Process towards an executable engine compliant with the Interoperable Test and Operation Procedure Language 70-32

Workshop on Simulation for European Space Programmes (SESP)
24-26 March 2015

ESA-ESTEC, Noordwijk, The Netherlands

Nieves Salor Moral[(1)], Dionisi Simone[(2)], Massimiliano Mazza [(1)]

[(1)] *Vitrociset Belgium*
Huygensstraat 34, Noordwijk, 2201DK, Netherlands
Email: n.salor_moral@vitrocisetbelgium.com, m.mazza@vitrocisetbelgium.com

[(2)] *Vitrociset Belgium in Germany*
Lise-Meitner-Strasse 10, 64293 Darmstadt (Germany)
Email: s.dionisi@vitrocisetbelgium.com
:

INTRODUCTION

Exchanging information is not a problem anymore with the communication technologies available to everyone. However, understanding the exchanged data to be able to work with it is still an issue, if not a bigger problem than before due to the amount of data and formats systems need to manage nowadays.
Although system interfaces allow exchanging information between integrated systems, the meaning of the data to be exchanged has to be the same for both parties. This constitutes the underlying problem of the data interoperability. Additionally, each time data is modified, further effort to assess, develop the changes is required; greatly increasing the project cost. This situation has led to the definition of multiple standards targeting procedural language in ESA; furthermore it is still an active discussion to decide which procedure format to follow in the EGS-CC.
This paper presents a combined approach to solve first, the validation of the correct understanding of the requirement specification by the experts in a contextual global way (in the target project, this is the procedural standard ECSS-E-ST-70-32) and second, the way to automate the code generation required at different stages of a procedure preparation and execution engine. Some of these stages are the logical database (e.g. E/R, XSD), physical (e.g. Oracle, PostgreSQL, MySQL); the business processes (e.g. BPEL), the persistence and applications data layers (e.g. Beans, Factories, Providers, Web Services) or the procedure editors.
Following this approach, the definition of several customizations of the same standard has been performed according to user needs while automatically validating the contents of the procedure (i.e. the semantics), transforming from one customization to another and providing a safe manner of extending a language without risking information loss. This mechanism is based in the applicability of different DSLs compliant with the same data model representing the semantics of the standard.
The methodology presented in this paper has been used in the ASE5 project for the development of a data model compliant with the ECSS-E-ST-70-32, the definition of two different textual procedure editors (two different DSLs) which allow interaction between each other and the creation of a unique execution engine for any DSL already defined or to be defined while they are compliant with the data model. After assessment of other procedural standards (e.g. OTX, FCP) and previous experience, new structures and functionalities such as transactions, global constants, new data types or bitwise support were requested to be added to the procedural language. The paper explains how the extensions were created, the time it took to develop them and the lack of impact in the already existing procedures while ensuring interoperability between already existing DSLs and extended ones.

CONCEPTUALIZATION

In order to provide harmonisation between different tailoring of [1], the methodology begins with the functional split between data and process requirements of [1]. This classification discovers the data the system is going to assess and manage on one site and the processes requiring that data on the other. Any assumption, inconsistency or missing aspect will be raised by two different validation tests as each process follows different standards.

Data Conceptualization

The methodology, used to conceptualise the standard, assesses each of the defined hierarchical structures composing the procedure, extracting its semantics (i.e. the data to be specified) and specifying the relationships between that data (e.g. arities, constraints, dependencies, hierarchical, exclusivity, etc.).

The result of the assessment of all the structures is a conceptual model which has been used for the validation of the standard by the targeted standard experts (i.e. the ECSS standardisation board) and some of the current users (e.g. EUMETSAT. EGS-CC).
Once this conceptual model was validated, different logical models were generated automatically using different software factories (e.g. NORMA) as described in [4] and [5]. The same process was repeated from logical models to physical implementations without having to invest more time in repeating the model creation for each of the technologies to be applied. Therefore, branching (as depicted in Fig.2) from a unique and validated conceptual model achieves two goals:

- The correct exchange of information and reusability between different tailoring according to needs (e.g. AIT, Operations, Simulations, Backwards Systems…)
- The creation of a unique extension point whenever modifications are required instead of the maintainability and consistency checking of several partial grammars, implementations, etc.

