# Agents Modeling under Fairness Assumption in Event-B

Irina Mocanu, Lorina Negreanu, and Adina Magda Florea

**Abstract.** Multi-agent systems, which are composed of autonomous agents are present in applications that span a wide range of domains: ambient intelligence, telecommunications, finance, Internet, energy, health. Therefore, it is critical to have rigorous, effective design and verification methods to ensure their development. In this paper, we present a formal modeling and proof of a multi-agent system for requesting services, under the fairness assumption. The model is specified and verified using Event-B and the Rodin platform.

## 1 Introduction

This paper specifies and verifies, using the Event-B [5], [6] formal specification method, a multi-agent system for requesting services under the fairness assumption. This should be read as an extension of the model we previously specified in [9], who will be integrated in an ambient intelligent system.

Fairness plays an important role in software specification, verification and development. Fairness properties state that if something is enabled sufficiently often, then it must eventually happen [8].This becomes important in our model when the requests are satisfied. It is possible that a request cannot be satisfied for an indefinite period of time while other requests continue normally. This may occur if the satisfying scheme of the requests is unfair.

The paper is organized as follows: Section 2 describes the proposed system, Section 3 presents the refinement of the model specified in [9] while fairness constraints are stated using the Event-B method, Section 4 lists conclusions and future work.

Irina Mocanu · Lorina Negreanu · Adina Magda Florea
University "Politehnica" of Bucharest,
Computer Science Department,
Splaiul Independentei 313,
060042 Bucharest, Romania
e-mail: {irina.mocanu,lorina.negreanu,adina.florea}@cs.pub.ro

F. Zavoral et al. (eds.), *Intelligent Distributed Computing VII,*
Studies in Computational Intelligence 511,
DOI: 10.1007/978-3-319-01571-2_35, 

## 2 System Description

We refine the former specification of the system for managing requests for services [9] under the strong fairness assumption [8]. Consider a supervised person that wants to do an activity at a relaxing centre (e.g. swimming, physiotherapy or talking to somebody). Since the person's well being is important we need to have a system that will manage his request based on some measured parameters.

The system we propose is composed of several agents [9]: (i) an agent to interrogate the ambient factors - *Temperature Agent*; (ii) an agent to verify the health status of the supervised person -*Pulse Agent*; (iii) an *User Agent* associated with the user; (iv) *Service Agents* (there are n agents - each agent has a specialization and a maximum available time) and (v) a *Community Agent* (this agent knows all the services available and unavailable). The user makes a request and the *User Agent* will analyze the request by computing the priority and the duration of the service using the monitoring information (health status of the person and performed activities). The *User Agent* will send a verification message to the *Temperature Agent* and to the *Pulse Agent* in order to verify if the supervised person can perform that activity. If these parameters (the ambiental temperature/pulse) are in normal values for that person, the *User Agent* will send a request message to the *Community Agent* (*new_request*). If the answer for the request is not received in a predefined time, the *User Agent* will send a message to the *Community Agent* (*modify_request*) in order to increase the priority of the requested service for that user. The request will be canceled by the *User Agent* (*cancel_request*). If there is available time for requested service, the corresponding *Service Agent* will inform the *User Agent* through the *Community Agent* ( *satisfy_request*). Some services may be requested more frequently than others. Thus the *Community Agent* will request to increase the maximum available duration for a *Service Agent*, by taking some available time from other services (*request_ available*). The *Service Agent* associated with the requested service with fewer requests will add some extra time to that service (*release_available*).

## 3 Formal Specification

In the previous work [9] we specified the request as the main modeling element. A request is defined as a member of the set of requests. It has a *status* (*pending* or *satisfied*), a *service* and a *duration* associated, that can be accessed through the functions

$$status \in requests \rightarrow \text{STATUS},$$

where *status* assigns a status to each request,

$$reference \in requests \rightarrow \text{SERVICES},$$

where reference assigns a service to each request,

$$duration \in \text{requests} \rightarrow \text{N}^*,$$

where *duration* assigns a duration to each *request* (with the assumption that if a service is requested, the duration is at least 1). The status of the request is changed

into *satisfied*, if the duration of the requested service is either less than or equal to the time availability of the service, defined as:

$$available \in \text{SERVICES} \rightarrow \text{N}.$$

In the modeling [9] we started by considering that all requested references exist in the set of available services and that this set and the set of requests are given in an up-to-date state. We derived by refinement [7] the situation in which we have to take into account new requests, modification of requests, cancelation of requests and update of services time availability.

