DYNAMIC CODE DEPLOYMENT ENABLES FAULT DIAGNOSIS IN THE CONNECTED HOME

P. C. Utton

Queen Mary, University of London

ABSTRACT

This paper discusses the role of the OSGi Service Platform in support of dynamic code deployment within home networks and its use to enable the operation of an extensible fault diagnosis capability. This capability takes the form of a prototype agent based system that can identify faults in software and hardware components by operating on a largely autonomous basis.

Index Terms— Connected home, home networks, OSGi, fault diagnosis

1. INTRODUCTION

Connected homes, involving the networked interconnection of a wide range of consumer electronics and home appliances, offer the prospect of increased comfort, convenience and control for consumers but can necessitate the use of complex technology. With complexity comes increased risk of operational failure. Automated fault diagnosis which relieves the consumer of the burden of identifying the root causes of problems may be critically important for the widespread acceptance of future home networks. Certainly the concept of 'zero user admin' has long been a goal in the world of the connected home.

With this end in mind, at QMUL we have developed a prototype Fault Diagnosis System (FDS) that provides an agent based approach to diagnosis in home networks. This paper outlines the capabilities of FDS (Section 2), then walks through an example diagnostic process (Section 3) and then focuses on the role that the OSGi Service Platform [1] plays in the system (Section 4). Section 5 provides brief conclusions.

2. THE FDS PROTOTYPE

FDS performs distributed diagnosis by employing a network of software agents which individually offer a variety of capabilities, are distributed across the nodes of a home network and collectively cooperative to exchange information to achieve a network wide diagnosis of faults. In essence, FDS generates diagnoses that identify suspect components by analysis of symptoms (observations of the behaviour of the home network and its applications) with reference to models of various aspects of the system, see Figure 1.

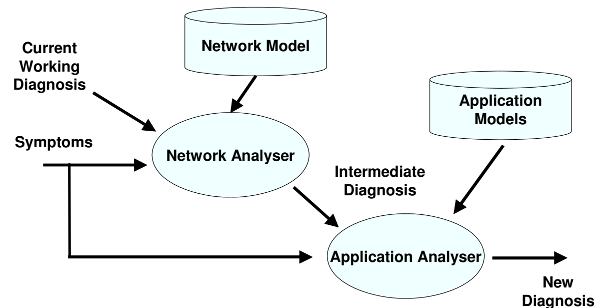

Figure 1: Diagnoses emerge from analysis of symptoms

Initial diagnoses, which typically involve multiple alternative suspects, are refined towards a final, definitive diagnosis of faulty components by exploiting information gleaned from different system viewpoints and by the use of test procedures. Such procedures know how to exercise specific suspect components within their operational environment so that additional symptoms may be generated to feed into the diagnostic process. In some cases test procedures will have knowledge of the expected outputs of specific components and can directly confirm or exonerate the components as faulty.

FDS incorporates an open architecture with the following key features:

- The use of plug-in diagnostic components including test procedures, plans and add-on agents.
- Separation of the knowledge of symptom and model types from knowledge of symptom and model details.
- Separation of diagnostic methods from collaboration mechanisms. The former use the symptoms and models, the latter do not.
- Use of a minimal ontology that is understood by both diagnostic methods and collaboration mechanisms.

In this way coupling between generic and problem specific diagnostic components is minimized. The open architecture of FDS is described in more detail in [2]. The

concept of application modeling and the use of associated test procedures is described in [3]. Note that an important design goal of FDS has been to allow dynamic changes to diagnostic components so that the diagnostic capability can be extended whilst the system is in operation. These extensions can, in principle, allow the resolution of new types of problem that emerge as new types of device and application are deployed within the network. This facility is underpinned by use of OSGi Service Platforms.

3. FDS PROCESSING

Having provided an outline of FDS capabilities in the previous section, this section will walk through an example end to end diagnostic process from initial problem detection to the availability of a final diagnosis.

A diagnostic process is triggered by the creation of one or more symptoms indicating abnormal behaviour. Such symptoms can come from self aware applications using a problem notification API offered by FDS, from users via a user agent offering an appropriate user friendly interface or by pro-active testing of components on a routine basis. A final FDS diagnosis can localise faults to application components, individual nodes or network links within the home network, see Figure 2. The diagnosis in Figure 2 indicates three faults, all of them software components, deployed on different nodes in the home network.

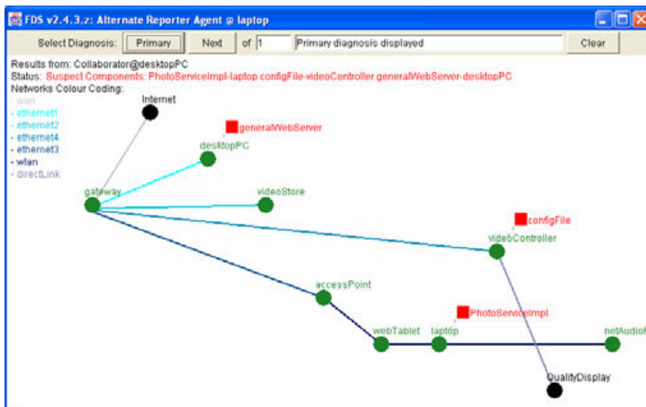

Figure 2: Example FDS Diagnosis

A Java application encountering a run time problem will typically throw an exception. In some cases the application may be able to identify the cause and indeed recover from it. In many cases however, the code will abandon execution, possibly only providing a cryptic error message (e.g. a stack trace) by way of explanation. At this point, an FDS enabled application will call the problem notification API from within a Java exception handler to declare an application problem. This triggers generation of an abnormal application symptom on the local node where the application code was executing and activates a diagnosis, if one is not already active.

