Data-Based System Engineering: ICDs management with SysML

Thierry Le Sergent (SCADE System Product Manager)
Alain Le Guennec (SCADE System Project Manager)

Esterel Technologies

9, rue Michel Labrousse, 31100, Toulouse, France
thierry.lesergent@esterel-technologies.com
alain.leguennec@esterel-technologies.com

Keywords: Model Based System Engineering, Software Engineering, SysML, ICD

Abstract

System engineering best pratices are well described in handbooks and guidelines such as the International Council on Systems Engineering (INCOSE) handbook [1] and certification guidelines such as the ARP4754A Guidelines for Development of Civil Aircraft and Systems [2]. These clearly define the development and verification processes including system requirements, functional decomposition, and architecture design.

The OMG (Object Management Group) has defined the Systems Modeling Language (SysML) standard [4] specifically to support the system engineering development processes through models. The benefits of models versus Documents-based processes has been highlighted in many papers; it allows in particular, automated verification of design consistency.

Several tools such as Papyrus [6] from the Eclipse foundation support the SysML language. Even if, in practice, discrepencies still exist amoung tools, it allows, in principle to move, with limited efforts, models from one vendor to another one, removing the fear from users to be prisoners from proprietary languages.

Despite these good arguments, the usage of SysML tools is not yet widely deployed for large industrial projects. One concern is the management of Interface Control Documents (ICDs) [4] that is at the center of most industries' system engineering processes, and that is not supported in a straightforward way by SysML.

This paper highlights the challenge in supporting ICDs by SysML tools, and demonstrates how these requests are supported by the SysML-based Esterel Technologies' SCADE System® product [7].

Introduction

Interface Control Documents, know as ICDs, are widely used in all industries, at different levels of the system development and verification processes. One definition, independent from the standards, is given at http://www.chambers.com.au/glossary:

"*An Interface Control Document (ICD) describes the interworking of two elements of a system that share a common interface. For example, a communications interface is described in terms of data items and messages passed, protocols observed and timing and sequencing of events. An ICD may also describe the interaction between a user and the system, a software component and a hardware device or two software components. This class of document is typically used where complex interfaces exist between components that are being developed by different teams. It is jointly prepared by the interfacing groups.*"

As many teams are involved at different levels, different times, and different expertise, in the development of complex systems, the difficulty faced by industries is to manage the consistency of the interface of each individual component of the system, from the textual requirements to the detailed specification of the interfaces for the development of each component separately. The NASA handbook [3] states that a clean process must be set up to "*identify and resolve interface incompatibilities and to determine the impact of interface design changes*".

The manner proposed in this paper is to rely on model-based technologies to manage all required information in a consistent model and generate the ICDs from the model.

That model shall:

- Handle “data” that is the information exchanged by each item of the system (as stated in [3], it can be “*cognitive, external, internal, functional, or physical*”)
- Handle the system architecture topology(ies) (there may be different architectures; functional, logical, physical)
- Handle fine grain traceability with textual requirements and allow impact analysis of requirements or design changes

This paper details the development of ICDs from the tool point of view; it answers to the question “what features shall be supported by an integrated toolset to support a model-based development of ICDs that ensure efficiency of user work and consistency of the result. Other aspects, such as organizations, qualification w.r.t. standard such as DO-1178C are not elaborated here.

The paper is structured in the following way:

First, the principal item of ICDs, the data, is discussed. Second, a major issue with SysML is explained, and a solution is proposed. This allows the third section to show how data and system architecture topology specified in SysML can be reconciled. The last section can then cover the complete process development, from the textual requirements, to the generation of consistent ICDs, and the initialization of the development of software components.

Data dictionary

A common technique used for the development of complex software predominant systems relies on a set of “data” exchanged between the software components of the system.

These databases are used as detailed specifications exchanged between OEM and equipment manufacturers. Also, a set of in-house tools have been developed over the years for verification activities and code generation, so it is important for a new system engineering tool to be able to import and export data dictionaries saved in these databases.

The inconvenience of the management of these data with databases is that they are technically separated from the model defined in the system engineering tool. Keeping the information coherent, such as data names and topological information (source and target of each data), is a challenge.

