CMU Analysis of Software Artifacts

Lab2

Thursday, September 20, 2001

Jeff Ditillo

Mike Lester

Aliyassoun Djibrilla

1. Travel Requisition Process

This paper attempts to represent the Travel Requisition formally in the Zed notation. First, some given sets are needed:

```
[PERSON]          // all employees in the company
[MANAGER]         // all managers in the company
```

Next is a list of status types, similar to enumerated types found in programming languages:

```
STATUS_TYPES ::= = SUBMTD, APPRD, REJCTD, FILDOUT

| approval_limit: ℤ     // maximum approvalprice for middle manager
```

A travel request has several attributes such as status, price, traveler, and approver, and is represented by the TravelReq schema. The relation reg is adds an ID to the n-tuple of attributes. Currentreqs is a set of these tuples, acting as a "database."

TravelReq

```
status: STATUS_TYPES                          // status of req
price: ℤ                                      // price of req
traveler: PERSON                              // owner of req
approvedby: MANAGER                           // approving mgr
reqid: ℕ                                      // unique req ID
req: STATUS_TYPES x Z x PERSON x MANAGER → ℕ
currentreqs: ℙ {req}
```

```
MANAGER ⊆ PERSON          // all managers are in person set

traveler ≠ approvedby     // a person can not approve own req
```

The following operation, SubmitReq, modifies TravelReq by adding a "record" to the currentreqs set of tuples. Inputs to SubmitReq include price, traveler, and reqid (the request number).

SubmitReq

ΔTravelReq	// TravelReq changes
price?: $\mathbb{Z}$	// price of req
traveler?: PERSON	// owner of req
reqid!: $\mathbb{N}$	// assigned unique req ID

approvedby = NULL
status = SUBMTD
req = (status, price?, traveler?, approvedby) $\rightarrow$ reqid!
currentreqs' = currentreqs $\cup$ {req}

When a manager approves a request, the "record" stored in currentreqs has its status field set to the constant APPRD.

ApproveReq

ΔTravelReq	// TravelReq changes
approvingmgr?: MANAGER	// input approving manager
reqid?: $\mathbb{N}$	// input req ID

$\exists$ req: currentreqs $\bullet$ reqid? $\in$ dom req
req' = req $\wedge$ (req.approvedby = approvingmgr?) $\wedge$ (req.status = APPRD)
currentreqs' = currentreqs $\wedge$ {req'}

The schema RejectReq is similar to ApproveReq, except that req.status is set to REJECTED. This schema is not shown.

Instead of approving or rejecting, if the purchase cost is above a threshold, a manager may send the request to upper management.

SendToUpperMgmt

Δ TravelReq	// TravelReq changes status
reqid?: $\mathbb{N}$	// input req ID
$\exists$ req: currentreqs $\bullet$ reqid? $\in$ dom req	
req.price > approval_limit	
currentreqs' = currentreqs	

Finally, tying in various operations, the Approval can be represented by "schema calculus" as follows:

$$\text{APPROVAL} \mathrel{\widehat{=}} \text{SubmitReq} \;\wedge\; (\text{SendToUpperMgmt} \vee \text{ApproveReq} \vee \text{RejectReq})$$

2. State Diagrams

2.1 Approver's Viewpoint

This section describes the travel reimbursement system's states from the vantage of the approval authority. The approver, after receiving a request either:

- Approves the request
- Rejects the request
- Sends the request to his boss for approval and waits for the answer

This will be modeled in LTSA, which focuses on transitions between states. First, the set of transitions chosen for the Approver include:

- Receive_request - indicates that a request is in the approver's inbox
- Send_to_upper_management – purchase amount too high, send to boss
- Wait_for_approval – wait for boss' answer
- Approve_request, Reject_request

