Visual Programming of Fault-Tolerant Distributed Applications*

B. Muganga F. Pacull K.R. Mazouni A.-D. Wolff
Laboratoire de Systèmes d'Exploitation, Département d'Informatique
École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne (Switzerland)
e-mail: muganga@lse.epfl.ch

Abstract

The design of fault-tolerant distributed applications is a complex task. In addition to application functionalities, the programmer must consider issues related to both replication and distribution for every application component concerned with fault-tolerance.

This paper describes an approach which combines two environments (SystemSpecs and GARF) so as to: (1) graphically design applications using high level Petri nets and (2) discharge the programmer of fault-tolerance issues.

1 Introduction

The design of fault-tolerant distributed applications is a complex task. Considering replication and distribution at design time complicates the programmer's work and may distract him from his main task: implementing application functionalities. Moreover, making a component fault-tolerant is independent of its functionality and may be automatically performed.

This paper presents the integration of two environments: GARF [1, 2] and SystemSpecs [3] in order to automatically build fault-tolerant applications. For each application, we consider two orthogonal aspects: (1) the *functional aspect*, i.e. what the application is supposed to do and (2) the *behavioural aspect*, i.e. the behaviour of each component in the presence of failures. The functional aspect is application specific and under the programmer's responsibility. In contrast, the behavioural aspect is generic and provided by GARF libraries.

Both the functional aspect code and the fault-tolerance specification needed by GARF are automatically generated from a SystemSpecs description of the application. SystemSpecs allows the design of applications using SPECS nets (high level Petri nets) and generates Smalltalk objects as output.

We have defined an extension of SPECS nets called CS-nets expressing the client/server model on which object interactions are based. Thanks to this model, fault-tolerance specification can be graphically associated to the initial Petri net.

*Research funded by the Fonds National Suisse de la Recherche Scientifique as part of the Programme Prioritaire Informatique (SPP IF) under project number 5003-034344.

This approach is of great interest since (1) the formalism of Petri nets permits easily both to simulate the application and to check its properties [4] (2) the fault-tolerant application is generated from a validated Petri net.

Section 2 shows how CS-nets are defined from Petri nets. Section 3 presents the fault-tolerant specification step which is a refinement of the application net. Section 4 discusses how the GARF environment proceeds for binding the behavioural aspect to each application object. Finally, Section 5 concludes the paper.

2 From Petri nets to CS-nets

Using Petri nets is now a well-known way to graphically specify and validate applications [4]. SystemSpecs [3] considers high level Petri nets that in particular allow to associate a transition with a complex action (*data transformation*) described in *SpecsLingua*, a pascal-like programming language and to manipulate tokens with complex structures.

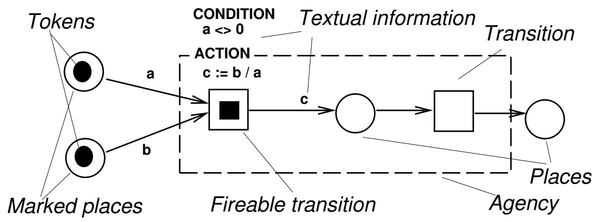

Figure 1: Example of a SPECS net with an agency

Figure 1 gives an example of a simple SPECS net with two entry places. The code associated with the transition specifies that if the condition ($a <> 0$) holds then the transition is fireable and the corresponding action ($c := b/a$) will be performed. Moreover, it is possible to group several places and transitions into a subnet called *agency*[1] (cf. Figure 1).

Distribution is possible if the application is correctly decomposed into components in accordance with the client-server model. If two components share information stored in main memory, they should be located on the same site. In contrast, if they interact

[1] an agency can also contain other agencies.

in a client-server manner they can be distributed over different computers.

When an application is written using an object-oriented language, this decomposition is implicit since objects naturally express the client-server model. With the Petri nets approach, nothing prevents the programmer from designing a monolithic application that cannot be distributed. Thus, the client-server interaction should be considered at the Petri net level.

A client-server interaction between two components is characterized by an invocation scheme between the client and the server. A component is modelled as a special category of agencies called CS-nets (for client/server nets). CS-nets have special places which allow to express the client-server interaction model. They may act as client, server or both since a server can also be the client of another server.