The solution proposed is the following:

1. Data can be defined as SysML values of a block, with a name and a datatype, on which a set of properties can be defined (see more details below),
2. Data are managed in tables allowing easy find/sort/filtering and direct editing of their properties,
3. Tables of data can be exported/imported as .csv files, allowing inter-exchange with external databases,
4. Data can be allocated on SysML connectors or ports.

These steps are detailed here.

Data properties

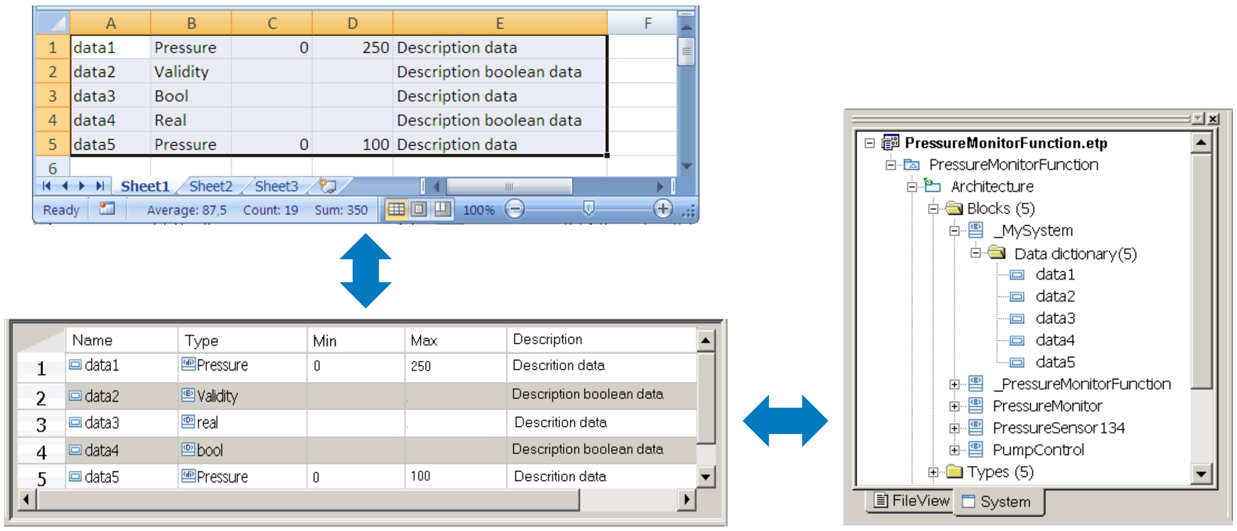
Data, functional or logical, shall carry information, such as value range, timing information, position in a message frame, etc. Defining this set of properties and the value for these properties must be supported in a friendly way by the engineering tool. The technical foundation of SCADE System, the Eclipse MDT Papyrus project [6], allows for defining such information through the standard profiling mechanism. This customization has the advantage of being an effective standard. But, the profiling mechanism is a rather low-level mechanism, not easily managed by system engineers who are experts in their domain but not necessarily in UML technology.

To provide the necessary means to system engineers, SCADE System comes with an annotation mechanism reused from the SCADE Suite® toolset [7] that has been proven to be very effective in many complex industrial projects. A simple textual language allows defining structured information types (aka “annotation schema”) associated to any model object, e.g. the SysML block values, that implement the data. Annotations are transparently implemented in terms of UML profiles, for interoperability’s sake.

The data and allocations done as part of the model are, of course, traced with high-level requirements where appropriate, and reported in the documentation generated automatically.

Import/export data

As all model objects, data are presented in the object tree view of the IDE. A table representation is also available both to provide a comprehensive view of the data defined in a block, with customizable columns to display any data properties, are as the key pivot for import/export capability to common separated value (.csv) files or MS-Excel tool as presented in the following screenshot.