## 3.1 Refinement under the Fairness Assumption

Bellow we refine the previous specification described by [9] addressing the problem of fairness. The previous specification captures the notion of flow by a set. In fact, it is possible that a request remains always pending and is never satisfied, because there are always other requests which are processed.

Our solution is to add a priority to each request that is increased the longer it waits and to satisfy the request with the highest priority. If a request is waiting too long (more than a specific deadline) the request is canceled. The event *new_request* gives each new request a priority using a parameter $p$ ($p \in \text{N}$). The variable *priority*, $priority \in requests \rightarrow \text{N}$, records the priority of the request and the new condition strengthens the guard of the event *satisfy_request*:

$$\forall\, p \cdot (p \in requests \wedge status(p)=pending \wedge duration(p) \leq available(reference(p)) \Rightarrow priority(p) \leq priority(r)).$$

In order to manage the waiting time of a request we add a time stamp recorded in the variable *timer*, $timer \in requests \rightarrow \text{N}$. The event *new_request* gives each new request a time stamp using the variable *clock*, $clock \in \text{N}$, that grows larger as each successive operation is invoked:

$$timer(r) := clock.$$

The event *cancel_request* is enabled if the difference between the current clock value and the time stamp of the request is larger than the deadline of the request:

$$clock - timer(r) > deadline(r),$$

where the constant *deadline*,

$$deadline \in \text{REQUESTS} \rightarrow \text{N},$$

records the maximum time for a request to be processed. The event *modify_request* increases the priority of the request if it took longer than the amount of time given by the constant *oldies*,

$$oldies \in \text{REQUESTS} \rightarrow \text{N}$$
$$clock - timer(r) > oldies(r).$$

The context of the described refinement together with the added variables are given in Fig. 1. Invariants for the variables *timer*, *clock* and *priority* together with the corresponding initialization are given in Fig. 2.

The event *satisfy_request* is given in Fig. 3.

**CONTEXT**
Services_c1
**EXTENDS**
Services_c0
**CONSTANTS**
deadline
pq
oldies
**AXIOMS**
axm1: deadline $\in$ REQUESTS $\rightarrow$ N
axm2: pq $\rightarrow$ N
axm3: oldies $\in$ REQUESTS $\rightarrow$ N
**END**

**MACHINE**
Services_2
**REFINES**
Services_1
**SEES**
Services_c1
**VARIABLES**
...
timer
clock
priority

**Fig. 1** The context and variables for the refinement

inv1: timer $\in$ requests $\rightarrow$ N
inv2: clock $\in$ N
inv3: priority $\in$ REQUESTS $\rightarrow$ N

...
act6: timer := $\emptyset$
act7: clock := $\emptyset$
act8: priority :$\in$ REQUESTS $\rightarrow$ N

**Fig. 2** The invariants and initialization for the refinement

**satisfy_request:**
**REFINES**
satisfy_request
**ANY**
r
**WHERE**
grd1: r $\in$ requests
grd2: status(r) = pending
grd3: duration(r) $\leq$ available(reference(r))
grd4: $\forall$ p $\cdot$ (p $\in$ requests $\wedge$ status(p)=pending $\wedge$
duration(p) $\leq$ available(reference(p)) $\Rightarrow$ priority(p) $<$ priority(r))
**THEN**
act1: status(r):= satisfied
act2: available(reference(r)) := available(reference(r)) - duration(r)
act3: clock := clock + 1
**END**

**Fig. 3** The *satisfy_request* event

The events *new_request* and *cancel_request* are given in Fig. 4 while *modify_request* is given in Fig. 5. The events *request_available* and *release_available* are not further refined. They have the same specification as in [9].

