The 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC), Monterey, California.

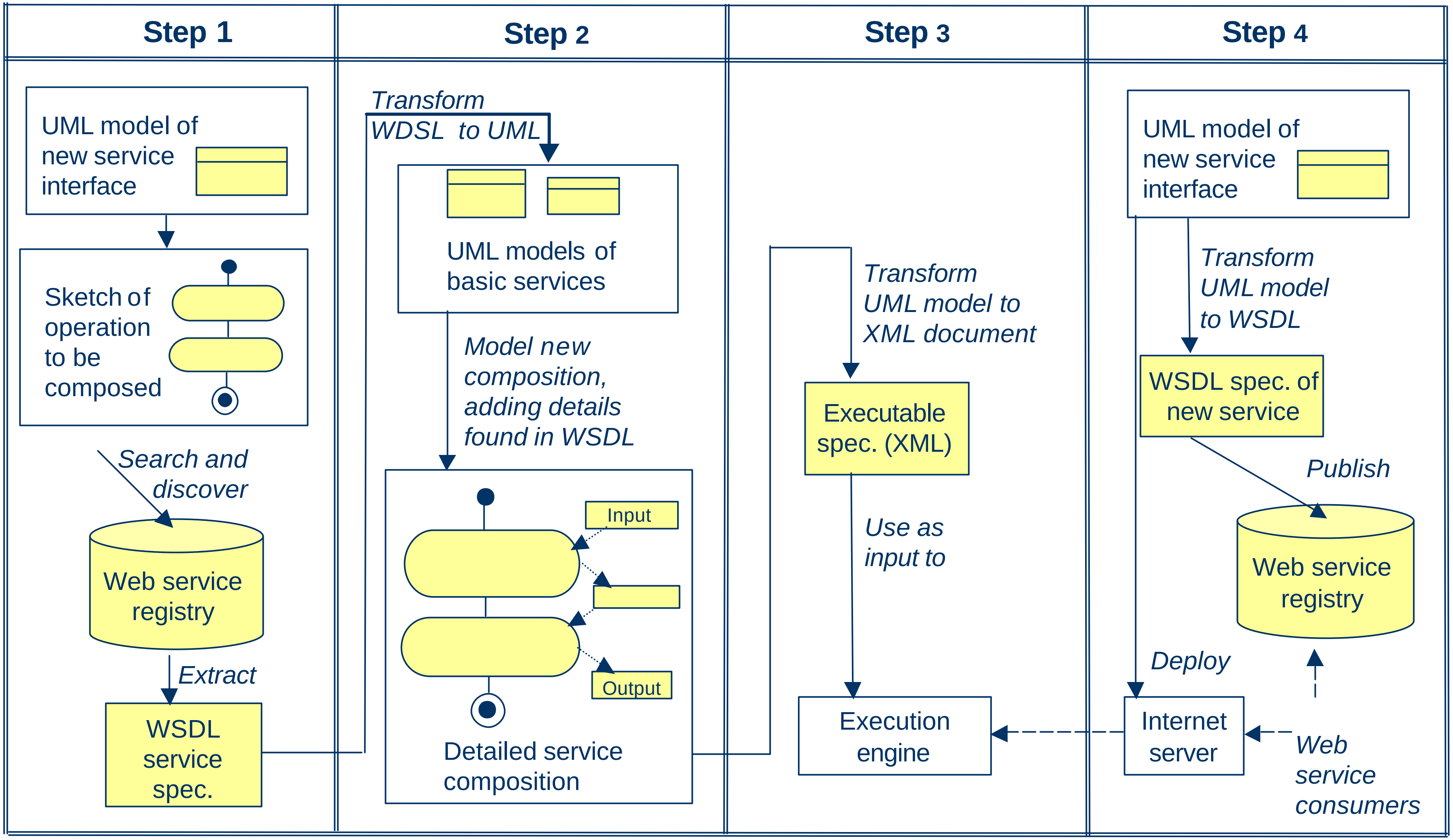
Modeling Web Service Composition in UML

David Skogan, Roy Grønmo, Ida Solheim

SINTEF, Norway

This work has been funded by the European Union project IST,
Adaptable and Composable E-commerce and Geographic Information Services (ACE-GIS)