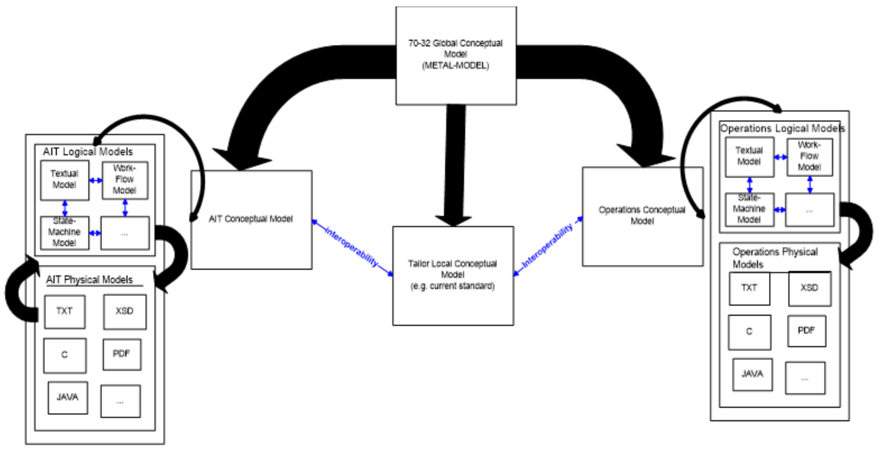


Figure 1. Three Level Conceptual Formalization

Exchanging information will be possible and correct due to the inherited semantic compatibility between local models. Sharing semantics ensures exchanged concepts and their relationships are the same, because they must comply with the same rules and constrains.
Adding or modifying semantic-structures creates minor divergences between sibling customizations. However, different tailoring creates a tree-like structure exponentially difficult to merge back in a single model. As branches diverge, incompatibilities usually appear between tailoring; which makes reusing specific extensions among them more difficult. Through the use of the common conceptual global model, any extension must be performed first in the meta-model, hence validating the extension, and afterwards pushing the extension to the concrete tailoring through the software factories, only requiring coding the specifics for the concrete implementation (e.g. label providers, graphical aspects).

Process Conceptualization

From a general point of view, the targeted standard describes an algorithm to execute procedures. This algorithm is based on a series of tasks, called steps, which must be performed at run-time.
Every algorithm is in essence, a process which has a goal to achieve. The process might be defined as a sequence of activities to be performed within the steps context by one (or several) specific actor(s). Consequently, the standard describes conceptual processes where each step is an embedded process.
Additionally, processes are supported by data (input, output and intermediate), which has been formalised in its totality in the previously explained meta-model.
All the required information for modelling a process (steps, data, actors, order) can be represented in a formal notation in BPMN (see [2]). This standard describes the creation of process diagrams in its composite tasks specifying the consumed and created data of each task, the actors or systems participating in each task, the possible interactions, contingency actions, etc. As such, this representation of the standard will be fully conceptual, hence allowing logical views for each different procedure that can be created according to the model.
Starting from the verified conceptual data model, the behaviour of each structure was analysed, based on the definitions contained in the standard and the experience VTCB has of previous implementations. With the behaviour described as functional requirements and constraints being explicitly declared, all the actions the system shall have to perform during the execution of the structure were defined. For each of this action, the dynamic instances of the procedure conceptual data are created and the corresponding BPMN diagram was generated referencing the instances which will be modified during the process and/or any information requested to the user from/in the execution engine.
The result of the process modelling and conceptualization allowed answering questions raised after several implementations of the standard have provided different behaviours if the same procedure is run by each of them.

VALIDATION

The validation of the models followed an agile approach to ensure all stakeholders agreed on the procedure standard needs before any further development was done. Several meetings were hold between parties until all procedure structures were discussed. Before the models' conceptualization documents were submitted two validation tests were performed:

1. The models are compliant with the normative of the applied methodologies (i.e. ORM and BPMN). The used tools contain validators to check models correctness (i.e. compliant with the normative) and complete (i.e. they do not depend on any other model and all possibilities are covered). In this case NORMA was the tool used for modelling in ORM and Intallio Designer for BPMN.
2. Already in-use procedures and every example contained in the targeted standard were tested in the conceptual models to verify no data was loss and the behavior would be the described one.

