A REVIEW OF COMPONENT INTERACTION APPROACHES FROM THE TESTING PERSPECTIVE

Angelina Espinoza-Limón, Juan Garbajosa
Technical University of Madrid (UPM)
Ctra. de Valencia, Km. 7. E-28031.Madrid, Spain
T.: (+34)-913365081/7885, Fax. (+34)-913367520
aespinoza@zipi.fi.upm.es, jgs@eui.upm.es

ABSTRACT

Complex systems are usually made of heterogeneous components, either hardware or software or both. Component interactions, mostly those unexpected, are a source of conflict, since one of the main concerns for system reliability and predictability is precisely this component interaction. This paper reviews a number of approaches, produced over an eight year period, to component interaction focussing on component interaction modelling, testing and testing coverage. Other topics such as component interaction observation and pure monitoring/visualization of component interactions are outlined.

1 INTRODUCTION

Complex systems are usually made of heterogeneous components, either hardware or software or both. An increasingly common trend in system and software engineering is assembly of components, either built on purpose of components of the shelf. The justification of this trend is outside of the scope of this paper. Component interactions, mostly those unexpected, are a source of conflict, since one the main concerns for system reliability is precisely this component interaction. As Williams and Probert outline in [11] a common the risk is magnified when, for each element in a system, there are a number of interchangeable components.

Therefore there a number of issues that are of interest to designers and testers, and having good approaches to achieve them must be considered as goals:

- Modelling of component interaction, in order to get a better understanding of the “fact” of interaction.
- Approaches to test applications in the presence of component interaction
- Measurements of the test coverage considering component interaction

Additionally two more complementary issues are related though they are just outlined within this paper:

- Component interaction observation intended for integration testing
- Pure monitoring/visualization of component interactions

While some approaches search for test generation, system validation relies heavily on the tester experience and knowledge. This work intends to provide new inputs to the system validation process, and indirectly to provide new requirements to environments such as the presented in [XX] This paper reviews a number of approaches to component interaction. These approaches have been produced over an eight year period. Some of the researchers reviewed in this work, study component interaction as a result of searching better methods for system integration and testing. In any case each one provides an interesting contribution to the above mentioned goals. Last section, Conclusions, provides a critical and comparative analysis of each approach contributions and to the two complementary issues.

2 THE JORGENSEN AND ERICKSON APPROACH

Jorgensen and Erickson study in [1] integration testing for Object oriented systems. Object orientation raised a number of issues, and brought up discussion, even when some of these topics were well known before.

Jorgensen and Erickson raise the attention about structure versus behaviour, and they introduce two constructs that are behavioural rather than structural to test OO system interactions. This is still true nowadays, since too often software engineers focus their attention mainly on structure, probably because behavioural issues are much more complex to model, more if modelling is intended to be consistent with that of structure. But its importance respect to system testing and correctness is proportional to its complexity.

Jorgensen and Erickson assert that the event-driven nature of object oriented systems forces a "declarative spirit", as opposed to imperative, on testing; and state that this is not evident at the unit level (as most object-oriented languages are imperative), but it is pronounced at the integration and system levels.

According to [1] functional decomposition has been the natural extension of system analysis, either prescriptive, such as development with functional languages (such as Lisp) or descriptive. This one has become more popular because is more tolerant of the way people work and exhibits several senses of hierarchy: levels of abstraction, lexical inclusion, information hiding and corresponding data structures. The problem is that functional decomposition has deep implications for testing as it emphasizes levels of testing, creates questions of integration order(bottom-up or top-down) and, last but not least, stress structure over behaviour. This is because the objective of integration, more in the context of the waterfall model, is to fit the units together into the functional decomposition tree. Thus the structure is the goal, not the behaviour.

Module interconnection usually focuses on interfaces, stressing functional decomposition, usually ready at preliminary design, and addresses structural issues, rather than behaviour. Therefore correct behaviour can be inferred from correct structure.

Both the event-driven nature and dynamic binding creates an indefiniteness that resembles of declarative programming. The shift to composition adds another dimension of difficulty to object oriented software: it is impossible to know the full set of "adjacent" objects with which a given object may be composed: two objects maybe correct but when composing errors may result.

Another issue is execution threads: for [1] authors a sequence of method executions linked by messages in the object network. Threads can be considered individually and in terms of interaction.

