Egon Börger (Pisa)

An Architecture for Web Service Mediation and Discovery

Università di Pisa, Dipartimento di Informatica, I-56127 Pisa, Italy
boerger@di.unipi.it

In collaboration with
Michael Altenhofen, Andreas Friesen and Jens Lemcke
SAP Research, Karlsruhe, Germany

Goal

Provide a Programming Language Independent Precise Mediation Model

for mediation between message-based interactions of heterogeneous systems. We want the model to be 'designed for change':

- refinable (instantiatable) to current mediation concepts
- offering accurate practical composition concepts
- providing a basis for defining rigorous equivalence notions supporting
  - discovery algorithms and service selection procedures in real-life applications
  - proofs of properties of interest in complex mediation schemes
- offering abstractions for both data and data transformations (abstract *state* and abstract *behavior*) that go beyond pure message sequencing or control flow analysis
- adaptable to different underlying communication mechanisms

The Method: using Machines operating on Abstract States

- within a single *precise yet simple conceptual framework*

the ASM method naturally supports and uniformly links the major activities occuring during the software life cycle:

- **requirements capture** by constructing rigorous *ASM ground models*, i.e. accurate concise high-level system blueprints (contracts)
- **architectural and component design** bridging the gap between specification and code by *piecemeal, systematically documented detailing* of abstract models via intermediate models to code (*general ASM refinement notion*)
- **validation** of models by their tool-supported *simulation*
- **verification** of model properties by tool-supported *proof techniques*
- **documentation** for *inspection*, *reuse* and *maintenance* by providing, through the intermediate models and their analysis, explicit descriptions of the *software structure* and of the major *design decisions*

Variety of applications of ASMs (1)

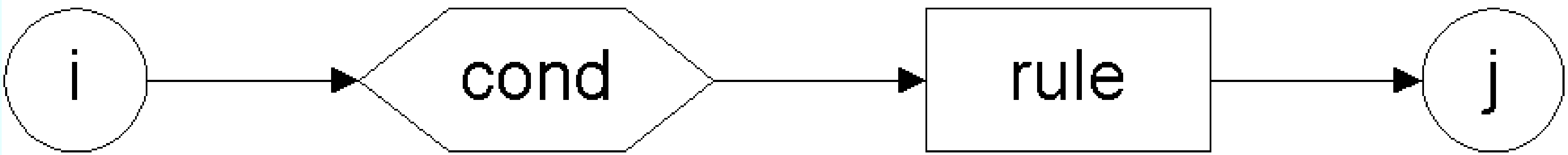
- **industrial standards**: *ground models* for the standards of
  - OASIS for Business Process Execution Language for Web Services
  - ECMA for C#
  - ITU-T for SDL-2000
  - IEEE for VHDL93
  - ISO for Prolog
- **design, reengineering, testing of industrial systems**:
  - railway and mobile telephony network component software at Siemens
  - fire detection system in German coal mines
  - implementation of behavioral interface specifications on the .NET platform and conformence test of COM components at Microsoft
  - compiler testing and test case generation tools

Variety of applications of ASMs (2)

- **programmming languages**: definition and analysis of the semantics and the implementation for the major real-life programmming languages, among many others for example
  - SystemC
  - Java/JVM (including bytecode verifier)
  - domain-specific languages used at the Union Bank of Switzerland

  including the verification of numerous compilation schemes and compiler back-ends
- architectural design: verification (e.g. of pipelining schemes or of VHDL-based hardware design at Siemens), architecture/compiler co-exploration
- protocols: for authentication, cryptography, cache-coherence, routing-layers for distributed mobile ad hoc networks, group-membership etc.
- modeling e-commerce and web services (at SAP)

ASMs = FSMs with Abstract States



$\textbf{if } ctl\_state = i \textbf{ then}$

$\quad \textbf{if } cond \textbf{ then}$

$\qquad\quad rule$

$\qquad\quad ctl\_state := j$

$\textbf{where } cond \equiv input = a \qquad rule \equiv output := b$ for FSM

ASMs use *parameterized locations* and *first-order conditions*:

- rule = set of updates $f(t_1, \ldots, t_n) := t$
- cond = arbitrary first-order formula

Basic Request Structure: Seq/Par Trees

- each arriving request viewed as root of a *seq/par tree* of subrequests, forwarded to and answered by subproviders
- subrequests (seq-subtree nodes) can be elaborated in sequence
  - forwarded to and to be answered by subproviders before proceeding to the next subrequest, until the final answer can be compiled
- subrequests may consist of multiple independent subsubrequests (par-subtree nodes)
- next sequential subrequest may depend on received answers to the subsubrequests of the current sequential subrequest

Nestings of such alternating seq/par trees and other more sophisticated hierarchical subrequest structures can be obtained by appropriate compositions of VPs.