	A	B	C	D	E	F
1	data1	Pressure	0	250	Description data	
2	data2	Validity			Description boolean data	
3	data3	Bool			Description data	
4	data4	Real			Description boolean data	
5	data5	Pressure	0	100	Description data	
6						

Sheet1 Sheet2 Sheet3

Ready Average: 87,5 Count: 19 Sum: 350 100%

	Name	Type	Min	Max	Description
1	data1	Pressure	0	250	Descrition data
2	data2	Validity			Description boolean data
3	data3	real			Descrition data
4	data4	bool			Description boolean data
5	data5	Pressure	0	100	Descrition data

- **PressureMonitorFunction.etp**
  - PressureMonitorFunction
    - Architecture
      - Blocks (5)
        - _MySystem
          - Data dictionary(5)
            - data1
            - data2
            - data3
            - data4
            - data5
        - _PressureMonitorFunction
        - PressureMonitor
        - PressureSensor134
        - PumpControl
      - Types (5)

FileView System

The column of SCADE System tables are not simple textual columns; they are associated to the data properties, as defined in the previous section, or predefined property, for example the datatype property inherited from the SysML definition. Exporting content of these tables to Excel or .csv files is a straightforward copy of the string contained in each cell. Importing is more challenging, as the textual information imported becomes model information. In particular the string “pasted” in the Type column is automatically bound to datatype with the same name available in the visibility scope of the block. If such datatype is not available in the model, a feedback is given to the user to enrich the model.

As the amount of data can be considerable in an industrial model, a specific caution has been taken in the development of Papyrus to support of tables of more than 10.000 lines.

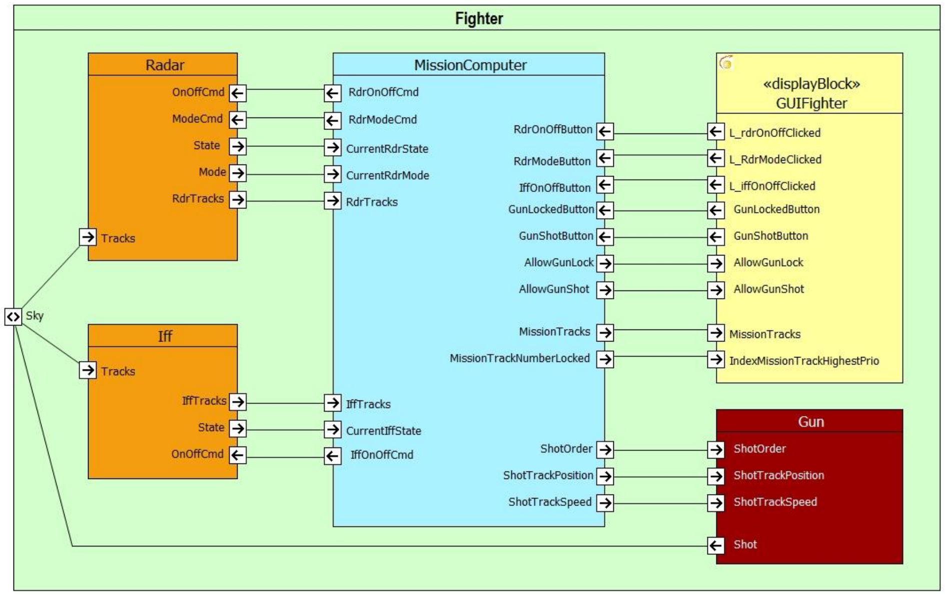
Data allocation on topology

The key aspect of the methodology proposed is to “map” the data as defined in the previous section to the architecture topology defined as a SysML model.