In [1] the concept of method/message path is proposed. It is a sequence of method executions linked by messages. It starts with a method and ends when it reaches a method which does not issue any messages of its own. Objects, as themselves, are not represented by this path. This is complementary to another concept that reflects the event-driven nature of object-oriented software: execution begins with an event that triggers the method-message sequence of a method/message path. Finally coverage can be obtained from this approach.

3 THE JIN AND OFFUTT APPROACH

Jin and Offut main objective in [3] and [4] was to present and approach for integration testing based on couplings between software components, and also provided some test coverage guidelines.

Therefore, from the Jorgensen and Erickson´s perspective presented in the former section, Jin and Offutt focus on structure rather than on behaviour. Following [3] coupling between two units reflects the interconnection between units; faults in one unit may affect the coupled unit. Coupling provides summary of information about the design and the structure of the software.

Jin and Offutt state that coupling is exactly where faults found during integration testing typically occur and therefore. They propose a new coupling-based testing technique, and assert that only three coupling types are needed with this objective, as opposed to a former work of Offut et al. [5], which in 1993 considered up to 12 levels. Coupling between two units increases the interconnections between the two units and increases the likelihood that a fault in one unit may affect others. The coupling levels are used to evaluate the complexity of software system designs. In [3] they define criteria that require that each connection between programmes units be covered. The three types of coupling they considered are:

- *Parameter coupling*: Refers to all parameter passing.
- *Shared data coupling*: Refers to procedures that refer to the same objects. This type combines non-local coupling and global coupling.
- *External device coupling*: Refers to procedures that both access the same external medium. It is analogous to external coupling.

The coupling-based criteria are based on the design and data structures of the program, and on the data flow between the program units: *Control flow graph* (CFG): Of a program is a directed graph that represents the structure of the program; n*odes:* Are basic blocks, and edges represent potential control flow from node to node; d*efinition (def):* It is an occurrence of a variable where a value is stored into memory (assignment, input, etc); u*se: It* is an occurrence of a variable where its value is accessed; c*aller:* It is a unit that invokes another unit, the *callee*; a*ctual parameter*: An actual parameter is in the caller, its value is assigned to a *formal parameter* in the callee; and, i*nterface:* Between two units is the mapping of actual to formal parameters.

Jin and Offutt propose in [3] a coupling-based testing scheme starting from a number of definitions. Coupling-based testing requires that the program executes from definitions of actual parameters through calls to uses of the formal parameters. Therefore, it is defined different coupling paths based on the three types of couplings.

The concept of testing path, used as the testing criteria, also defined in [3] helps us to get a better understanding of component interaction. Three types are proposed: *Parameter Coupling path*, *shared Data coupling path, and external Device coupling path*. Each of the three coupling criteria can be applied to four testing levels. Testing criteria levels are requires issues such that the set of paths executed by the test set T covers the call-site where A calls B; *or* that for each coupling-def of a variable x in A, the set of paths executed by the test set T contains a def-clear subpath from the coupling-def to at least one coupling-use of variable y in B.

Finally, coverage is defined in terms of a coupling graph. This coupling graph comes out as a result of the needed structural coverage analysis. A coupling graph is structured hierarchically and root node is the main program that calls a sequence of the other modules; this sequence of modules then becomes the next layer of the coupling graph, and they can call other sequences of modules and son on. The approach is describing coverage measurement schemes for each call-coupling, all-coupling-defs, all-coupling-uses, and all coupling paths.

4 THE LIU AND DASIEWICZ APPROACH

Liu and P. Dasiewicz in [6] describe a major issue in testing the integration of software components: the selection of tests to ensure that the components work together correctly. Their goal is to detect subtle interaction errors without duplicating the work performed in unit testing. The approach they use is to capture the assumptions made by each component about how other components should interact with it. The assumptions become new and formal test requirements that specify what test cases are needed to exercise the interactions. Interactions of components are model using a mathematical model that allows concurrency and synchronous communication. It is similar to creating test cases for conformance testing of communication protocols, but focussing in modelling software components, and errors in interactions.

For Liu and P. Dasiewicz, most problems in interaction come from the fact of using an object in an improper order (try to read a file before opening

it). Since a “protocol” describes what order things are expected to happen, then the interaction problems can be described as “violations of the correct protocol”.