Separating Tree Processing and Communication

VP defined as interface with five methods:

- ReceiveReq for receiving request messages from clients
- SendAnsw for sending answer messages back to clients
- Process to handle ReceivedRequests via the seq/par tree of auxiliary subrequests and answers received for them
- SendReq for sending request messages to (sub-) providers
- ReceiveAnsw for receiving answer messages from (sub-) providers

MODULE VirtualProvider =

ReceiveReq

SendAnsw

Process

SendReq

ReceiveAnsw

SEND/RECEIVE Machines (Abstract Msgg Passing)

$\textsc{ReceiveReq}(\mathit{inReqMssg}, \mathit{ReqObj}) =$

if *ReceivedReq*$(\mathit{inReqMssg})$ **then**

$\textsc{CreateNewReqObj}(\mathit{inReqMssg}, \mathit{ReqObj})$

where $\textsc{CreateNewReqObj}(m, R) =$

let $r = \mathit{New}(R)$ **in** $\textsc{Initialize}(r, m)$

$\textsc{SendAnsw}(\mathit{outAnswMssg}, \mathit{SentAnswToMailer}) =$

if *SentAnswToMailer*$(\mathit{outAnswMssg})$ **then** $\textsc{Send}(\mathit{outAnswMssg})$

$\textsc{SendReq}(\mathit{outReqMssg}, \mathit{SentReqToMailer}) =$

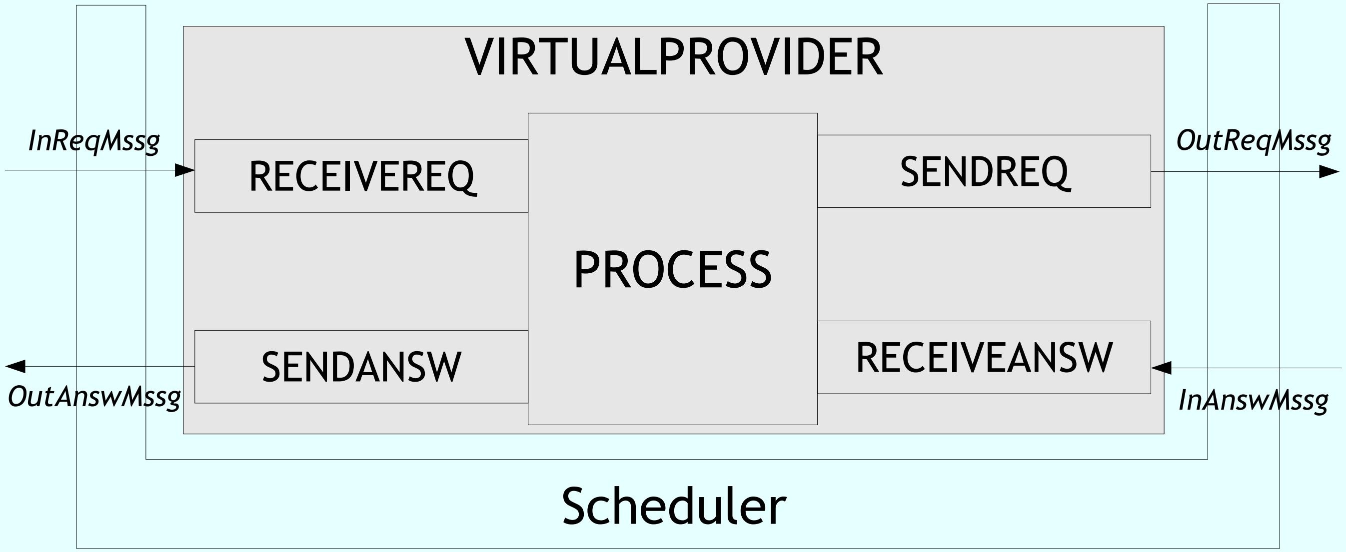
if *SentReqToMailer*$(\mathit{outreqMssg})$ **then** $\textsc{Send}(\mathit{outReqMssg})$

