

PROCESS TOWARDS AN EXECUTABLE ENGINE COMPLIANT WITH THE INTEROPERABLE TEST AND OPERATION PROCEDURE LANGUAGE 70-32

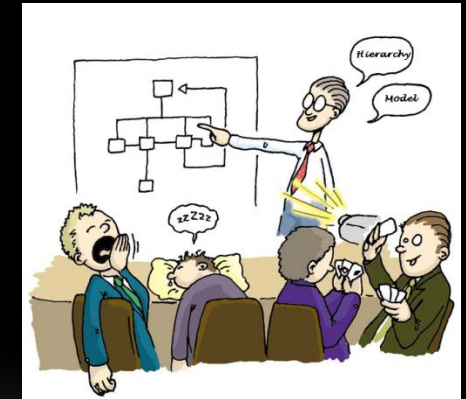
Nieves Salor Moral - Vitrociset Belgium

SESP 2015 – Noordwijk, March 2015



AGENDA

- ISSUES
- CONCEPTUALIZATION
 - DATA
 - PROCESS
- VALIDATION
- DEVELOPMENT
 - EDITORS
 - SCALABILITY
 - UNIQUE ENGINE
- RE-USE OF METHODOLOGY
- CONCLUSIONS



ISSUES IN PROCEDURE INTEROPERABILITY (I)

Let's cook...



...or better not

ISSUES IN PROCEDURE INTEROPERABILITY (II)

The collage includes:

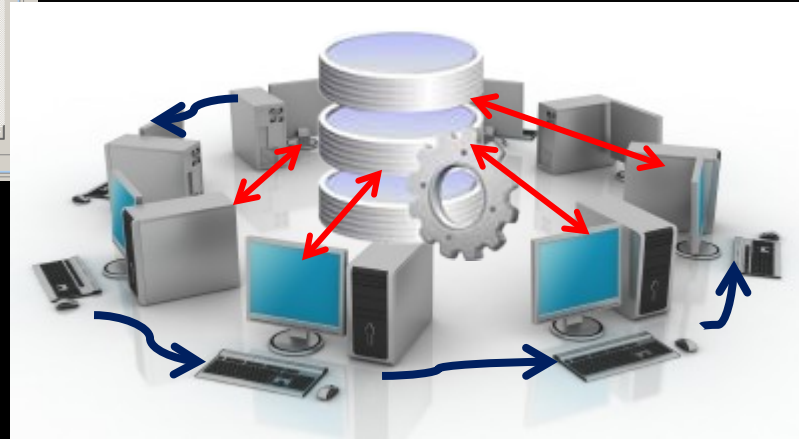
- A flowchart titled "START of Procedure" with decision points like "Check status of CACE" and "Check ADCE is Thruster voltage level".
- A code editor window showing PLUTO code:

```
procedure
initiate and confirm step thrustervalues
declare
  boolean ix
end declare
inform user "Signal time difference between expect
inform user "Corrective Thruster evaluated (Roll)
inform user "Corrective Thruster evaluated (Pitch)
inform user "Corrective Thruster evaluated (Yaw)
ix := ask user("Execute values?") expect boolean;
end step;
end procedure
```
- A graphical user interface with a "Select" menu and various activity blocks like "declaration body", "preconditions body", "main body", and "watchdog body".
- Technical documents such as "Callipers (Digital, Dial, Vernier)", "Height Gauge", and "Microscope".

PLUTO

MOIS

OTX



OBCP

TOPE

FCP

JAVA

EGS-CC AP LANGUAGE

ECSS-E-ST-70-32 CONCEPTUALIZATION

- Requirements

- From the standard
- From experience
- From other implementations

- Compare
- Unify
- Add missing info.

Formalize:

- A Procedure specifies:

- Data to be handled
- Process to be performed with the specified data
- Format to specify the information

Data

Process



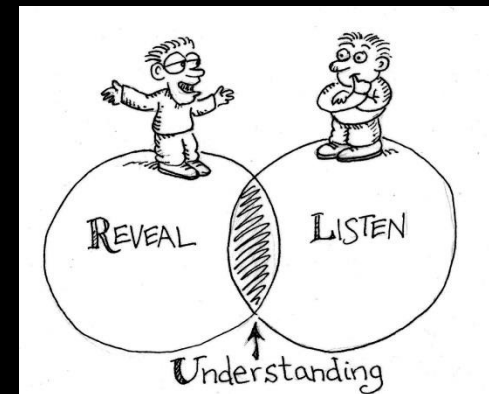
CONCEPTUALIZATION: DATA (I)

- The standard defines a complete set of information and structures to be handled.
- Exchanging procedures → semantics have to be shared between parties.

