First International ICSE Workshop

TESTING DISTRIBUTED COMPONENT-BASED SYSTEMS

Los Angeles, California, USA
17 May 1999

Organization

Andreas Ulrich, Gunther Chrobok-Diening, Peter Zimmerer
Siemens AG, ZT SE 1
81739 Munich / Germany

Formalizing integration test strategies for distributed systems

Frédéric Mercier†, Pascale Le Gall†, Antonia Bertolino‡

†: LaMI, Université d'Evry
Cours Monseigneur Romero
91025 Evry Cedex, France

{mercier,legall}@lami.univ-evry.fr

‡: Istituto di Elaborazione della Informazione
CNR, Via S. Maria, 46
56126 Pisa, Itatly

bertolino@iei.pi.cnr.it

Abstract: *This work focuses on integration testing. For centralized systems with hierarchical structure, conventional approaches to systematically cover the constituent modules in incremental way, following a top-down or a bottom-up order, have been used. Such approaches are no longer adequate for integration testing of complex, distributed systems. We introduce here a novel approach, based on a dedicated specification formalism, called* information spaces. *Such approach supports a systematic decomposition of the system into sub-systems relevant for specified testing objectives, driven by both structure and information processing. By using a slicing technique over information spaces, we are able to obtain* test classes, *which intuitively define* what and where to test *in order to check a given information flow between system components.*

Keywords: *integration testing, distributed systems, test class*

1 Introduction

This research involves the integration testing of complex, distributed systems. Although probably the most extensively used of the three testing stages conventionally associated with the V development model, integration testing still remains the least rigorous and the least systematic phase.

Upstream, many works in the literature have early addressed *unit testing* and produced numerous empirical observations and exact results, allowing for the emergence of systematic methods [Bei90, Jor95]. However, the number of units to test in modern systems has dramatically increased. So to proportionally increase the cost of a generalized and demanding unit testing to prohibitive levels. To such a degree that the deployment of advanced techniques for this kind of testing is in practice reserved to those few system's elements guaranteeing a critical function.

Downstream, system testing: this does not yield the rigorousness of the unit testing tool box, however its design is closer to the final use of the system and thus seems to be more intuitive. In practice, in addition to a set of high abstract level test cases (usually designed in partnership with the customer), the project manager relies on experts of the system to deepen and vary the test scenarii in order to guarantee the expected quality level.

Between unit testing, too expensive to be made systematically, and system testing, intervening too late in the development cycle, integration testing is the verification phase likely to compensate the weakness of a partial unit testing and to incrementally guarantee system correctness against the specifications, avoiding the pitfalls of a big-bang effect.

In this paper, we first overview conventional approaches to integration testing and speculate why these ones no longer fit the needs of

modern distributed systems. Then in the next section, we introduce a rigorous approach we are currently working on aimed at remedying identified problems.

1.1 Conventional approaches

The culture relative to integration testing basically formed around the very classical case of sequential systems. A large variety of methods, based on a hierarchical decomposition of the system, can be generally derived from the two principles of common sense that are top-down and bottom-up approaches (figure 1).

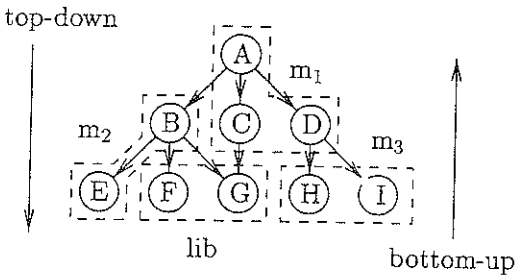

Figure 1: *Hierarchical decomposition*

Usually, a hierarchical decomposition (figure 2) is a set of elements with a transitive strict order relation $<$ (which is the transitive closure of "$\rightarrow$" in figure 1), having a single minimum element ($\forall X \in \{B, C, \ldots, I\}, A < X$) and possibly some local maxima ($B < G$ and $C < G$).

