A Fault Diagnosis System for the Connected Home

Peter Utton and Eric Scharf, Queen Mary, University of London

ABSTRACT

The connected home of the future, in which all consumer appliances in a home are networked together, is close to becoming the connected home of today. This article explores the role of fault diagnosis in such an environment and explains how agent technology may be applied. The article outlines the need for future standards that can maximize the benefit of a shared diagnostic system.

INTRODUCTION

Although the idea of the "connected home" is not new — after all, the concept formed a cornerstone of the Jetsons cartoons dating from 1962 — it is beginning to look as if its time may have finally come. With widespread Internet access from homes, the growth in the use of broadband and wireless LAN technologies, and the emergence of new devices such as networked audio players, the vision of "devices working together with synergy to realize the potential of the digital decade" (Bill Gates [1]) may soon become a reality.

Given this context, this article explores the role of fault diagnosis in the connected home and reports on how agent technology may be used to address this issue. It draws on experiences gained through the development of an experimental prototype fault diagnosis system (FDS). It closes by looking to the future and argues the need for standards to assist with successful resolution of faults in multivendor home systems of the future.

THE CONNECTED HOME CONCEPT

In the authors' opinion, most current home networks can be categorized as first-generation systems: they typically encompass a few PCs sharing an Internet connection and possibly a printer. However, we believe the future lies with a second generation of systems built around a *service gateway* at the heart of the system.

A service-gateway-based system extends the home network vision to include a range of devices, not just PCs, attached to a variety of types of local networks with differing properties, and in so doing should help to realize the truly connected home (Fig. 1). For example, both wired networks, such as IP over Ethernet and Home Audio Video Interoperability (HAVi) over Firewire (IEEE1394), and wireless networks, such as Wi-Fi (IEEE802.11b), might be included. Indeed, connectivity for some subnetworks may include powerline technologies such as X10, where low-bit-rate signals are transmitted over mains cabling.

The goal of such a home network is to enable a wide range of different applications to execute within the home environment. Applications would involve processes running locally on one or more devices and also possibly incorporate external services executing on remote service provider servers. Hence, such a system would provide the framework for deployment of so-called *Internet appliances*.

The essential characteristic of a service gateway is that it should support the dynamic download and installation of program code to enable on-the-fly selection and execution of new services and the straightforward introduction of new local devices and networks. Arguably, the leading standard in this area stems from the Open Services Gateway Initiative (OSGi) [2]. The OSGi was formed to develop and promote open specifications for the network delivery of managed services to devices primarily in the home. To this end, the OSGi have defined a standard known as the OSGi service platform [3]. It is important to note the change in emphasis from service gateway to service platform as the standard has evolved and matured, and its value as a general-purpose distributed services platform has been recognized.

Figure 2 provides a simplified representation of a home networking system based on the OSGi standard. This is based on Java, which enables the required dynamic code download. It is hardware- and operating-system-independent, merely requiring a small footprint Java virtual machine, and is designed to work with multiple network technologies, device access technologies, and services. OSGi functionality is structured into a core set of framework services and an extensible set of *bundles*. Bundles form the basis for dynamic code download.

 0163-6804/04/$20.00 © 2004 IEEE

Plug and play is a fundamental goal of OSGi, and support for automatic discovery is built into the OSGi device access manager. For this to work smoothly end to end, with a minimum of user interaction, the system requires devices to be able to announce themselves on connection to their LAN. This feature can be provided by a number of complementary technologies, such as Jini [4], UPnP [5], Bluetooth [6], and HAVi [7]. As a result, OSGi in combination with these other technologies offers the basis for a sound service-based home network environment.

The OSGi specification dictates that OSGi platforms must support administration by a remote service provider, hence the presence of the remote manager role in Fig. 2. In fact, the OSGi promotes a philosophy it terms *zero user admin* to reflect the fact that negligible effort on the user's part should be devoted to systems administration.

Another relevant emerging standard is the home electronic system, part 1 of which was published in 2003 [8]. Informally known as HomeGate, it defines the architecture for a modular platform to act as a gateway for delivery of broadband information to the home and facilitate the interoperability of different LAN technologies. In contrast to OSGi, HomeGate is more hardware-oriented, mandating the use of physical plug-in modules rather than remotely downloadable software bundles. It does, however, have the backing of an international standard.

