<thead>
<tr>
<th>Observation</th>
<th>Diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contradicting sensor readings</td>
<td>Broken sensor</td>
</tr>
<tr>
<td>Component delay timeout</td>
<td>Wires disconnected</td>
</tr>
<tr>
<td>Intermittent actuator activity</td>
<td>Degraded power supply lines</td>
</tr>
<tr>
<td>Segmentation fault</td>
<td>Bug in library x, function y</td>
</tr>
<tr>
<td>Deadlock</td>
<td>Communication fault in process p</td>
</tr>
</tbody>
</table>
Various ways of fault diagnosis

• **Manual**
  • Let your system engineers analyze the test results and deduce the root cause and decide for a repair action. This may be very time-consuming and occupy expensive and scarce resources.

• **Automated**
  • Symptom based using the results of a one time manual analysis. Does not evolve with system design, only covers anticipated faults.
  • Inference of possible explanations of failures through model-based diagnosis (MBD)
  • Localization of fault components. This technique is called spectrum-based fault localization (SFL)
2. Model-based Diagnosis

- Basics
- Diagnosis Models
- Example
- Diagnosis Algorithm
- Diagnostic quality
- Entropy and Uncertainty
Models

- Describe system behavior
  - Correct behavior (good weather)
  - Faulty behavior (bad weather)
- Model details:
  - Granularity
  - Strength

Stronger models capture more “bad weather” behavior
Model used for error detection
Models used for diagnosis

To the nominal functionality

We add “health” information...
Models used for diagnosis

\[ y = f(x, h) \]

\( h_i = 1 \) means \( f_i \) is healthy,
\( h_i = 0 \) means \( f_i \) is at fault

We would like to find:
\[ h = f^{-1}(x, y) \]
But in general \( f^{-1} \) cannot be determined.
In practice we compute consistent solutions for \( h \) with an efficient search algorithm. (analog to numeric solving)
Our model-based diagnostic process

**Process:**
1. map $f$ to propositional logic
2. observe $x$ and $y$
3. find all $h$ for which $y = f(x,h)$ is consistent
   (i.e., the diagnosis or “numeric solution” for by $h = f^{-1}(x,y)$)

$$y = f(x,h)$$
**Simple example**

for $x=1$, $y_1=1$, $y_2=0$

$i_1, i_2 \Rightarrow \text{ok}$

$i_1, i_3 \Rightarrow \text{not ok}$

conclusion $i_3$ is root cause?

using the behavior and structure of the model we can find more solutions, e.g., $i_1$ and $i_2$ fail
A simple example

Step 1: map f to propositional logic

Model of component $i$: $y_i = \neg x_i$

Logic proposition: $h_i \Rightarrow (y_i = \neg x_i)$

Normal form: $\neg h_i \vee (x_i \wedge \neg y_i) \vee (\neg x_i \wedge y_i)$

Reasoning in normal form:

For $z$: $\neg h_1 \vee (x_i \wedge \neg z) \vee (\neg x_i \wedge z)$

For $y_1$: $\neg h_2 \vee (z \wedge \neg y_1) \vee (\neg z \wedge y_1)$

For $y_2$: $\neg h_3 \vee (z \wedge \neg y_2) \vee (\neg z \wedge y_2)$
Step 2: Observe x and y

Symptom: $x = 1, (y_1, y_2) = (1, 0)$
(Expected was $(y_1, y_2) = (1,1)$)

Step 3: Infer diagnosis

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>$h_2$</th>
<th>$h_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Diagnosis Algorithm

Simply generating all combinations and checking them is not possible (increases with $2^N$).

