

# Reaction Graphs for the Testing and Analysis of Software Architectures

**Antonia Bertolino**

Istituto di Elaborazione della Informazione  
C.N.R.  
Via S. Maria, 46  
Pisa, Italy  
bertolino@iei.pi.cnr.it

**Paola Inverardi**

Dip. Di Mat. Pura ed Applicata  
Universita' dell'Aquila  
Via Vetoio, Localita' Coppito  
L'Aquila, Italy  
inverard@univaq.it

## 1 Introduction

In previous papers we have addressed the problem of deriving from an architectural description pieces of information useful to drive testing [2, 3]. Our approach is based on the use of the CHAM formalism for architectural descriptions, in the way proposed by [5]. In this paper we provide a new method for the analysis and testing of a software architecture. It consists of modeling a software architecture (SA) described in CHAM as a *reaction* graph. This graph supports both the automatic derivation of selective test plans and the consistency analysis of the SA specification.

In this paper we illustrate the method on a case study which is a teleservice and remote medical care system (TRMC).

The paper is organized as follows:

In the next section we present the notion of reaction graph, and discuss its application to testing. Section 3 presents our case study, the TRMC system, and gives the CHAM description of its software architecture. Section 4 applies the reaction graph method to the case study and discusses possible testing criteria.

## 2 Reaction Graphs and their application

For the sake of brevity we omit background material about the CHAM formalism and its use for the description of Software Architectures. We refer to [5].

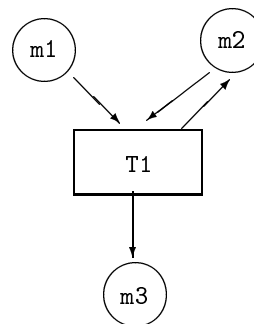
Starting from the CHAM description of a SA we derive a synthetic description of its dynamic potentiality in the form of a bipartite graph, called a *reaction* graph. A bipartite graph is a graph whose vertices can be partitioned into two disjoint sets, such that no two vertices in the same set are adjacent.

A reaction graph  $R = \{M, \mathcal{T}, \mathcal{R}\}$  has nodes of two kinds: molecule nodes  $m \in M$  and transition nodes  $T \in \mathcal{T}$ . An arc  $r \in R$  going from molecule  $m_i$  to transition  $T_j$  means that  $T_j$  needs  $m_i$  to be activated ( $m_i$  is in the left-hand side of transition  $T_j$ ); an arc going from transition  $T_j$  to molecule  $m_i$  means that  $T_j$  produces  $m_i$  ( $m_i$  is in the right-hand side of  $T_j$ ). Thus for instance the following

transition rule:

$$T_1 \equiv m_1, m_2 \longrightarrow m_2, m_3$$

would be modelled by the graph below:



Keeping with the chemical metaphore, the reaction graph depicts in graphical form the chemical transformations which can occur among the molecules in the CHAM. Looking at the system which is modelled by the CHAM, the reaction graph models the reactive behavior of the system, i.e. how it can react to events.

Thus on one side this description is compact since its dimension is directly bound to the length of the CHAM description, ultimately to the number of transition rules. On the other side it is possible to dynamically interpret reaction graphs to obtain all the possible system behaviors, as outlined below. Interestingly, reaction graphs exhibit a strong resemblance with Petri Nets, that we will better exploit and discuss in future papers. At any given instant during the life of the system, some molecules are activated and others are deactivated. A transition can fire if all the arcs entering it come from activated molecules. After a transition has fired, its descendant molecules are activated, while parent molecules are deactivated. In some cases, a transition can have an arc coming from and an arc going to the same molecule: this represents a transition which regenerates a molecule, like molecule  $m_2$  in the above figure.

At start, only the initial molecules are activated; at any instant, any transition that descends only from initial molecules can (nondeterministically) be taken. By analogy with Petri Nets we can say that a transition consumes the tokens of entering molecules, and passes a token to all descendant molecules. Let us denote by the triple  $\{S_i^s, T_i, S_i^f\}$  a transition  $T_i$  which is fireable in the starting solution  $S_i^s$  and makes the system evolve to final solution  $S_i^f$ . A path on a Reaction Graph, also called a *reaction path*, is defined as a sequence of  $k + 1$  transitions  $\{S_0^s, T_0, S_0^f, \dots, S_i^s, T_i, S_i^f, \dots, S_k^s, T_k, S_k^f\}$ , where  $S_0^i = S_0$  is the initial solution, and for  $i \in [0, k - 1]$ ,  $S_i^f = S_{i+1}^s$ , and if  $T_i \equiv m_1^s, \dots, m_n^s \rightarrow m_1^f, \dots, m_m^f$  then  $(m_1^s, \dots, m_n^s) \in S_i^s$  and  $(m_1^f, \dots, m_m^f) \in S_i^f$ .

