Towards Microagent based DBIST/DBISR

Liviu Miclea*, Szilárd Enyedi*, Gavril Toderean**
Technical University of Cluj-Napoca, Romania
*Department of Automation
**Department of Telecommunications

Alfredo Benso, Paolo Prinetto
Politecnico di Torino, Italy
Dipartimento di Automatica e Informatica

Abstract

In this paper, we present some ideas and experiments on using microagents for testing and repairing a distributed system, whose elements may or may not have embedded BIST (Built In Self Test) and BISR (Built In Self Repair) facilities.

The microagents are software modules that perform monitoring, diagnosis and repair of the faults. They form together a society whose members communicate, set goals and solve tasks.

The platforms taken into consideration for mobile tester microagents include Java Micro Edition, BREW, Symbian, PalmOS, as well as more general small scale platforms. Experimental tester agents in Java 2 Micro Edition and PalmOS are also presented, a solution that ensures portability, flexibility, but also a relatively small memory footprint.

1. Introduction

One of the current trends in BIST technology is Distributed BIST, or DBIST [1-5]. The distributed nature of DBIST means that each of the modules of the tested system has its own BIST routine, which runs the test more or less independently from the other modules. This way, the actual BIST of the whole device is decomposed into smaller, dedicated BISTs, which should be simpler and easier to develop and maintain. If the communication is expensive, a decentralized test management can be more efficient. This testing solution is especially suitable for large systems, with many subsystems, possibly of different types. One such system is shown in figure 1.

Distributed BIST, or DBIST, usually implies that each module of the system has its own BIST, and the testing is not done centrally, but locally, in a distributed manner. The system may or may not have a central DBIST management module. Most DBIST approaches [1-5] use a central control authority to start/stop the remote BIST tests, to generally organize the DBIST process and gather together the results.

2. Agent based DBIST and DBISR

2.1 Generalities

This work is a natural continuation of the multi-agent approach presented in [6-8]. We extended the work to other low scale platforms as well, like BREW (see 2.4.2. below) or Symbian (see 2.4.3. below), for example.

The term "microagent" is preferred to "agent", in this context, due to the low processing power and memory requirements of the agents used.

The IEEE 1232 family of standards, analyzed in [9], describes common exchange formats and software services for reasoning systems used in system test and diagnosis. The goal is to make the data exchange between two different diagnostic reasoners easy. The standard also defines software interfaces, for the use of external tools that can access the diagnostic data in a consistent manner. It allows exchanging diagnostic information and embedding diagnostic reasoners in any test environment.

Intelligent agents are software modules able to make decisions on their own, communicate with each other, learn new things and even "travel" from system to system (see also [10]).

Most of the large systems we talk about are heterogeneous, comprising a large number of devices of different types. The devices we talk about usually have different hardware and/or software, tasks, dependability requirements, but all are capable of running software (in order to be able to run the agent code). If not, an agent from a nearby device can test this device.

A multi agent approach and diagnosis ontology for diagnosis of spatially distributed technical systems is presented in [11]; however, in that approach, each subsystem has its own agent monitoring and diagnosing it, which can be costly in some cases. The memory holding the agent could be used for system purposes.

In this paper, we propose an innovative solution based on microagent approach for diagnosing distributed systems. It offers many advantages like flexibility, easy mainte-