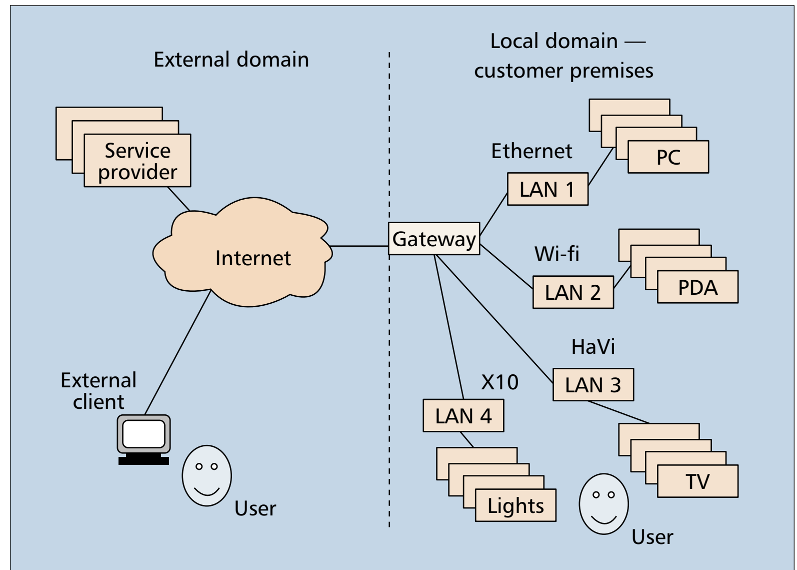

Figure 1. *The connected home: system outline.*

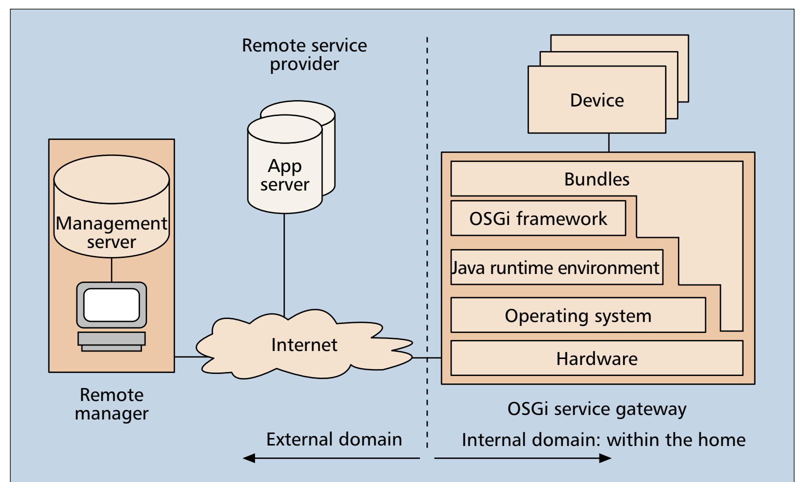

Figure 2. *An OSGi -based system.*

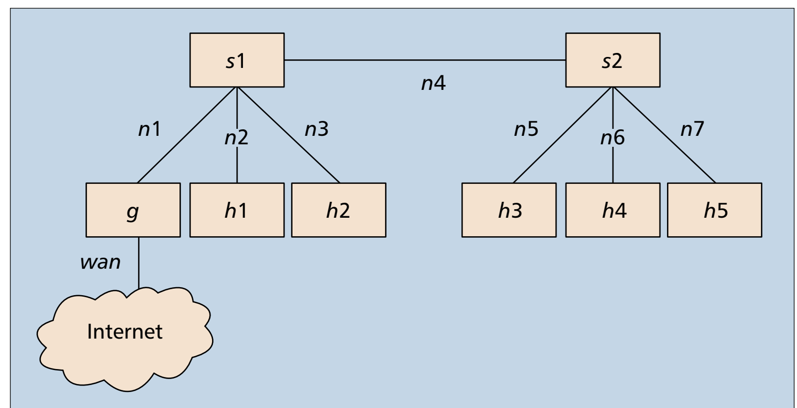

Figure 3. *An example home network.*

The Need for Fault Diagnosis

Within the home environment, the management of faults will be critically important. Second-generation home networks will be inherently complex, but must appear simple to users. Faults will occur and as systems are likely to be assembled from components from different suppliers, diagnosing problems and localizing them to an individual component will be a key requirement. It is likely that component suppliers will be reluctant to accept responsibility for resolving faults unless there is clear evidence that the blame lies at their door. For home networking to achieve widespread acceptance, it will demand automated support for identifying and resolving faults.