➔ **SEMANTICS**

- Shared semantics

Shared Process Engine
Translations
Decoupling editors
Shared Persistence Model



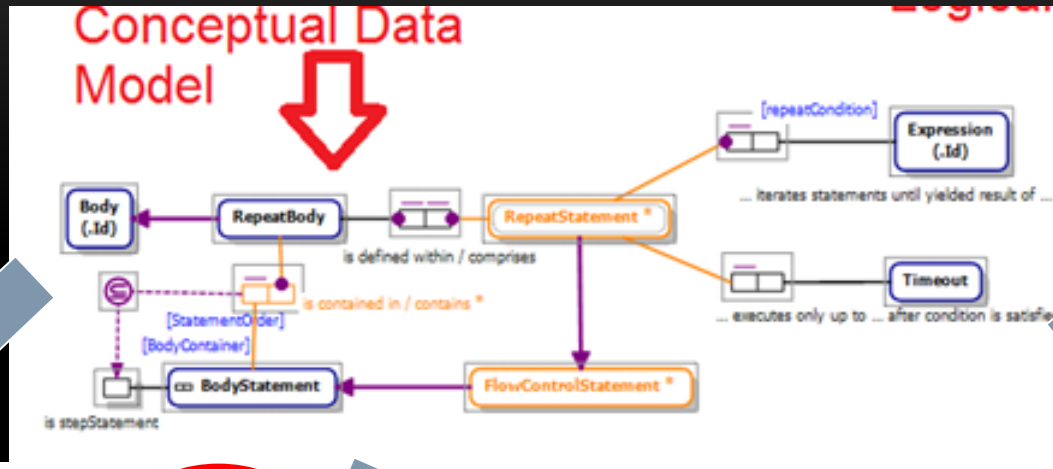
CONCEPTUALIZATION: DATA (II)

- Process for conceptualization:
 - Define the structure of the language
 - For each structure
 - Get the data concepts required
 - Get the relationships within the structure (cardinalities)
 - Define the constraints applicable per structure
 - Verify all structures are reachable and complete
 - Add missing structures.



Complete Conceptual Model

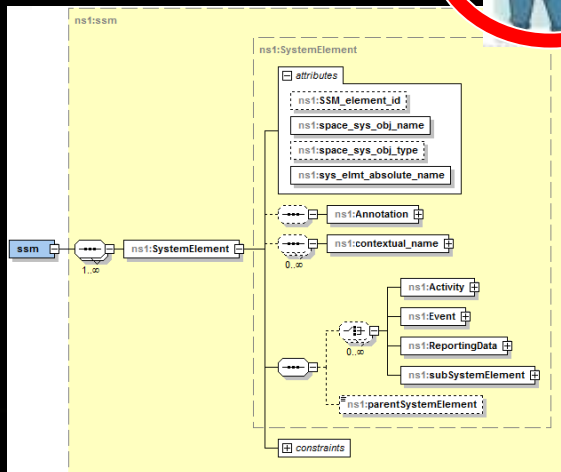
CONCEPTUALIZATION: DATA (III)



- 20130611-70-32 Formal.MySQL.sql
- 20130611-70-32 Formal.Oracle.sql
- 20130611-70-32 Formal.PostgreSQL.sql



BodyStatement is stepStatement.
 In each population of BodyStatement is stepStatement, each BodyStatement occurs at most once.
 RepeatBody contains BodyStatement.
 Each RepeatBody contains some BodyStatement.
 Each BodyStatement is contained in at most one RepeatBody.
 It is possible that some RepeatBody contains more than one BodyStatement.
 Derivation Note: Fully derived from the entries in the repeatBody.
 If some BodyStatement is contained in some RepeatBody then that BodyStatement is stepStatement.
 Body is an entity type.
 Reference Scheme: Body has Body_Id.
 Reference Mode: .Id.