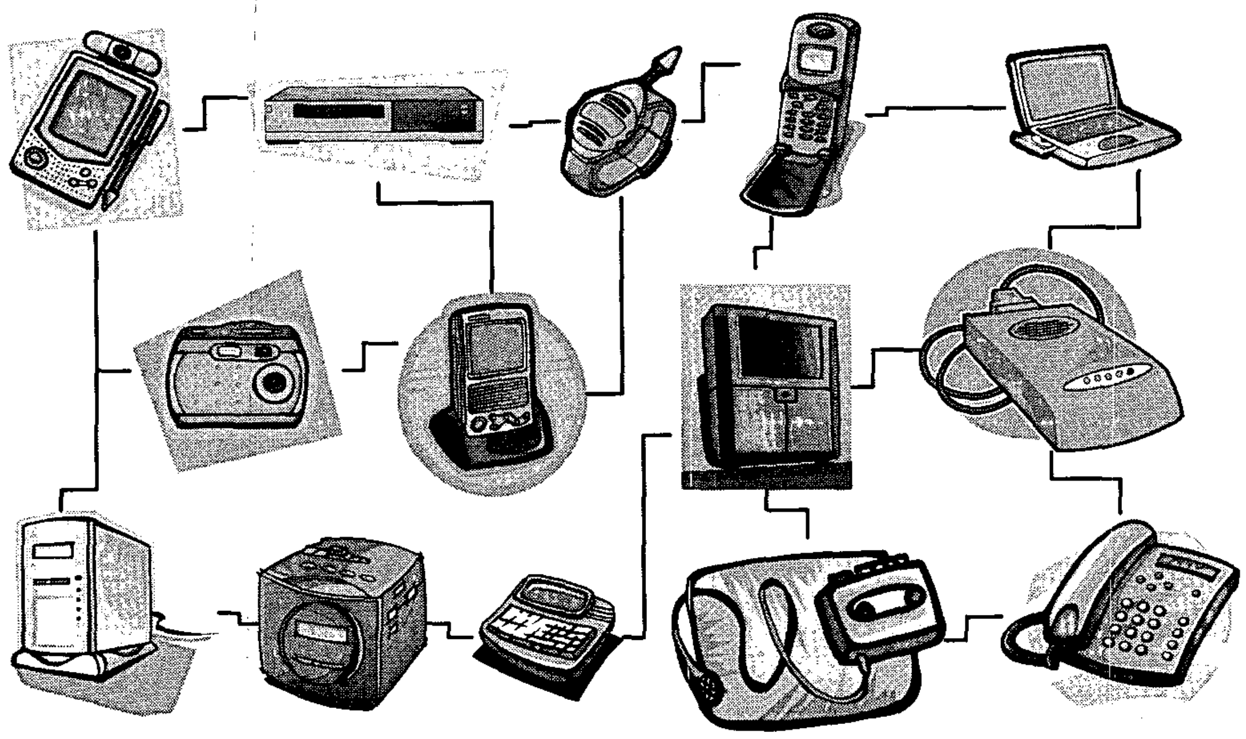


Figure 1 – Heterogeneous distributed system. The devices are in connection with each other. In this ex-ample, the components of the system are several computing enabled modules of different makes and versions. Of course, the more classical example is an industrial system with diverse components, but we chose consumer devices, for demonstrational purposes.

nance, diagnosis tool for parts of the overall system, and fault tolerance due to the Built in Self Repair. Monitoring and diagnosing faults is one of the application areas for agent-based systems. Some modern complex devices have also BIST ed components, so we can decompose the diagnosis of the whole system to the diagnosis of components. Our approach differs from other multi-agent approaches, because the agents are portable, highly platform independent, require relatively low resources, they can deal with many types of devices and the system administrator can use various, inexpensive and friendly tools to supervise the devices, tests, agents and the agent society in general.

An agent based approach has the advantage of distributing the processing among many distinct components and, due to the autonomy of the agents, reduces the communication in the system. Moreover, the mobility of the agents increases the ability to efficiently solve a problem that appears in a region of the system, by increasing the local knowledge.

2.2 Agent Society

The agent society, as exemplified in figure 2, is able to share resources and repair the faults whenever possible. One or more agents diagnose each subsystem.

The agents travel from device to device, try to detect and repair errors, either by themselves or with the help of other agents or a central database. They can also gather "experience" through their work.

When an agent cannot detect a cause of an observed fault or cannot repair it, it appeals to other agents to start cooperation. We use a decentralized diagnosis model, which reduces the complexity and communication overhead of centralized solutions. Due to the diversity of devices in modern complex systems, heterogeneous agents can be implemented that take care of device(s) in their responsibility area.