The management of fault conditions in networks is a long established discipline (and of course gives rise to the F in the FCAPS mnemonic for standard network management functions, along with configuration, accounting, performance, and security), but second-generation home networks are likely to exhibit certain characteristics that differentiate them from traditional networks. They are likely to:

- Be smaller in scale than typical business or telecommunications networks
- Incorporate a wide diversity of device types
- Perhaps crucially have limited user expertise available to resolve problems

Certainly there will not be a team of skilled human experts on hand to act as technical support. Consequently, maintaining the ethos of zero user admin will demand a fault diagnosis system that:

- Is highly autonomous
- Provides expert advice to users
- Exploits multiple knowledge sources to enable the most informed decisions to be made

This article will consider some fault scenarios

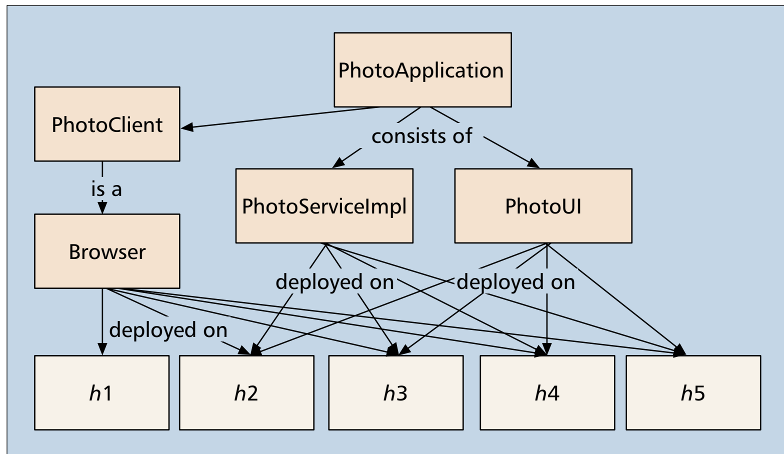

Figure 4. *The deployment model for a photo application with an example home network.*

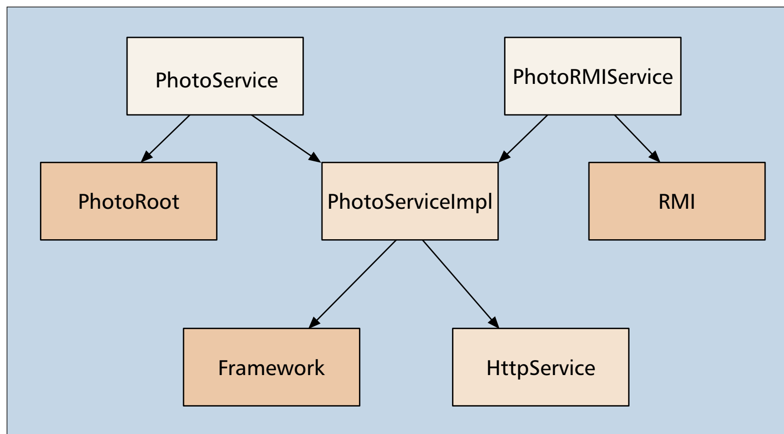

Figure 5. *The dependency model for the PhotoService implementation bundle.*

for the example network shown in Fig. 3. In this hypothetical network configuration, $s1$ and $s2$ are 4-port switches, node g acts as the gateway for external connectivity, and hosts $h1$–$h5$ are available to run applications. Network links are labeled $n1$–$n7$, apart from the wide area network link, labeled *wan*.

Our example uses a simple photo gallery application built as a service using the OSGi application programming interface (API). OSGi enables services running on one service platform to call other services running on the same platform. Unfortunately, at present OSGi does not support federations of platforms and does not directly support calls to services on remote platforms. However, the expectation is that a family's electronic photos would be distributed across multiple nodes in the network, each node under the control of a different family member. The facilities offered by an OSGi framework provide benefits for deployment of software on internal hosts as well as the gateway, so nodes g and $h1$–$h5$ in our scenario are all OSGi enabled platforms. The photo application allows users to browse JPEG images on both local and remote nodes, and achieves this by using the Java remote method invocation (RMI) API to make calls between PhotoService instances on different nodes.

The photo application is structured into two types of OSGi bundle: a PhotoUI bundle, which offers a browser based user interface and calls the underlying PhotoService to access photo resources; and a Photo bundle, which implements the PhotoService itself. Within the example home network, these bundles are deployed on nodes $h2$–$h5$ and can be accessed from browsers on any of nodes $h1$–$h5$. Figure 4 illustrates a deployment model.

