IEEE Ninth International Conference on Research Challenges in Information Science, May 13-15 2015, Athens, Greece

Formal Modeling for Verifying SCA Composition

Lazhar Hamel, Mohamed Graiet and Mourad Kmimech

High School of Computer Science and Mathematics
Monastir University, Tunisia

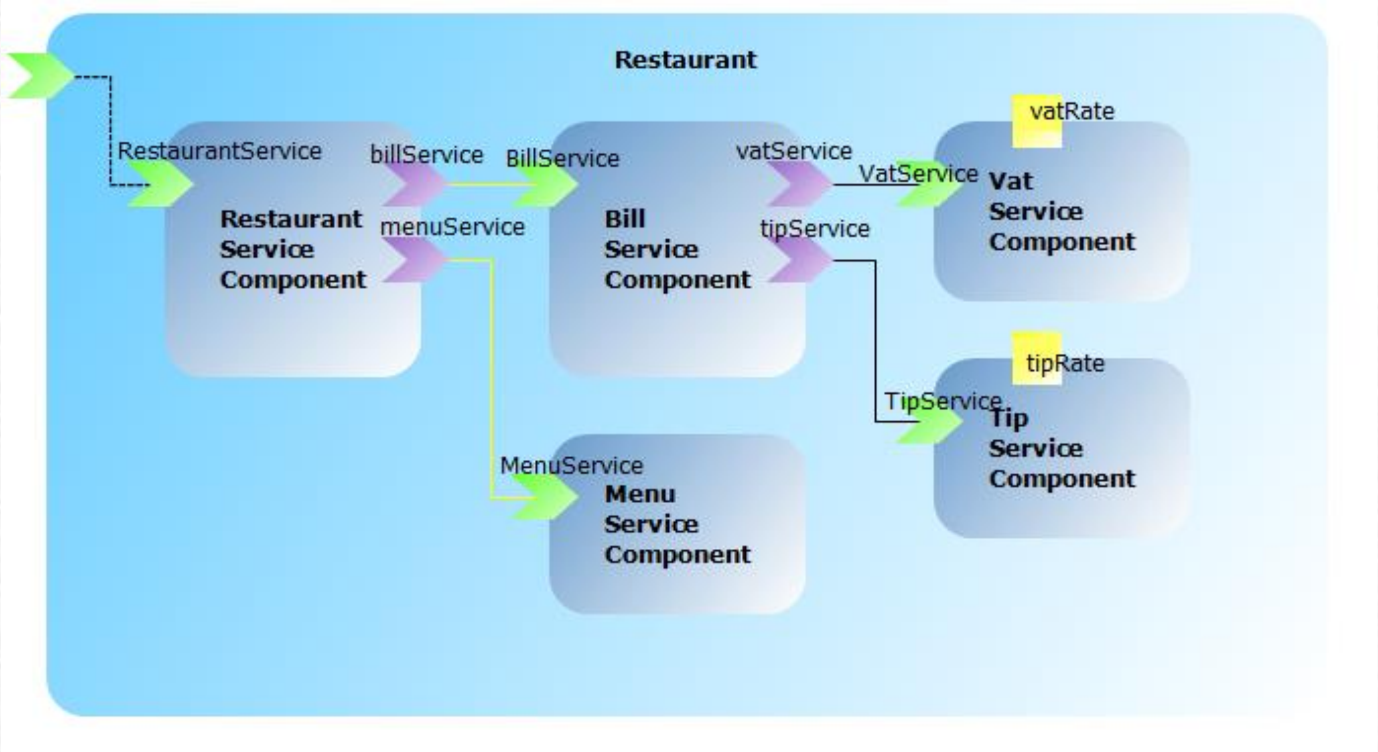
lazhar.hamel@gmail.com

Outline

I. Motivations
II. Event-B Method
III. Formal SCA behavior
IV. SCA dynamic reconfiguration
V. Modeling and Verification approach
VI. Conclusion and Perspectives.

Scope and objective(1/2)

Service Component Architecture



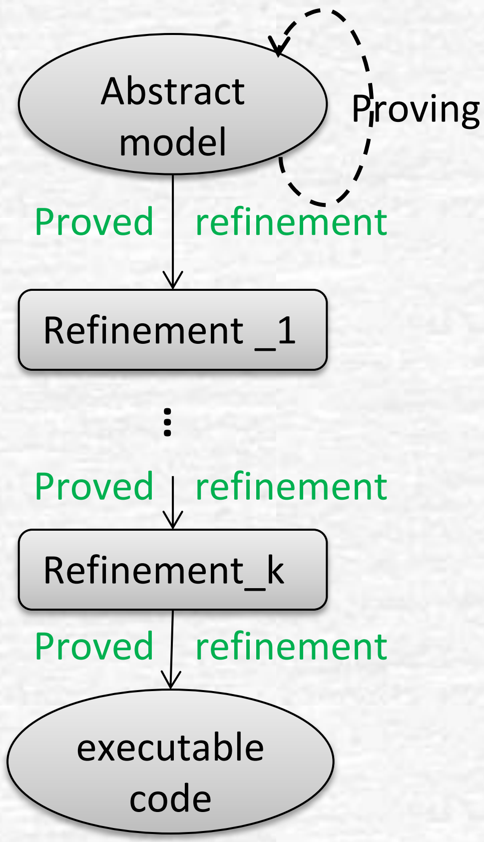
How to ensure a reliable interaction between component Services?