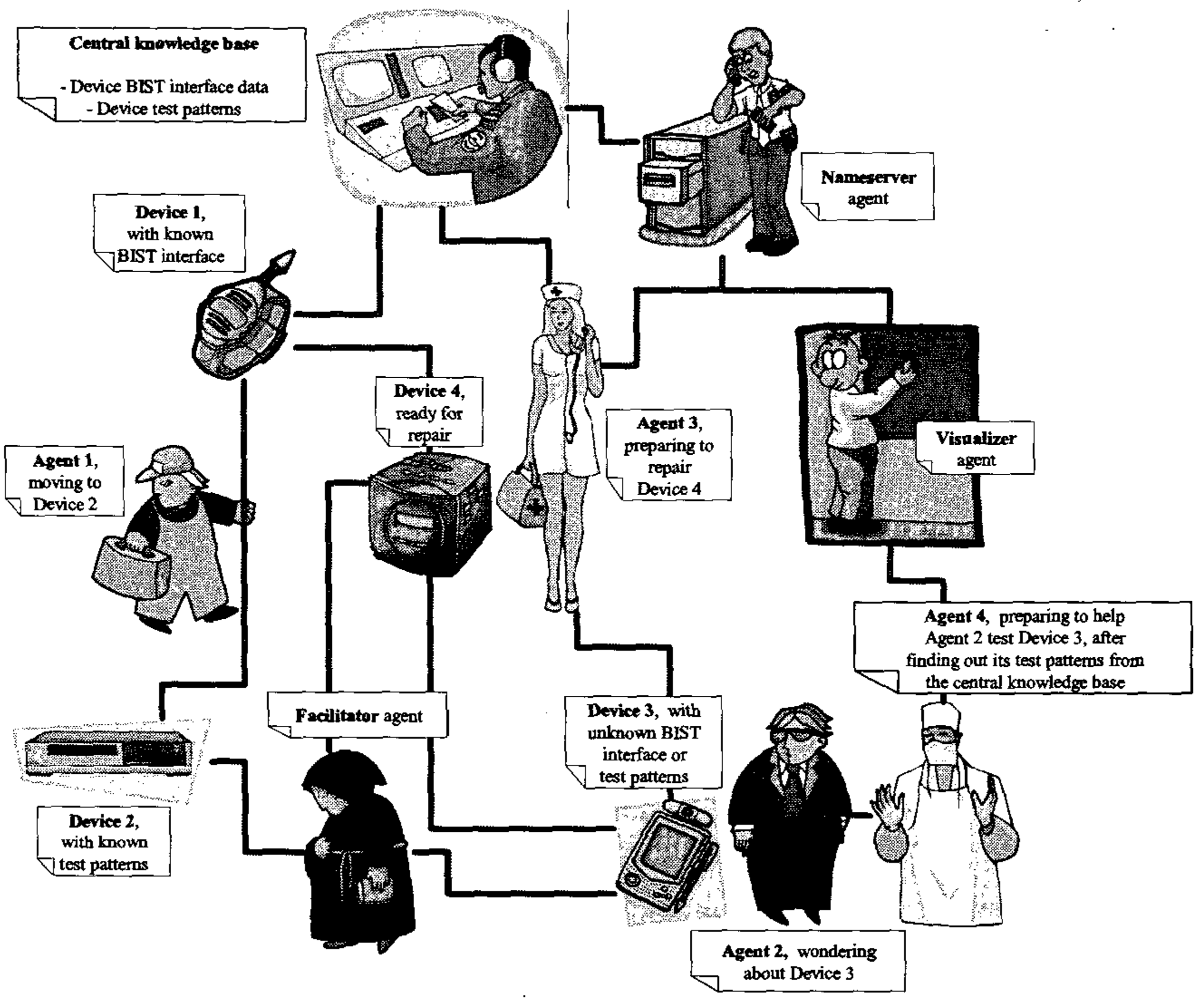

Central knowledge base
- Device BIST interface data
- Device test patterns
Nameserver agent
Device 1, with known BIST interface
Device 4, ready for repair
Agent 1, moving to Device 2
Agent 3, preparing to repair Device 4
Visualizer agent
Agent 4, preparing to help Agent 2 test Device 3, after finding out its test patterns from the central knowledge base
Facilitator agent
Device 3, with unknown BIST interface or test patterns
Device 2, with known test patterns
Agent 2, wondering about Device 3

Figure 2 – Agents of the society, in action.

