Towards ASM-Based Formal Specification of Self-Adaptive Systems

Elvinia Riccobene[1] and Patrizia Scandurra[2]

[1] Computer Science Department, Università degli Studi di Milano, Italy
[2] Engineering Department, Università degli Studi di Bergamo, Italy

Abstract. This paper shows how to use multi-agent Abstract State Machines to specify self-adaptive behavior in a decentralized adaptation control system. A traffic monitoring system is taken as case study.

1 Introduction

Modern software systems typically operate in dynamic environments and are required to deal with changing operational conditions: components can appear and disappear, may become temporarily or permanently unavailable, may change their behavior, etc. Self-adaptation (SA) has been widely recognized [4,6] as an effective approach to deal with the increasing complexity, uncertainty and dynamics of these advanced systems. A well recognized engineering approach to realize self-adaptation is by means of a feedback control loop conceived as a sequence of four computations: Monitor-Analyze-Plan-Execute [6].

One major challenge in self-adaptive systems is to assure the required quality properties (e.g., flexibility, robustness, etc.). Formal methods are an attractive option for solving this problem as they provide a means to precisely model and reason about the behaviors of self-adaptive systems. The survey in [7] shows that the attention for self-adaptive software systems is gradually increasing, but the number of studies that employ formal methods remains low, and is mainly related to runtime verification. However, formally founded *design* models that cover structural and behavioral aspects of self-adaptation, and of approaches to validate behavioral properties are of extreme importance in order to provide guarantees about qualities at the early stages of the system design.

By exploiting the theoretical framework of the Multi-Agent Abstract State Machines (ASM) [2], we here show how to model the behavior of self-adaptive distributed systems with decentralized adaptation control, where the MAPE control loop is naturally formalized in terms of agents' actions (transition rules). A traffic monitoring application, inspired from [5], is taken as case study.

This is a first work of our ongoing research activity on answering the request of precise models that help reasoning about adaptation at design time. In the conclusion we report some lessons learned from our experience that reveal the high potentiality of the ASMs in the context of self-adaptive systems.

Y. Ait Ameur and K.-D. Schewe (Eds.): ABZ 2014, LNCS 8477, pp. 204–209, 2014.

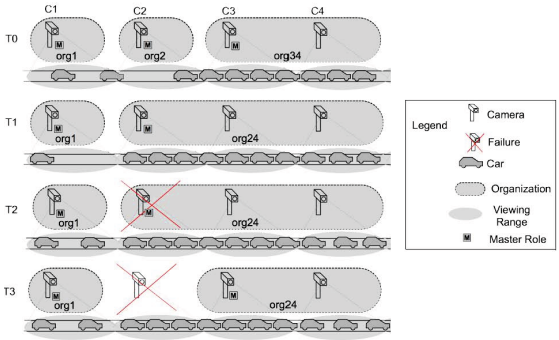

Fig. 1. Adaptation scenarios (adapted from [5])

2 The Traffic Monitoring Case Study

We present a traffic monitoring application inspired by the case study in [5].

A number of intelligent cameras are distributed along a road, each with a limited viewing range (see Fig. 1). Cameras are equipped with a data processing unit capable of processing the monitored data, and a communication unit to communicate with other cameras. Traffic jams can span the viewing range of multiple cameras and can dynamically grow and dissolve. Each camera monitors the traffic state within its viewing range. Because there is no central point of control, cameras have to aggregate the monitored data to determine the position of the traffic jam on the basis of the head and tail of it. Cameras enter or leave the collaboration whenever the traffic jam enters or leaves their viewing range.

There are two main adaptation concerns. The first is system *flexibility* for the dynamic adaptation of an organization. See, e.g., the scenario in Fig. 1 from configuration T0 to T1, where camera 2 joins the organization of cameras 3 and 4 after it monitors a traffic jam. The second is related to *robustness* due to camera failures, i.e., when a failing camera becomes unresponsive. This scenario is shown in Fig. 1 from T2 to T3, where camera 2 fails.

3 Multi-Agent ASM Specification

Because of the distributed nature of adaptive systems, we use the notion of *multi-agent ASMs* where multiple agents interact in parallel in a synchronous/ asynchronous way. Each agent executes its own (possibly the same but differently instantiated) ASM-based program that specifies the agent's behavior. Some agents form the *managing ASM* part encapsulating the logic of self-adaptation. Some other agents form the *managed ASM* part encapsulating the functional logic.