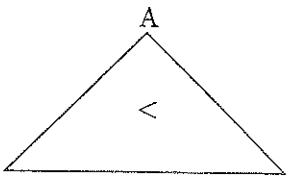

Figure 2: *Synthetic representation*

When integration becomes effective, this hierarchical decomposition corresponds to an acyclic condensation of the *call graph* of system's functions[1]. The interdependent functions being gathered, conceptually and physically, in the same functional unit. However this relatively fine level of conception arises late in development cycle, so that it cannot be reasonably the only criterion considered in order to define the integration strategy. At a more abstract level, it is possible to take advantage of a *functional decomposition* of the system. This is another form of hierarchical decomposition, according to which the call graph is first divided into functional areas conceptually associated to modules (figure 1) and then the integration of these last is planned accordingly.

The functional decomposition also lays on a hierarchical structure which can be qualified as heterogeneous because of the different nature of the elements it contains. The following ones can be more especially identified:

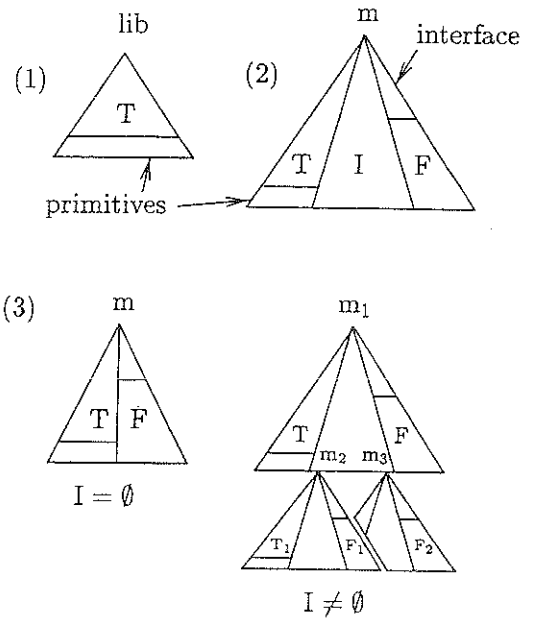

Figure 3: *Functional decomposition*

[1] $f < g$ mean then that the function f (will) call the function g in its code.

- user defined *libraries*, denoted T in the figures 3-1 and 3-2. These libraries induce the specification of abstract data types (ADT) and related primitives. The notion of ADT is especially well suited for applying algebraic methods of specification and test [AGM96, BBG97] just as it supplies a helpful guideline to decompose system into sub-systems in such a hierarchical way.

- partial abstractions of the *call graph*, denoted F in the figure 3-2. These ones correspond to the description of the behavior of a module in terms of functions and procedures. The composition and the refinement of these partial abstractions produce the call graph issued from the design phase (figure 4) on which the integration strategy is articulated.

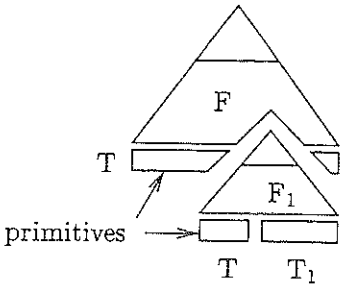

Figure 4: *Call graph*

- a set of *modules* denoted I in the figure 3-2. This one describes the importation architecture of modules. It is however possible to distinguish (figure 3-3) final modules which do not import any module ($I = \emptyset$), from non final modules ($I \neq \emptyset$) importing a set of modules to ensure their own functionalities (F).

The main correlative property between the functional decomposition and the call graph is that a function f associated to a module m ($m < f$) can use a function g ($f < g$) only if (1) g is also associated to m, or (2) g is associated to a module m' which is imported by m ($m < m'$). What can be expressed in a more concise manner as follow:

$$f < g \Rightarrow \begin{array}{l} m < f \\ m < g \end{array} \quad \text{or} \quad \begin{array}{l} m < f \\ m' < g \\ m < m' \end{array}$$

In general, the choice of an integration strategy is not only conditioned by the checking of the preceding property, but also by certain characteristics of the elements to be integrated. We limit ourselves here to the enumeration of some traditional cases without going into the details. Thus, the strategy of integration will be more particularly:

- top-down style in case of early validation of high level functions or behaviors (for instance F in figure 3-3);

- bottom-up style in case of incremental integration of reused modules. It allows for an early check point on the adequacy or the inadequacy of the reuse.

In practice, the two approaches are rarely applied purely, but mixed strategies thereof are generally devised accordingly to schedule requirements on one side, and relative criticality of building modules (more risky ones ought to be verified earlier).