Model-driven Web Service Composition - Method

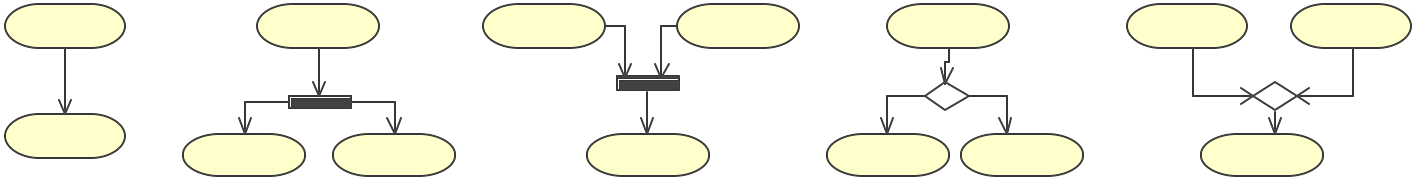

Step 1
Step 2
Step 3
Step 4
UML model of new service interface
Sketch of operation to be composed
Search and discover
Web service registry
Extract
WSDL service spec.
Transform WDSL to UML
UML models of basic services
Model new composition, adding details found in WSDL
Input
Output
Detailed service composition
Transform UML model to XML document
Executable spec. (XML)
Use as input to
Execution engine
UML model of new service interface
Transform UML model to WSDL
WSDL spec. of new service
Publish
Web service registry
Deploy
Internet server
Web service consumers

UML as an integration platform for modeling web service compositions

- The UML models are sufficiently expressive for web service composition;
- The UML models can be transformed to directly executable composition specifications; and
- The UML models are independent of the executable composition languages.

The five basic control flow patterns

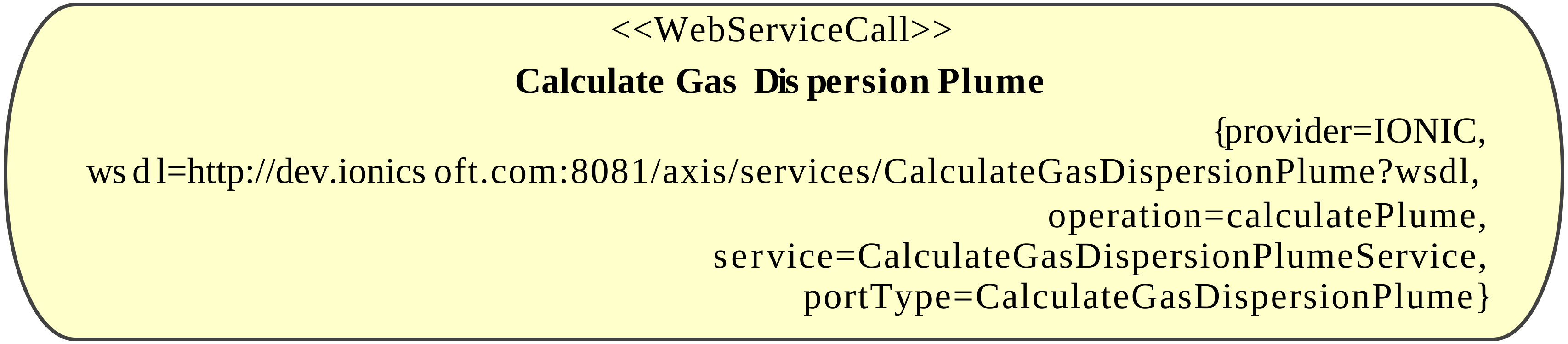
PATTERN NAME	Sequence	Parallel split	Synchronization	Exclusive choice	Simple merge
description	Execute activities in sequence	Execute activities in parallel	Synchronize two parallel threads of execution	Choose one execution path from many alternatives	Merge two alternative execution paths
UML	Control-Flow	Fork	Join	Decision-Node	Merge



Additional composition patterns

- **Discriminator** – control flow pattern. Alternative services are executed in parallell and the first one to return an answer is used.
- **Selector** – control flow pattern. The choice of alternative services is based on QoS selection criteria.
- **Data transformations** – The data output of previously executed services needs to be transformed into the required input of the next service to execute.