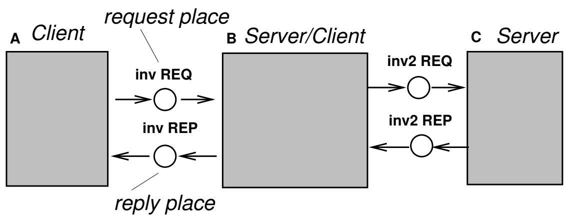

Figure 2: Client-server interactions

Figure 2 shows the basic scheme for modelling invocations between CS-nets. Every method offered by a server is always modelled by a couple of places (REQ and REP) respectively for *request* and *reply place*. Hence, the invocation mechanism needs two new places, shared among the client and the server and located outside the components. A client performs a request by marking the *request place* with a token. It receives the reply as a token put in the *reply place*.

3 Fault-tolerance specification

Fault-tolerance is achieved by replicating and distributing critical components of the application [5]. The service offered by a component is guaranteed as long as there is at least one of its replica available in the system. The whole application is able to carry on despite the failure of some hosts of the underlying distributed system.

Once the application is decomposed into CS-nets, the programmer has to consider the specification of fault-tolerance. He adds to the textual part of each CS-net information describing its behaviour in the presence of failures. Figure 3 gives an example of fault-tolerance specification.

By specifying a *replication rate* (e.g. NbReplicas = n), the programmer defines that the corresponding component will be replicated n times. As a result, it will tolerate n-1 failures. However, if the programmer wishes to control the *location* of each replica, he has to specify a list of hosts (e.g. Nodes = {lsesun8.epfl.ch,lsesun10.epfl.ch}).

The choice of the *replication policy* is important as it specifies how replicas interact [2], it depends on the

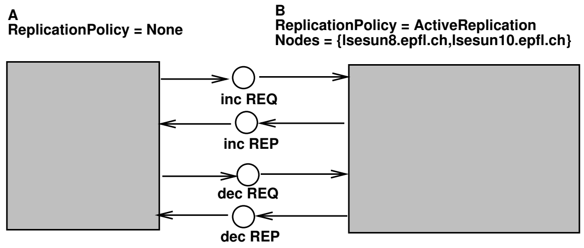

Figure 3: Fault-tolerance description of a CS-net

property the programmer wants to guarantee for a given component. For instance, if the server maintains an internal state, it must ensure that this value is accurate in any case. Hence, *active replication* is chosen. If it performs computations based on the parameters given as input: no internal state is changed and *passive replication* can be used.

4 Fault-tolerance generation

The GARF system is an object oriented environment aimed to simplify the programming of fault tolerant distributed applications. GARF provides libraries of fault tolerant classes which are used by programmers in their applications.

The GARF system assumes a model in which every fault tolerant object is made of two distinct objects: a *data object*, generated from the CS-nets by SystemSpecs (section 2) and a *behavioural object*, from the fault-tolerance specification (section 3). Data objects are application specific because they express application functionalities without caring about distribution or fault tolerance. Behavioural objects are provided by the GARF libraries and bound to data objects according to the fault-tolerance specification. They implement generic mechanisms related to distribution and fault tolerance.

A behavioural object intercepts all the invocations sent to or received by its associated data object. Figure 4 shows how an invocation of method m on server B by client A (Figure 4-(a)), is transformed by the GARF system (Figure 4-(b)).

A behavioural object is actually implemented using two objects: an *encapsulator* and a *mailer*, as shown in Figure 4 (b). Each data object has a dedicated encapsulator located on the same node. A copy of the mailer resides on every node where at least one *potential client* is located. Only encapsulators can be invoked across the network and the corresponding mailers alone can do it. The location of a data object is therefore transparent to its clients.

In Figure 4, the server B has the following fault-tolerance specification:
ReplicationPolicy = ActiveReplication;
Nodes={lsesun8.epfl.ch,lsesun10.epfl.ch}.
The behaviour corresponding to ActiveReplication is built from an encapsulator of class ActiveReplica and a mailer of class Abcast. An ActiveReplica encapsulator is located on each node where a data object replica resides: lsesun8.epfl.ch and lsesun10.epfl.ch in this case.