The BIST/BISR can be run off line or on line in the background, depending on the capabilities of the device under test. In our experiments, we used both methods – for example, in figure 5, you may observe that the exemplified test is not only off line – taking over the device during the test – but it also needs user intervention. The PC version (not shown) of the BIST/BISR process is on line and runs in the background.

Different agents have different repair capabilities and they have to ask their colleagues if they cannot repair the fault by themselves.

When an agent has to analyze a specific subsystem (device), it executes three major steps:

- detection
- diagnosis
- repair

For each step, the agent has to:

- make a plan
- get the necessary information to execute the plan
- execute the plan
- analyze the results (not compulsory)
- decide (not compulsory)

The first step is to see if there is a fault or not. This may or may not be possible, depending on the agent's capability in finding a way to check that specific device.

The simplest case is when the device has BIST, and the agent knows how to access it. If the agent cannot access the BIST module, it can ask other agents or a database about it.

Another case is when the device does not have BIST, but has some pre generated test sequences in its memory. In this situation, if the agent knows how to access them, it can extract and apply these test patterns. If not, the agent can ask other agents or a database about how to access these test patterns in the device.

Of course, there may be cases when the device does not contain the test patterns in its memory, thus the agent has to request them from other agents or databases.

After detecting the fault, the agent starts a diagnosis (although most fault detection methods include diagnosis as well). In order to do this, the agent uses the same sources of information as for detection.

When the fault has been correctly diagnosed, the agent tries to repair it. It uses the same sources of information as for the detection and diagnosis. Of course, being software by nature, the agent is limited mainly to software repairs.

There may also be cases when the diagnosis is ambiguous, i.e. there may be more causes of the failure. In that case, the agents conduct further tests, eventually "discuss" the problem.

There are four basic types of agents in the society:

- Tester agents
- Nameserver agents
- Facilitator agents
- Visualizer agents

Tester agents are the ones "working", i.e. effectively testing the devices.

Nameservers are like phone books, they make easier for the agents to find each other.

Facilitators are like the Yellow Pages, they know who has what and who knows how to detect or fix what problem.

Visualizers are the interfaces between the agent society and other systems, for example accepting commands from the system administrator and supplying information about tested devices and society status.

More about agent management can be found in [12].

2.3 Agent Communication

At software level, the agents communicate with each other through the FIPA (Foundation for Intelligent Physical Agents) ACL (Agent Communication Language) [12]. FIPA ACL specifications describe aspects of the structure of messages and the ontology service. For now, our agents have a reduced language set, mainly allowing sharing test sets, device test/repair data and system coverage plans.

The FIPA MTP (Agent Message Transport Protocol) specifications [12] present different ways of communication for the agents to exchange data. IIOP (Internet Inter ORB Protocol), WAP (Wireless Application Protocol) and HTTP (HyperText Transfer Protocol), TCP/IP over wireline are described, as well as generic wireless solutions. They also deal with bit oriented, string oriented and XML oriented message representations. Our agents, in their current development status, use TCP/IP over wireline and wireless connections, with the messages in ASCII string format. They ask information from the central database through HTTP. A newer version, with XML, is being developed, to simplify inter agent, agent to database communication and use of protocols like HTTP and WAP.

At hardware level, the agents use whatever communication layer is available for the device (serial, I2C, Ethernet or other). We have also considered embedded TCP/IP solutions.

For a system with mobile subsystems to be tested, short range, standardized radio based Bluetooth chips can be used. For large scattered systems, radio based Wi Fi solutions or GPRS boards are available. Wi Fi works even with public Access Points, while GPRS boards are adequate for low cost, always on sporadic communication over large distances.

2.4 Agent Platforms

2.4.1 Java Micro Edition

Sun's Java 2 Micro Edition [13, 14] is standardized, portable, has a small footprint (Sun's KVM reference implementation has about 128 kiloytes), optimized for networking and very flexible.