$\textsc{ReceiveAnsw}(\mathit{inAnswMssg}, \mathit{AnswerSet}) =$

if *ReceivedAnsw*$(\mathit{inAnswMssg})$ **then**

insert $\mathit{answer}(\mathit{inAnswMssg})$ into

$\mathit{AnswerSet}(\mathit{requestor}(\mathit{inAnswMssg}))$



Compositional VP Architecture

Sequential composition $VP_1 \ldots VP_n$ by connecting the communication interfaces:

- $\textsc{SendReq}$ of VP_i to $\textsc{ReceiveReq}$ of VP_{i+1}
  – data mediation bw VP_i-*OutReqMssg* and VP_{i+1}-*InReqMssg*
- $\textsc{SendAnsw}$ of VP_{i+1} to $\textsc{ReceiveAnsw}$ of VP_i
  – data mediation bw VP_{i+1}-*OutAnswMssg* and VP_i-*InAnswMssg*

Composing VP Mediator Structures: Example

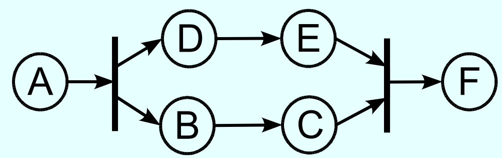


Fig. 0.1.

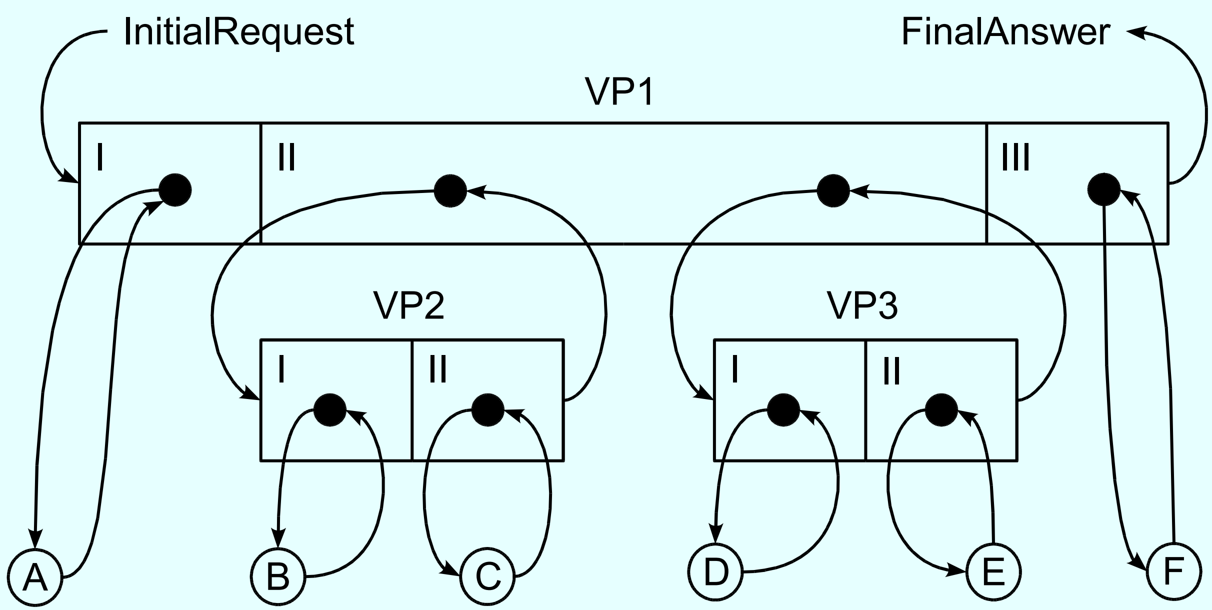
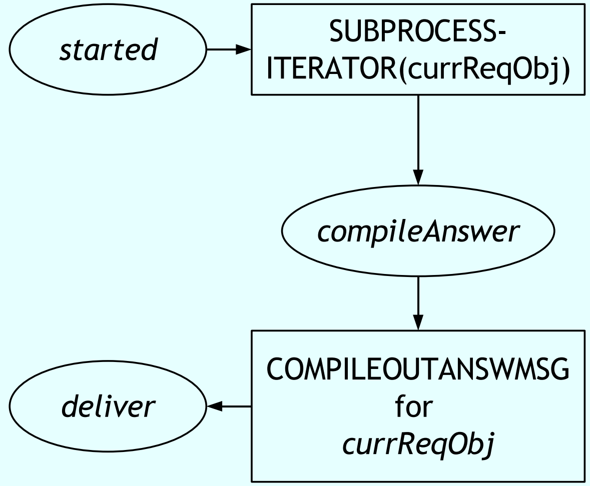


Fig. 0.2.