Activity realized by a web service call



<<WebServiceCall>>

Calculate Gas Dispersion Plume

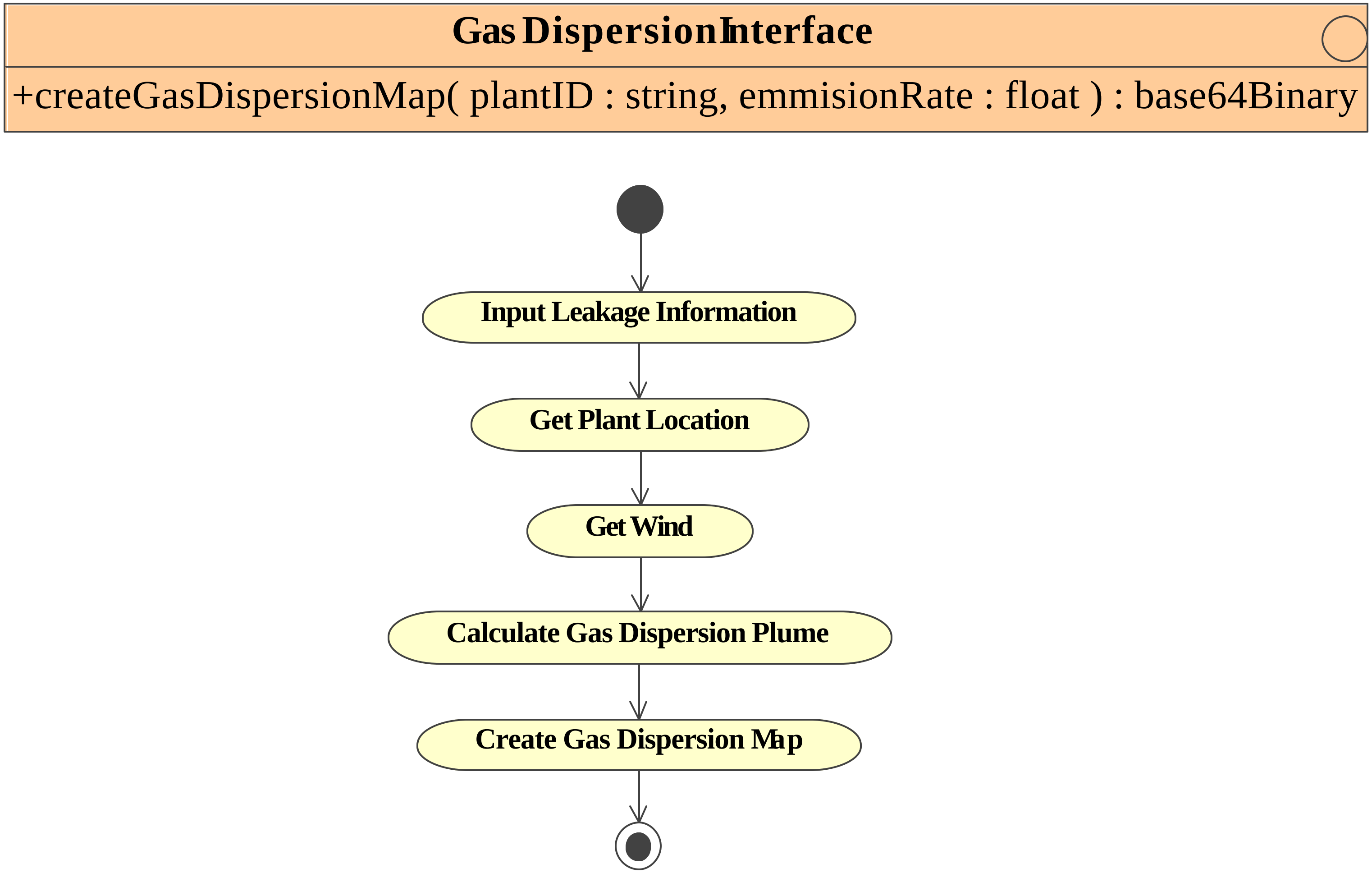
{provider=IONIC,
wsdl=http://dev.ionicsoft.com:8081/axis/services/CalculateGasDispersionPlume?wsdl,
operation=calculatePlume,
service=CalculateGasDispersionPlumeService,
portType=CalculateGasDispersionPlume}

UML profile for Web Service Composition - Summary

- Five basic control flow patterns
- Additional control flow: Selection and Discriminator
- Data transformations
- Web service calls