An FDS collaborator agent resides on each FDS enabled node in the network and collectively they orchestrate the diagnostic process. Once activated, the first action of the collaborator agent on the local node is to evaluate the available symptoms. This normally involves initiating one or more analysis steps. Analysis is performed by analyser agents that are deployed on selected nodes within the home network. These nodes would normally be relatively powerful computational devices compared to typical CE devices.

On startup, an analyser registers its availability with all collaborator agents and declares what types of symptoms it knows about and the treatment rules to be applied to those symptoms. A treatment rule will normally dictate that the analyser concerned should be called to analyse the available symptoms and in these cases the analyser also declares what types of model it needs to perform the analysis. Different types of analyser will understand different types of problem and use different diagnostic methods which will typically operate on different types of model. Within the FDS prototype, a number of network level analysers have been implemented with different analysis algorithms operating on models of the topology of the network. In addition two different application level analysers have been implemented: one employs dependency models of the application and the other applies experiential diagnosis and embodies a small case base allowing interpretation of selected Java exceptions.

The active collaborator inspects the symptom types belonging to its available symptoms and selects an analyser to call. The analyser's registration identifies the models required and the collaborator makes a call to a modeller agent to retrieve the necessary files. There will be a single modeller agent deployed within the home network and it acts as the custodian for all models.

The internal form of a model will depend on its type and neither the collaborator nor modeller agents necessarily understand a model's content. However analysers do. Models can be statically defined, for example representing application structure, or dynamically constructed, for example representing the deployment of application components across a network. FDS enabled applications register models with the modeller agent on application startup. The modeller agent then provides copies of them to collaborator agents on demand. The collaborator agent will cache a copy of a model locally to avoid unnecessary repeat requests.

Having determined the analyser to call and retrieved all required models, the active collaborator sends an analysis request to the analyser and includes all available symptoms, the requested models and the collaborator's current working diagnosis. If the system has recently been reset, then the

current working diagnosis may be empty. The analyser processes the request and returns a new diagnosis.

The active collaborator determines which symptoms have been resolved by the analysis step (some analysers handle multiple symptom types) and continues to call selected analysers until all symptoms have been resolved. It then attempts to share its latest working diagnosis with all other collaborators within the home network. This collaboration step attempts to achieve a consensus diagnosis across the home network.

Multiple problems may have occurred concurrently at different nodes in the network and different collaborators may have become active at the same time. They will each have their own working diagnosis which may relate to the same or completely separate faults. Working diagnoses include information on confirmed faults and cleared components as well as suspects. By exchanging this information, agents with one viewpoint can help refine the viewpoint of another by exonerating or confirming suspects. Such viewpoints can involve different locations and conclusions derived from different types of symptoms. In addition, viewpoints of different analysers can overlap and FDS allows for suspects suggested by one analyser to be subsumed by a diagnosis generated by another.

However *opportunistic* refinement of this kind will rarely lead directly to a final diagnosis. It will often need to be supplemented by one or more *deliberative* refinement steps which involve the explicit testing of selected suspects via tester agents. Refinement steps will typically generate further symptoms and trigger further analysis followed by further collaboration steps. The diagnostic process terminates either when a final diagnosis is achieved or all testing options have been exhausted.

The above walkthrough concentrates on the most straightforward sequence of operations within FDS. Of course processing can be more complicated. For example, if no analyser has registered a treatment rule for a particular symptom type and symptoms of this type are created, then the active collaborator will mark these symptoms as unrecognized and move on. Additional analysers may be deployed at any stage to 'sweep up' such leftovers. It is also possible that an analyser may be unable to process an analysis request. In these situations it will return a failure message rather than a new diagnosis and a different analyser (if available) will be selected.

If collaborations are initiated by two collaborators in the same time frame, then one will defer to the other as only one active collaborator is needed to lead the diagnostic thread within the home network. However, the other collaborators will still contribute to the refinement of diagnoses and so distribute the processing load across the network.

It is possible that analysis of different symptoms in different parts of the network can lead to contradictions within a working diagnosis – for example the same component may be rated as definitely faulty and also definitely clear, apparently at the same time. In such situations the working diagnosis will be reset and a recovery process initiated to explicitly test all suspects and components believed to be 'confirmed' faults.

Rather than triggering invocation of an analyser, a symptom can be associated with a treatment rule that indicates that a different (non-standard) sequence of diagnostic operations is required. In such cases a named diagnostic plan is loaded and executed. The current plan is suspended until the new plan has completed. Such plans can activate additional specialist agents to monitor particular aspects of system behaviour (for example traffic on the network) and create additional symptoms for subsequent analysis before de-activating these agents when they are no longer required.