Fault Diagnosis Approaches

In general, the purpose of fault diagnosis within systems is to identify *least replaceable units* (the smallest faulty components that can actually be replaced). Once such a component has been identified there is no need to diagnose further inside the component. The level of granularity of components will depend on the system under consideration. In the context of a connected home, we can regard network links, nodes, devices, and software as such components. To generate a diagnosis we will need to collect observations of operational behavior that we will treat as a set of symptoms (i.e., the input to our diagnostic process). The output is a diagnosis that can be regarded as a logical proposition asserting that a particular combination of components is faulty. A diagnosis may be a complex proposition covering sets of alternate suspects.

Candidate fault diagnosis techniques include experience-based techniques using heuristic knowledge [9] and model-based techniques involving models of expected system structure and behavior [10]. We have taken the model-based approach.

Different models can be used to address different concerns. For example, models could represent user behavior, application structure, application behavior, application deployment, or network connectivity. This article shows how models of application structure, deployment, and network connectivity can help us resolve problems.

Each of the bundles for our Photo application exposes a number of external interfaces (ways in which the functionality they contain may be invoked) and embodies a number of dependencies (other system entities on which the code relies for successful operation). Figures 5 and 6 illustrate dependency models for the two bundles. White nodes represent external (callable) entry points to the bundle, tan nodes represent code within bundles, and orange nodes represent miscellaneous other entities that may range from complex subsystems (e.g., RMI) to simple application configuration settings (e.g., PhotoRoot, which represents the fact that a property identifying the root directory for photos needs to be set for PhotoService to operate successfully).

Dependency models are a common technique within fault diagnosis and involve the application of a simple principle: the fault free-operation of an entity is reliant on the fault-free operation of all dependent entities. Therefore, the successful operation of an entity can be used to *exonerate* all its dependents, whereas the detection of a failure in an entity will *implicate* that entity and all its dependents until they can be cleared.

These principles hold for analysis of both applications and networks, and provide us with a basis for diagnosing faults across a wide range of systems. However, we do need a means of testing our diagnoses. At a network level we can incorporate and automate the long established practice of pinging another network node to check connectivity. This normally involves the use of Internet Control Message Protocol (ICMP) messages at a network level. At an application level we can incorporate test procedures for entities defined in a dependency model. This article will explain how these features are incorporated within an *agent-based* fault diagnosis system (FDS).

Agent-Based Solutions

Arguably the leading innovation in the field of artificial intelligence during the last decade has been the emergence of so-called *intelligent* agents operating in the context of distributed or *multi-agent* systems. Developments in agent technology have benefited immensely from the increasing availability of low-cost distributed processing power and the ubiquity of TCP/IP networks. The technology has also received an impetus from the emergence of Java as a platform-independent programming environment and, more recently, from the emergence of the Foundation of Intelligent Physical Agents (FIPA) [11] as a standards body encouraging interoperation between competing agent platforms.

The key characteristic of a multi-agent-based solution lies in the idea that the whole is more than the sum of its parts. Individual agents need not be very complex, but the emergent behavior arising from their interactions should be able to solve problems that are impractical or impossible for an isolated agent to tackle.

There are a several reasons why agents are suited to fault diagnosis within home networks:

- This task is naturally distributed. Agents can operate within the distributed nodes of the home network, execute local tests on components, and cooperate with other agents to identify problems. Decentralizing processing also removes potential single points of failure, and the processing load on a remote manager can certainly be reduced if we process diagnostics locally.
- Different agents can implement different analysis techniques (e.g., heuristic or model-based), and an open agent architecture allows the straightforward introduction of alternate analyzers within the FDS.
- The goal of the system being highly autonomous aligns particularly well with agent philosophy. We can visualize such a system as embedding various rule-based and model-based analyzers within the home network, exploiting a range of knowledge sources and taking the initiative to run a variety of system tests in order to provide proactive assistance to human users. Furthermore, in the event of network disruption, agents operating in different partitions can still reason about their view of the problem and maintain some level of service to users.