For the traffic monitoring application, we introduce four ASM agents: the agents *OrganizationController* and *SelfHealingController* representing the managing components, an agent *Camera* and an agent *TrafficMonitor* (the sensor that in case of "congestion" or "no longer congestion" notifies the organization

Listing 1.1. Organization controller's program

```
macro rule r_organizationControl =
 seq //MAPE control loops
   r_selfFailureAdapt[] //Adaptation due to internal failure
   r_failureAdapt[] //ROBUSTNESS: Adaptation due to external failure (silent nodes)
   r_congestionAdapt[] //FLEXIBILITY: Adaptation due to congestion
 endseq
```

controller) both representing the managed camera subsystem. An ASM module, called *knowledge*, is used as knowledge for the MAPE loops to define the ASM signature (domains and functions symbols) shared among the managing agents. We specify the self-adaptive behavior of the managing components of a camera using two main MAPE loops: the first loop deals with flexibility concerns to restructure organizations in case of congestion, the second loop deals with robustness concerns to restructure organizations in case of failing cameras. Both the two loops start with the program associated to the agent *OrganizationController*, however due to the decentralized nature of MAPE computations, part of the monitoring functionality of the second loop is on the *SelfHealingController* behavior. A further MAPE loop to deal with internal failures of the camera is also executed by the two managing agents.

For the lack of space, we describe only part of the behavior of the *OrganizationController*. The complete specification is available online[1].

Organization middleware for Flexibility. An organization controller runs on each camera and is responsible for managing organizations depending on the data it gets from the traffic monitor and from the self-healing controller of the camera. A master/slave control model is adopted to structure organizations in case of congestion. Each camera has a unique ID (a static integer-valued function *id*). To keep the master election policy simple, we assume the camera ID is monotonically increasing on the traffic direction and the camera with the lowest ID becomes master. Traditional election algorithms (like the Bully algorithm and the Ring algorithm) or new ones are out of the scope of this paper.

Each camera has four basic states (the function *state*). In normal operation, the camera can be master with no slaves (i.e., master of a single organization), master of an organization with slaves, or it can be slave. Additionally, the camera can be in the failed state, representing the status of the camera after a silent node failure. Initially, all cameras are master. A camera state is changed by the organization controller as part of the adaptation logic. The organization controller has the same four basic states of the camera it manages. The organization controller's program (see the rule `r_organizationControl` in Listing 1.1) executes sequentially the three MAPE control loops.

We here focus on the third MAPE loop for adapting organizations in case of traffic congestion notified by the traffic monitor. Such a behavior is represented by the rule `r_congestionAdapt` defined in Listing 1.2.

[1] See the examples directory in the ASMETA repository `http://asmeta.sf.net/`

Listing 1.2. ASM rule `r_congestionAdapt`

```
 //@M_ c context−aware monitoring
macro rule r_congestionAdapt =
par
 if state(self) = MASTER
 then
      if ( cong(camera(self)) and not congested(self) )//Congestion detected!
      then //@P Planning
            par
             congested(self) := true
             cong(camera(self)) := false
             if isDef(next(camera(self))) then s_offer(next(camera(self))) := true endif
            endpar
      else if congested(self) then r_analyzeCongestion[] endif endif endif
 if state(self) = SLAVE
 then
      if no_cong(camera(self)) //No longer congested!
      then //@P Planning
            par
             no_cong(camera(self)):= false
             congested(self) := false
             slaveGone(getMaster(camera(self)),camera(self)) := true
             r_turnMaster[]
            endpar
      else
        r_receiveOrgSignals[] endif endif
 if state(self) = MASTERWITHSLAVES
 then if no_cong(camera(self)) //No longer congested!
       then //@P Planning
             par r_removeSlavesTurningMaster[]
             no_cong(camera(self)):= false
             congested(self):= false endpar
       else r_analyzeOrganization[] endif endif
endpar
```

In the role of master of a single member organization, when a congestion is detected (the signal *cong*) the organization controller sends a request (the predicate *s_ offer*) to the next alive camera (if any) in the direction of the traffic flow to join the organization as slave. Depending on the traffic condition of the next camera and its role, the organizations may be restructured according to the rule *r_ analyzeCongestion* reported in Listing 1.3. If traffic is not jammed (the controlled predicate *congested* is false) for the next camera, organizations are not changed, otherwise organizations are joined. The next camera becomes slave of the requester camera by executing the rule `r_turnSlave` that changes the camera state, sets the requester camera as new master and informs back it by setting the shared function *newSlave* and (indirectly) the derived predicate *m_ offer* to true. When the *m_ offer* signal is set the requester camera becomes master of the joined organization executing the rule `r_turnMasterWithSlaves` (see Listing 1.3) to concretely add the new slave to its list and change state.