As indicated above, the basic, generic FDS infrastructure may be dynamically extended in a number of respects: specifically by adding component specific test procedures, new analysers and other specialist agents and additional diagnostic plans.

4. THE ROLE OF OSGI

As others have identified, [4] [5], handling dynamism in systems is difficult. These projects, along with FDS, have used OSGi to support the dynamic deployment of program code and other resources. An OSGi framework provides a value add functional layer on top of a Java Virtual Machine for deployment and management of applications and can be regarded as providing a general purpose computing platform. OSGi originally stood for Open Services Gateway initiative. However the 'gateway' element of its name is now largely historic. The expectation when OSGi was originally conceived was that residential gateways in the home would, to an extent, be 'programmable' to allow third party software and services to run on them However this market has not (yet) developed in the home and in fact most of today's service providers seem determined to control the whole end to end network under the auspices of providing a managed environment for their services and so safeguard delivery of the required performance (e.g. current IPTV initiatives). In these circumstances there seems little room for third party services. Nevertheless OSGi seems to be taking off in the enterprise market as evidenced by its use with Spring [6] and Eclipse [7] and the general rise in interest in service oriented approaches to building systems [8].

Perhaps the key issue to consider with regard to the use of OSGi within the home is that if you want FDS-like capabilities then you may need to have OSGi installed on a substantial number of your devices to enable the dynamic deployment of test procedures and other diagnostic components. The use of OSGi on a single network node (the gateway for arguments sake) enables the dynamic download

of test procedures that can be used to test components on other home network nodes but only if they provide a remotely accessible interface and then purely on a 'black box' basis. Use of OSGi across all nodes enables a more direct form of 'glass box' testing with the potential for more detailed and accurate diagnoses as a result. Such testing would require internal knowledge of the target component so this effectively means that the component developer would have to write these test procedures. But they are packaged separately to the component itself so they may be deployed at a later date than the target component. For Java based components, the test procedure would only be able to access operations/data that the original developer was prepared to allow access to. However the annotations facility introduced in Java 5.0 offers the prospect of being able to label 'significant' variables and data structures within a class so that appropriate accessor methods can be automatically generated to facilitate inspection by the testing facility and assist future problem resolution.

5. CONCLUSIONS

This paper has provided an outline description of the FDS prototype which uses distributed software agents running on a home network of devices equipped with OSGi Service Platforms to diagnose operational problems and identify root causes. This might be considered as an unusual role for OSGi until one recognises the utility that OSGi adds to a standard Java Virtual Machine.

The open architecture of FDS, enabled by OSGi, allows new analyser agents to be introduced which understand the detail of new types of symptom created by new types of applications/devices. These new applications/devices can register new types of model which are understood by the new analysers. Clearly there is a degree of coupling between analysers, symptoms, applications and test procedures but coupling with the basic diagnostic infrastructure is minimal: the common ground between generic and problem specific parts is the understanding that systems are composed of components which have a status of suspect, fault or clear. This understanding constitutes the minimal ontology referred to in Section 2.

The intention is to continue research in this area at QMUL by developing analyser/symptom/model sets for new types of problem to further validate the FDS concepts. It is also hoped to make a version of the FDS code available in an open source form in due course.

For the future, if home networks are to evolve under the ownership of consumers rather than the ownership of service providers then open standards and facilities such as those provided by OSGi and FDS may well prove to be essential.

6. ACKNOWLEDGEMENTS

This paper was inspired by a conversation with Peter Kriens, software architect with aQute consultancy and renowned OSGi guru, after the "Service Platform for the Home, OSGi deja vu" technology application panel at CCNC 2008.

7. REFERENCES

[1] OSGi Alliance, OSGi Service Platform Core Specification, Release 4, August 2005. Available: [online] http://www.osgi.org

[2] P.C. Utton, "An Open Diagnostic Infrastructure for Future Home Networks", CCNC2008, Proceedings of the 5th IEEE Consumer Communications and Networking Conference, January 2008

[3] P. Utton, E. Scharf, "A fault diagnosis system for the connected home", IEEE Communications Magazine, vol 42, no. 11, pp128-134, November 2004. DOI: 10.1109/MCOM.2004.1362556.

[4] C. Escoffier, J. Bourcier, P. Lalanda, and J. Yu, "Towards a home application server", CCNC2008, Proceedings of the 5th IEEE Consumer Communications and Networking Conference, January 2008

[5] H. Cervantes, D Donsez and L. Touseau, "An Architecture Description Language for Dynamic Senssor-Based Applications", CCNC2008, Proceedings of the 5th IEEE Consumer Communications and Networking Conference, January 2008

[6] Spring Application Framework, Available: [online] http://www.springframework.org/osgi

[7] Eclipse Development Environment, Available: [online] http://www.eclipse.org/

[8] Z. Stojanovic and A. Dahanayake, , Service-Oriented Software System Engineering: Challenges and Practices, IGI Publishing, Dec 2004