After both tests were passed, the documentation was submitted so the procedure experts assessed them prior to its acceptance. The change requests generated after the conceptualization were also send to the ECSS standardization board and the need to create a working group for updating the standard has been the received official answer.
After the automatic generation of the physical models and the process implementation, the different codes had to be integrated. Coming from the same conceptual model, the only integration issues to solve were those of communication due to the decoupling between client and server parts of the system architecture of the ASE5 system, in which the models are used to their full capability, by creating a preparation and execution environment for procedures

PROCEDURE SUPPORT SYSTEM DEVELOPMENT

Once the conceptual models were designed, the development of the system took two directions. On one side the client side had to provide editors for creating/visualising procedures compliant to the conceptual data model and the tracing of their execution. And on the server side, a single execution engine which handles the execution following the processes standardised in the process model.

Procedure Editors

While the semantics of a language is compliant with the common conceptual data model, the user is free to define any syntax that better suits its needs. This is the major goal and benefit of the methodology provided. As shown in Fig.2, the language syntaxes, a user may define, are instantiation of the conceptual data model of the standard. After the initial compilation of the language, a single result is produced which is harmonised for both editors.

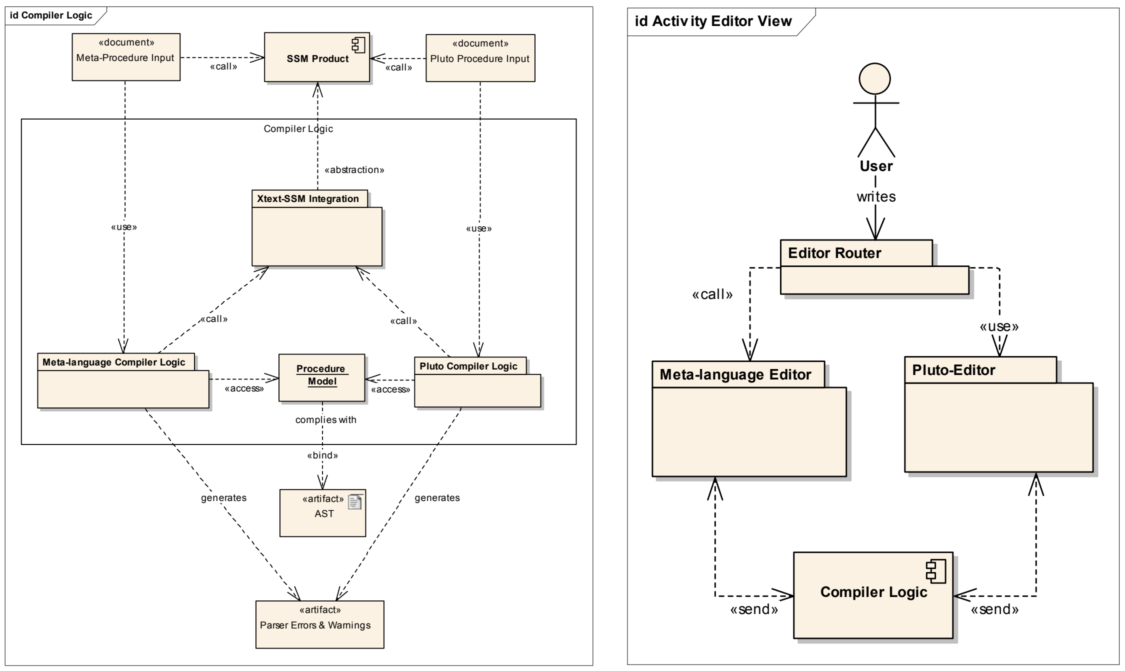
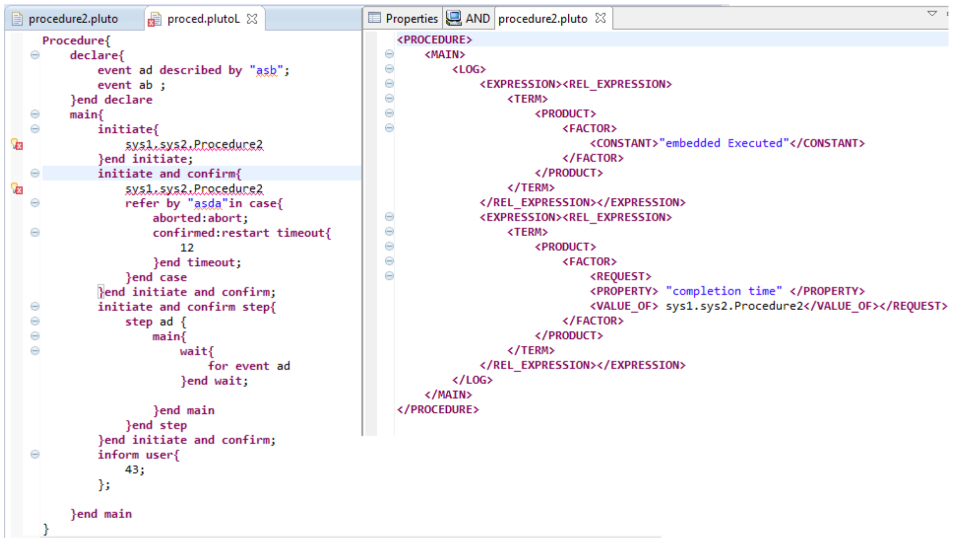