**new_request:**
**REFINES**
new_request
**ANY**
r
d
s
p
**WHERE**
grd1: r ∈ REQUESTS \ requests
grd2: d ∈ N1
grd3: s∈SERVICES
grd4: p ∈ N
**THEN**
act1: requests := requests ∪ {r}
act2: status(r) := pending
act3: duration(r) := d
act4: reference(r) := s
act5: priority(r) := p
act6: timer(r) := clock
act7: clock := clock + 1
**END**

**cancel_request:**
**REFINES**
cancel_request
**ANY**
r
**WHERE**
grd1: r ∈ requests
grd2: status(r) = pending
grd3: clock - timer(r) > deadline(r)
**THEN**
act1: requests := requests \ {r}
act2: status := {r} << | status
act3: duration := {r} << | duration
act4: reference := {r} << | reference
act5: priority := {r} << | priority
act6: timer := {r} << | timer
act7: clock := clock + 1
**END**

**Fig. 4** The *new_request* and *cancel_request* events

**modify_request:**
**REFINES**
modify_request
**ANY**
r
**WHERE**
grd1: r ∈ requests
grd2: status(r) = pending
grd3: clock - timer(r) > oldies(r)
**THEN**
act1: priority(r) := priority(r) + pq
act2: clock := clock + 1
**END**

**Fig. 5** The *modify_request* event

### *3.2 System Evaluation*

The proof statistics given in Table 1 show that 52 proof obligations were generated by the Rodin platform [3], [2]. 47 proof obligations were discharged automatically while the others were discharged by interactive proofs. In Table 1 Services_0 represents the abstract model; Services_1 represents the refinement described in [9] and Services_2 represents the refinement under the fairness assumption.

**Table 1** The statement of the development

| Element_name | Total | Auto | Manual | Reviewed | Undischarged |
|---|---|---|---|---|---|
| **Services** | **52** | **47** | **3** | **0** | **0** |
| Services_0 | 25 | 20 | 3 | 0 | 0 |
| Services_1 | 4 | 4 | 0 | 0 | 0 |
| Services_2 | 23 | 23 | 0 | 0 | 0 |

## 4 Conclusions and Future Work

In this paper, we have presented a specification and verification technique of a multi-agent system for requesting services under the fairness assumption. The informal specification of the system is translated into the Event B notation to verify the required properties. The model refinement that Event-B emphasizes simplifies proofs by providing a progressive and detailed view of the system.

As future work we intend to use UML-B, specified in [4], to model the system and translate the specifications into Event-B for verification. Since the final target is to have an executable code we intend to generate Java code from the Event-B specification with the aid of EB2J plug-in, [1].

**Acknowledgements.** The work has been funded by Project 264207, ERRIC- Empowering Romanian Research on Intelligent Information Technologies/FP7- REGPOT-2010-1 and the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/89/1.5/S/62557.

## References

1. EB2J Tool (2013), `http://eb2all.loria.fr/`
2. Rodin Platform (2013), `http://wiki.eventb.org/index.php/Rodin_Platform`
3. Rodin User's Handbook v.2.5 (2013), `http://handbook.eventb.org/current/html/index.html`
4. UML-B (2013), `http://wiki.eventb.org/index.php/UML-B`
5. Abrial, J.-R.: The B book. Cambridge University Press (1996)
6. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge Press (2010)

7. Abrial, J.-R., Cansell, D., Méry, D.: Refinement and reachability in event_B. In: Treharne, H., King, S., Henson, M., Schneider, S. (eds.) ZB 2005. LNCS, vol. 3455, pp. 222–241. Springer, Heidelberg (2005)
8. Sun, J., Liu, Y., Dong, J.S., Wang, H.H.: Specifying and verifying event-based fairness enhanced systems. In: Liu, S., Araki, K. (eds.) ICFEM 2008. LNCS, vol. 5256, pp. 5–24. Springer, Heidelberg (2008)
9. Negreanu, L., Mocanu, I.: Formal verification of service requests in a multi-agent system using Event-B method. In: 8th Workshop on Workshop Knowledge Engineering and Software Engineering (KESE 2012), ECAI 2012, Technical Report TR-2012/1, Montpellier, France, August 27-31, pp. 62–65. University of Almeria, Almeria (2012)