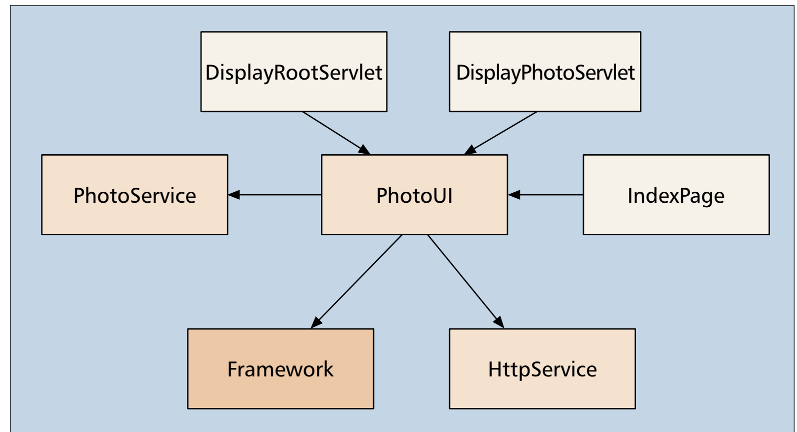

Figure 6. *The dependency model for the PhotoUI bundle.*

Of course, an agent approach is not without its drawbacks. For example, the computing resources at some nodes may be limited. However, by separating essential "per node" functionality from more computationally demanding processes, it may be possible to accommodate a balance between "lightweight" and "heavyweight" agents.

The FDS Prototype

The FDS prototype is designed to take advantage of three pre-existing software packages: an OSGi framework developed by Gatespace [12], the FIPA-OS Agent Framework [13], and JESS, the Java Expert System Shell [14].

FIPA-OS provides an open source Java-based implementation of a FIPA compliant agent platform. As such it provides facilities for message passing between distributed agents and directory lookup of the identities of registered agents.

JESS provides a rule-based programming system with good performance characteristics that may be readily integrated with other Java-based systems. JESS is used to implement various analyzer agents within FDS, including both application- and network-level analyzers.

In addition to JESS- based analyzers, a pure Java alternate analyzer agent has also been implemented. This embodies a graph-based algorithm to analyze network connectivity. FDS also includes Modeller, Reporter, Tester and Collaborator agents. The Modeller maintains dynamic models of the system (e.g., of network topology). Static models (e.g., of application structure) are generally packaged within application bundles.

The FIPA-OS, JESS, and FDS software has been packaged into a set of OSGi bundles that allow OSGi mechanisms to be used to deploy FDS code by remotely and dynamically updating the FDS software code for each distributed node. To achieve this, compiled FDS code is made available on a predefined code server within the network.

Generating Diagnoses

Diagnosis results consist of alternate sets of suspects (i.e., possibly faulty components) and a set of cleared (i.e., fault-free) components. These can be represented as AND/OR tree structures