Not necessary as only likely solutions are interesting.
Diagnosis Algorithm

Basic algorithm:

1. generate seed candidates in queue
   {} becomes { (h1=0), (h1=1) }
2. pop most likely candidate (based on a priori probability heuristic)
   (h1=1)
3. check if candidate is consistent with model and observations
   (h1=1) is consistent
4. if consistent add “sibling candidates” to queue
   siblings of (h1=1) are (h1=1,h2=0) and (h1=1,h2=1)
   queue becomes { (h1=0), (h1=1,h2=0) , (h1=1,h2=1) }
5. continue with 2 until the queue is empty or user interrupt
Diagnosis Algorithm

Algorithm can be improved by

- compiling an efficient knowledge representation (e.g., one that exploits system hierarchy)
- using conflicts for a more efficient search
Diagnostic quality

Diagnostic quality is determined by:

• the number of constraints the model imposes (model strength)

• the number of observations

and can be expressed by entropy.
Model strength

- Nominal behavior:
  \[ \text{inverter c healthy} \Rightarrow \text{c.out} = \text{neg(c.in)} \]

- Nominal behavior and failure modes:
  \[ \text{inverter c healthy} \Rightarrow \text{c.out} = \text{neg(c.in)} \]
  \[ \text{inverter c stuck at 0} \Rightarrow \text{c.out} = 0 \]
  \[ \text{inverter c stuck at 1} \Rightarrow \text{c.out} = 1 \]
  \[ \text{inverter c IO shorted} \Rightarrow \text{c.out} = \text{c.in} \]
Observation quality

• Spatial
  • Number of points in the model where we can measure system behavior
  • Naturally depends on model granularity
• Temporal
  • Number of measurements
Uncertainty

- a measure for information content (Shannon, 1948)
- used for next best measurement and test selection heuristic within MBD
- a measure of the uncertainty of $D$

$$H_D = - \sum_{D} P(d_k | (x, y)) \log_2 P(d_k | (x, y)) \ [\text{bits}]$$
Uncertainty

- e.g. $P(h_i=\text{True}) = 0.99$

| $z$ | $a$ | $(x, y_1, y_2)$ | $(1, 0, 1)$ | $P(d_{j_k} | a)$ | $H(d_{j_k} | a)$ |
|-----|-----|-----------------|------------|----------------|----------------|
| $D$ | 1   | $(1, 0, 1)$     | 0.970492   | 0.041937       |                 |
|     | 2   | $(0, 0, 1)$     | 0.009803   | 0.065411       |                 |
|     | 3   | $(1, 0, 0)$     | 0.009803   | 0.065411       |                 |
|     | 4   | $(0, 1, 0)$     | 0.009803   | 0.065411       |                 |
|     | 5   | $(0, 0, 0)$     | 0.000099   | 0.001317       |                 |

$H$  $0.239486$
$E[H | z]$ $0.136815$
Reduction of expected uncertainty

Three methods for reducing $E[H] \ (= 0.1368 \text{ bits})$:

- adding more variables to $z$ (spatial, $0.1361$ for $z$) 
  (quality aspect, some a better than others hence the heuristic)
- conjugation of multiple $z^{(c)}$ (temporal, $0.0763$ for $2$)
- adding more constraints to the model, e.g. explicitly defining fault modes ($0.0808$ strong model) 
  (makes a model less robust!)
Reduction of expected uncertainty

$C$ is number of conjunctions, temporal observability

$\sigma$ is fraction of observable variables, spatial observability

\[
E[H_2]\]

$C$

$\sigma$

$\{(x), (z), (y_1), (y_2)\}$, $1/4$

$\{(z, x), (z, y_1), (z, y_2)\}$, $1/2$

$\{(x, y_1), (x, y_2), (y_1, y_2)\}$, $1/2$

$\{(z, x, y_1), (z, x, y_2), (z, y_1, y_2)\}$, $3/4$

$\{(x, y_1, y_2)\}$, $3/4$

$\{(z, x, y_1, y_2)\}$, $1$
Model-based Diagnosis background

• Model-based Diagnosis (MBD) first proposed by Reiter 1987 and De Kleer 1987: General Diagnostic Engine (GDE)
• major performance improvements since then
• practical examples: NASA’s Deep Space 1 and Earth Observing 1, XEROX PARC, car industry
• active community: DX workshop
• http://fdir.org/lydia (our open source implementation, language, models, converters, diagnosis, and simulation engines)
Spectrum-based Fault Localization

• For MBD models are crucial

• What if models are not available? E.g., in the case of software.

• a brief intro...
Spectrum-based fault localization

Ingredients:

- First of all you need to know when the system is in a correct state and when it enters an error state: An Error Oracle is needed.