The method proposed, the *CIT method, is a* method to test interaction errors called Component Interaction Testing (CIT). The method requires creating a model that shows how each component should be used, and a list of problematic sequences of interactions, called test requirements**,** used to select the test cases that will exercise the interactions or sequences of interactions.

The basis of the CIT model is to use a simple, formal model of component interactions, similar to finite state machines used in many OOA/D methods. It views interactions as exchanges of messages between concurrently executing objects. It is a simplified model of an object that applies well to all levels of abstraction.

The model can be defined in TTCN or PROMELA languages, to show abstract states and transitions. An example would be to specify the model for forwarding a telephone call. Then test requirements are defined, also in TTCN, as specified in [7], or PROMELA [8]. An example of a test requirement is “whatever happens when a user forwards an extension for a phone, the call has to terminate”. Then the authors of [6] generate test cases from the test requirements and the model. Model and test requirements graphs are combined representing interaction models and tests are generated.

The main limitation of the method is scalability, due the to state explosion problem. This problem is worse for software as many more components have to be considered, as outlined in [6].

In [9] Liu and P. Dasiewicz evolved their approach using a formal extension of UML, called ObjectState. This extension is made of an architectural description language (ADL), with a representation of connections and components in line with UML for Real Time; a behavioural language, with finite state machine representation of the behaviour of each component; and a data manipulation language, for detailed modelling of the effect of transitions on local component data. The formal basis for ObjectStore is provided by labelled transitions systems [10].

To write formal test requirements, as in the case of [6], ObjectStore has to be extended, according to [9]. Interactions, similarly to the Jorgensen and Erickson approach are defined with paths, to show sequencing. Authors

of [9] claim that only important interactions have to be specified. A usual model checking approach is then followed.

Finally test coverage is related to paths. It is noticeable that asynchronous communication and dynamic component creation, and dynamically allocated data structures are left outside, since they result in large models.

5 THE WILLIAMS AND PROBERT APPROACH

Williams and Probert in [11] present a metric that can be used to measure component interaction coverage of a system test configurations. The trade–off that a system tester faces is the thoroughness of test component configuration coverage, versus limited resources of time and expense.

A manufacturer of these system components would want to test as many of the potential system configurations as possible, to reduce the risk of interaction problems. The tester will have to select a subset of all possible configurations to use during testing.

Following Williams and Probert [11] there are two approaches for selecting a set of test configurations: one would be to decide in advance what the interaction test coverage criterion will be, and generate a set of test configurations directly to meet this criterion. In general this approach is used to generate a complete set of test configurations all at once. The second is to evaluate the interaction test coverage of any set of test configurations; in particular a set not specifically created to meet an interaction test coverage criterion. One case is, for instance, an operational profile.

The approach is based on the concept of *interaction element*. An interaction element consists in selecting a subset of parameters, and a number of specific values assigned to these parameters. Interaction degree is the size of the subsets of parameter values for which it wishes to detect unwanted interactions. This approach uses somehow the pair-wise coverage approach. A number of references related to this approach and the results of applying it can be found in [11]. The objective is that every interaction element be covered by a selected test configuration. In [11] interaction elements are used as test units for system interaction testing, as

one would use control flow branches or definition-use associations in other type of test coverage criteria.

Authors of [11] take the view that an unwanted interaction is usually not caused by the particular values of the entire set of parameters, but by the values of only a (hopefully, small) subset of parameters. The aim is to reduce the number of test configurations to the point where the testing can be conducted with a feasible cost in time and money, and still have a good probability of detecting unwanted system interactions. The coverage metric measures the coverage of potential interaction-degree-way interactions in a selected set of test configurations.

6 CONCLUSIONS

Jorgensen and Erickson make a thorough study of the topic, which they use to justify their approach. But the analysis they provide deals with topics simplified in the rest of the approaches, such as the case of the "event" nature. This could lead us to topics such as combining synchronous and asynchronous aspects within the same system. The approach makes a number of assumptions and, as Liu and Dasiewicz mention, Jorgensen and Erickson view interaction as exchange of messages between objects in an object oriented programme, but they do not provide guidelines to produce them, while Liu and Dasiewicz do.