To ensure portability among different manufacturers' devices, the MIDP 1.0 (Mobile Information Device Profile) and specification establishes some basic functionality for the first generation Java enabled mobile devices. This guarantees that the programs – "midlets" – will run on any MIDP 1.0 certified hardware.

MIDP 1.0 offers only HTTP type connections by default, but there are a few workarounds to have always on, flexible, raw socket connections – proprietary network connections – between the server and the mobile device. MIDP 2.0 is more flexible in this respect, but few mobile devices comply with it.

On need, the j2me agents can be easily extended with additional functions, enabling a device's additional testing abilities.

The drawback of the j2me solution is that from its conception, Java (Enterprise, Standard or Micro) has been designed for portability. This means that it does not allow native access to the hardware, only through the functions of the virtual machine. On the other hand, special, device-specific classes can be developed, which bypass the virtual machine and access the hardware directly.

Another drawback is that the "midlets" – j2me programs – can be installed and run only on the user's request. This is a security measure, aiming at protecting the user's handheld – the original target of j2me – from unwanted programs. However, if there is already a midlet running on the device, with an active network connection, it can send and receive data, including microagents.

2.4.2 BREW

Qualcomm's BREW platform [15] is similar to Java Micro Edition, but the programs can be developed in C++, as well. There is a Micro Java virtual machine for BREW, so that even the j2me programs are able to run. The main advantage of BREW over Java Micro Edition is that it can run native applications that access the hardware. Its main disadvantage is that its use is not widespread, but the number of BREW enabled devices is increasing.

BREW is mainly embedded into CDMA communication devices.

2.4.3 Symbian

Symbian [16] is actually a low scale operating system, supported by Ericsson, Panasonic, Nokia, Psion, Samsung, Siemens and Sony Ericsson. It is mainly for, but not limited to, enhanced mobile phones. It can even run a Java Micro Edition virtual machine, allowing the j2me solution, presented above, to run. Still, the main advantage of Symbian is that it accepts programs that access the underlying hardware directly, circumventing the problems of the aforementioned Java Micro Edition.

Unfortunately, Symbian also requires more resources than the j2me virtual machine, making it more expensive as embedded agent platform.

2.4.4 PalmOS

PalmOS [17] was originally an operating system for Personal Digital Assistants. Later, some PalmOS PDAs became smartphones, and PalmOS got wireless.

The main advantage of PalmOS, like Symbian's, is that its programs can access the hardware directly. The disadvantage is that it was not designed for background applications, but for programs that interact a lot with the user. However, the latest versions (PalmOS 5 and 6) are promising.

2.4.5 Embedded Linux

Linux, the most acclaimed open operating system, also has many downscaled embedded versions. μCLinux [18], for example, runs on microcontrollers.

Linux, in its embedded versions, is the most powerful and resource efficient platform for embedded computational tasks. The downside is that since the native programs contain native machine instructions, they are not portable to other processors.

For more about devices with embedded Linux, see [19].

2.4.6 Single Board Computers

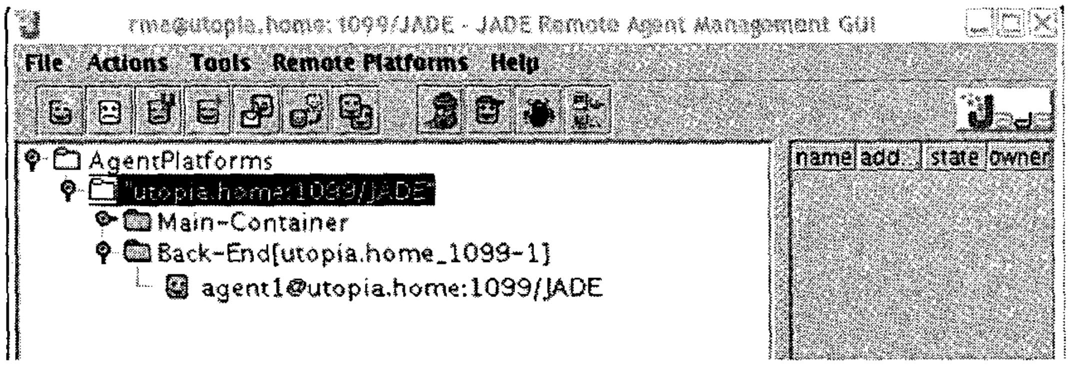
An SBC is, in fact, a hardware platform. It is a powerful computer, usually with network access, audio and video capabilities, lots of processing power, but all crammed on one small printed circuit board. There are even 45x45mm SBC boards.

