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
I. Motivations
II. Event-B Method
III. Formal SCA behavior
IV. SCA dynamic reconfiguration
V. Modeling and Verification approach
VI. Conclusion and Perspectives.
How to ensure a reliable interaction between component Services?
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 represent 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.

\[
\text{inv\_wire\_interfaces\_compatibility} : \forall s, r \cdot r \in \text{componentReferences} \land s \in \text{componentServices} \land r \rightarrow s \in \text{Wire} \Rightarrow (\text{InterfaceOf}(s) \rightarrow \text{InterfaceOf}(r) \in \text{CompositesInterfaces} \lor \text{InterfaceOf}(s) \rightarrow \text{InterfaceOf}(r) \in \text{CompositesSuperset})
\]
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.
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.
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)
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 : \( \forall s, r \cdot s \in \text{componentServices} \land r \in \text{componentReferences} \land s \rightarrow r \in \text{Wire} \Rightarrow \text{protocolOfService}(s) \leftrightarrow \text{protocolOfService}(r) \in \text{compatibleProtocols} \)

Inv_protocols_compatibility_check : \( \forall p_1, p_2 \cdot p_1 \leftrightarrow p_2 \in \text{compatibleProtocols} \Rightarrow p_1 \rightarrow p_2 \in \text{DeadlockFree} \)

Inv_Unspecified_reception : \( \forall p_1, p_2 \cdot p_1 \leftrightarrow p_2 \in \text{NoUnspecifiedReception} \Rightarrow (\forall i \cdot p_1 \leftrightarrow i \in \text{interactionsOfP} \land i = \text{send} \Rightarrow ((\text{stateOfP}(p_1) = \text{sendState}) \land (\text{stateOfP}(p_2) = \text{receiveState})) \land \text{sendMessage}(p_1) = \text{TRUE} \land \text{receiveMessage}(p_2) = \text{TRUE} \land \text{sendAck}(p_2) = \text{TRUE} \land \text{receiveAck}(p_1) = \text{TRUE} \land \text{TRUE}) \)

Inv_Deadlock_Free : \( \forall p_1, p_2 \cdot p_1 \leftrightarrow p_2 \in \text{DeadlockFree} \Rightarrow (\forall i_1, i_2, m_1, m_2 \cdot p_1 \leftrightarrow i_1 \in \text{interactionsOfP} \land p_2 \leftrightarrow i_2 \in \text{interactionsOfP} \land \text{time}(i_1) = \text{time}(i_2) \land i_1 = \text{send} \land i_2 = \text{send} \land \text{MessageOfInteraction}(i_1) = m_1 \land \text{MessageOfInteraction}(i_2) = m_2 \land \text{PriorityOfMessage}(m_1) \geq \text{PriorityOfMessage}(m_2) \Rightarrow \text{sendMessage}(p_1) = \text{TRUE} \land \text{wait}(p_2) = \text{TRUE} \land \text{time}(i_2) = \text{time}(i_2) + 1 ) \)
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 triggered
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 ?