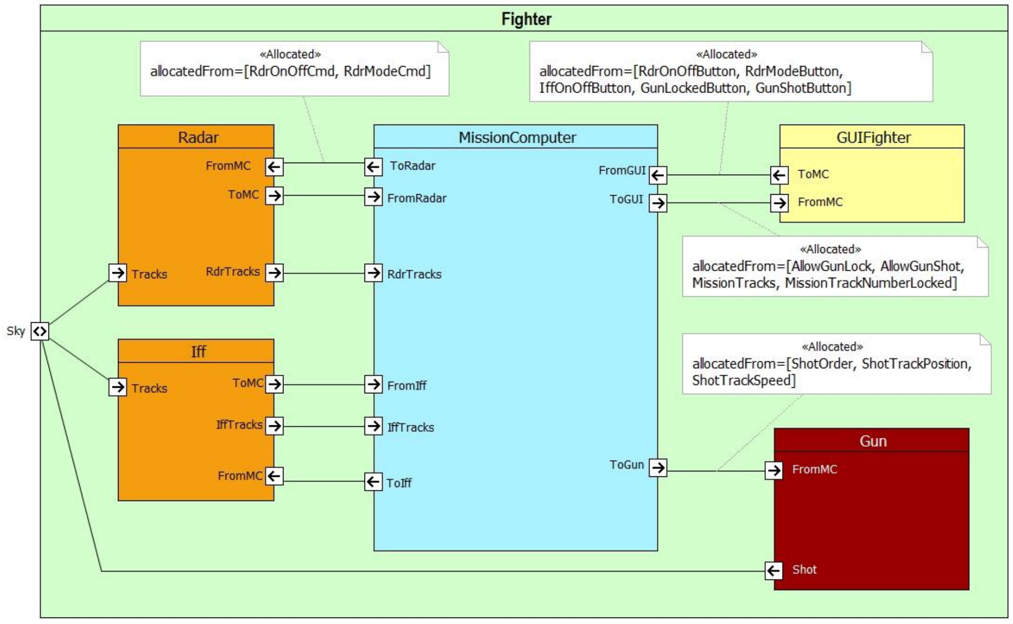
The constructs used from SysML to define an architecture topology are only the Packages, Blocks, Flow Ports, and connectors. The mapping is realized by SysML allocations.

The following figures introduce, with a simple example, the proposed modeling style. More details are given in the remaining portion of the paper after a major SysML issue is detailed in the next section, as it must be solved beforehand.

Classical representation in SysML. The information exchanged between the components are expressed by the ports, carrying in particular a name and a direction.



Data-based representation in SysML. The ports represent only “gates” through which the information implemented as “data” can circulate. Data are allocated to Ports or to the connectors carrying the information between the ports (all allocations not displayed here).


Fighter
«Allocated» allocatedFrom=[RdrOnOffCmd, RdrModeCmd]
«Allocated» allocatedFrom=[RdrOnOffButton, RdrModeButton, IffOnOffButton, GunLockedButton, GunShotButton]
Radar
FromMC
ToMC
Tracks
RdrTracks
MissionComputer
ToRadar
FromRadar
RdrTracks
FromIff
IffTracks
ToIff
FromGUI
ToGUI
ToGun
GUIFighter
ToMC
FromMC
«Allocated» allocatedFrom=[AllowGunLock, AllowGunShot, MissionTracks, MissionTrackNumberLocked]
«Allocated» allocatedFrom=[ShotOrder, ShotTrackPosition, ShotTrackSpeed]
Sky
Iff
Tracks
ToMC
IffTracks
FromMC
Gun
FromMC
Shot

Note that SysML allocations are also supported in SCADE System tables, allowing a more scalable view than the nice, but limited to the size computer screen, graphical representation. This also allows for import/export capability.