For brevity we will denote a path by the list of its transitions  $\{T_0, \dots, T_i, \dots, T_k\}$ . A CHAM specification can include a molecule which is not terminal, as for instance in (this is one of the transition for the CHAM used as an example in Section 3):

$$T_5 \equiv m \diamond \mathbf{Router}, o(\mathit{clock}) \diamond \mathbf{Timer} \longrightarrow m \diamond \mathbf{Router}$$

In these cases the non terminal molecule  $m \diamond \mathbf{Router}$  is still one molecule node in the reaction graph but identified by the special symbol  $\&$ . We recall this information when we analyse the graph, in order to substitute the non terminal molecule with one (or every) possible instantiation of it.

## 2.1 Applications of the Reaction Graph

We believe that the reaction graph is a very useful tool for the analysis and testing of SAs. In fact, as control flow graphs represent the flow of control in a program, reaction graphs represent the *transition* flow in a SA. Thus, reaction graphs can play, for SA, the same role flowgraphs play for programs. For example, a reaction path represents a potential behavior of the system, and can thus constitute a test case. Therefore, many different criteria to cover the reaction graph can be devised, which would correspond to as many testing strategies for the system. For instance, we can immediately think of the two following test coverage criteria.

The simplest coverage criterion that one could imagine is each transition in the transition graph  $R$  at least once: we call this the T\_basic criterion. Of course, given an  $R$ , we can derive different lists of paths satisfying the T\_basic criterion: most appropriate ways for covering the graph should be studied.

Another more accurate criterion is to require that all possible paths in  $R$  are covered: this criterion could not always be feasible, e.g., when  $R$  presents cycles. So, similarly to what is done in control-flow based testing,

we can fix an upper bound to the number of iterations of cycles, say 1. Hence, we define the T\_path criterion as requiring that all paths in  $R$  iterating at most once possible cycles are covered.

Finally, we can use the reaction graph to identify sensible decompositions of the system. Thus it can also be used as a tool for guiding integration test. In fact, the degree of connection of the reaction graph is an indication of the coupling between components; for instance, if the reaction graph is not connected we can decide to divide the testing of the system into those parts corresponding to the subgraphs.

Furthermore, we can think of applying suitable reduction rules to the graph, to obtain reduced reaction subgraphs describing interesting classes of behaviours. If we know that a transition is particularly critical we can consider all and only those paths that include such transition. Or, if in a system we are going to replace one component and are interested to understand which functions would be affected, we can reduce the graph by leaving in it all and only those molecules including this component, and then perform the usual analyses on the reduced graph.

In the next section we provide an example of application of the reaction graph to a real world case study.

## 3 The TRMC System

In this section we briefly sketch the Teleservice and Remote Medical Care (TRMC) system architecture, already introduced in [1]. This system provides and guarantees assistance services to users with specific needs, like disabled or elderly people. It is composed of a set of *Users* connected to a *Router*, which interacts with a *Server*. An external component, the *Timer*, allows the modeling of time. The four types of units operate as follows:

- **User** sends either alarm (i.e. help requests) or check signals (i.e. control messages about the subsystem user state or the users's health state, respectively).
- **Router** accepts signals (control or alarm) from **User** s. It forwards the alarm requests to the *Server* and checks the behavior of the subsystem user through the control messages.
- **Timer** sends a clock signal for each time unit.
- **Server** dispatches the help requests.

The CHAM specification of the system consists of the molecule syntax  $\sum_{TRMCS}$ , transformation rules  $T_0, \dots, T_{10}$  and the initial solution  $S_0$ .