The core PROCESS(currReqObj) machine

- currReqObj yields a **sequence** of **SubRequests**, to be elaborated by an *Iterator* on *SeqSubReq(currReqObj)*
- **AnswMsg** to the **currReqObj**ect is compiled from the *AnswerSet(seqReq)* of all answers collected from the subrequests

COMPILEOUTANSWMSG for $o =$

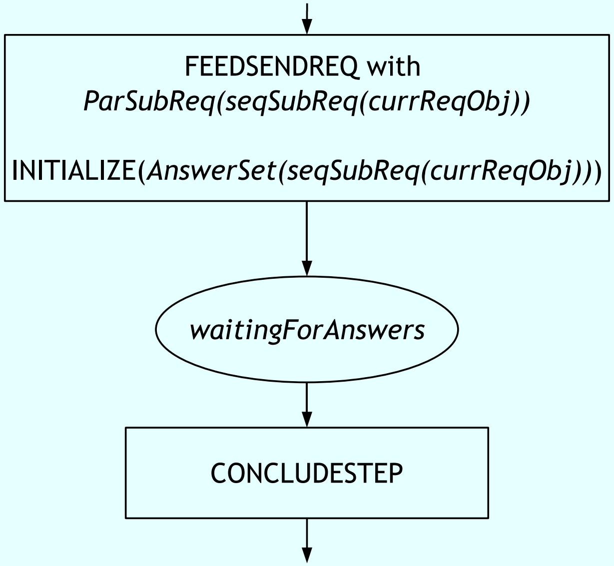
if $AnswToBeSent(o)$ **then**

$SentAnswToMailer(\textsf{outAnsw2Msg}(outAnswer(o))) := true$

Elaboration of Sequential Subrequests: SubProcessIterator

$\textsc{SubProcessIterator}(currReqObj) =$
$\quad \textsc{InitializeIterator}(currReqObj)$ **seq**
$\quad \textsc{IterateSubReqProcessg}(currReqObj)$ **until**
$\qquad FinishedSubReqProcessg$
$\quad$ **where**
$\qquad yes(FinishedSubReqProcessg) = compileAnswer$
$\qquad no(FinishedSubReqProcessg) =$
$\qquad\quad initStatus(\textsc{IterateSubReqProcessg})$

Realizes the sequential part of the hierarchical VP request processing view: each incoming (top level) request object $currReqObj$ triggers the sequential elaboration of a finite number of immediate subrequests, members of a set $SeqSubReq(currReqObj)$



Elaboration of Parallel Subrequests: IterateSubReqProcessg

- each sequential **SubReq**uest triggers *forwarding* finitely many independent *parallel SubRequests* and *waitingForAnswers*
- ReceivedAnswers are collected in the *AnswerSet(seqSubReq)*
- until AllAnswersReceived triggers PROCEEDing to NextSubRequest

$$\text{FEEDSENDREQ with } ParSubReq(seqSubReq) =$$
$$\textbf{forall } s \in ParSubReq(seqSubReq)$$
$$SentReqToMailer(\textsf{outReq2Msg}(s)) := true$$

