# Unimplemented functionality of Rotalumis

This is a list of requests that are in the rotalumisrequest.xsd but are not implemented in the current version of Rotalumis. (Reactions by Leo van Bokhoven are displayed in Blue).

* Storage API
  + Save
* Compiler API
  + Serialize
  + Deserialize (only poosl\_specification is implemented)
  + Compile (only poosl\_specification is implemented)
* Model API
  + List classes
  + Get top level specification
  + Get class definition
  + Get method definition
  + Rename object
  + Insert object
  + Delete object
* Execution engine API
  + Inspect is missing the implementation for types(er zijn al C++ objecten voor gemaakt die al netjes gepropageerd worden, zoals voor de al bestaande object typen: zie api/api\_inspect.cc, API::INSPECT\_CLUSTER, INSPECT\_CHANNEL, INSPECT\_PORT; de invulling van deze requests ontbreekt zo blijkt nu)
    - Cluster class
    - Channel
    - Port
  + Observe (niet geimplementeerd)
  + Set breakpoint condition (condities op breakpoints zijn nog niet geimplementeerd, dit zou een uitbreiding zijn op klasse ROTALUMIS::BREAKPOINT, in include/core/bp.h)

# Missing in the API

There are also a number of requests that are not specified in the xsd but are necessary to build a decent debugger.

* A request to get a handle for a specific statement in a process method definition that we can relate to the original xml input. This is necessary for at least:
  + Setting a breakpoint (especially to parameter control\_point):

<xs:complexType name="t\_create\_breakpoint\_request">

<xs:sequence>

<xs:element name="name" type="t\_identifier" />

<xs:element name="control\_point" type="xs:integer" />

</xs:sequence>

</xs:complexType>

The control pointer is a handle to a node in the parse tree corresponding with the control point at which the breakpoint is set: statement/expression node or top-level design for breakpoint on currentTime

* + Interpreting the PET. These are all the handles that are returned in a inspect response.

<xs:complexType name="t\_executiontree\_base">

<xs:attribute name="process" type="xs:integer" />

<xs:attribute name="parent" type="xs:integer" />

<xs:attribute name="handle" type="xs:integer" />

<xs:attribute name="global" type="xs:integer" />

<xs:attribute name="local" type="xs:integer" />

</xs:complexType>

Example of possible output is:

<execution\_tree>  
<method\_call name="someMethod" state="prebind\_input" process="1" parent="0" handle="2" global="3" local="0"/>  
</execution\_tree>

In addition, it looks like the current handles in the PET don’t refer to statements in the original xml.

# Communication with Leo van Bokhoven

Questions asked to Leo:

1. voor een aantal request/responses schijf je dat deze geimplementeerd zijn, maar voor de Model API requests ziet het ernaar uit dat dit niet zo is. Er is wel een functie voor, maar daar staat dan: std::cout << "MODEL::list\_classes() to be implemented" << std::endl;  
  
2. In 'missing in the API', is de hoofdvraag hoe we PET informatie kunnen teruglinken aan de source code (XML spec) voor debugging. Er lijken weliswaar velden in datastructuren te zijn die naar source code zouden kunnen wijzen, maar uit een debugsessie blijkt dat die velden niet gezet worden. We weten dus niet hoe we in het algemeen de toestand van een PET kunnen linken aan de corresponderende punten in de XML spec.

Response:

1) ik heb vandaag toevoegingen gemaakt om list\_classes, get\_top\_level\_specification en get\_class\_definition te voltooien. Voor get\_method\_definition is een voorbeeld implementatie gemaakt (het is op dit moment onduidelijk hoe jullie naar objecten willen gaan verwijzen), en retourneert op dit moment enkel de gezochte method definition als een string, niet als een XML tree. (Dat is overigens wel voor get\_top\_level\_specification en get\_class\_definition het geval.) Voor get\_object\_handle heb ik geen implementatie die uit API::MODEL informatie oplepelt. De verwarring mijnerzijds is ontstaan omdat ik wel get\_object\_handle geimplementeerd heb om objecten te vinden van de draaiende simulatie, dus van IMPL::EXECUTION\_ENGINE.  
  
2) ieder API::OBJECT object heeft een location(). Deze locatie vul ik momenteel enkel vanuit cps/cps\_parse.bison middels convert\_location(...). Voor libxml heb ik nooit een goede API kunnen vinden die me (line, column) informatie teruggeeft voor een XML-tree node. Als deze API bestaat, kan de locatie vanuit iedere subklasse aan de constructor van API::OBJECT worden meegegeven, net als dat nu al voor de oude poosl-syntax gebeurd.  
  
De wijzigingen zijn zojuist gepusht naar de rotalumis git repository onder branch "api". Een en ander wordt getoond in test/inspect/test\_list\_classes.py, zie:  
     list\_classes()  
     get\_top\_level\_specification()  
     get\_class\_definition("Nil", "data")  
     get\_class\_definition("P1", "process")  
     get\_method\_definition("NP/initialize")  
  
     get\_object\_handle("/I1", "process")  
     get\_object\_handle("/I1/iv1", "variable")