$\Sigma_{TRMCS}$ :

$M ::= P \mid C \mid M \diamond M$   
 $P ::= \mathbf{User} \mid \mathbf{Router} \mid \mathbf{Server} \mid \mathbf{Timer}$   
 $D ::= check \mid alarmUR \mid alarmRS \mid ackRU \mid$   
 $ackSR \mid nofunc \mid clock$   
 $C ::= i(D) \mid o(D)$

The set  $P$  represents the active components of the system. **Timer** sends clock messages at regular times. As it will be clearer in the following we do not explicitly model passing of time, we simply assume that the **Timer** sends its messages periodically. The clock message lets the **Router** start to check whether all the **User** have sent a check message during the last time period. Based on this check the **Router** can send a message of not-working (*nofunc*) to the **Server**.

The set  $D$  represents the data (messages in this case) exchanged among the various system components upon communication.

*check* represents the message the user sends to the **Router**. It can be a control message or a message containing information about the users' health state. At this level of description we do not model this difference since we are interested in analyzing the interactions among components.

*alarmUR* is the message from the **User** to the **Router** to signal an alarm.

*alarmRS* is the message from the **Router** to the **Server** that forwards the **User** alarm request.

*ackRU* is the acknowledgment message from the **Router** to the **User**.

*ackRS* is the acknowledgment message from the **Server** to the **Router**.

The set  $C$  specifies connecting elements, which characterize the kind of possible communications among system components.

$i(D)$  represents an input communication port, acting of a particular input data in  $D$

$o(D)$  represents an output communication port, acting of a particular output data in  $D$

The initial solution of the CHAM is:

$S_0 = \mathbf{User} \diamond o(alarmUR) \diamond i(ackRU),$   
 $\mathbf{User} \diamond o(check), \mathbf{Timer} \diamond o(clock),$

$i(alarmUR) \diamond o(alarmRS) \diamond i(ackSR)$   
 $\diamond o(ackRU) \diamond \mathbf{Router},$   
 $i(check) \diamond \mathbf{Router},$   
 $i(alarmRS) \diamond o(ackSR) \diamond \mathbf{Server},$   
 $i(nofunc) \diamond \mathbf{Server},$

The rules are the following:

$T_0 \equiv \mathbf{User} \diamond o(check) \longrightarrow$   
 $o(check) \diamond \mathbf{User}, \mathbf{User} \diamond o(check)$   
 $T_1 \equiv \mathbf{User} \diamond o(alarmUR) \diamond i(ackRU) \longrightarrow$   
 $o(alarmUR) \diamond i(ackRU) \diamond \mathbf{User},$   
 $\mathbf{User} \diamond o(alarmUR) \diamond i(ackRU)$   
 $T_2 \equiv i(check) \diamond \mathbf{Router}, o(check) \diamond \mathbf{User}, \longrightarrow$   
 $i(check) \diamond \mathbf{Router}$   
 $T_3 \equiv i(alarmUR) \diamond o(alarmRS) \diamond i(ackSR) \diamond$   
 $o(ackRU) \diamond \mathbf{Router},$   
 $o(alarmUR) \diamond i(ackRU) \diamond \mathbf{User} \longrightarrow$   
 $o(alarmRS) \diamond i(ackSR) \diamond o(ackRU) \diamond \mathbf{Router},$   
 $i(alarmUR) \diamond o(alarmRS) \diamond i(ackSR) \diamond$   
 $o(ackRU) \diamond \mathbf{Router},$   
 $i(ackRU) \diamond \mathbf{User} \diamond o(alarmUR)$   
 $T_4 \equiv m \diamond \mathbf{Router}, o(clock) \diamond \mathbf{Timer} \longrightarrow$   
 $o(nofunc) \diamond \mathbf{Router}, m \diamond \mathbf{Router}$   
 $T_5 \equiv m \diamond \mathbf{Router}, o(clock) \diamond \mathbf{Timer} \longrightarrow$   
 $m \diamond \mathbf{Router}$   
 $T_6 \equiv i(alarmRS) \diamond o(ackSR) \diamond \mathbf{Server},$   
 $o(alarmRS) \diamond i(ackSR) \diamond o(ackRU) \diamond \mathbf{Router} \longrightarrow$   
 $o(ackSR) \diamond \mathbf{Server},$   
 $i(alarmRS) \diamond o(ackSR) \diamond \mathbf{Server},$   
 $i(ackSR) \diamond o(ackRU) \diamond \mathbf{Router}$   
 $T_7 \equiv o(ackSR) \diamond \mathbf{Server},$   
 $i(ackSR) \diamond o(ackRU) \diamond \mathbf{Router} \longrightarrow$   
 $o(ackRU) \diamond \mathbf{Router}$   
 $T_8 \equiv i(ackRU) \diamond \mathbf{User} \diamond o(alarmUR),$   
 $o(ackRU) \diamond \mathbf{Router} \longrightarrow$   
 $\mathbf{User} \diamond o(alarmUR) \diamond i(ackRU)$   
 $T_9 \equiv \mathbf{Timer} \diamond o(clock) \longrightarrow o(clock) \diamond \mathbf{Timer},$   
 $\mathbf{Timer} \diamond o(clock)$   
 $T_{10} \equiv i(nofunc) \diamond \mathbf{Server}, o(nofunc) \diamond \mathbf{Router} \longrightarrow$   
 $i(nofunc) \diamond \mathbf{Server}$