Standard	Conceptual Model
26 Mandatory Pages	137 Mandatory Pages
88 Informative Pages	>400 static data requirements
	>60 Change Requests

CONCEPTUALIZATION: DATA (IV)

- After conceptualization, users can exchange procedures in:
 - Database Sets (e.g. SQL)
 - XML
 - ORM formats
- Editors are needed.
 - Define edition formats (i.e. grammars) to create procedures



CONCEPTUALIZATION: PROCESS (IV)

- Procedure Goal → Execute
- Each data structure → Specific behavior
- Behaviors need to be modelled → Interfaced
 - Require the definition data
 - Implies the creation of dynamic information (instances)
 - Follow a set of interactions between elements
 - Have to handle user/system inputs



DEVELOPMENT: EDITORS

The image shows a development editor with two panes. The left pane displays a procedure code in a structured text format, and the right pane displays its corresponding XML representation. The code in the left pane is as follows:

```
Procedure{
  declare{
    event ad described by "asb";
    event ab ;
  }end declare
  main{
    initiate{
      sys1.sys2.Procedure2
    }end initiate;
    initiate and confirm{
      sys1.sys2.Procedure2
      refer by "asda" in case{
        aborted:abort;
        confirmed:restart timeout{
          12
        }end timeout;
      }end case
    }end initiate and confirm;
    initiate and confirm step{
      step ad {
        main{
          wait{
            for event ad
          }end wait;

          }end main
        }end step
      }end initiate and confirm;
    inform user{
      43;
    };
  }end main
}
```

The XML representation in the right pane is as follows:

```
<PROCEDURE>
  <MAIN>
    <LOG>
      <EXPRESSION><REL_EXPRESSION>
        <TERM>
          <PRODUCT>
            <FACTOR>
              <CONSTANT>"embedded Executed"</CONSTANT>
            </FACTOR>
          </PRODUCT>
        </TERM>
      </REL_EXPRESSION></EXPRESSION>
      <EXPRESSION><REL_EXPRESSION>
        <TERM>
          <PRODUCT>
            <FACTOR>
              <REQUEST>
                <PROPERTY> "completion time" </PROPERTY>
                <VALUE_OF> sys1.sys2.Procedure2</VALUE_OF></REQUEST>
              </FACTOR>
            </PRODUCT>
          </TERM>
        </REL_EXPRESSION></EXPRESSION>
      </LOG>
    </MAIN>
  </PROCEDURE>
```

Timeout returns Timeout:
'<TIMEOUT>' expression=Expression ('<RAISE_EVENT>' event =[LocalEvent] '</RAISE_EVENT>')? '</TIMEOUT>'
;

DEVELOPMENT: EDITORS

The screenshot displays a development editor interface. The main workspace is titled "Procedure" and is divided into two primary sections: "Procedure Declaration Body" on the left and "Procedure Main Body" on the right. The "Procedure Declaration Body" contains a diagrammatic representation of a procedure's start, featuring a black circle at the top, a square box with a plus sign, a rectangular box labeled "Local Event", another square box with a plus sign, and a grey circle at the bottom. The "Procedure Main Body" is currently empty. Below the main body, there is a section labeled "Watchdog Body". To the right of the main workspace is a vertical "Objects" palette, which lists various graphical elements available for use in the diagram, including Continuation Action, Watchdog Body, step Definition, Flow Control Statement, For Statement, Step Main Body, Step Statement, Case Statement, Case Branch Type, While Statement, Repeat Statement, In Node Type, Within Node Type, Relational Expression Node, Comparative Expression Node, and Expression Node.

not committed, continue

Variable loopcounter has not been previously declared. `loopcounter </COUNTER>`

Declare missing variable

Create variable not declared

DEVELOPMENT: SCALABILITY

- Global constants
- Augment the possible data types with:
 - Byte Fields
 - Arrays (Simple or complex)
 - Record: Set of complex values which can mix both simple values and complex ones.
- Allow the explicit raising of events during the execution.
- Assign a criticality to the declared events
- Mathematical exceptions handling (i.e. arithmetic and class cast exceptions)
- Transactional bodies