1.2 Distributed context

The integration techniques we have briefly exposed all rely on the assumption that there exist one or several correlated hierarchical decompositions. With respect to which it is possible to plan the integration process (functional decomposition) and to actually perform the integration (call graph).

The study of distributed systems reveals two new dimensions which affect the verification of this assumption.

Indeed, even if it remains practically possible to establish a functional decomposition of a distributed system (figure 5), one of the immediate effects of the distribution is that the base of this decomposition still expresses the membership of elements to the system, but does not explicitly corresponds to any hierarchical structure between them.

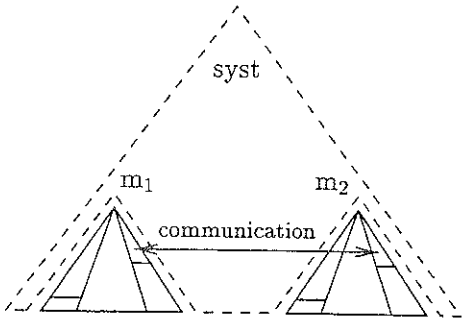

Figure 5: *Distributed system*

Another characteristic related to distributed systems is that the communication architecture, which is instinctively perceived as an equivalent to the call graph, does not maintain the same hierarchical properties as for conventional systems.

We can thus conclude that the techniques classically used do not bring any general solution to integration testing on this type of architecture. We thus should conceive an approach based on a representation alternative to hierarchical decomposition, in order to allow for systematic integration testing in distributed contexts. The main guideline underlying our approach is to provide functional decomposition of distributed systems with additional ordered elements on which integration testing strategies could be based.

Intuitively, these elements will be "messages" transmitted between sub-systems. More precisely, these messages will be formalized by information terms expressing some dependence links between different exchanged messages on the distributed system. Our principal motivation is that these links are correlated to the global system functionalities.

The testing integration strategy we propose closely follows the underlying structure of message passing sequence. In that sense we will retrieve some of the intuitions that guide integration testing in the context of hierarchical structures. In the next section we formalize distributed systems under the form of *information spaces*, which simply figure sub-systems exchanging terms of information. And then, based on this description, we present our approach to integration testing.

2 Information spaces

2.1 Capturing obviousness

There is a general agreement that integration testing, much more than any other test stage, must gather correlatively both structural and functional testing criteria.

In the case of distributed systems, the structural part may naturally be supported by the communication topology of the system. To stimulate the system and to observe which communication channels are actually used may ensure that some information travels throughout the communication topology. But this observation would carry nothing more with regards to any specific functionality.

The functional counter-part states a major difference with centralized systems. Indeed, distributed systems hold by distributed functionalities. No top function which solves a given problem using sub-functions exists, but communication channels transferring information, which is the solution to a problem, between specific sub-systems. Moreover, the

problem data are submitted to the system via one or several stimulations to possibly several sub-systems.

More than ever, what we want to test conditions what we describe. To know if a system correctly solves the type of problem it is supposed to, we need at least to know *what is the problem type, where to supply input data and where to observe the result(s).*

For instance, in figure 6 the communication nodes A,B, ... ,G compose a sub-system, part of a bigger one (dashed items). This sub-system solves a problem standing as a precise relation between the information $\{a, b, c\}$ and the information $f(h(b), g(a, c))$.

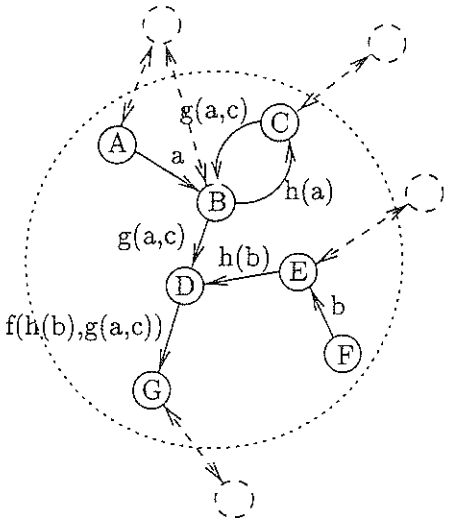

Figure 6: *Distributed functionalities*

The problem data $\{a, b, c\}$ may be impulsed from the components respectively A,F and C. The problem solution for $\{a, b, c\}$ standing in the information $f(h(b), g(a, c))$ can be observed on the component G. Note that at the abstraction level we are, this intuitive test description might be understood as a scenario, to be instantiated by specific values for $\{a, b, c\}$.