Most of them use x86 compatible processors, thus are able to run MS Windows. Nevertheless, the majority uses Linux, for its flexibility. See [19] on Linux enabled SBCs.

3. Experimental Results

3.1 JADE-LEAP

The first implementation of the specifications above originates in the extension of the work presented in [1] so that the testing society holds tester microagents. The "mass" of the society was implemented in JADE (Java Agent Development Platform), a powerful agent framework, fully compliant with FIPA standards. The designed


rma@utopia.home:1099/JADE - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help
name
add...
state
owner

AgentPlatforms
"utopia.home:1099/JADE"
Main-Container
Back-End[utopia.home_1099-1]
agent1@utopia.home:1099/JADE

Figure 3 – JADE-LEAP microagent backend.

tester microagent runs on the JADE-LEAP extension of JADE. JADE-LEAP (Lightweight Extensible Agent Platform) is capable of running under Java2ME, thus enabling an agent to exist on an embedded system. The implemented agent community contains agents running both on Java2SE and Java2ME systems, communicating transparently with each other via the middleware provided by JADE-LEAP. Thus, tester agents capable of performing, storing and searching different test procedures for various devices can reside either on PCs, mobile phones or other embedded devices.

The demonstrative scenario tests the display of the mobile phone the tester agent resides on. The following simple set of tests was devised:

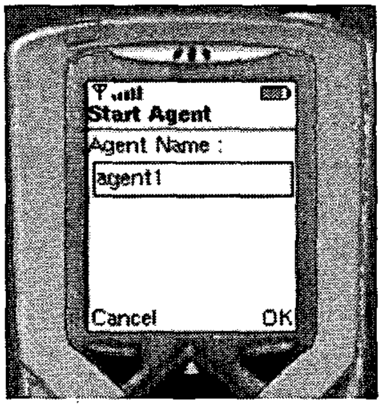
- basic graphical elements tests (labels, tickers etc.) required by the Java2ME specification.
- base color tests (red, green, blue) for color displays.
- black and white patterns (such as a dot/blank/dot grid) for monochrome displays.

The resources of the microdevice being limited, storage of all the data related to the agent on the mobile phone is not feasible. JADE-LEAP allows the split of the agent's data between a container running in a PC environment and the device under test. A backend of the agent will be stored in the PC environment, as seen in the screenshot presented in figure 3. This backend is an interface between the agent society container and the microdevice.

Each microagent has a name unique in the society, chosen by the operator prior to agent deployment. Figure 4, left shows a screenshot of the connection screen, running in a simulated mobile environment.

Microagents store data about testing procedures in special structures called record stores. A record store consists of a collection of data which will remain persistent across multiple invocations of the agent. Each testing procedure is identified by name. The operator issues a test by entering the name of the test (figure 4, right). The microagent searches the required testing procedure in the local record stores. If the test procedure is not found locally, the agent asks the agent society for it, and if no other agent knows it, the database is queried. Finally, the agent will store the procedure persistently in a record store.

The procedure exemplified in figure 5 tests a mobile phones' implementation of the Java2ME specification related to the user interface. A testing procedure consists of a list of display items (labels, gauges, tickers etc.) and questions to be asked for each one. The microagent reads the testing procedure, displays one by one the items contained therein and asks the questions. A test is considered passed if the user answers affirmatively to all questions. The instance showed in the figure tests a ticker (or marquee) which is a piece of text that runs continuously along the display.