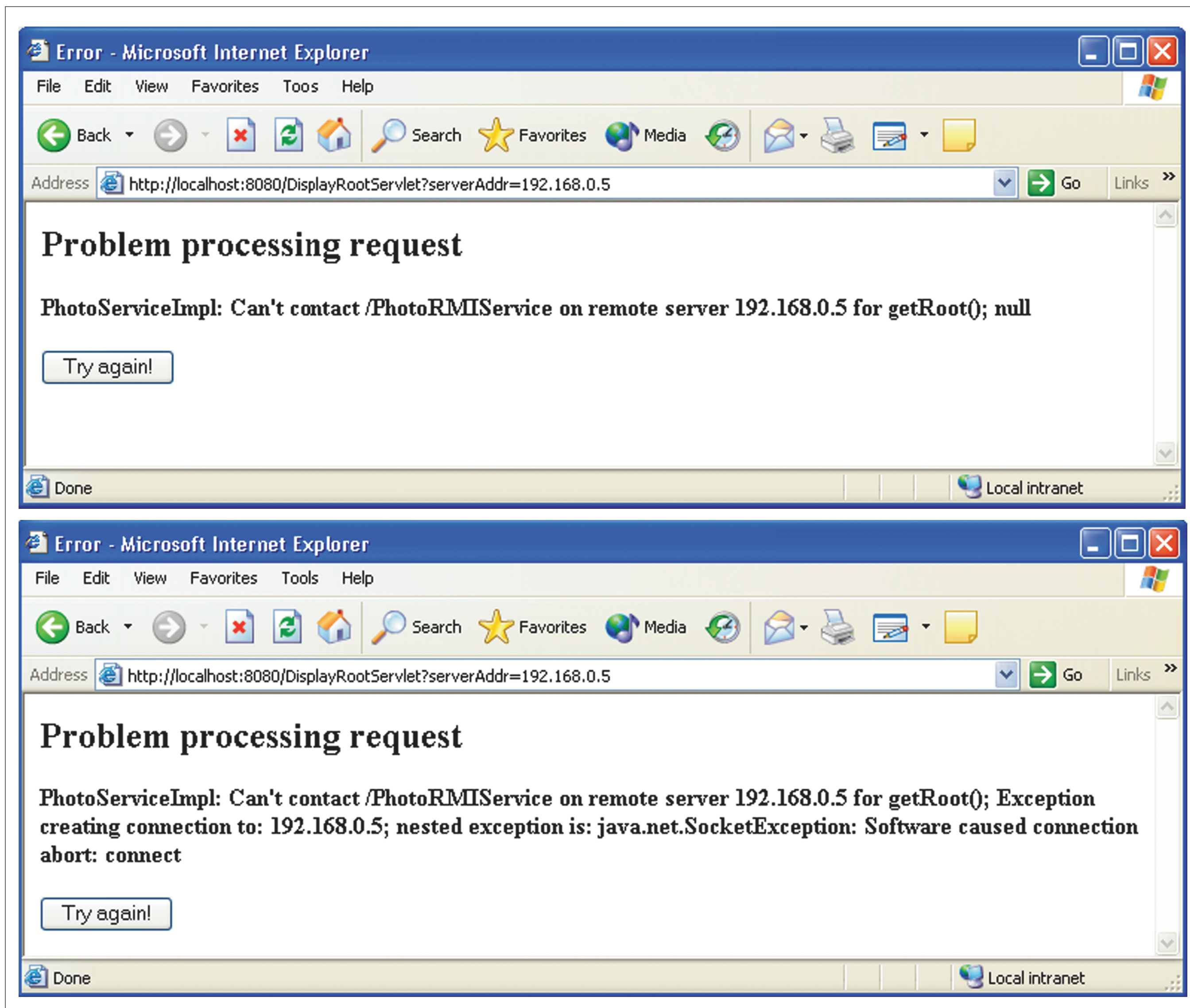

Figure 7. *Example application error displays.*

or logical propositions. Within FDS, diagnoses are generated from symptoms by analyzer agents. Symptoms may be collected proactively by periodically "routining" components within the network (i.e., interrogating devices and invoking test procedures) or by reacting to notifications of abnormal behavior from instrumented applications or external agencies. The FDS prototype exposes an interface, FDSService, as an OSGi service that instrumented applications such as the Photo application can use to pass information into FDS when they encounter a problem. In a Java application, this would typically follow an exception being thrown. Although instrumentation may be regarded as intrusive, without the active participation of applications, diagnostic ability will be limited. In such a case, periodic automated checks of network connectivity are still possible, and FDS also provides a User agent to allow manual entry of observations. However, this still requires application models or heuristic knowledge to drive the diagnosis, and manual symptom entry is intrusive for the user. Essentially more informed diagnosis is possible with the cooperation of applications and, of course, the availability of more information.

Let us consider some fault scenarios for our example network. Imagine that an instance of PhotoService running on *h*2 encounters a problem getting a response from the instance on *h*5. This may be because the *h*5 node is unreachable, perhaps the node itself is down, or network link *n*3, *n*4, or *n*7 has become disconnected, or switch *s*1 or *s*2 is faulty. Alternately, perhaps the PhotoService on *h*5 has experienced a runtime failure.

The application itself may attempt to present advice to the user of the problem experienced, although this may not prove to be helpful, even to an experienced user. Figure 7 shows examples produced by the PhotoUI bundle when encountering different exceptions. The second example at least indicates there may be a network-level problem, but the first is less than helpful (it was actually triggered by deliberately seeding an ArrayOutOfBounds fault in PhotoServiceImpl).

As mentioned, FDS employs different analyzers for different models, but the results of these different analysis viewpoints can be integrated

and — a key point — faults at a lower level can be used to *subsume* faults at higher levels. This means that the application analyzer will look no further if a network level analyzer has already identified a problem that has generated suspects that match any of its own suspect set and so explain the symptoms being experienced.

The model used by the application analyzer is generated by combining the bundle dependency models and deployment models. In our example scenario, PhotoUI running on *h*2 notifies its local FDS agent of a problem with PhotoService on *h*2 and in turn the PhotoService on *h*2 reports a problem with PhotoRMIService on *h*5. This triggers invocation of the application analyzer, which in the absence of further symptoms concludes there is a problem with one of the components associated with the Photo application, with either one of its dependent entities or one of the platforms on which they are deployed. This results in an initial 54 suspects: 49 software components (the 11 distinct entities shown in Figs. 5 and 6 deployed on each of platforms *h*2–*h*5 plus the five instances of browser from Fig. 4) and platforms *h*1–*h*5.