Figure 2 Procedure Compilers and Editors Diagram

However, the implementation of the compilers and editors required is not as cumbersome as it could be some years ago. Following the trend of using new technologies, the conceptual data model was generated from the resulting XSD into an Eclipse Modelling Framework (EMF) model. With it, and using the Xtext Library to define grammars explictly populating the procedure model elements; the compilers and fullly functional editors are automatically generated (see Fig.3) .



```
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
}
```

```
<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>
```

Figure 3 Procedure Editors

The main features of these editors are:

- Syntax checking/verification (i.e. static consistency check of the syntax);
- Auto-completion of the language syntax and support for coding templates;
- Bracket and word matching, lines numbers display, keywords highlighting;
- Undo/Redo of modifications in all views;
- Text formatting and indentation;
- Simultaneous edition of multiple procedures;
- Statically check the consistent use of data types inside the procedures;
- Ease of SSM services insertion with a procedure;
- Colouring syntax

The remaining work to be done is providing semantic validation checks (e.g. restrictions on values, constrains on the possible data types, etc.), quick fixes for some situations (e.g. missing declarations), custom messages for errors and special formatting rules.

Synchronisation between different procedure syntax

In order to prove interoperability between the languages and to allow the users to define in their preferred format while others can visualise it into theirs; transformation between the supported formats is required. As all formats comply with the same data model and this is kept as a model instance, specifying the syntax strings while traversing the model is required. This process is supported by the Xtext tool by the definitions of generator classes containing the support of the model and its structures.
This work is required to be performed n*(n-1) times, where n is the number of different languages supported by the system. However, the task does not take a long period of time and if, performance is not an issue, transitive translations can reduce the number of translators required.
A step forward in this translation is the synchronization between formats at edition time. As soon as the user modifies the procedure in one language and saves the changes, the other language representations are also modified to reflect the change. This is achieved by invoking the transformation of the code at saving time. The same behaviour, although not implemented, should be performed between textual and graphical representations.

Unique Engine Development

The Procedure Execution Engine is unique to the whole system because it does not process syntaxes but semantics. As the engine handles EMF Java Objects instead of textual representations, the input format of a procedure is not important, so a complete decoupling is ensured between client representation and engine input.
The procedure execution is based on a set of Executor objects created according to the definition of the common conceptual model. These objects contain the behavior methods (i.e. execute, start, stop, resume, pause). At high level, there is one executor per structure of the procedure (e.g. Procedure Main Body, Watchdog Body, Statement, Declaration Body, Repeat Body...). As the executor contains the reference to the structure EMF Java Object, the engine implements at high level the processes which were formalized in BPMN.
The engine will receive a Procedure EMF Object (which is a Java object). It will check the existence of the preconditions body and if it exists, it will execute it by creating the respective Executor object and checking its internal statements. Per statement type an implementation is provided which is fed by the data contained in the statement EMF Object.
Thanks to this approach a direct feedback can be provided to the final user between statement, structure and its timely execution result.