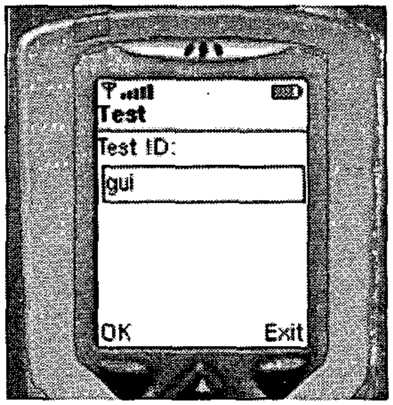


Figure 4 – JADE-LEAP tester microagent screenshots.

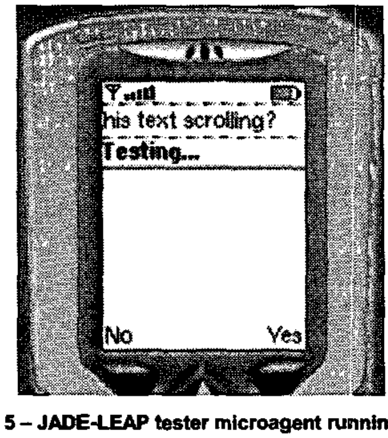
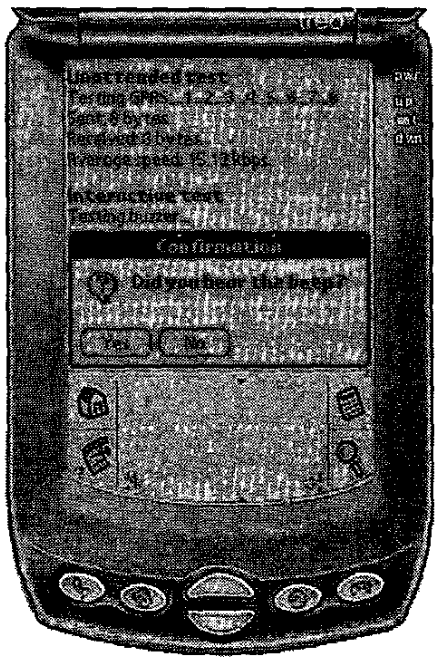


Figure 5 – JADE-LEAP tester microagent running a test.

3.2 PalmOS

The PalmOS port of the microagents can be of two types: the first one does not actually need porting, since it is the same Java Micro Edition implementation described above. The other version runs native PalmOS code. We could not find a PalmOS agent platform, so we started to write our own. Since agent communication is based on FIPA ACL, the agent can communicate with other agents outside the PalmOS device.


Unattended test:
Testing GPRS...1..2..3..4..5..6..7..8
Sent: 8 bytes
Received: 8 bytes
Average speed: 15.12kbps

Interactive test
Testing buzzer...

Confirmation
Did you hear the beep?

Figure 6 – A microagent testing a Treo 180g device.

As you can see in figure 6, the microagent is able to run some unattended, as well as interactive tests. It displays test progress details for debug purposes, but for the unattended tests, it can also run with no output at all. There is one problem though: only the latest PalmOS operating systems support multitasking, so in earlier devices the tests take control of the device and they have to be regarded as off line BIST, at least from the point of view of the user.

4. Conclusions and Future Work

We presented here a few ideas and experiments regarding DBIST and DBISR with microagents, on various small scale platforms.

The agents are able to work together in order to find and possibly solve device problems.

The agents travel from device to device, try to detect and repair errors, and learn new solutions. They can "live" on their own, or work together with other agents and/or a central database.

When an agent cannot detect a cause of an observed fault or cannot repair it, it appeals to other agents to start cooperation. We use a decentralized diagnosis model, which reduces the complexity and communication overhead of centralized solutions. Due to the diversity of devices in modern complex systems, heterogeneous agents can be implemented that take care of device(s) in their responsibility area.

Different agents have different repair capabilities and they have to ask their colleagues if they cannot repair the fault by themselves.

Tester agents do the testing and repair what is repairable. Visualizers supply the interface between the agent society and the outer world. Nameservers and Facilitators provide lookup services for the agents, so they find each other and also offer their services and knowledge.

Of course, device specific routines are both more efficient and more economical, but they lose in portability and ease of development.

The agent management and communication follow FIPA specifications, which describe the management services and communication protocols and formats.