Jorgensen and Erickson also point out that it is not sufficient to test structure but also behaviour. Being this obvious, structural approach seems more common as in the case of Jin and Offutt, and then in Williams and Probert, though in this last case using somehow pair-wise testing. A conclusion can be that behaviour issues are still a great concern from testing and to understand component interaction. Jin and Offut´s approach is purely structural. It presents some for real systems with a big component number.

In the Liu and Dasiewicz view, their formal method does not depend on any programming paradigm or packaging technique, and formal test requirements capture knowledge of problematic interactions. In this case the main limitation of the method is scalability, due to the state-explosion problem: the number of states to explore grows exponentially with the number and complexity of components.

In the Williams and Probert´s view, their approach, though structural, is more ambitious than that of Jin and Offut´s. It covers interactions not

necessarily coming from direct invocation. Profiles as a tool to identify interactions is also interesting. However their investigations are concentrated on only degree 2 interactions and the situation where each parameter has the same number of values. They comment that is needed to generalize these results to interaction coverage of higher degrees, and finding efficient ways of handling parameters with varying numbers of values.

Summarizing existing models focus considerably on structural properties as opposed to behavioural. Liu and Dasiewicz proposed a model that deals with behaviour but describing all interactions leads to state explosion, and specific models of interactions must be built for each application. Concerning behaviour it is worth while mentioning the efforts reported in [12] and [13]. These efforts address the observation of a number of issues that could be relevant from a testing point of view, and that would provide a view on the component interaction, and pure monitoring/visualization of component interactions. Both approaches may help to improve component interaction models.

Each author proposes tests coverage approaches metrics according to their approaches, but together with the efficacy of each metric it is not possible to forget that approaches analysed, mostly, are affected by combinatorial explosions, when applied to big systems.

7 REFERENCES

[1] Paul C. Jorgensen, Carl Erickson; Object-Oriented Integration Testing; communications of the ACM; v 37, n 9, pages 30-38, September 1994.

[2] Hong Zhu, Patrick A. V. Hall, John H. R. May; Software Unit Test Coverage and Adequacy; ACM Computing Surveys; v 29 n 4, 366-427, Dec 1997.

[3] Zhenyi Jin and Jeff Offutt; Coupling-based criteria for Integration Testing; Second IEEE International Conference on Engineering of Complex Computer Systems (ICECCS ’96); pages 10-17; Montreal, Canada, IEEE Comp. Soc. Press, October 1997.

[4] Zhenyi Jin and Jeff Offutt; Integration testing based on software couplings; In Proc. of the Tenth Annual Conference on Computer Assurance (COMPASS 94) pages 13:23, June 1995, IEEE Computer Soc. Press.

[5] A. J. Offut et al.; A software metric system for module coupling. The Journal of Systems and Software, 20(3):295-3008, March 1993.

[6] Wayne Liu, P. Dasiewicz; Formal Test Requirements for Component Interaction Testing Using Model-Checking.

[7] B. Baumgarten, A. Giessler; OSI Conformance Testing Methodology and TTCN; North Holland; 1994.

[8] G. J. Holzmann; Design and Validation of Computer Protocols; Prentice Hall, 1991.

[9] Wayne Liu and P. Dasiewicz; Formal Test Requirements for Component Interactions; Proceeding of the 1999 IEEE Canadian Conference on Electrical and Computer Engineering; pages 295-299; Shaw Conference Center, Edmonton, Alberta, Canada, May 1999.

[10] R. Millner; Communication and Concurrency; Prentice-Hall; 1989.

[11] Alan W. Williams and Robert L. Probert; A Measure for Component Interaction Test Coverage; Computer Systems and Applications, ACS/IEEE International Conference on. pages 304-311, 2001.

[12] Zhu, H. and He, X., A Methodology of Component Integration Testing, to appear in Testing COTS Components and COTS-based Systems; Sami Beydeda (ed.), Springer, 2004-10-29

[13] Harold Batteram, Wim Hellenthal, Willem Romijn, Andreas Hoffmann, Axel Rennoch, Alain Vouffo; Implementation of an Open Source Toolset for CCM Components and Systems Testing; Testcom 2004

[14] Pedro P. Alarcón, Juan Garbajosa, Belén Magro, Alberto Crespo; Automated Support For Requirements And Validation Tests As Development Drivers; System Testing and Validation Workshop SV04, published by Fraunhofer Institute, 2004.