Submachine Macros

$\textsc{ConcludeStep} =$

if $AllAnswersReceived$ **then**

$\quad \textsc{ProceedToNextSubReq}$

$\quad status(currReqObj) :=$

$\quad\quad Nxt(status(currReqObj))$

where $Nxt(waitingForAnswers) =$

$\quad testStatus(FinishedSubReqProcessg)$

$AllAnswersReceived =$

let $seqSubReq = seqSubReq(currReqObj)$ **in**

$\quad$ for each $req \in ToBeAnswered(ParSubReq(seqSubReq))$

$\quad\quad$ there is some $answ \in AnswerSet(seqSubReq)$

$\textsc{Initialize}(AnswerSet(seqSubReq)) =$

$\quad AnswerSet(seqSubReq) := \emptyset$

Adapting Standard Iterator Pattern to *SeqSubReq*

$\text{InitializeIterator}(currReqObj) =$

$\quad \textbf{let } r = FstSubReq(SeqSubReq(currReqObj)) \textbf{ in}$

$\qquad seqSubReq := r$

$\qquad ParSubReq(r) := FstParReq(r, currReqObj)$

$FinishedSubReqProcessg =$

$\quad seqSubReq(currReqObj) = Done(SeqSubReq(currReqObj))$

$\text{ProceedToNextSubReq} = \textbf{let}$

$\quad o = currReqObj$

$\quad s = NxtSubReq(SeqSubReq(o), seqSubReq(o), AnswerSet(o)) \textbf{ in}$

$\qquad seqSubReq(o) := s$

$\qquad ParSubReq(s) := NxtParReq(s, o, AnswerSet(o))$

$NxtSubReq$ and $NxtParReq$ *may depend on answers accumulated so far*

Analysis of Mediators

- Definition of *ServiceBehavior*

  $ServiceBehavior(VP) =$
  $\{(inReqMssg, outAnswerMssg) \mid$
  $originator(outAnswerMssg) = inReqMssg\}$

  – $originator$ is retrievable by $\textsc{CompileOutAnswMssg}$ from $currReqObj$ if recorded as part of $\textsc{Initialize}$ by $\textsc{CreateNewReqObj}(inReqMssg, ReqObj)$

- Definition of *Service Equivalence*

  $VP \equiv VP'$ iff
  $ServiceBehavior(VP) \equiv ServiceBehavior(VP')$

  where the equivalence of ServiceBehavior can be defined in terms of message contents extracted from *InReqMssg* and *OutAnswMssg*

  – opens space for practical, not syntax-based but content-driven semantical $\equiv$-notions

Refinement of Mediators: A Simple Example

- Refine VP by internal state component
  - for recording request and answer data:

$$
\begin{array}{l}
\textsc{ReceiveReq}(inReqMssg) = \\
\quad \textbf{if } ReceivedReq(inReqMssg, ReqObj) \textbf{ then} \\
\quad\quad \textbf{if } \textit{NewRequest}(inReqMssg) \textbf{ then} \\
\quad\quad\quad \textsc{CreateNewReqObj}(inReqMssg, ReqObj) \\
\quad\quad \textbf{else} \\
\quad\quad\quad \textbf{let } r = \textit{prevReqObj}(inReqMssg) \textbf{ in} \\
\quad\quad\quad\quad \textsc{RefreshReqObj}(r, inReqMssg)
\end{array}
$$

NB. This is a simple (but frequently occurring) case of the general *ASM refinement* concept.

Refinement of VP for Semantic Web Service Discovery

- concept instantiations (data refinement)
- rule extensions

Concept instantiation: changing "view" of the abstractions from requests/answers to goals/webservices, formally resulting in the following substitutions:

- $Req \rightarrow Goal$
- $Answ,\ Answer,\ AnswerSet \rightarrow \{SetofWS, WS\}$
- $\textsc{Process} \rightarrow \textsc{ProcessGoal}$
- $ParSubReq(seqSubReq(currReqObj)) \rightarrow ParGoalQuery(currGoalObj)$
- $SentReqToMailer \rightarrow SentGoalToProvider$ (in $\textsc{SendGoal}$)
- $SentAnswToMailer \rightarrow SentSetOfWSToRequestor$ (in $\textsc{SendSetOfWS}$)
- Reducing SubReqSeq to *Singleton* determined by currReqGoal

Extending VP ReceiveGoal for DiscoveryServiceProvider DSP

$\textsc{ReceiveGoal}(inGoalMsg, GoalObj) =$

$\quad$ **if** $ReceivedGoal(inGoalMsg)$ **then**

$\quad\quad \textsc{CreateNewGoalObj}(inGoalMsg, GoalObj)$

$\quad$ **where**

$\quad\quad \textsc{CreateNewGoalObj}(m, R) =$

$\quad\quad\quad$ **let** $g = new(R)$ **in**

$\quad\quad\quad\quad \textsc{Initialize}(g, m)$

$\quad\quad\quad\quad \textsc{Initialize}(SetOfWS(g))$

$\quad\quad\quad\quad$ **if** $NewGoal(g, m)$ **then**

$\quad\quad\quad\quad\quad status(g) := started$

$\quad\quad\quad\quad$ **else**

$\quad\quad\quad\quad\quad status(g) := loopDetected$

$\quad\quad \textsc{Initialize}(SetOfWS(g)) = (SetOfWS(g) := \emptyset)$

Extending ProcessGoal for DiscoveryServideProvider DSP

Detection of loops (receiving a request for an already processed goal) to guarantee that no goal query is serviced twice