In the role of slave, if the traffic in the viewing range of the camera is no longer jammed (the signal *no_ cong*), the organization controller leaves the organization it belongs to (by setting the function *slaveGone*) and becomes master of a single member organization (by executing the rule `r_turnMaster` in Listing 1.2). Otherwise (still congested), the organization controller waits (by

Listing 1.3. ASM rules for analysis computations

```
//@A Analyzing
macro rule r_analyzeCongestion =
 if m_offer(camera(self)) then r_turnMasterWithSlaves[]
 else if s_offer(camera(self)) then r_turnSlave[prev(camera(self))] endif endif
//@A Analyzing
macro rule r_receiveOrgSignals =
   par
     if change_master(camera(self)) then //Master changed!
              //@P Planning
              par
               r_setMaster[prev(getMaster(camera(self)))]
               newSlave(prev(getMaster(camera(self))),camera(self)) := true
               change_master(camera(self)) := false
              endpar endif
     if masterGone(camera(self)) then r_turnMaster[] endif
     if m_offer(camera(self)) then r_notifyPendingSlavesMasterChanged[] endif
   endpar
//@A Analyzing
macro rule r_analyzeOrganization =
if m_offer(camera(self)) then r_addNewSlave[]
else if isEmpty(slaves(camera(self))) //Simply turn master
     then r_turnMaster[]
     else if (s_offer(camera(self)) and congested(self))
          then r_turnSlave[prev(camera(self))] endif endif endif
//@P Planning
macro rule r_addNewSlave =
  forall $s in Camera with newSlave(camera(self),$s) do
      par
        r_addSlave[$s]
        newSlave(camera(self),$s):= false
        s_offer($s):=false
      endpar
```

the rule `r_receiveOrgSignals`) for a trigger from its master. The rule `r_receiveOrgSignals` is reported in Listing 1.3. If (in the slave role) the controller receives a signal *change_ master* as effect of a restructuring of the organization, it is responsible for planning adaptations to change its master to the new master. If it receives that the master is gone (by the shared predicate *masterGone*), it restarts the camera as master of a single member organization (by invoking the rule `r_turnMaster[]` already shown in Listing 1.2). Finally, if it receives an *m_ offer* signal, it means there are slaves not effectively engaged when in the role of master it asked them to join the organization as slave. In this last case it is responsible for notifying them that the master changed by executing the rule `r_notifyPendingSlavesMasterChanged` in Listing 1.3.

Finally, in the role of master with slaves, when the traffic is no longer jammed (the signal *no_ cong*), the organization controller notifies all its depending slaves that the master is gone (setting the predicate *masterGone* to true) and leaves the organization becoming master of a single member organization (see rule `r_removeSlavesTurningMaster` in Listing 1.2). Otherwise (still congested), the organization controller analyzes the organization by the rule `r_analyzeOrganization` (see Listing 1.3) to add and remove slaves dynamically. When no slaves remain, the master with slaves becomes master of a single member organization again. During analysis of its organization, it has also to

wait for a trigger *s_ offer*, and if notified it has to plan to become slave of the requester camera by executing `r_turnSlave` in Listing 1.2.

For the lack of space, macro rules (such as `r_turnSlave`, `r_turnMaster`,etc.) and those annotated with @E for atomic adaptation actions in a master/slave organization (such as `r_clearSlaves`, `r_addSlave`, etc.) are not reported.

4 Conclusion and Future Directions

Besides modeling, we were also able to validate the Traffic Monitoring case study by exploiting model simulation and scenario construction. We focused on two qualities: flexibility (i.e., the ability of the system to adapt dynamically with changing conditions in the environment), and robustness (i.e., the ability of the system to cope autonomously with errors during execution). By means of the ASM tools[1,3], we simulated different scenarios with increasing number of cameras. In particular, we reproduced the adaptations scenarios shown in Fig. 3 from T0 to T2 for flexibility, and from T2 to T3 for robustness.

From modeling and validation, we learned some lessons briefly reported. We were able to achieve a clear separation of concerns: (i) separation between adaptation logic and function logic, (ii) separation between behavior of managing and managed components, (ii) separation between the specification of the MAPE functions. This helps the designer to focus on one adaptation concern at a time, and, for each concern, separate the adapting parts from the adapted ones.

In the future, we plan to define a formal framework providing high level constructs for expressing context-awareness, self-awareness, adaptation actions, distributed communication patterns. We plan to investigate on the verification of self adaptive systems by using the ASMETA tool set.

References

1. Arcaini, P., Gargantini, A., Riccobene, E., Scandurra, P.: A model-driven process for engineering a toolset for a formal method. SPE J. 41(2), 155–166 (2011)
2. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer (2003)
3. Carioni, A., Gargantini, A., Riccobene, E., Scandurra, P.: Scenario-Based Validation Language for ASMs. In: Börger, E., Butler, M., Bowen, J.P., Boca, P. (eds.) ABZ 2008. LNCS, vol. 5238, pp. 71–84. Springer, Heidelberg (2008)
4. de Lemos, R., et al.: Software engineering for self-adaptive systems: A second research roadmap. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 7475, pp. 1–32. Springer, Heidelberg (2013)
5. Iftikhar, M.U., Weyns, D.: A case study on formal verification of self-adaptive behaviors in a decentralized system. In: Kokash, N., Ravara, A. (eds.) FOCLASA. EPTCS, vol. 91, pp. 45–62 (2012)
6. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer 36(1), 41–50 (2003)
7. Weyns, D., Iftikhar, M.U., de la Iglesia, D.G., Ahmad, T.: A survey of formal methods in self-adaptive systems. In: C3S2E, pp. 67–79. ACM (2012)