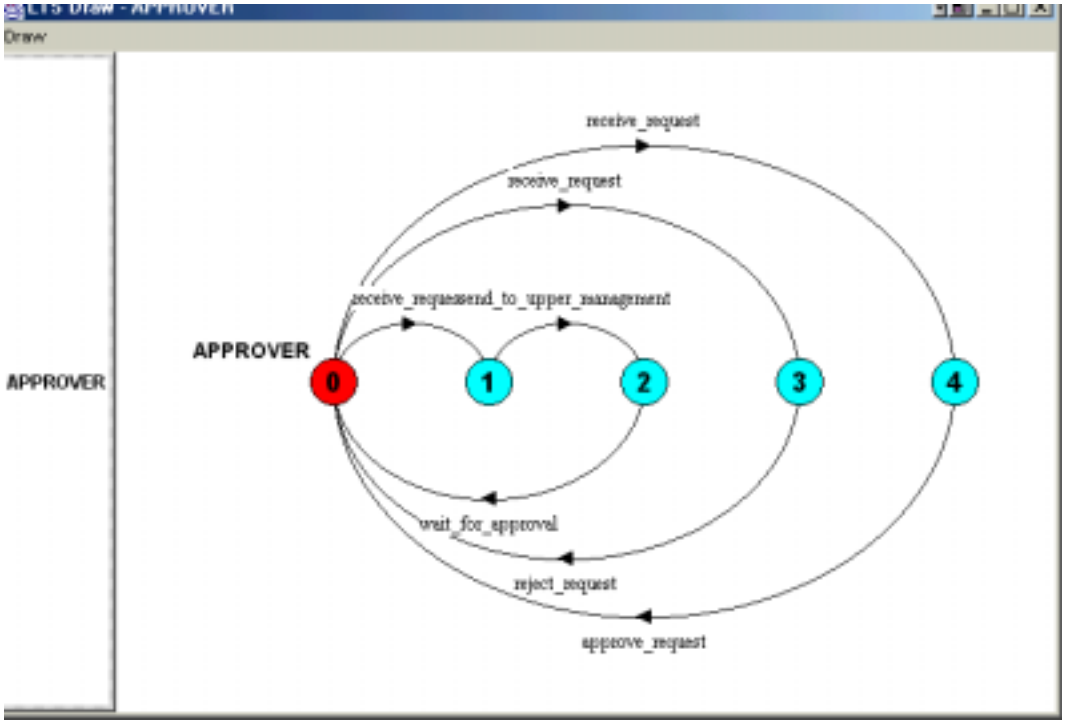
Finally, this is modeled in LTSA syntax as follows:

APPROVER = (
receive_request->send_to_upper_management->wait_for_approval->APPROVER|
receive_request->reject_request->APPROVER|
receive_request->approve_request->APPROVER).

Note: It would make more sense to distribute *receive_request* then continue on, but the LTSA tool would not allow this syntax. For example, it should be possible to state this in the form:

APPROVER= rr-> (s|r|a)->APPROVER

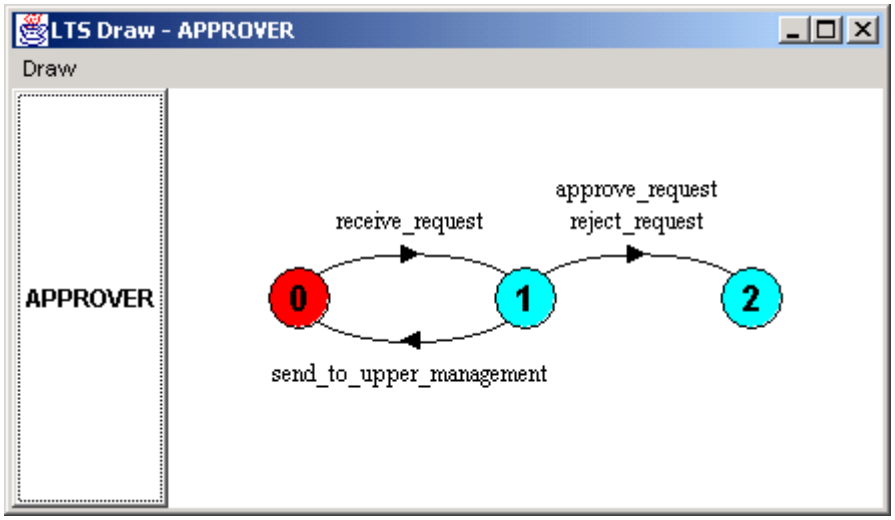
Following is the state diagram as output by the LTSA tool.



- Figure 1, Approver State Diagram

Another approach is to model the system as follows, assuming that the APPROVER just processes one request. State 2 represents the "STOP" or completed state.

APPROVER = (receive_request ->
(send_to_upper_management ->APPROVER|
reject_request->STOP|
approve_request ->STOP)
).



2.2 Traveler's Viewpoint

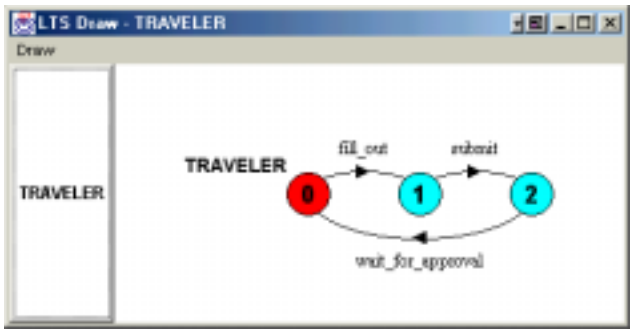
From the viewpoint of the traveler, things are much simpler. He simply creates a request, submits the request, then waits for an answer. The corresponding transitions are then:

<fill_out, submit, wait_for_approval>

This is modeled in LTSA as follows:

TRAVELER = (fill_out->submit->wait_for_approval->TRAVELER).

The resulting state diagram follows.



- Figure 2, Traveler State Diagram