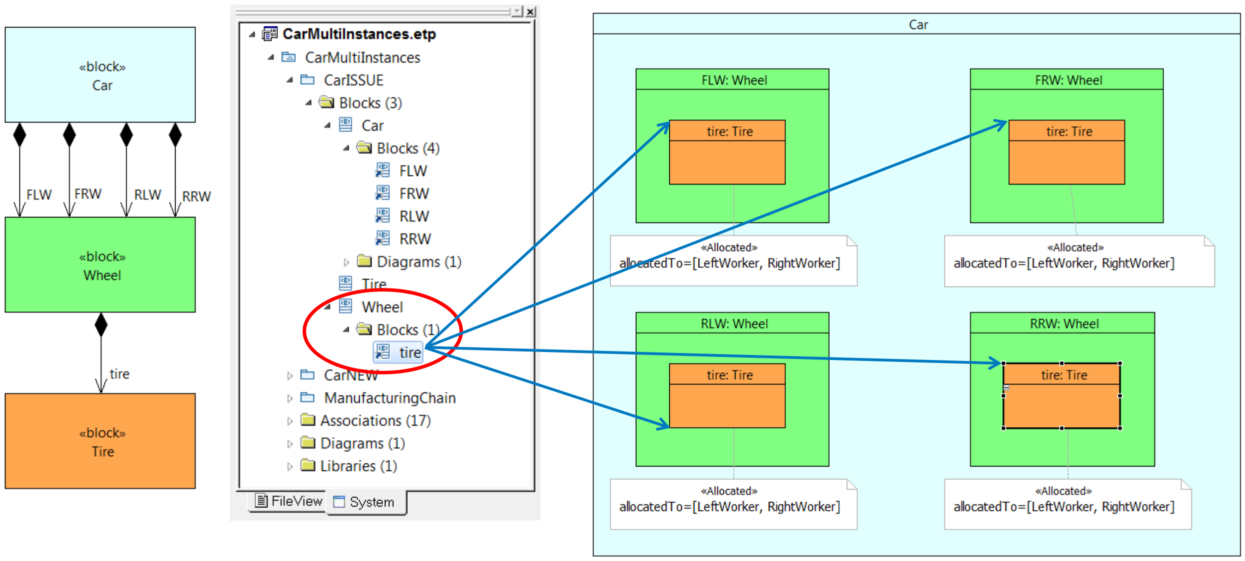
Challenges with SysML

One main issue to deal with is the “multi-instantiation” of architecture components; it is indeed a great benefit of modeling language, such as SysML, to support the definition of “blocks”, whatever they represent, that can be reused several times in the model through “part”. This feature is naturally used to model the fact that, for example, an equipment is “duplicated” in the system.

Without entering the formal definition of SysML parts versus. blocks, the following example highlights the issue system engineers are faced with for the specification of information related to each individual component.

The model as specified with a SysML Block Definition Diagram on the left side of the following figure represents a Car with four Wheels, FLW (Front Left Wheel), FRW, RLW, RRW, each Wheel containing one tire.

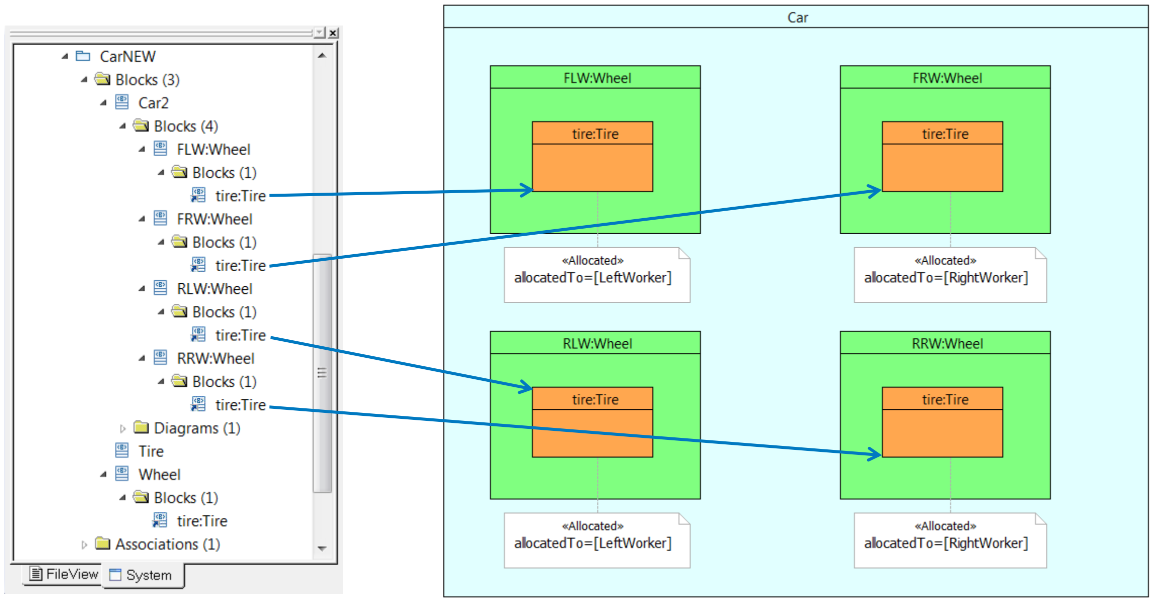
Classical SysML modeling tools handle model objects, as pictured in the tree view in the center of the following figure: flat list of blocks, each containing parts. The problem arises on the second level of the “conceptual hierarchy”: the tools handle only one object to represent the four conceptual tires. There is no direct mean to specify, e.g. the pressure of each of the four tires as there is only one “object handle”. It is not possible to specify that the tires on the left side of the car are allocated to a worker on the left side of the manufacturing chain, while the tires on the right side of the car are allocated to a worker on the right side of the manufacturing chain. Trying to set such allocations leads to the allocation of all tires (in fact _the_ object tire) to both the LeftWorker and RightWorker, as pictured on the right side of the figure with the SysML Internal Block Diagram representing the car; all tire graphical representations are bound to the same object.