The application analyzer knows nothing of network connectivity or network-level problems but is able to consult with a network-level analyzer for this viewpoint. The network connectivity model in conjunction with reachability symptoms enables a network analyzer to identify possible faulty intermediate nodes and subnets, not just faulty destination nodes. The more symptoms input to an analysis, the more tightly focused the results.

Let us assume that node *h*5 is powered down. With the benefit of unreachable symptoms for *h*5 and reachable symptoms for all other nodes, the network-level analyzer would be able to conclude that either component *h*5 or *n*7 is faulty. Either possible fault would explain why *h*5 is unreachable but all other nodes are reachable.

The network analyzer contributes this conclusion into the working hypothesis, and the application analyzer is satisfied that its suspects are subsumed by the network-level suspects. The presence of the network fault is sufficient to explain the symptoms. However, in this scenario we need additional information to be able to arbitrate between a fault in the IP network at *n*7 or a fault in *h*5 itself. For these situations, FDS implements a low-bit-rate secondary communications channel using the powerline technology X10. As a final test it sends a message to *h*5 over the mains cabling. X10 is a broadcast medium, and all powered up hosts will receive the message via their serial port. However, only *h*5, assuming it is up, would reply. The message times out, and FDS concludes that *h*5 is indeed faulty.

Alternately, if our network is fully operational, FDS will start to test suspect software components using Tester agents that invoke test procedures devised specifically for the components concerned. Test procedures are supplied in Java classes named after specific entities in a dependency model (e.g., *Test_PhotoRoot* is the class containing the test procedure for entity *PhotoRoot*) and deployed via OSGi bundles.

The test procedure embodies knowledge of how to trigger execution of the particular entity and what its expected results should be. The procedure therefore returns a value indicating whether the test has been passed or failed. This enshrines the principle of information hiding and decouples applications from the diagnostic system: dependency models are built into bundles as resource files and registered with FDS on bundle activation; test procedures are provided in separate bundles that can be dynamically updated. Therefore, FDS does not need to know anything about the internals of the entities under test.

It is computationally expensive to test every suspect, and the order in which tests are applied can have a significant impact on the speed of problem resolution. Remember that a successful test that demonstrates full functionality of a target entity can be used to exonerate its dependents, so top-down testing of those components more likely to be okay (as assessed using heuristic knowledge) can be effective in rapidly pruning the search space. Bottom-up testing of suspects with known fault history can help identify the culprit. Early targeting of suspects on the primary problem node (*h*5) is also a valid strategy.

However, the diagnostic procedure is only as good as the models and test procedures available to it. Identifying the ArrayOutOfBounds fault would be dependent on the effectiveness of the *Test_PhotoServiceImpl* procedure, and it is not practical to guarantee complete test coverage for every eventuality. Operational fault diagnosis should not be considered a substitute for thorough development testing. Configuration problems such as those detected by *Test_PhotoRoot* are more typical of the class of problem this type of approach can economically solve.

However, during this test phase, FDS would proceed to test selected suspects, accumulate additional symptoms, and rediagnose until a single suspect set remains (i.e., the diagnosis is that this combination of components is faulty) or test options are exhausted.

Diagnosis, as we have considered it thus far, has been driven from a single viewpoint within the network, typically the location where an abnormal notification has originated. However, multiple problems may well be notified at different locations in a distributed system, both in the same timeframe and at different times. These may relate to the same causal fault or different ones. Rather than pursuing each problem independently, agents within a multi-agent system can interact with each other in various ways to formulate a collective solution. We have experimented with two distinct methods we call:

- *Holistic* diagnosis, to signify analysis using all relevant observations (i.e., those emanating from multiple locations and viewpoints) in one operation
- *Collaborative* diagnosis, which involves agents comparing the results of local diagnoses and agreeing on a shared result

Hopefully, with the editors' permission, a future article will be able to explore these different interaction methods and highlight their pros and cons. Certainly we have found different situations where each method performs better than the other.

The network connectivity model in conjunction with reachability symptoms enables a network analyzer to identify possible faulty intermediate nodes and sub-nets, not just faulty destination nodes.

To maximize the utility of a shared diagnostic infrastructure, applications and devices would need to be able to register models of their expected behavior, thus enabling the diagnostic mechanisms to be applied as the home network is extended.