Scope and objective(2/2)

- Previous works (ASM, UML4SOA, etc.):
  - A complete model in order to check the composition
  - Need for a translation step to verify and validate a given composition.
  - Risks losing the semantics of such specification.
- Our approach:
  - Step1: a proof based approach for modeling service composition based on SCA specifications. The extension includes behavioral properties and the dynamic reconfiguration of composite service.
  - Step2: incrementally combine model-check and theorem proving for discharging proof obligation;
  - Step3: validate the event-B specification by using ProB animator.

➔ Proof and model-check based approach.

- formal method for modeling secure information systems.
- Set theory and first order logic.
- A full software lifecycle:
  - Specification.
  - Refinement.
  - Implementation.
- Proof obligations.



The basic concepts

- The initial specification:
  - includes a context and a machine.
  - fixes the main definition of the basic concepts on which the general specification is built.
- Modeling the SCA assembly model (Lahouij et al., 2013).
- Formal SCA behavior :
  - We reuse the services interaction patterns given by Barros and Boerger, 2005.
  - Behavioral constraints are defined to express the behavioral compatibility.
- SCA Dynamic reconfiguration.

Formal structural model for SCA

- The model includes :
  - A context and a machine to fix the vocabulary and definitions on which the general specification is
  - Sets : Composites, Components, Service, Reference, etc.
  - Variables : used to represents the composition elements.
- Some invariants as consistency
  - Wire : from a service exposed to reference that requires this service.
  - Wired services: the interface of a reference connected to a service must be an equal set or a subset of the interface provided by the service.

$$\texttt{inv\_wire\_interfaces\_compatibility}: \forall s, r \cdot r \in componentReferences \wedge s \in componentServices \wedge r \mapsto s \in Wire \Rightarrow (InterfaceOf(s) \mapsto InterfaceOf(r) \in CompatiblesInterfaces \vee InterfaceOf(s) \mapsto InterfaceOf(r) \in CompatiblesSuperset)$$

Formal SCA behavior :

- The machine includes :
  - A formalization of patterns proposed by Barros and borgoer
  - Invariants and Events to express those patterns : functions defining the current state of each message during the communications.
- In this paper we define the Event-B model for Send and receive patterns.
- Pattern send, pattern receive

Service interaction patterns

- Send pattern :
  - Send Without Guaranteed Delivery
  - Guaranteed Non-Blocking Send
  - Guaranteed Blocking Send
- Receive pattern
  - Basic receive where the recipient is ready to receive
  - Basic receive where the message has to be discarded
  - Receive where the recipient is ready to and the action request an acknowledgment
  - Receive where the recipient is ready to and the action is blocking

Service interaction patterns

- The event “BasicSend” :
  - AckRequested and BlockingSend must be always FALSE.
  - OkSend(m) and Arriving(m) setted to TRUE.
- OkSend set to TRUE refers to message correctly sent.
- Arriving(m) informs the recipient of m that the message is arriving so to be ready to receive it.

Event $BasicSend \mathrel{\widehat{=}}$
where
grd1 : $sendMode(m) = TRUE$
grd2 : $AckRequested(m) = FALSE$
grd3 : $BlockingSend(m) = FALSE$
then
act1 : $sendMode(m) := FALSE$
act2 : $OkSend(m) := TRUE$
act3 : $Arriving(m) := TRUE$
end

Service interaction patterns

- SendAckNonBlocking :
  - AckRequested to TRUE and BlockingSend must be always FALSE.
  - OkSend(m) and Arriving(m) setted to TRUE.
- Set WaitingForAck(s) to TRUE, sendTime(m) := CurrentTime, and deadline := 3.

```
Event  SendAckNonBlocking ≙
  where
    grd1 : AckRequested(m) = TRUE
    grd2 : BlockingSend(m) = FALSE
    grd3 : m ↦ s ∈ sender
  then
    act1 : WaitingForAck(s) := TRUE
    act2 : sendTime(m) := CurrentTime
    act3 : deadline(m) := 3
  end
```