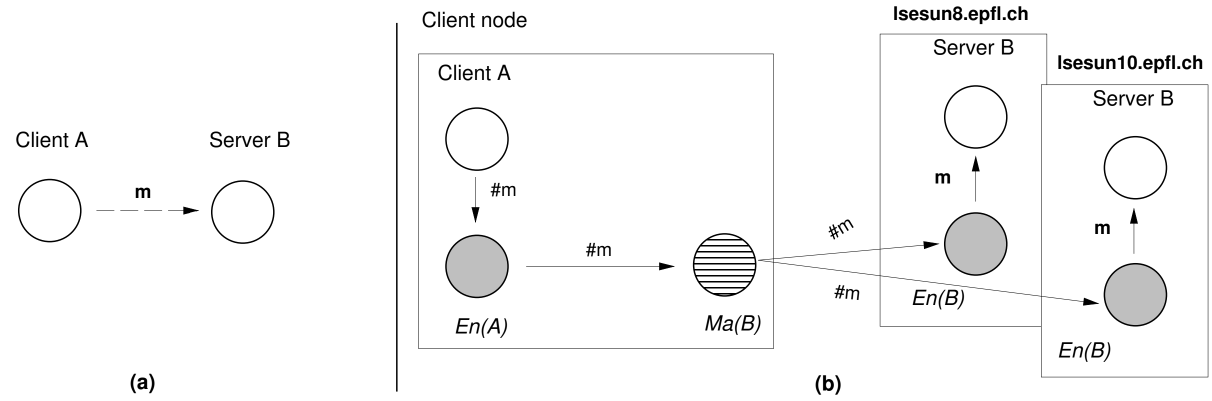

Figure 4: Behaviour, mailer and encapsulator

It holds a reference to the local data object replica and belongs to an object group. On client nodes, Abcast mailers act as proxies of the encapsulator group. Mailers of class Abcast multicast invocations to the encapsulator group and ensure that every group member will receive the invocations in a total order.

Therefore, the invocation of method m on server B by client A, is transformed by the GARF system into an invocation on the client encapsulator *En(A)*. The latter transmits the reified invocation #m to *Ma(B)*, the server mailer which calls each server encapsulator in order to forward #m to the server nodes. At last, each encapsulator *En(B)* invokes m on their local replica of server B. Once method m has been performed, the result returns to A by the reverse path.

4.1 Implementation

Among the Smalltalk classes generated by SystemSpecs we find the instance methods corresponding to the functional aspect of an object of this class and the class method garfnew which is invoked at instantiation time. This method contains information concerning the behavioural aspect: the number of replicas, their location and the behavioural objects (mailer and encapsulator) associated with every new instance.

Both classes ActiveReplica (the encapsulator) and Abcast (the mailer) are provided by GARF and implemented in Smalltalk [6] on top of the *Dependable Object Toolkit* (DOT), the layer of the GARF system implementing replication management and group communications at the object level. DOT[1] is built on top of the Isis toolkit [7, 5] which provides support for both membership management and multicast communication of Unix process groups.

The final result is a collection of objects (data and behavioural) which compose a fault-tolerant application that runs on top of the GARF run time.

5 Conclusion

Designing an application using Petri nets is not new. The originality of our approach using SystemSpecs and GARF is to add a new dimension to graphical design of applications: support for fault-tolerance.

SystemSpecs flexibility at the graphical interface level and the output format allowed to quickly combine it with the GARF environment without reconsidering the initial concepts of GARF.

The advantage of our prototype is immediate since the target fault-tolerant application is automatically generated from a specification tested and validated with the powerful Petri nets formalism.

Acknowledgments

We are especially grateful to R. Mattmann of IvyTeam for his help concerning SystemSpecs.

References

[1] B. Garbinato, R. Guerraoui, and K. Mazouni, "Implementation of the GARF replicated object platform," *(to appear in Distributed Systems Engineering Journal)*, 1995.

[2] B. Garbinato, R. Guerraoui, and K. Mazouni, "Distributed programming in GARF," in *Object Based Distributed Programming* (R. Guerraoui, O. Nierstrasz, and M. Riveill, eds.), no. 901 in Lecture Notes in Computer Science, Springer Verlag, 1994.

[3] Ivy Team, *SystemSpecs TM 2.1 - Reference Manual*, Oct. 1993.

[4] W. Reisig, *Petri nets – an introduction*, vol. 4 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, 1985.

[5] K. Birman, "The process group approach to reliable distributed computing," *Communications ACM*, Dec. 1993.

[6] A. Goldberg and A. Robson, *SMALLTALK-80: The Language and its Implementation.* Addison Wesley, 1983.

[7] K. Birman, "Replication and fault-tolerance in the ISIS system," in *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pp. 79–86, Dec. 1985.