The straightforward means used in practice consists in copying the block Wheel as many times as needed, leading the system engineers to manage, manually, the consistency of the copies.

To allow system engineers to manage, in a straightforward manner, their architecture designs, SCADE System manages, automatically, the “replica” from the specification specified with blocks and parts as in the BDD shown above. All intermediate “blocks” and “inheritances” are automatically created internally to comply with the SysML standard, and are managed by the tool to remain consistent with the user design. A simple view, as shown in the following picture, is shown to the user; it abstracts the internal objects, but provides “handles” allowing setting property value and allocations to individual conceptual component.

Thanks to this management, data and architecture topology can now be assembled taking into account the replication of component defined once through block types.



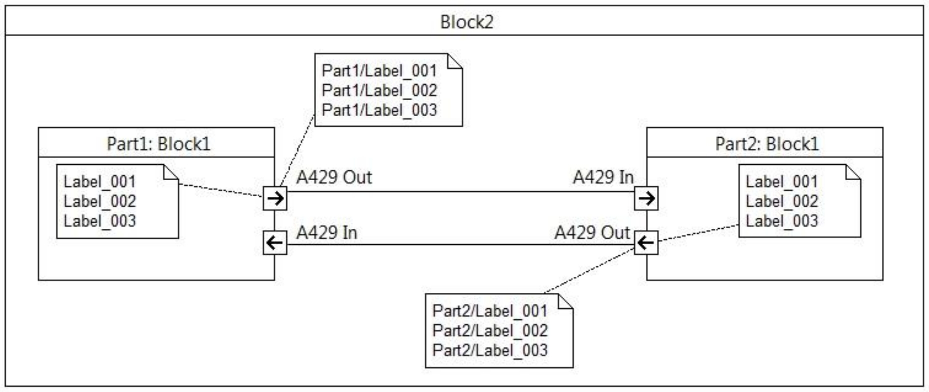
Data and topology reconciliation

Managing a huge set of data in an industrial model is cumbersome. Two mechanisms facilitate this management:

1. Automatic replication of the data produced by component replica,
2. Propagation of the data along the hierarchical architecture,

Data in blocks

Data being implemented as SysML properties of the blocks, the replication of blocks detailed in the previous section is of great help to manage the data. Consider the example presented in the following figure.



Block2 contains two identical parts, Part1 and Part2, defined from the block Block1. The “internal interface” of these parts are the same; it is made of data Label_001, Label_002 and Label_003, defined in Block1 (for example imported from a .csv file), allocated to its port A429_Out.

Now, in Block2, these data must represent different information. The replication of Block1 handled by the tool makes the data produced by the two parts indeed different information, accessed from Block2 through a single path. As shown in the next section, aliasing these names with names local to Block2 is also possible.

Data propagation

Independently of the multi-instantiation topic discussed in the previous sections, propagating data along a path of connectors through intermediate levels of blocks would be better supported by tools than requiring many manual, error prone, actions. An interactive IDE allows the system integrator to efficiently specify how the data are propagated in the model.