Service interaction patterns

- Basic receive where the recipient is ready to receive :
  - ReadyToReceive(m) is set to TRUE, AckRequested(m) is set to FALSE and BlockingSend(m) too.
  - Consume(m) is set to TRUE.
- Consume event
  - the message m is added to the set of received messages of recipient(m)
  - the data contained in m is added to the local data of recipient(m)

Event $BasicReceiveReady \mathrel{\widehat{=}}$
where
grd1 : $Arriving(m) = TRUE$
grd2 : $ReadyToReceive(m) = TRUE$
grd3 : $AckRequested(m) = FALSE$
grd4 : $BlockingSend(m) = FALSE$
then
act1 : $Consume(m) := TRUE$
end

Event $Consume \mathrel{\widehat{=}}$
where
grd1 : $m \mapsto s \in sender$
grd2 : $Consume(m) = TRUE$
grd3 : $m \mapsto r \in recipient$
then
act1 : $receivedMessages(r) := (receivedMessages(r) \cup \{m\})$
act2 : $receivedDatas := (receivedDatas) \cup \{d \cdot d \in \{DatasOfmessage(m)\} | r \mapsto d\}$
end

Behavioral compatibility

- Each service of component and reference of component, having a wire relation, must be protocol compatibles
- Two protocols are said to be compatibles if they have no unspecified receptions and they are deadlock-free

Inv_wire_protocols_compatibility : $\forall s, r \cdot s \in componentServices \land r \in componentReferences \land s \mapsto r \in Wire \Rightarrow protocolOfService(s) \mapsto protocolOfService(r) \in compatibleProtocols$

Inv_protocols_compatibility_check : $\forall p1, p2 \cdot p1 \mapsto p2 \in compatibleProtocols \Rightarrow p1 \mapsto p2 \in NoUnspecifiedReception \land p1 \mapsto p2 \in DeadlockFree$

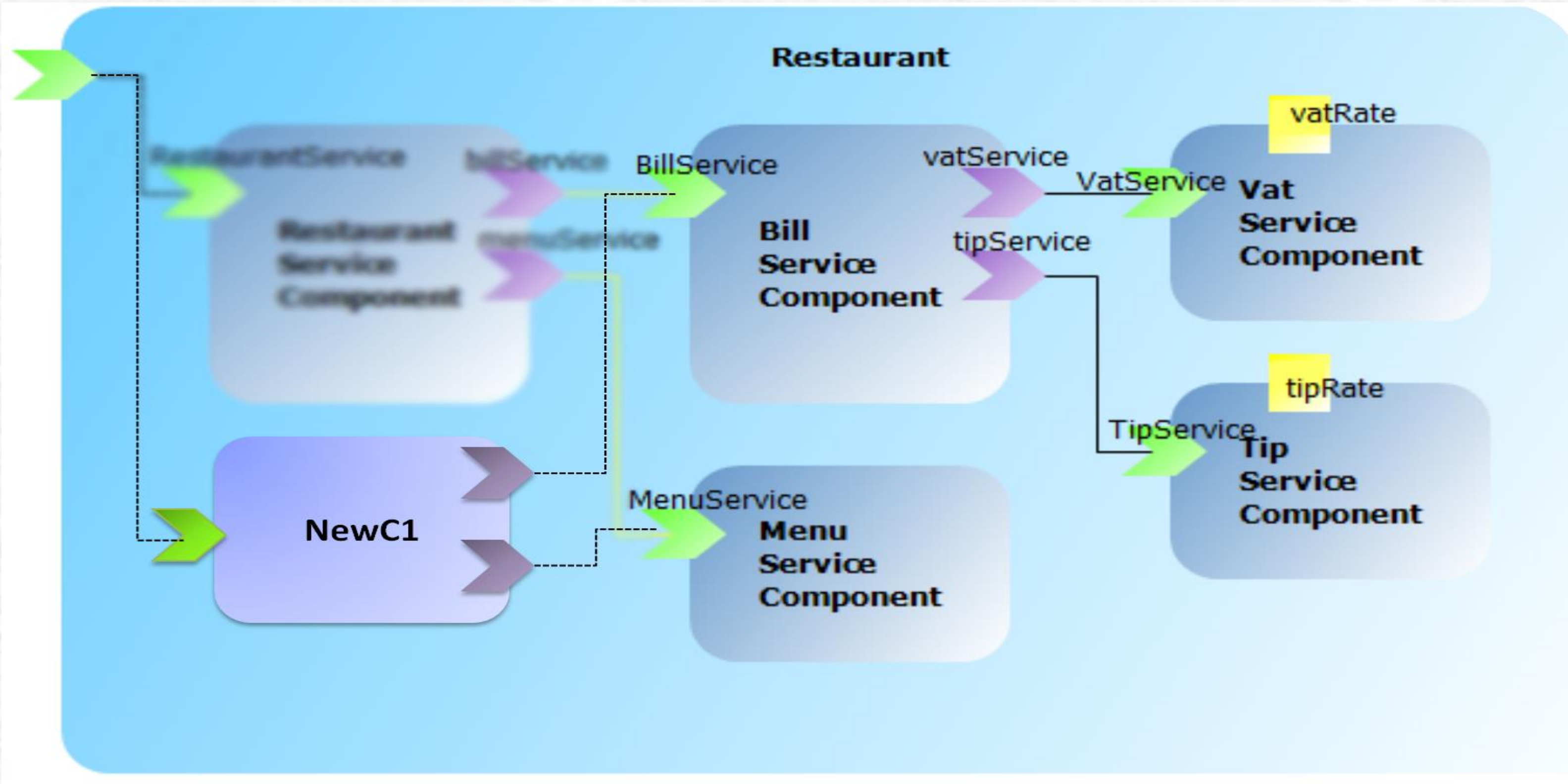
Inv_Unspecified_reception : $\forall p1, p2 \cdot p1 \mapsto p2 \in NoUnspecifiedReception \Rightarrow (\forall i \cdot p1 \mapsto i \in interactionsOfP \land i = send \Rightarrow ((stateOfP(p1) = sendState) \land (stateOfP(p2) = receiveState)) \land sendMessage(p1) = TRUE \land receiveMessage(p2) = TRUE \land sendAck(p2) = TRUE \land receiveAck(p1) = TRUE)$