The main motivation we have is to be systematically able, given the formal description of a distributed system, to deduce such relevant test role as well as any scenario devoted to test a specific information term. A strong coverage criteria being then to test every information terms everywhere it is possible.

In the following we define the funding notions of *architecture topology* and *event expressions* before introducing the formalism of *information space* on which our approach relies.

2.1.1 Architecture topology

The most intuitive representation of a complex system made of interacting components remains that of an oriented graph. So, we introduce the underlying topological structure of the system as being an oriented graph $G =< N, A >$ where N is a set of nodes corresponding to the components of the described system and $A \in N \times N$ is a set of arcs corresponding to the communication channels between components. Such a graph is called an *architecture* and allows us to describe a constrained space of communication.

2.1.2 Event expression

The information associated to any system is basically of two kinds: primary or basic information, like that derived from physical measurements, or more generally any piece of information indivisible at the level of abstraction we are (as a,b and c in figure 6). The second kind corresponds to the information that is the one of a computation and thus contributes to carry out the system's functionalities. These information processings are formalized by functions (as f,g and h in figure 6). Finally, we define the set I of *information terms* containing all the basic informations and the all terms recursively derivable by functions in F.

To express the computation occurring between a basic information and a synthetised

one, we define a language of event expressions. With the conventions that t denotes an arbitrary information term and e_1 and e_2 are arbitrary event expressions, the set E of all event expressions results from all the well-formed combinations built on the following elementary compositions given :

$i[t]$ (resp. $o[t]$) holds for the input (resp. output) event of the information t.

$e_1.e_2$ holds for a sequence of events.

$e_1|e_2$ holds for an exclusive disjunction of events. This operator expresses that either e_1 or e_2 are expected to occur but not the two.

$e_1;e_2$ holds for a set of independent events. Each of e_1 or e_2 car occur independently of the other.

Our event expression language is actually extended with some other useful connectors. Nevertheless these ones can be reduced to expressions only built on the four language constructors defined above.

2.1.3 Information spaces

We can finally define information spaces that gather the preceding concepts of architecture topology, information terms and event expressions. Precisely, in an information space, the behaviour of each component of the architecture is described by an event expression and each communication channel is labelled with carried information terms. So, an information space on an architecture $G =< N, A >$ and a set I of information terms is described by means of a couple $\vec{\mu} = (\mu_A, \mu_N)$ composed of mapping functions $\mu_A : A \rightarrow I$ and $\mu_N : N \rightarrow E$.

The behaviour of a component is a loop on the associated event expression which figures both the internal semantic of the component and the way it perceives its neighborhood's reaction. For instance the event expression associated to the component D in our example is :

$$(i[g(a,c)]\&i[h(b)]).o[f(h(b),g(a,c))]$$

where & is a connector standing for

$$i[g(a,c)].i[h(b)] \mid i[h(b)].i[g(a.c)]$$

The interested reader may refer to [MBGB98] for a more concrete instantiation of information spaces on a Tele-Remote Medical Care System case study, which is also treated under a slightly different perspective in [BIMR98].

2.2 Test support

In this section we formalize our view of integration testing developed in section 2.1 from an intuitive point of view. We also give an exhaustive set of integration test scenarii for the illustrative example given in figure 6.

2.2.1 Information slices

Deriving from an information space, which implicitly defines the global event schedule over the system, all and only the part of the specification relevant to a specific information corresponds to extracting a *slice* of the information space. Given an input event $i[t]$ and its location, an *information slice* is an information space segment whose behaviour description is identical to the global information space with respect to $i[t]$.

2.2.2 Test classes

We now introduce *test classes* that constitute a first step towards how to test information slices. Given an information slice, defining a test class corresponds to assigning specific test

roles to some of the slice components (see figure 7). Basically, we distinguish between those components from which test input data are launched (the launchers) and those other ones on which the test oracle could be located.