- UML profile for the interface modeling

Applying the method to an example – step 1: Preliminary composition model



Gas DispersionInterface

+createGasDispersionMap(plantID : string, emmisionRate : float) : base64Binary

Input Leakage Information

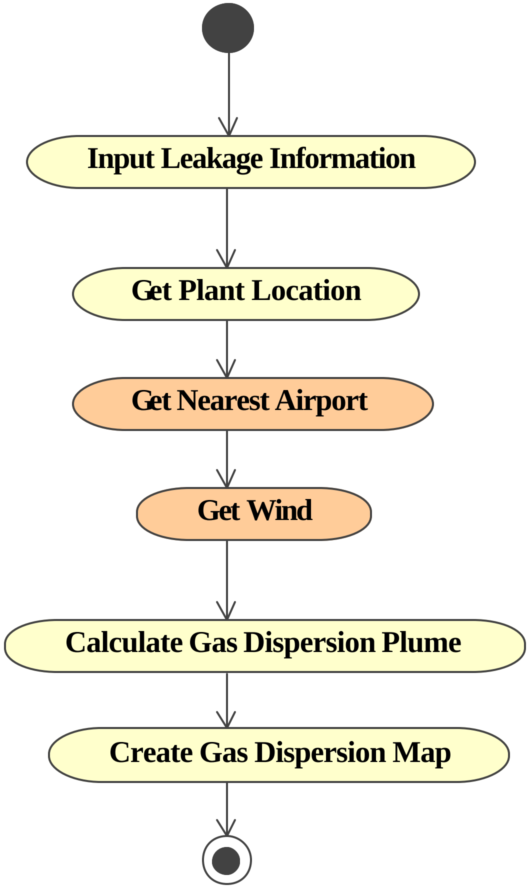
Get Plant Location

Get Wind

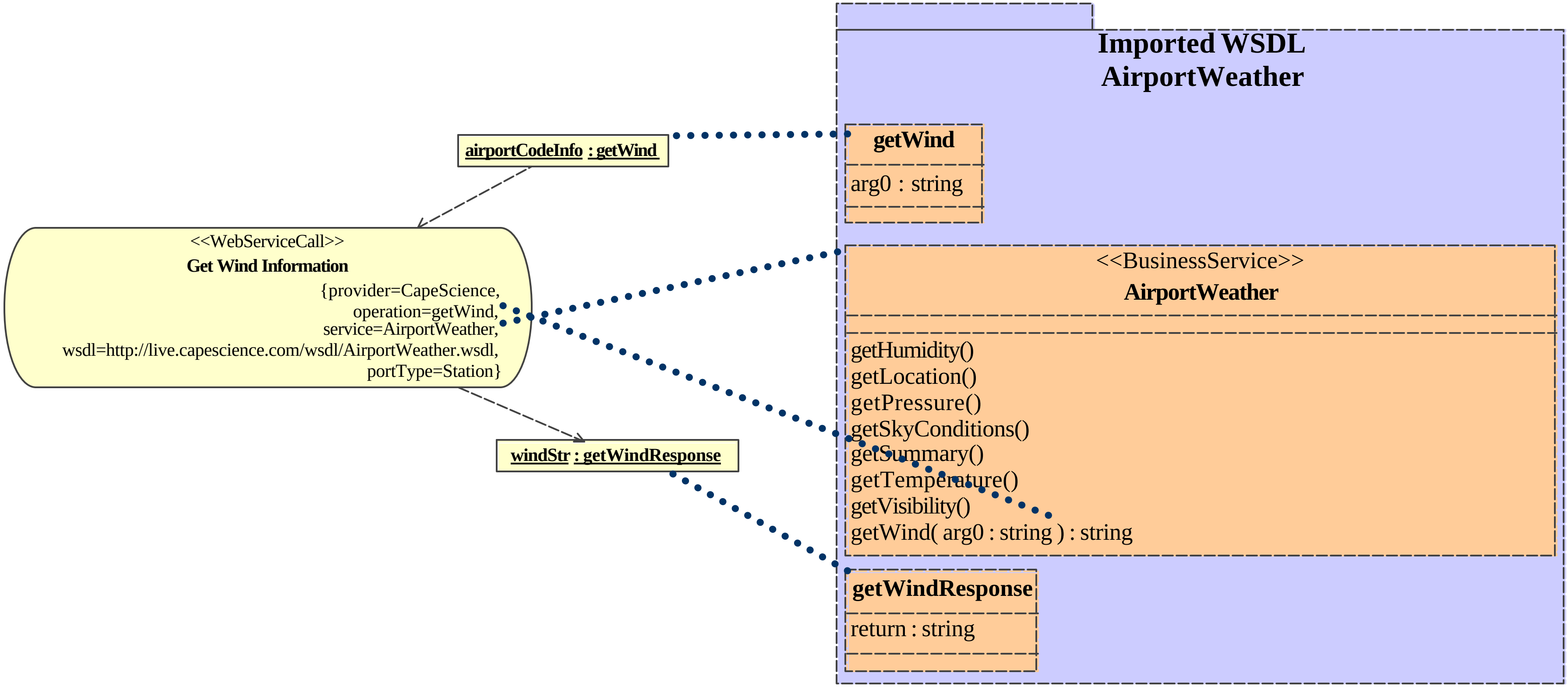
Calculate Gas Dispersion Plume

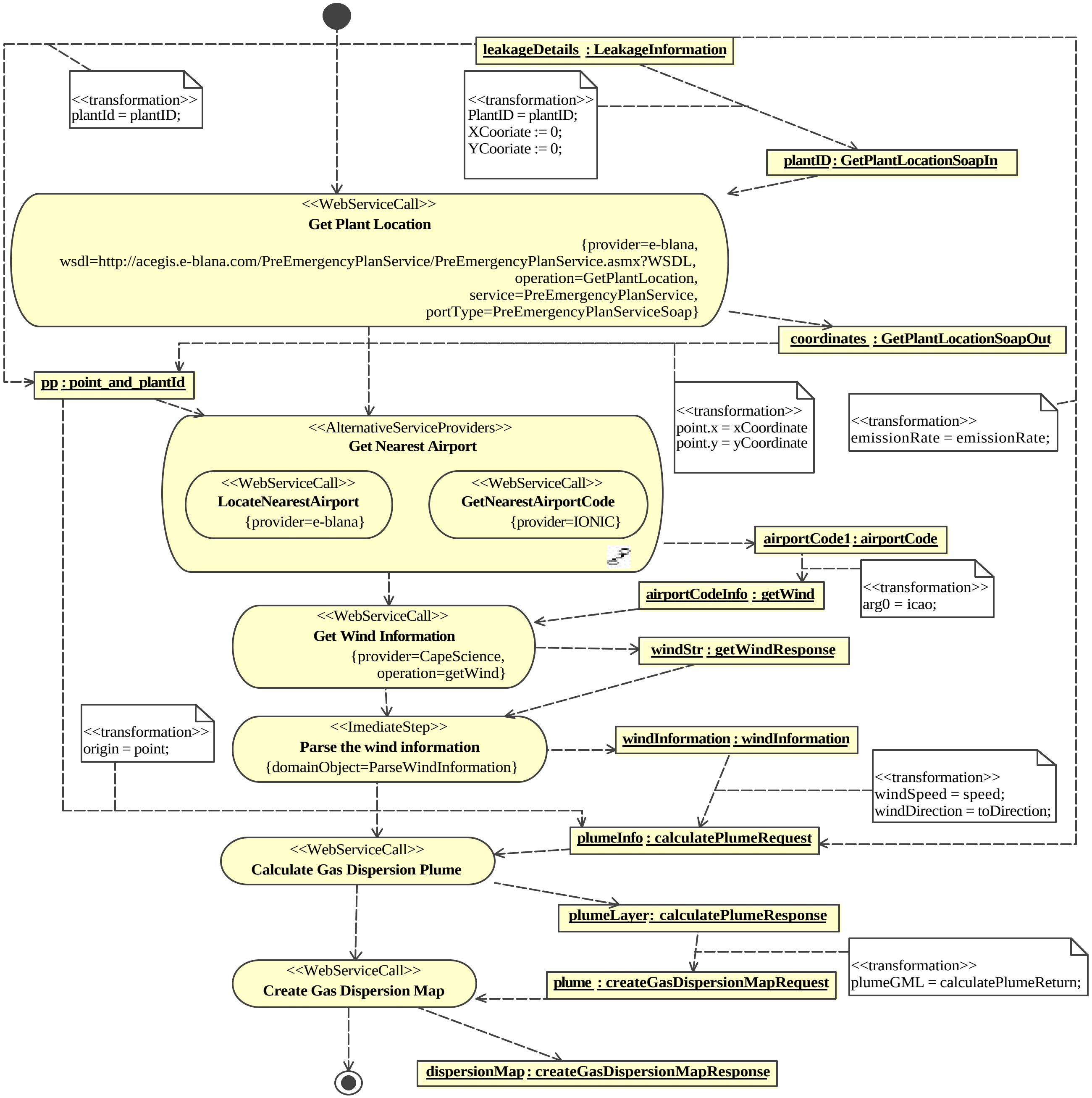
Create Gas Dispersion Map

Applying the method to an example – step 2: Adjusted composition model


Input Leakage Information
Get Plant Location
Get Nearest Airport
Get Wind
Calculate Gas Dispersion Plume
Create Gas Dispersion Map

Applying the method to an example – step 2: Adding WSDL details





leakageDetails : LeakageInformation

<<transformation>>
plantId = plantID;

<<transformation>>
PlantID = plantID;
XCooriate := 0;
YCooriate := 0;

plantID : GetPlantLocationSoapIn

<<WebServiceCall>>
Get Plant Location
{provider=e-blana,
wsdl=http://acegis.e-blana.com/PreEmergencyPlanService/PreEmergencyPlanService.asmx?WSDL,
operation=GetPlantLocation,
service=PreEmergencyPlanService,
portType=PreEmergencyPlanServiceSoap}

coordinates : GetPlantLocationSoapOut

pp : point_and_plantId

<<AlternativeServiceProviders>>
Get Nearest Airport

<<WebServiceCall>>
LocateNearestAirport
{provider=e-blana}

<<WebServiceCall>>
GetNearestAirportCode
{provider=IONIC}

<<transformation>>
point.x = xCoordinate
point.y = yCoordinate

<<transformation>>
emissionRate = emissionRate;

airportCode1 : airportCode

airportCodeInfo : getWind

<<transformation>>
arg0 = icao;

<<WebServiceCall>>
Get Wind Information
{provider=CapeScience,
operation=getWind}

windStr : getWindResponse

<<transformation>>
origin = point;