LOOKING TO THE FUTURE

The ideas presented here could be extended to more detailed models for addressing application functionality and also to models covering the internal behavior of devices. For example, application models derived from UML-style sequence diagrams (message sequence charts) could be used to expose expected application behavior to enable more informed model-based reasoning about fault conditions and help target suspect software components. Furthermore, different problems, such as streaming related issues for multimedia applications, will demand different types of models and/or heuristic knowledge and appropriate supporting analyzers.

Of course, individual applications could embed more intelligence to resolve operational problems for themselves, but there is a clear benefit in using a shared diagnostic infrastructure to avoid the need for repeated implementation of common functionality. In addition, components cannot be expected to reason about the effect of interactions between entities about which they know nothing; hence, the need for an independent diagnostic capability that is aware of the applications deployed on the network and the network topology.

To maximize the utility of a shared diagnostic infrastructure, applications and devices would need to be able to register models of their expected behavior, thus enabling the diagnostic mechanisms to be applied as the home network is extended. This implies the need for standards for the exchange of data between diagnostic clients and the infrastructure.

Universal Plug-and-Play (UPnP), mentioned earlier as a complementary technology to OSGi, may provide a more elegant solution for the interfacing of client applications and devices to the diagnostic infrastructure than the current mechanisms employed within the FDS prototype. UPnP employs wireline protocols, which means it is platform-neutral, and uses XML for data exchange. In the context of FDS with standardized analyzers, the data to be exchanged would be problem notifications and models. For an open FDS where individual analyzers and other agents could be replaced, this would also mean symptoms and diagnoses. For effective communication, ontology and taxonomy agreements are needed between participants. This is an area where HomeGate could make valuable contributions. Its purpose is to define the means for disparate systems in the home to interoperate. One aspect of this would be a common language for communication; indeed, part 2 of the standard is intended to address taxonomy issues.

CONCLUSIONS

This article has argued that fault diagnosis will be important for home networks of the future. It has outlined how agent-based diagnostic solutions might be applied in this domain by means of a worked example. It closes by looking to the future and the need for standards to facilitate fault diagnosis within a connected home embodying components from multiple vendors.

If shared diagnostic infrastructure becomes a reality and standardization bears fruit, the benefits will be considerable: consumers will enjoy an improved user experience, vendors will find their products more highly regarded by users, the volume of calls to help desks will be reduced, and residual calls should be more quickly resolved as relevant diagnostics would be readily at hand.

REFERENCES

[1] "Come Over To My House," *Guardian Online*, 23 Jan., 2003; http://www.guardian.co.uk/online/
[2] OSGi, http://www.osgi.org
[3] OSGi Service Platform, rel. 3, Mar. 2003, IOS Press.
[4] Jini, http://www.sun.com/jini
[5] UPnP, http://www.upnp.org
[6] Bluetooth, http://www.bluetooth.com
[7] HAVi, http://www.havi.org
[8] Home Electronic System Standards, ISO/IEC 15045-1, committee ISO/IEC JTC1 SC25 WG1; http://hes-standards.org
[9] C. Price, *Computer-Based Diagnostic Systems*, Springer Verlag (Practitioner Series), 1999.
[10] W. Hamscher, L. Console, and J. de Kleer, *Readings in Model Based Diagnosis*, Morgan Kaufmann, 1992.
[11] FIPA, http://www.fipa.org
[12] Gatespace's Service Gateway Application Development Kit, http://www.gatespace.com/
[13] FIPA-OS, http://fipa-os.sourceforge.net; http://www.emorphia.com
[14] Sandia Nat'l. Labs, JESS, http://herzberg.ca.sandia.gov/jess/

BIOGRAPHIES

PETER UTTON (peter.utton@elec.qmul.ac.uk) holds degrees in materials science, computer science, and telecommunications. He has spent a substantial part of his career working at the main BT research laboratories at Adastral Park, United Kingdom. He currently lectures on a range of software related subjects for the Electronic Engineering Department at Queen Mary, University of London. The connected home is his personal research interest.

ERIC SCHARF (e.m.scharf@elec.qmul.ac.uk) is a lecturer in the Department of Electronic Engineering at Queen Mary, University of London. Since 1989 he has participated in many European Union funded projects on intelligent networks and risk assessment. In particular, he coordinated the recent successful EU-funded TORRENT project on intelligent residential access networks. His interests are in communication networks and computing, and he has written and co-authored several papers in these areas.