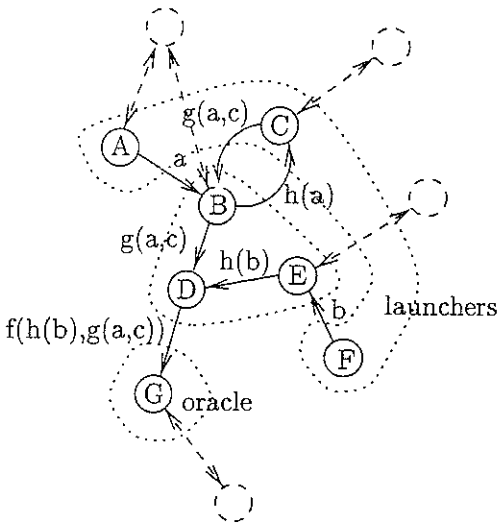

Figure 7: *A "$f(h(b), g(a, c))$" test class*

What we do is a partition of the set of components related to the slice. Oracles and associated launchers are the border of the slice, since the launchers produce basic information necessary to build the information tested on the oracle. The other components in the slice form the part of the system that is actually tested to verify their interactions (here, components B,D and E).

The table 1 gives an extended set of scenarii related to our abstract example. These results figure all possible scenarii to test each information. That is to say, testing each information at every possible verification point (oracle) by considering every possible associated stimulation point (launchers).

Info.	oracle	launchers
a	B	$\{A\}$
b	E	$\{F\}$
$h(a)$	C	$\{B\}$
$h(b)$	D	$\{E\}$
$h(b)$	D	$\{F\}$
$g(a,c)$	B	$\{C\}$
$g(a,c)$	B	$\{B;C\}$
$g(a,c)$	B	$\{A;C\}$
$g(a,c)$	D	$\{B\}$
$g(a,c)$	D	$\{C\}$
$g(a,c)$	D	$\{B;C\}$
$g(a,c)$	D	$\{A;C\}$
$f(\ldots)$	G	$\{D\}$
$f(\ldots)$	G	$\{B;E\}$
$f(\ldots)$	G	$\ldots$
$f(\ldots)$	G	$\{A,F,C\}$

Table 1: *Example test classes*

3 Looking forward

Our work is clearly needs-driven. In the context of distributed systems the testing needs are "how to test" as well as "what to test". We have chosen to develop our strategy from a deep understanding of a specific "what" (distributed functionalities) before trying to find or to identify an acceptable related "how".

Supported by information spaces and test classes, we have then presented an alternative integration testing approach based on the notion of information processing by interconnected components.

From this point we plan to relax some of the simplifying hypothesis holding over current definition of information spaces. In particular we would like to express that the topology may possibly vary in some pre-defined dimensions. The test classes would then address some topological regularity criteria like "Test ... with 1 (resp. 2,3, ...) component of X type". Some

of the features of information space may be then subject to some deep modifications that we hope to be fruitful.

Correlatively to the extension of the description capabilities of information spaces, we are trying to find a more concrete specification formalism capable to capture some specific data-wise aspects of distributed systems. The idea is obviously to make the two formalisms efficiently cooperating in order to fill test classes with relevant test cases.

Acknowledgments

This work was partly supported by the ESPRIT-IV Working Group 22704 ASPIRE, the ESPRIT-IV Working Group 23531 FIREworks.

References

[AGM96] A. Arnould, P. Le Gall, and B. Marre. Dynamic testing from bounded data type specifications. In *Dependable Computing - EDCC - 2*, Taormina Italy, October 1996. LNCS 1150.

[BBG97] G. Bernot, L. Bouaziz, and P. Le Gall. A theory of probabilistic functional testing. In *ICSE'97 Proc.*, Boston, Massachusetts USA, 1997.

[Bei90] B. Beizer. *Software Testing Techniques, Second Edition.* Van Nostrand Reinhold, New York, 1990.

[BIMR98] S. Balsamo, P. Inverardi, C. Mangano, and F. Russo. Performance evaluation of a software architecture. In *IEEE Proc. IWSSD-9*, April 1998.

[Jor95] P. C. Jorgensen. *Software Testing, a Craftsman's Approach.* CRC Press, 1995.

[MBGB98] F. Mercier, A. Bertolino, P. Le Gall, and G. Bernot. A systematic approach for integration testing of complex systems. In *ICSEA '98 Proceedings*, Paris, December 1998.