<<ImediateStep>>
Parse the wind information
{domainObject=ParseWindInformation}

windInformation : windInformation

<<transformation>>
windSpeed = speed;
windDirection = toDirection;

plumeInfo : calculatePlumeRequest

<<WebServiceCall>>
Calculate Gas Dispersion Plume

plumeLayer: calculatePlumeResponse

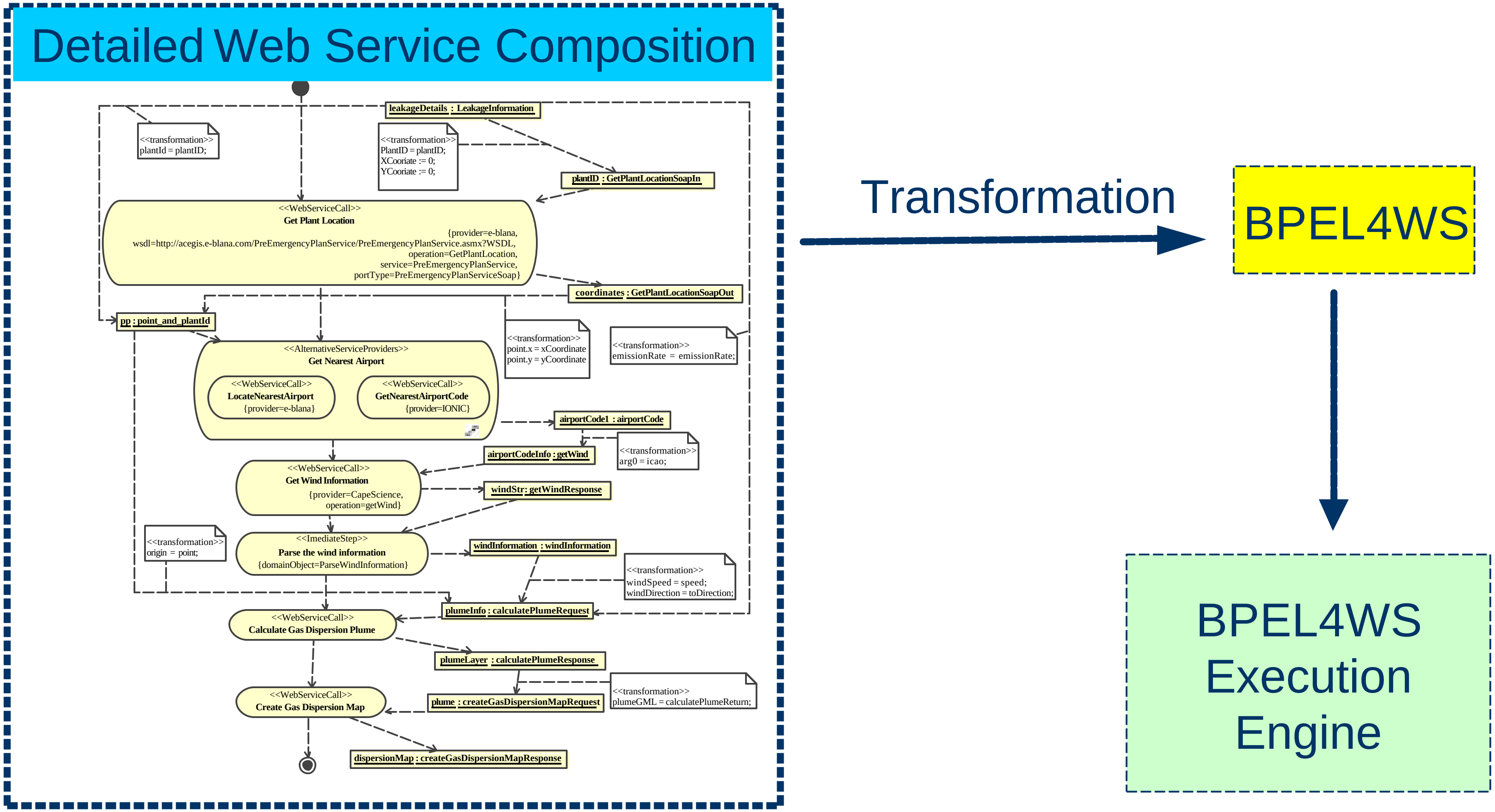
<<transformation>>
plumeGML = calculatePlumeReturn;

<<WebServiceCall>>
Create Gas Dispersion Map

plume : createGasDispersionMapRequest

dispersionMap : createGasDispersionMapResponse

Applying the method to an example – step 3 :Transforming UML into executable descriptions