<2 weeks

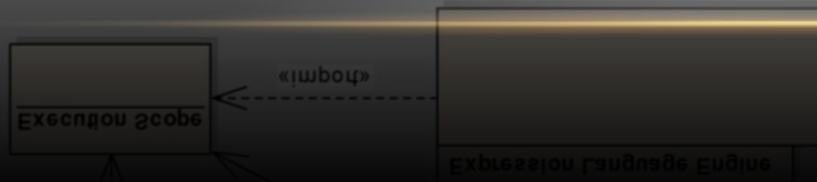
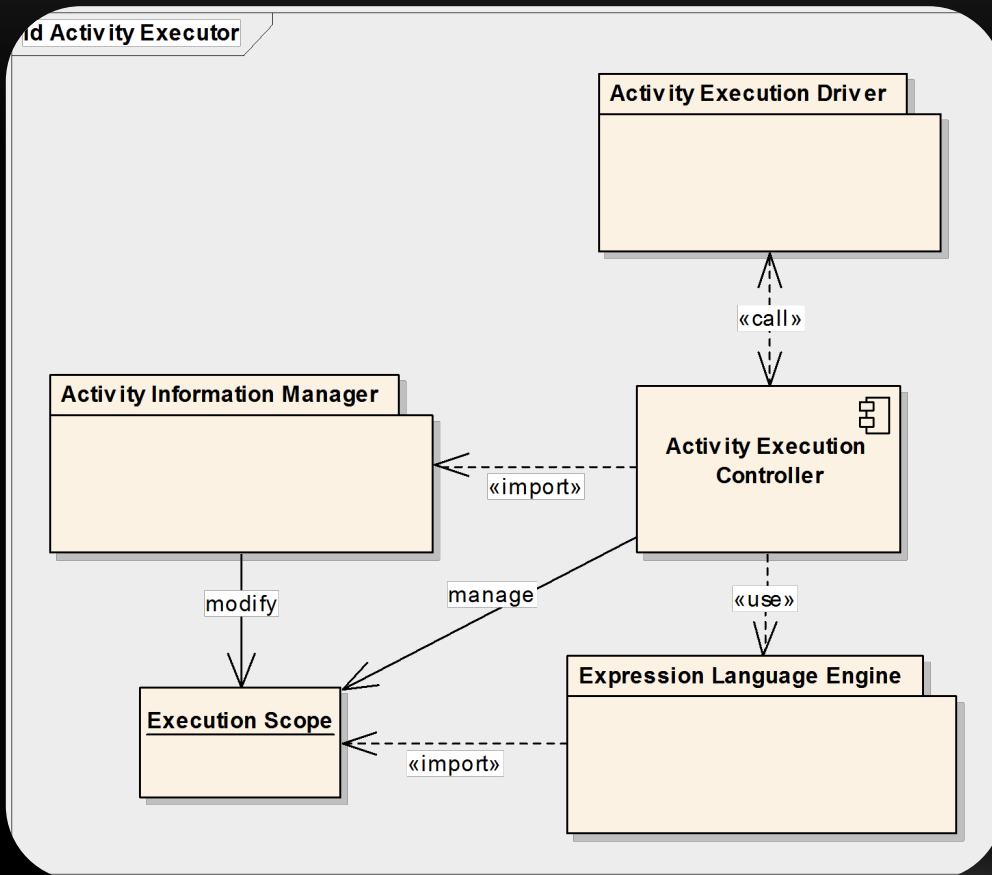
Editor

Validation

Transformation

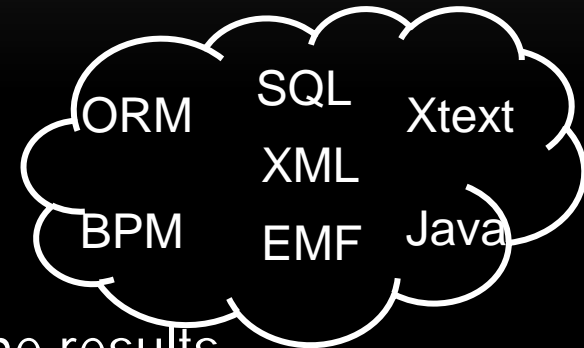
Execution

DEVELOPMENT: UNIQUE ENGINE



VALIDATION

- Verification & Validation Techniques
 - Validation of the methodologies used
- Experts in the standard have assessed the results
- Existing procedures have been assessed with the created editors.
- Interoperability among different formats have been empirically proven during compilation and execution.



METHODOLOGY-REUSE

- Application in EGS-CC environment
 - The AP Execution Language:
 - Definition of structures/behaviors/interfaces
 - Definition of DSL languages and transformations into AP Format.
 - The CDM Definitions:
 - Definition of several MCM browsers/editors through the formalization of the data model.
- The HMI Project (ongoing):
 - Definition of the data model
 - Integration between graphical and textual procedure editors.
- Technical Knowledge
 - Use of OSGI Services
 - Application and integration of EMF/Xtext/GMF Libraries

CONCLUSIONS

- Having common meta-model and execution engine:
 - Only new front-ends have to be developed
 - Expand the range of activities where space standards can be used.
 - Reduce updates/modifications impact
- Conceptualization of requirements
 - Ensure completeness and validity of requirements
 - Help design and development stages

Any Question?