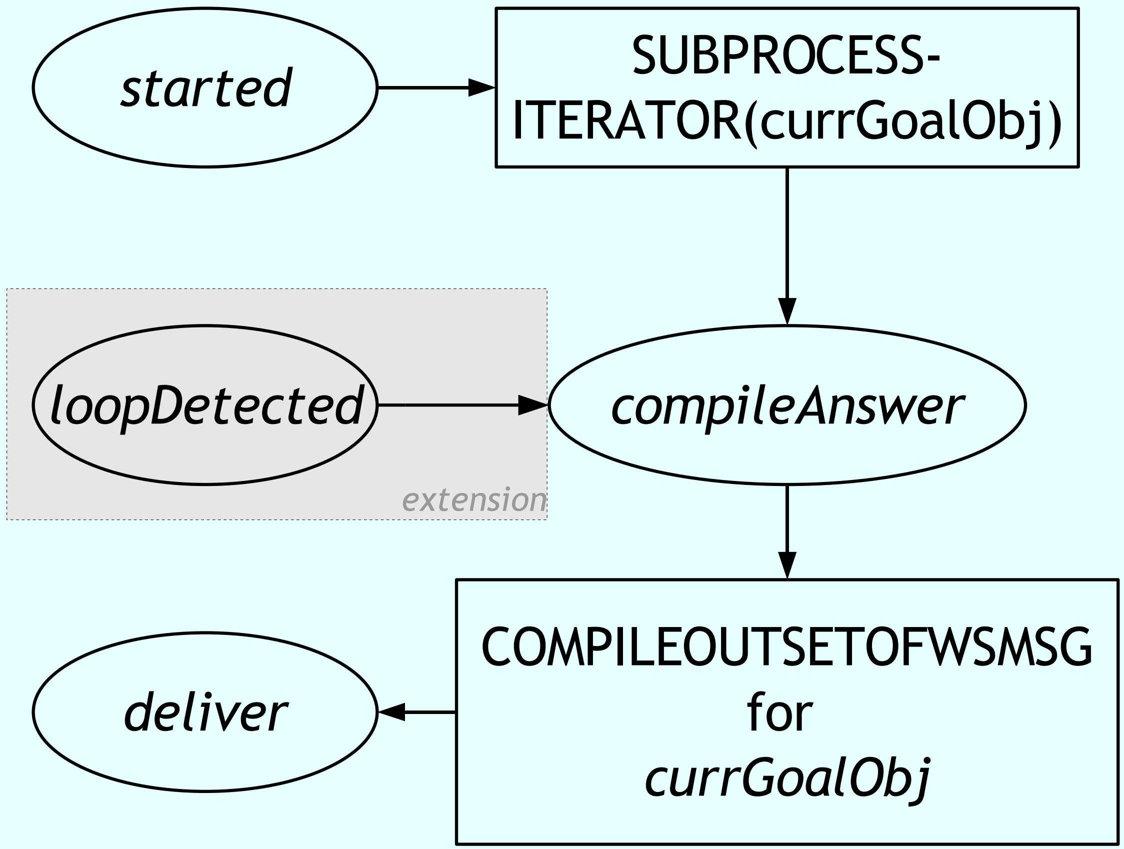


Fig. 0.3.