#### 4 The Reaction Graph for TRMC

With reference to the CHAM specification in the previous section, we have derived the Reaction Graph below:

From a first, visual analysis of the graph, we see that it

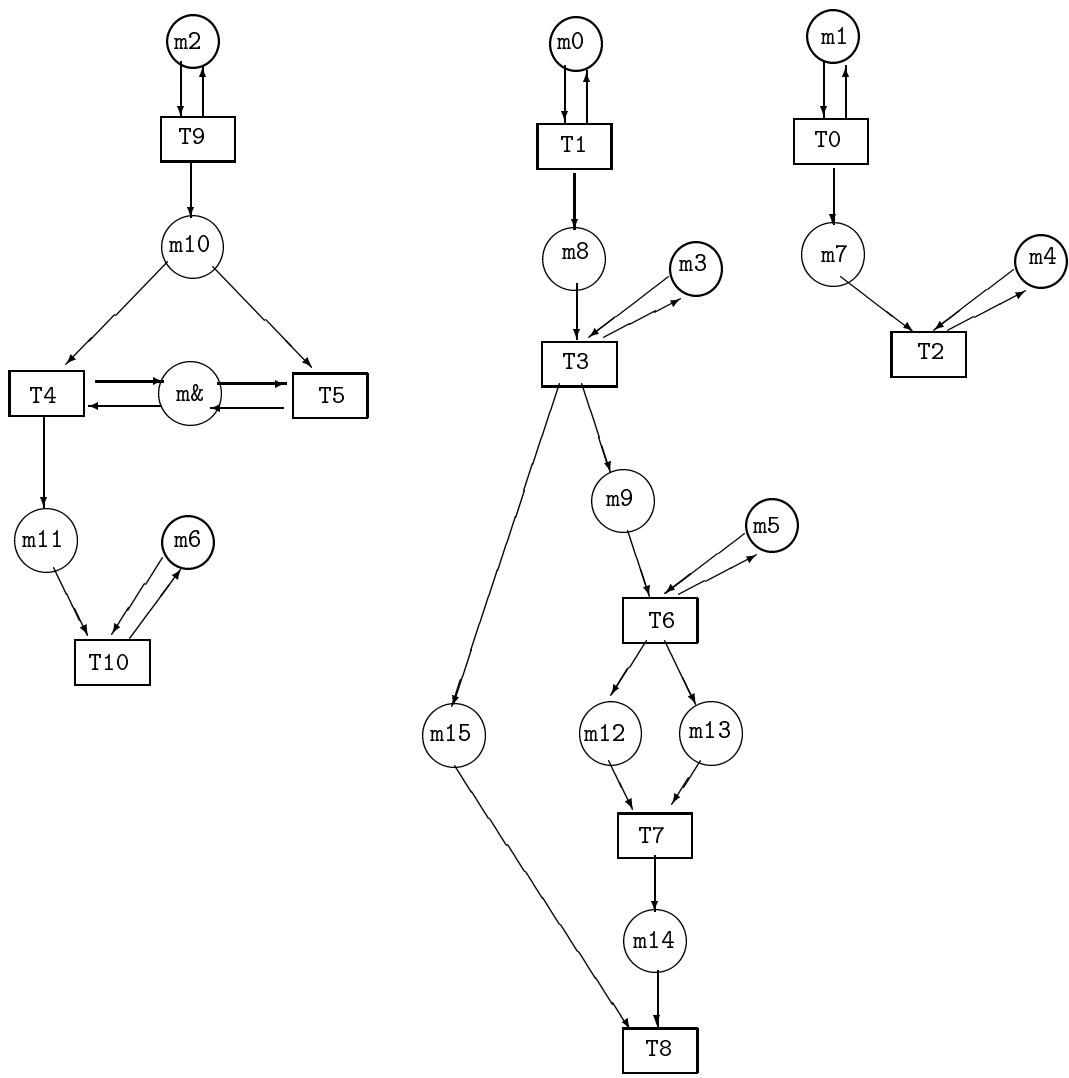


Figure 1: TRMC reaction graph

is a disconnected graph, with three components. This would suggest the separate testing of the three subsystems involved by the three subgraphs of  $R$ . The leftmost subgraph involves the interaction between the **Router** and the **Timer**; the central subgraph involves the three components **User**, **Router** and **Server**; finally the rightmost subgraph involves the interaction between **User** and **Router** during check transmittal and reception.

Let us apply the two testing criteria outlined above. For the  $T\_basic$  criterion, a possible list of reaction paths ( $rp$ ) fulfilling this criterion is:

- $rp_1 : \{T_0, T_2\}$  : the **User** sends a check and the **Router** receives it;
- $rp_2 : \{T_1, T_3, T_6, T_7, T_8\}$  : the **User** sends an alarm msg, the **Router** receives the alarm and transmits it to the **Server**; which in turn receives the alarm and sends an Ack back to the **Router**; the **Router** receives the **Server** ack and sends an ack to the **User**;
- $rp_3 : \{T_9, T_5\}$ : the clock sends a clock msg to the **Router**, which receives it;
- $rp_4 : \{T_9, T_4, T_{10}\}$ : the clock sends a clock msg to the **Router**, and this issues a "nofunc" msg to the **Server**, which receives it

If instead we try to apply the  $T\_path$  criterion, the list of reaction paths becomes much longer. Due to the cycles in  $R$ , we have to consider possible iterations of transitions. We fix to 1 the maximum number of iterations. For instance from  $rp_1$ ,  $rp_3$  and  $rp_4$  we also obtain:

- $rp_{1'} : \{T_0, T_0, T_2\}$ ;
- $rp_{3'} : \{T_9, T_9, T_5\}$ ;
- $rp_{4'} : \{T_9, T_9, T_4, T_{10}\}$