Performed Extensions and impacts

After the development of the editors where already finished, some new functionalities were required to be implemented. This, in a traditional compiler project would have a big impact as compilers tend to be not well maintainable due to language limitations (e.g. left and right conflict, recursion, etc.) and validation rules. Besides the grammar, modifications in the data model would also need to be performed, together with the implementations of their execution processes.
However and thanks to the methodology applied; after the extensions were identified, the data model and process models were updated to include them to discern if there was some conflict with existing behaviours or data. After rejecting some extensions due to the non-compliance with the standard (i.e. a process similar to cloning watchdogs), the following features were added as an extension to the normal behavior:

- Declaration of global constants: Users are allowed to declare constants which value do not change during the execution and can be used within calculations. Constants can be of any allowed data type.
- Augment the possible data types with:
    - Byte Fields: Set of bytes to be used for bitwise functions (xor, or, and, not). The bytes can be represented either by unsigned integers or hexadecimal values from 0…255.
    - Arrays ((Simple and Record): Resembling the list data types and using the definition of [AD1] for activity arguments, this has been upgraded to data type. There are two types of arrays: simple (a list of constant values) or complex (i.e. a list of other sets). The main property is that all elements of an array have to be of the same data type.
    - Record: Set of complex values which can mix both simple values and complex ones.
- Allow the explicit raising of events during the execution: The extended mechanism allows users to raise a previously declared event without the need to use a timeout or a continuation action as it is required in the current version of the standard.
- Assign a criticality to the declared events. Allowed criticalities are ‘critical’ or ‘not-critical’. The default behavior currently allowed in the standard [AD2] corresponds to the ‘critical’ property. However, if an event is ‘not-critical’, in case no watchdog monitors it, the event will be logged and the execution will resume with confirmation status not-confirmed for the statement raising the event.
- Declare and handle mathematical exceptions (i.e. arithmetic and class cast exceptions): Currently mathematical problems raised directly abort the procedure, such as division by zero. Currently the user can declare to watch for this type of errors and declare watchdog steps associated to them. In case no watchdog steps are defined, the default behavior of aborting will take place. Mathematical errors are considered ‘critical’.
- Define transactional parts within a procedure. Within a transaction the value of reporting data and variables are not updated (i.e. the notifications are not processed). As soon as the transaction ends the notifications are processed again and the references to RD will take the latest received value.

Specific syntaxes were created for each of these extensions in the supported languages (i.e. Pluto and Meta). As a consequence, transformations among them and validation rules were also developed in the client side.

At the server side, the EMF Java Objects representing these extensions were included and the behavior of some functions was modified according to the re-definition of the involved processes. For example, the behavior of the exception handling when events are not critical modified the traditional way of aborting a procedure if not catch

The development and testing time of these extensions were limited to less than two weeks.

CONCLUSSIONS

The targeted standard, up to the moment, is applied in a very constrained number of situations according to VTCB's feelings because its potential scope of applicability is by far greater. Having a formal language able to create procedures in customizable ways (including natural language) that control instruments, send/receive messages, call to other activities, etc. can be reused in several different business.

These convey not only the space-related one, but also robotics (i.e. controlling robotics devices, planning), augmented reality (e.g. creating the environment depending on the situation), management (e.g. creating the instructions for taking-up a position, filling forms, making deliveries to clients, organizing missions…), software engineering (e.g. defining requirements), creating templates, etc.

As the main logic behind the standard is to define everything as an activity, the examples above are only some of the possible cases where it can be applicable. However, it offers an idea of the potential the standard may have, if focused and explained correctly as it has been the result of the conceptualization process.

Defining and using properly the previously described meta-models, all these applications may be developed without having to invest a lot of work. They will only require the definition of the concrete activities that can be called upon by a procedure in the conceptual model and link them either with already existing commands or with newly developed ones.

In parallel, the process described in this paper can be applied to any standard or any new developed system. Not only for the proven results in terms of completeness, consistency and verified goals but also for the possibility to automate part of the development and reduce the maintenance of the system. This process is expected to be reused in the satellite projects (e.g. OPALE) around the Automation Exchange Language of the EGS-CC in order to reuse the single execution engine that is going to be developed in the Kernel of the EGS-CC.

REFERENCES

[1]ECSS-E-70-32C - Ground systems and operations-procedure definition language; issue 2.0,July 2008
[2]BPMN 2.0- http://www.omg.org/spec/BPMN/2.0/PDF, [cited 30 March 2014].
[3]ECSS-E-70-31C - Ground systems and operations-Monitoring and control data Definition; issue 3.0. July 2008
[4]Robert J. L. Schmaal, Herman Balsters, Serge Valera, "Formal Specification of a Meta Hierarchical Logical Data Model Using Object Role Modeling", OTM Workshops, 2011, pp. 370-379
[5]A Formalisation of the Normal Forms of Context-Free Grammars in HOL4, Aditi Barthwal and Michael Norrish; ISBN 3-642-15204-X 978-3-642-15204-7; 2010