Refined IterateSubReqProcessg for DSP

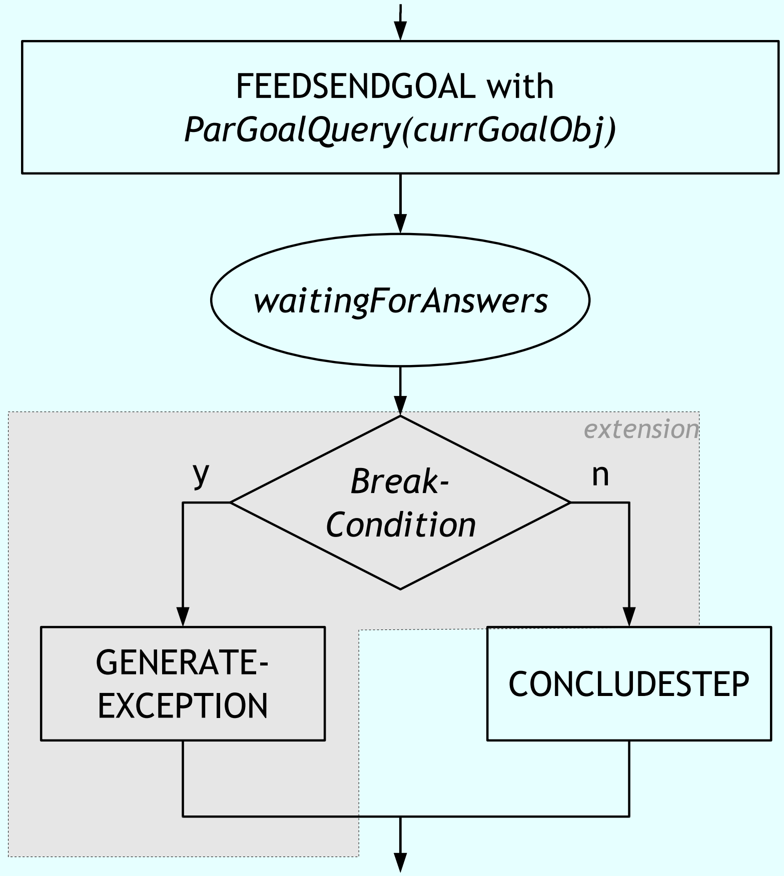


Fig. 0.4.

Typical BreakCondition: timeout. SubReqSeq reduces to singleton, reducing SUBPROCESSITERATOR

Discovery Engine

$\text{DiscoveryEngine} =$
$\quad \textbf{choose } M \in \{\text{ReceiveGoal}, \text{SendSetOfWS}\} \cup$
$\qquad \{\text{MatchGoal}\}$
$\qquad\quad M$

Interface with three main methods:

- ReceiveGoal for receiving goal queries from a requestor DSP
- SendSetOfWS for sending sets of found Web services back to the associated DSP
- MatchGoal to handle *ReceivedGoal*s (elements of a set $GoalObj$ of internal representations of received goals, say as goal objects), typically by filtering and matching the locally available set of Web services to service the currently handled goal request $currGoalObj$

MatchGoal submachine

Goal: stepwise reduction of the initial set $inSetOfWS$ of Web services to the final set of goal matching Web services, which is sent to DSP

$\textsc{MatchGoal}(currGoalObj) =$

if $status(currGoalObj) = started$ **then**

$\textsc{Prefiltering}(currGoalObj)$

seq $\textsc{SemanticMatchmaking}(currGoalObj)$

seq $\textsc{QoSMatchmaking}(currGoalObj)$

seq

$\textsc{CompileOutSetOfWSMsg}$ from $currReqObj$

$status(currGoalObj) := deliver$

$\textsc{Prefiltering}$, $\textsc{SemanticMatchmaking}$ and $\textsc{QoSMatchmaking}$ can be further and independently refined to implement different filtering and matchmaking methods or strategies.

Applications and Future Work

- Evaluate competing approaches in terms of the VP model abstractions
- Implement a VP platform as mediation pattern
- Analyse impact on VP of more general communication patterns
  - ReceiveReq and SendAnsw: basic bilateral service interaction patterns
  - FeedSendReq with SendReq: instance of basic multilateral mono-agent service interaction pattern OneToManySend
  - ReceiveAnsw until *AllAnswersReceived*: instance of basic multilateral mono-agent OneFromManyReceive pattern
- Formulate and prove properties for practical VP instances

References

- M. Altenhofen and E. Börger and J. Lemcke: An Abstract Model for Process Mediation
  - Proc. ICFEM 2005, Springer LNCS 3785, pp. 81-95
- M. Altenhofen and E. Börger and A. Friesen and J. Lemcke: An High-Level Specification for Virtual Providers
  - International J. for Business Process Integration Management 2006
- A. Barros and E. Börger: A compositional framework for service interaction patterns and communication flows
  - Proc. ICFEM 2005, Springer LNCS 3785, pp. 5-35
- E. Börger: The ASM Refinement Method
  - Formal Aspects of Computing 15:237-257, 2003.
- E. Börger and R. F. Stärk: Abstract State Machines
  - Springer 2003. pp.X+438.