Detailed Web Service Composition

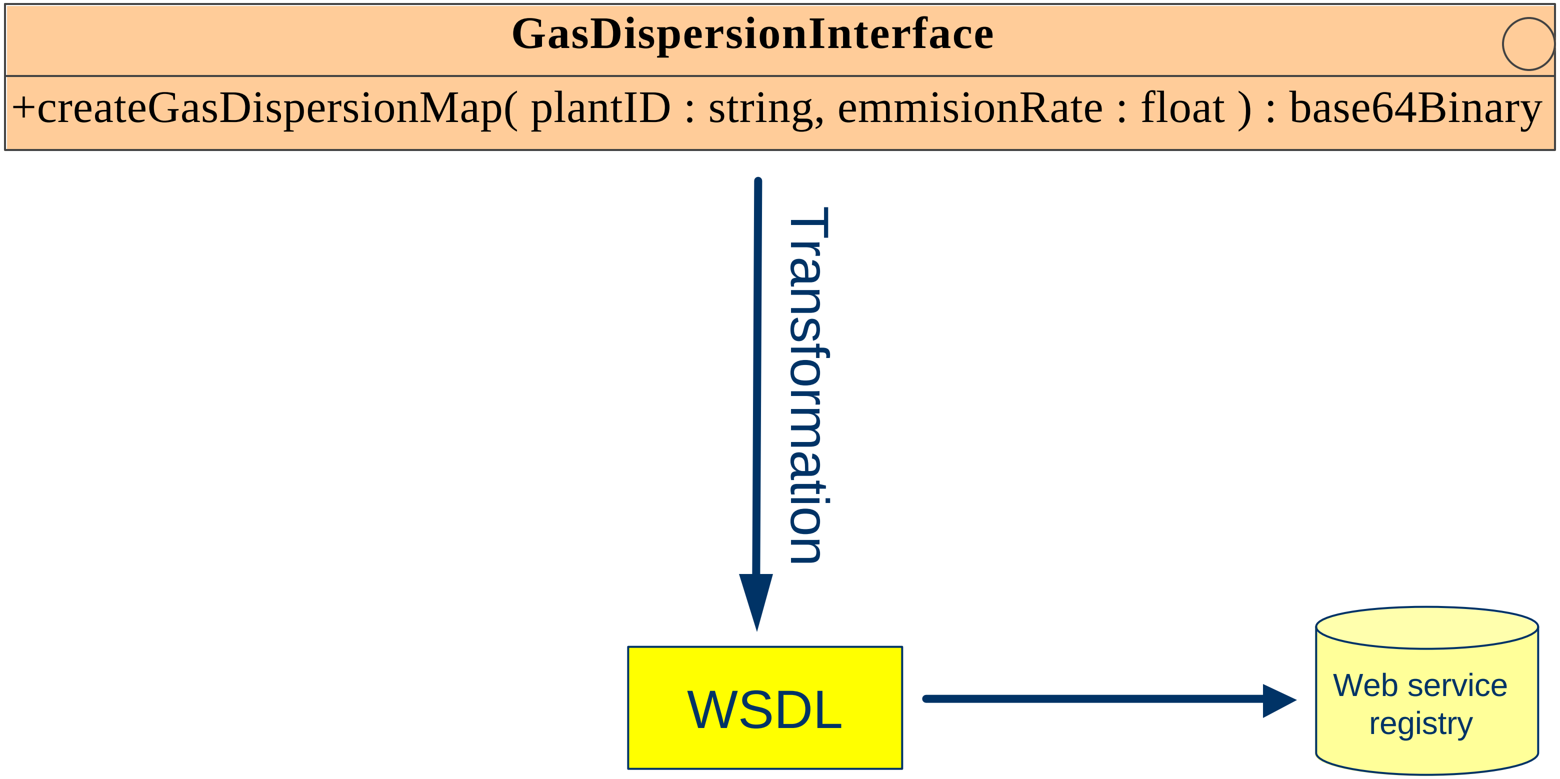
leakageDetails : LeakageInformation
<<transformation>>
plantId = plantID;
<<transformation>>
PlantID = plantID;
XCooriate := 0;
YCooriate := 0;
plantID : GetPlantLocationSoapIn
<<WebServiceCall>>
Get Plant Location
{provider=e-blana,
wsdl=http://acegis.e-blana.com/PreEmergencyPlanService/PreEmergencyPlanService.asmx?WSDL,
operation=GetPlantLocation,
service=PreEmergencyPlanService,
portType=PreEmergencyPlanServiceSoap}
coordinates : GetPlantLocationSoapOut
pp : point_and_plantId
<<transformation>>
point.x = xCoordinate
point.y = yCoordinate
<<transformation>>
emissionRate = emissionRate;
<<AlternativeServiceProviders>>
Get Nearest Airport
<<WebServiceCall>>
LocateNearestAirport
{provider=e-blana}
<<WebServiceCall>>
GetNearestAirportCode
{provider=IONIC}
airportCode1 : airportCode
airportCodeInfo : getWind
<<transformation>>
arg0 = icao;
<<WebServiceCall>>
Get Wind Information
{provider=CapeScience,
operation=getWind}
windStr: getWindResponse
<<transformation>>
origin = point;
<<ImediateStep>>
Parse the wind information
{domainObject=ParseWindInformation}
windInformation : windInformation
<<transformation>>
windSpeed = speed;
windDirection = toDirection;
plumeInfo : calculatePlumeRequest
<<WebServiceCall>>
Calculate Gas Dispersion Plume
plumeLayer : calculatePlumeResponse
<<transformation>>
plumeGML = calculatePlumeReturn;
plume : createGasDispersionMapRequest
<<WebServiceCall>>
Create Gas Dispersion Map
dispersionMap : createGasDispersionMapResponse