- Next you need to divide the system (software) in a number of small components

- Perform a number of (short) runs on the system:
  - Keep track which components of the system are touched
  - Keep track which runs produce errors and which runs are error-free runs
Spectrum-based fault localization

Error Oracle:
• System failures are clear indications that an error has occurred
• Examples of other error oracles / detection mechanisms,

• Application specific:
  • Expert knowledge (e.g., CPU load too high)
  • Precondition and postcondition checking
  • Assert statements added to the code
• Generic:
  • Array bounds checking
  • Deadlock detection
Spectrum-based fault localization

- **Measure** the activity of the various parts / components of the system at run-time

- **Compare** the activity measured in good runs with the activity when errors occur

- The parts whose activity resembles the occurrence of errors most are the most likely locations of the fault that causes these errors

- Measurements can be at any level: hardware / software components, modules, functions, blocks of code, statements
An Example

Consider this system:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Not touched**
- **Touched, good run**
- **Touched, bad run**
Spectrum-based and testing (1)

Test suite

- t1
- t2
- t3
- t4
- t5

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not touched</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Touched, good run</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Touched, bad run</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Spectrum-based and testing (2)

Test suite

<table>
<thead>
<tr>
<th>Status</th>
<th>t1</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td></td>
</tr>
<tr>
<td>t4</td>
<td></td>
</tr>
<tr>
<td>t5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not touched</th>
<th>Touched, good run</th>
<th>Touched, bad run</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>K</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

39
Spectrum-based and testing (3)

Test suite

<table>
<thead>
<tr>
<th>Status</th>
<th>t1</th>
<th>t2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Not touched
- Touched, good run
- Touched, bad run

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Spectrum-based and testing (4)

Test suite

Status

| t1    | √ |
| t2    | √ |
| t3    | × |

Not touched
Touched, good run
Touched, bad run

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Spectrum-based and testing (5)

Test suite

Status

| t1 | ✓ |
| t2 | ✓ |
| t3 | ✗ |
| t4 | ✓ |

Not touched

Touched, good run

Touched, bad run

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Spectrum-based and testing (6)

Status
- t1: ✓
- t2: ✓
- t3: ✗
- t4: ✓
- t5: ✗

Legend:
- Not touched
- Touched, good run
- Touched, bad run

A B C D E F G H J K L M
3 1 1 0 1 3 3 2 3 1 2 2
2 0 0 2 1 2 2 0 2 1 0 0
System components are ranked according to likelihood of causing the detected errors.

Status

- t1 ✓
- t2 ✓
- t3 ✗
- t4 ✓
- t5 ✗

First indications are by intuition. Can we motivate or understand our intuition?
Program spectra

• Execution profiles that indicate, or count which parts of a software system are used in a particular test case

• Many different forms exist; e.g.
  • Spectra of program locations
  • Spectra of branches / paths
  • Spectra of data dependencies
  • Spectra of method call sub-sequences
4. Tangible Results and Outlook
Summary: Model-based vs. Spectrum-based

**Model-based**
- Models used primarily for reasoning
- All generated explanations are valid
- Most likely diagnosis need not be actual cause
- Well suited for hardware

**Spectrum-based**
- Model used primarily for error detection (our error oracle – when are things going wrong)
- Ranking may lead to a wrong conclusion of the faulty component
- Well suited for software
Results From Research Projects

Model-based Diagnosis (MBD) (Tangram 3 years)
• modeling language and tooling is stable
  (available from http://fdir.org/lydia )
• technology transfer complete, initial results positive
• strong improvement of diagnosis algorithms
• entropy as quality quantifier (tools needed)
• me looking for a job

Spectrum-based Fault Localization (SFL) (Trader 2 years)
• tooling (compiler) is in place
• industry (NXP) is very much interested in “technology transfer”
Outlook

MBD:
• modeling and diagnosis of dynamic (time-dependent) systems
• better exploitation of model characteristics by the algorithm
• repair, reconfiguration, and system autonomy

SFL:
• transfer
• increasing accuracy through combination with models
  (hybrid approach between SFL and MBD)

Soon: a white paper on both methods,
(Embedded Systems 2007 Conferentie)
Discussion / Questions