Data is not necessarily defined in the block producer as shown in the previous section. It should also be possible to define all data in one upper block, for example allowing the import of a global data dictionary. Data could also be defined on the recipient side so that a block comes as a standalone component defining its complete interface independently from the way it is connected.

Data propagation consists of specifying, without ambiguities at any interface connected to several connectors, how each data is broadcasted or multi-casted through the connectors. For that purpose SCADE System offers an interactive window showing, for each connector selected, the list of data provided and required on both ends. Three actions are proposed:

- “Pushing” a data from source to target,
- “Pushing” a data from target to source,
- “Binding” data selected on the right and left side.

At the end, before generating ICDs, it is also important to verify globally on the model that the propagations have been kept consistent, in particular after model topology modification. To support that needed feature, SCADE System comes with a checker that automates the verification of rules that can be customized to adapt to the precise methodology defined. Typical rules would be:

- All defined data is produced, i.e. allocated to an output port
- No data is “lost” in intermediate connections

- No data arrive ex-nihilo on intermediate connections
- Etc.

The technical means proposed for data propagation is detailed below. It enforces an important property: all allocations between data and block ports is local to block. This allows:

- The automated replication of the dependencies with block replica,
- The capability to “export”, and then re-integrate, standalone blocks for efficient collaborative work.

A data of a block is considered as produced by the block (a “provided data” of the block) if it is allocated to an output port of the block. Symmetrically, a data of a block is considered as consumed by the block (a “required data” of the block) if it is allocated to an input port of the block. Data of a block that are not allocated to any port are internal data of the block.

When a block is used as a part of another block, the provided and required data corresponding to that part are automatically transposed into the enclosing block to make them available there. Initially, those data are internal data of the enclosing block (until they are themselves allocated to ports of the enclosing block, possibly), and are bound to the corresponding data in the inner block; binding between two data means they represent the same data but possibly at different levels in the hierarchy and under possibly different names. The initial name of the transposed data is formed by the qualified name of the data being transposed (but that initial name can later be changed for readability), and so on recursively.

Example:

If a block ‘B’ contains two parts ‘c1’ and ‘c2’ typed by a block ‘C’ having one data ‘d’ connected to a port of ‘C’, then two data are automatically transposed in ‘B’:

- ‘c1’.’d’ → ‘c1.d’ (can be later renamed by the user to ‘d1’ for example)
- ‘c2’.’d’ → ‘c2.d’ (can be later renamed by the user to ‘d2’ for example)

The binding relation forms equivalence sets (the set of data bound together). There must be at most one producer in the set. For any consumed data, it is possible to retrieve the producer, if any.

