Model Based Design of Reliable and Efficient Services

Yi Huang, Michigan State University

Advisors: L. K. Dillon and R. E. K. Stirewalt

Collaborators & supporters: AT&T Research Labs, LogicBlox Inc., Sun Labs
Overview

Motivating problem

Model-based solution

Discussion
Services
Services

HTTP
Database
RPC
VoIP
...

CSE 914 Fall 2010
Messaging

Synchronous  Asynchronous
Messaging

Synchronous  Asynchronous

Wait
Messaging

Synchronous

Asynchronous

Wait
Messaging

Synchronous

Wait

Easier to program

Asynchronous

More efficient
Why programming asynchronous services is hard
Why programming asynchronous services is hard
Why programming asynchronous services is hard
Why programming asynchronous services is hard
So, in sum

Sources of complexity in programming asynchronous services

• Need to split service logic and keep track of service proceeding progress
• Need to synchronize conflicting message processing
So, in sum

Sources of complexity in programming asynchronous services

- Need to split service logic and keep track of service proceeding progress
- Need to synchronize conflicting message processing
- The above “needs” complicate each other
Model-based solution
Model-based solution

Implement both “needs” in low level programs
Model-based solution

Implement both “needs” in low level programs

Design high-level models that hide handling of both “needs”
Model-based solution

Implement both “needs” in low level programs

Automatically generate both “needs”

Design high-level models that hide handling of both “needs”
State models

Suitable for designing asynchronous services

- A state => a milestone of servicing progress
- A transition => processing of a message
- A transition event => arrival of a message
**State models**

**Suitable for designing asynchronous services**
- A state => a milestone of servicing progress
- A transition => processing of a message
- A transition event => arrival of a message

**A “black hole” SIP service**
Multi-tiered state models
Multi-tiered state models

Generic model

Abstraction of basic messaging primitives (protocol-neutral, basic synchronization)
Multi-tiered state models

Specialized model

Abstraction of common protocol routines (protocol-specific, basic synchronization)

Generic model

Abstraction of basic messaging primitives (protocol-neutral, basic synchronization)
Multi-tiered state models

- **Specialized model**
  - Abstraction of common protocol routines (protocol-specific, basic synchronization)
  - Add-ons to support model composition (advanced synchronization)

- **Generic model**
  - Abstraction of basic messaging primitives (protocol-neutral, basic synchronization)
Generic state model

Idea: Abstract basic messaging primitives

Highlight: Protocol neutral

Synchronization

• Atomic intra-transition

• Serialized inter-transition
Generic state model

Idea: Abstract basic messaging primitives

Highlight: Protocol neutral

Synchronization

• Atomic intra-transition
  • Within a transition, activity is performed atomically
• Serialized inter-transition
  • Within a machine instance, transitions are fired serially
Generic model formalization

A generic model \( M \) is a set of transitions

A transition is in form of \( (src, \text{event}, \text{condition}, \text{activity}, tgt) \), where

- \( src \) and \( tgt \) are source and target states, respectively
- \( \text{event} \) signifies reception of a message: \( \text{channel}\text{?message} \)
- \( \text{condition} \) is predicate over \( \text{event} \)'s \( \text{channel} \) and \( \text{message} \)
- \( \text{activity} \) is sequence of message-sending \( \text{actions} \): \( \text{channel}\text{!message} \)
Generic model execution semantics

Let $Q$ be a queue of messages

Let $n$ be initial state

Then, the set of sequences of observable actions of model $M$ is $Trace(M, n, Q)$, where

$$Trace(M, s, Q) = \{ <atomic(t.activity)>::L \mid t \in M$$

$$\wedge t.src=s$$

$$\wedge Q=<m>::Q'$$

$$\wedge t.event=c?m$$

$$\wedge t.condition$$

$$\wedge L \in Trace(M, t.tgt, Q') \}$$
Generic model graphical notation

States are circles with names
- Special case: Initial state is black dot without name
- No explicit final states
- No nested-states

Transitions are arrows between states with labels:
  event [condition] / activity
- condition defaults True
- activity defaults nop
Generic model example: music pausing service

- Customer
- Pausing service
- Music server
- Agent
Generic model example: music pausing service
Generic model example: music pausing service
Generic model example: music pausing service

Customer → Pausing service → Agent

Customer

Pausing service

Music server

Agent
Generic model example: music pausing service
Generic model example: music pausing service

Customer

Pausing service

Music server

Agent
Generic model example: music pausing service
Generic model example: music pausing service

Customer

Pausing service

Music server

Agent

CSE 914 Fall 2010
Generic model example: music pausing service

CSE 914 Fall 2010
Observation: generic model is complex

Sources of complexity

- Model design must respect details of underlying concrete protocol
- Such details become overly tedious with multiple channels
Specialized state model