Inv_Deadlock_Free : $\forall p1, p2 \cdot p1 \mapsto p2 \in DeadlockFree \Rightarrow (\forall i1, i2, m1, m2 \cdot p1 \mapsto i1 \in interactionsOfP \land p2 \mapsto i2 \in interactionsOfP \land time(i1) = time(i2) \land i1 = send \land i2 = send \land MessageOfInteraction(i1) = m1 \land MessageOfInteraction(i2) = m2 \land PriorityOfMessage(m1) \geq PriorityOfMessage(m2) \Rightarrow sendMessage(p1) = TRUE \land wait(p2) = TRUE \land time(i2) = time(i2) + 1)$

Dynamic reconfiguration

- Service substitution.
- Component substitution
- For a service substitution (oldS by news) :
  - Select a service which its interface and the olds interface are structural and behavioral compatible.
  - A second selection based on non-functional properties (for our case a score selection)
  - After those three selection the service substitution is triggred

Dynamic reconfiguration



The composition after substituting a component

- Verification approach:
  - Step 1: Writing an event B model
  - Step 2: Discharging proof obligations
  - Step 3: Validation.
- The verification activity is based on:
  - Proofs of theorems.
  - Model-checking.
- We use for the specification and verification :
  - The Rodin platform.
  - ProB animator.

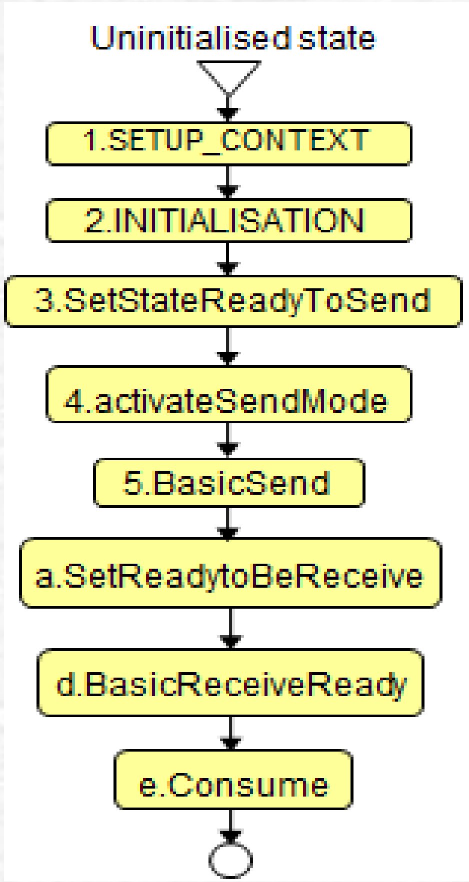
Discharging proof obligations

- Proof obligations automatically discharged.
- Proof obligations interactively discharged.
- proof obligations can't be interactively discharged .

ProB to easily validate several undischarged proof obligations.

The trace of a scenario with ProB

- Set the state of the MenuService to ready to send
- Choose arbitrarily a message to send from the MenuService operations messages
- Activate the send mode.
- Set the service to ready to receive
- Receive and consume the message.



Conclusion :

- An approach for modeling SCA composition : a formal behavioral modeling and formal dynamic reconfiguration model

- Complementarity of proof and model-checking.

Perspectives :

- Extending our approach by formalizing multi-directional patterns.

- Integrate those concept in our eclipse plug-in SCA2B

THANK YOU

Questions ?