Future development plans include porting the tester agents on more platforms, as well as making the tests more automated and transparent for the user. Implementing on line DBIST/DBISR is an important goal for high availability systems on one hand, and for impatient handheld users, on the other. This, however, implies that the device must have multitasking capabilities (or additional hardware for the purpose), and its peripherals supporting on line testing/repairs (e.g. marching memory tests or on the fly reconfigurable circuitry [20]).

A big problem of the approach we presented here is to find a balance between the simplicity and cost of the components of the distributed system and the depth and accuracy of the tests and repairs. If the processing power, memory and peripheral requirements of the DBIST/DBISR are high, test accuracy and system availability will be high as well, but the costs will increase accordingly, reducing the feasibility of the project. On the other hand, reducing the requirements will limit testing and repair capabilities as a result. Costs will be lower, but so will be availability, too.

5. Acknowledgements

The authors would like to thank dipl. eng. Lucian Buşoniu, MSC and dipl. eng. Andrei Vancea for their help with the microagent development.

6. References

[1] L.Miclea, Enyedi Sz., R. Orghidan, On line BIST Experiments for Distributed Systems, IEEE European Test Workshop ETW'2001, Stockholm, Sweden, May 29th – June 1st, 2001, pp 37-39

[2] L. Miclea, D. Cimpoca, M. Gordan, An On-Line BIST Structure for Distributed Control Systems, Digest of IEEE European Test Workshop ETW'2000, Cascais, Portugal May 23rd – 26th 2000, pp. 283-284

[3] A. Benso, S. Chiusano, S. Di Carlo, HD2BIST: a Hierarchical Framework for BIST Scheduling, Data patterns delivering and diagnosis in SoCs, ITC International Test Conference, pp. 899-901, 10 - 2000.

[4] Monica Lobetti Bodoni, A. Benso, S. Chiusano, G. di Natale, P. Prinetto, An Effective Distributed BIST Architecture for RAMs , Informal Digest of IEEE European Test Workshop ETW 2000, pp. 201-206

[5] R. Pendurkar, A. Chatterjee, Y. Zorian, A Distributed BIST Technique for Diagnosis of MCM Interconnections, International Test Conference 1996 Proceedings, pp. 214-221

[6] L. Miclea, Enyedi Sz., G. Toderean, P. Prinetto, A. Benso, Agent Based DBIST / DBISR and its Web / Wireless Management, International Test Conference ITC 2003, Charlotte, NC, USA, September 30 – October 2, 2003, pp. 952-960.

[7] L. Miclea, Enyedi Sz., A. Benso, Intelligent Agents and BIST/BISR - Working Together in Distributed Systems, Proceedings of International Test Conference, Baltimore, USA, 8th–10th October, 2002, pp. 940-946.

[8] L.Miclea, Enyedi Sz., Distributed Built-In Self-Test using Intelligent Agents, IEEE European Test Workshop ETW'2002, Corfu, Greece, May 26th–May 29th, 2002.

[9] J. Sheppard, M. Kaufman, IEEE 1232 and p1522 standards, AUTOTESTCON Proceedings, 2000 IEEE, 2000, pp. 388-397

[10] J. Ferber, Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison-Wesley, 1999

[11] I. A. Letia, F. Craciun, Z Köpe, A Netin, Distributed diagnosis by BDI agents, In M H Hamza (ed), IASTED International Conference "Applied Informatics", Innsbruck, Austria, 2000, 862-867, ACTA Press

[12] *** FIPA standards and specifications, http://www.fipa.org

[13] Qusay Mahmoud, Learning Wireless Java, O'Reilly, 2002

[14] *** Official Java 2 Micro Edition site, http://java.sun.com/j2me

[15] *** Official BREW site, http:// http://www.qualcomm.com/brew

[16] *** Official Symbian site, http://www.symbian.com

[17] *** Official PalmOS site, http://www.palmsource.com

[18] *** Official µCLinux site, http://www.uclinux.org

[19] *** Linux Devices site, http://www.linuxdevices.com

[20] Diederik Verkest, Machine Chameleon, IEEE Spectrum, vol. 40, issue.12, 2003, pp. 41 46