Transformation

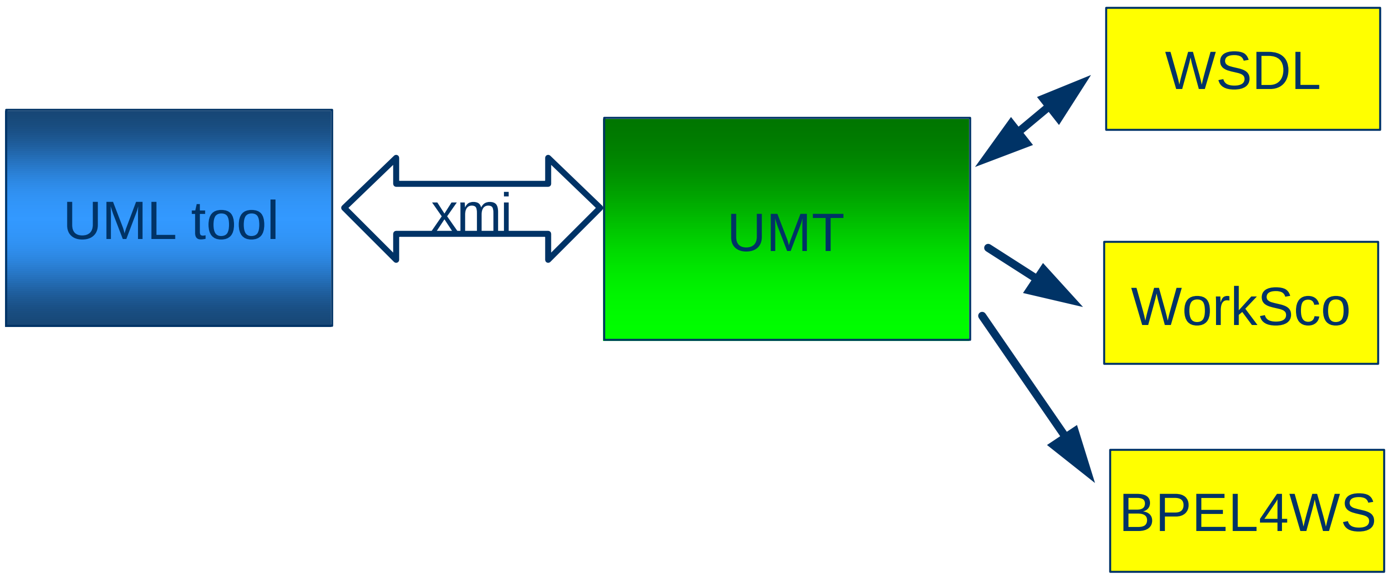
BPEL4WS

BPEL4WS Execution Engine

Applying the method to an example – step 4: Transforming UML into WSDL


GasDispersionInterface
+createGasDispersionMap(plantID : string, emmisionRate : float) : base64Binary
Transformation
WSDL
Web service registry

Method supported by transformations implemented in the UMT tool



Open Source tool available at Sourceforge: qvt-umt.sourceforge.net

Future improvements

- **QoS integration:** Modeling, search and discovery, static vs. runtime.
- **Semantic services:** Modeling, search and discovery of services based on precise semantics and well-known ontologies.

- *Immature*: Lack of established standards for web services.

Related work

- Control flow patterns: Aalst, “Don't go with the flow”, 2003
- Pure UML approach: Dumas and Hofstede, “UML Activity Diagrams as a Workflow Specification Language”, 2001
- BPEL4WS-dependent UML modeling: T. Gardner, “UML Modelling of Automated Business Processes with a Mapping to BPEL4WS,”, 2003.
- Petri net-based model: Hamadi and Benatallah, “A Petri Net-based Model for Web Service Composition”, 2003
- Similar approach: Thöne et al., “Process-Oriented, Flexible Composition of Web Services with UML”, 2002
- QoS integration 1: Zeng et al., “QoS-aware Middleware for Web Services Composition”, 2004.
- QoS integration 2: Jaeger et al., “QoS Aggregation for Web Service Composition using Workflow Patterns”, EDOC 2004 (*Thursday*!)
- Semantic integration tools: Probst and Lutz, “Giving Meaning to GI Web Service Descriptions ”, 2004.

Conclusions

- Reverse-engineering WSDL into UML allows the use of UML as a common integration platform
- Tool-supported conversions from UML to executable workflow descriptions, show that our UML workflow models are executable
- Our UML profile is independent of execution language
- The use of UML models improves the documentation and readability of service compositions compared to low-level XML descriptions