Idea: Abstract common protocol routines

Highlight: Protocol specific

Example: StratoSIP

- Abstract SIP invite & reinvite scenarios using high-level primitives
- Example high-level primitives:
  - Actions: $c1 = ctu(c0)$, $c1 = rev(c0, dest=...)$, $c = new(dest=...)$, $end(c)$
  - Events: $rcv(c)$, $succeeded(c)$, $ended(c)$
  - State annotations: $c$, $c0 \leftrightarrow c1$
Specialized model example: music pausing service

Agent, \texttt{agent=new()}, customer talking to agent, customer <-> agent

Music, \texttt{music=new()}, customer listening to music, customer <-> music

rcv(customer) / agent=new()

web?pause / music=new()

web?resume

Tuesday, December 7, 2010
Translate specialized model to generic model

Idea: Expand each high-level transition in specialized model into one or more implementing transitions in the equivalent generic model

Pitfall: Synchronization semantics must be preserved

- Atomic transition activities in specialized model must be also atomic in generic model
- Transitions in generic model that are expanded from the same transition in specialized model do not have to be fired atomically
State model translation example: StratoSIP

Let $Trans(M)$ be the equivalent generic model of specialized model $M$

- $Trans(\{ t \} \cup M') = TransSingle(t) \cup Trans(M')$
- $TransSingle(t)$ expands transition $t$ into a set of generic model transitions

$TransSingle(t=(src, asrc, rcv(c), condition, activity, atgt, tgt)) =$
\{ (src, c?invite[condition], <c!ok>, n), (n, c?ack, A0::A1, tgt) \} \cup T0 \cup T1$

$TransSingle(t=(src, asrc, succeeded(c), condition, activity, atgt, tgt)) =$
\{ (src, c?ok[condition \land t.method=invite], <c!ok>, n),
  (n, c?ack, A0::A1, tgt) \} \cup T0 \cup T1$

(...elided...)

where $n$ is a new state

$(A0, T0) = Expand(activity, t)$
$(A1, T1) = Implement(asrc, atgt, t)$
State model translation example: StratoSIP

\textit{Expand(activity, t)} expands \textit{activity} of transition \textit{t} into a tuple \((A, T)\)

- \textit{A} is a sequence of actions expanded from \textit{activity}
- \textit{T} is a set of transitions needed to add to generic model
State model translation example: StratoSIP

\textit{Implement}(\textit{asrc}, \textit{atgt}, \textit{t}) \textit{generates implementation of }\textit{t}'s\textit{ state annotation change into a tuple (A, T)}

- \textit{A} is a sequence of actions to change state annotation from \textit{asrc} to \textit{atgt}
- \textit{T} is a set of transitions needed to add to generic model
Observation on generic and specialized models
Observation on generic and specialized models

Both notations cannot efficiently model a certain class of services

- Consider services that handle large numbers of channels
- SIP examples
  - Large-scale conferences that involve hundreds of participants
  - Online chatting that allows callers to change chatting rooms
  - Tele-dating that randomly pairs dial-in users
Observation on generic and specialized models

Both notations cannot efficiently model a certain class of services

- Consider services that handle large numbers of channels
- SIP examples
  - Large-scale conferences that involve hundreds of participants
  - Online chatting that allows callers to change chatting rooms
  - Tele-dating that randomly pairs dial-in users

Synchronization makes models of these services inefficient

- Transitions are fired serially => clients must be served one after another
- Inefficient when the number of connected clients is large
Methods to improve efficiency for such services
Methods to improve efficiency for such services

**Method one: Introduce concurrent sub-machines**

- Need additional notations to support state hierarchy
- Need to typify states: AND-states, OR-states
- Disadvantages
  - Programmer has to explicitly specify concurrency
  - Programmer must keep in mind global behavior
Methods to improve efficiency for such services

Method two: Permit composition of model instances

- Need additional notations to communicate with different model instances
- No need to typify states
- Advantages
  - More consistent to the existing model notations
  - Concurrency is abstracted away from programmer
  - Programmer considers only local behavior
Notational add-ons for method two

Sending and receiving events through model instances

- $p!\text{event}$
- $p?\text{event}$

Referring to model instances

- $p\.\text{channel}$
- $p$ in $\langle\text{state name}\rangle$
Example: composition of model instances

Online dating service
Synchronization semantics
Synchronization semantics

Let \( p_0, p_1 \) be two concurrent instances of model \( M \)

Let \( t_0 \) be one of \( p_0 \)'s enabled transitions, \( t_1 \) be one of \( p_1 \)'s enabled transitions

\( t_0 \) and \( t_1 \) can be fired concurrently only if

- \( t_0 \) does not refer to \( p_1 \)
- \( t_1 \) does not refer to \( p_0 \)

Above rule can be generalized to the case having 3 or more transitions
Discussion
Discussion

Basic questions

• How helpful are the above models for designing asynchronous services?
• How much performance cost does generated code introduce?
Discussion

Basic questions

• How helpful are the above models for designing asynchronous services?
• How much performance cost does generated code introduce?

Other questions

• What other benefits does modeling services provide? Verification? Property reasoning?
• Is it possible to prove equivalency between a specialized model and a generic model?
• Can translator of specialized models be automatically generated from formal protocol specification?
Thank you!