Other reaction paths can be derived by instantiating the non terminal molecule in the leftmost component of the reaction graph. For instance, if we instantiate it by molecule  $m_9$ , we obtain, among others, the reaction paths:

- $rp_5 : \{T_9, T_1, T_3, T_4, T_{10}\}$ ;
- $rp_6 : \{T_1, T_9, T_3, T_5\}$

## 5 Conclusions

We have presented a new approach to model the dynamic behavior of a transition-based SA description.

The method has several advantages with respect to the traditional state transition modeling [4, 3]. Notably, it is very expressive while avoiding the state explosion problem. It is based on the notion of a reaction graph, which describes the transition flow in a compact way. By using reaction graphs it is possible to perform static analysis and define test criteria based on notions of coverage in the spirit of [6]. In this abstract we have just sketched the basic concepts. Although much work remains to be done we believe these initial attempts look very promising.

## REFERENCES

- [1] S. Balsamo, P. Inverardi, C. Mangano, F. Russo, Performance Evaluation of a Software Architecture: A Case Study. *IEEE Proc. IWSSD-9*, April 1998, Ise-Shima, Japan.
- [2] A. Bertolino, P. Inverardi. Architecture-based Software Testing. Second International Software Architecture Workshop ISAW-2. In *Joint Proceedings of the ACM SIFSOFT '96 Workshops*. ACM, October 1996.
- [3] A. Bertolino, P. Inverardi, H. Muccini, A. Rosetti. An Approach to Integration Testing Based on Architectural Descriptions, *IEEE Proc. ICECCS-97*, Como 1997.
- [4] T. S. Chow. Testing Software Design Modeled by Finite-State Machines. *IEEE Transactions on Software Engineering*, 4 (3), May 1978.
- [5] P. Inverardi and A.L. Wolf. Formal Specifications and Analysis of Software Architectures Using the Chemical Abstract Machine Model. *IEEE Transactions on Software Engineering*, 21(4):100–114, April 1995.
- [6] M. Marré and A. Bertolino. Reducing and estimating the cost of test coverage criteria. *Proc. ACM/IEEE Int. Conf. Software Eng. ICSE-18*, pp. 486–494, Berlin, Germany, March 1996.