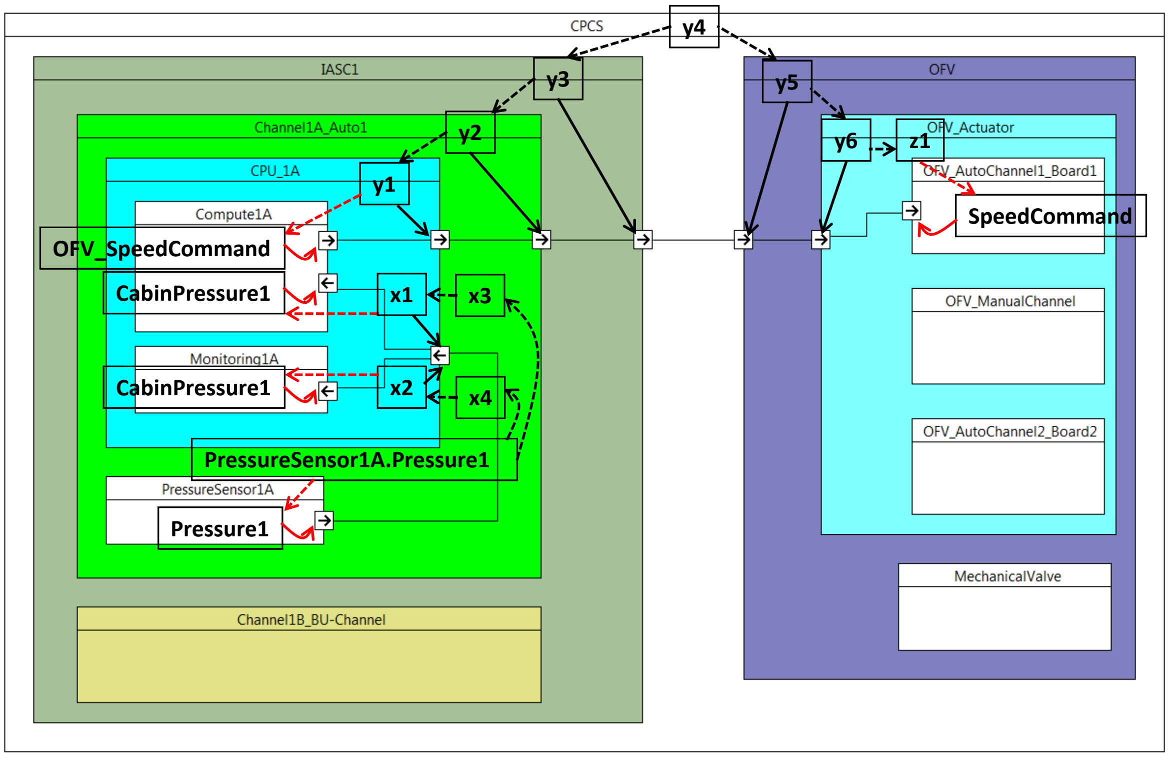
The propagation consists simply in binding data. The tool helps in creating the missing data and its allocation to the port. On each connector end, the local data allocated to the end connector port (for a port of the block), or local data bound to a data allocated to the port of an inner part, is proposed for propagation through that connector. From the selection of one of these proposed data there are three propagation cases:

- A corresponding data already exists at the other connector end: a simple binding between the two data implements the propagation.
- Upward propagation: the data to propagate is already allocated to the port of the inner part; so it has been transposed upwards in the block. Upward propagation consists of allocating the transposed data to the port of the block. Note that this leads automatically to the creation of a new transposed data in each block containing a part of that block, allowing further propagation of the data.
- Downward propagation: the tool creates a new data in the nested block on the other side of the connector and allocates it to the connected port of the nested block. This data is therefore transposed upwards at the same level as the connector, where it will be bound to the initial data. An optimization in downward propagation consists in binding directly the outer data to the just created inner data.

The result of this propagation is shown in the following figure representing a realistic model of an aircraft Cabin Pressure Control System (CPCS). Allocations are represented as plain arrows, binding represented as dashed arrows.

At the bottom of the picture: As data *Pressure1* of *PressureSensor1A* is allocated to its port, a transpose data *PressureSensor1A.Pressure1* (default name) is automatically created by the tool. Same for *CabinePressure1* from parts *Compute1A* and *Monitoring1A*, transpose data renamed *x1* and *x2* to make the name fit in the picture. Upward propagation of *x1* and *x2* is realized with their allocation on the input port of *CPU_1A*; this creates transposed data *x3* and *x4*. Propagating horizontally the data is realized through the binding between *PressureSensor1A.Pressure1* and *x3* and *x4*.

At the top of the picture, downward propagation is highlighted: *OFV_SpeedCommand* is allocated to the out port of *Compute1A*, so transposed data (renamed) *y1* is created by the tool. User propagates it upward until block *CPCS*, then downward until *OFC_Actuator*, local data named *y6*. In *OFV_AutoChannel1_Board1* a data *SpeedCommand* is defined and allocated to the port, so a transpose data *z1* is automatically created. Propagating *y6* to *SpeedCommand* is realized through the binding between *y6* and *z1*.



Data grouping can also be defined within a block to aggregate other data (including other data group). When a data-group is allocated to a port or connector, all its data are considered to be allocated too. When a data-group is transposed upwards in the block hierarchy, it becomes the group of its transposed data. This allows propagating data more quickly, by grouping them when they are related.

ICDs system development process

System development and verification processes involve many steps that are out of the scope of this paper. A comprehensive methodology handbook for the aeronautic domain can be found in [14]. The focus of this paper is on the technical aspects of the production of ICDs. From the tool features detailed in the previous sections, an efficient tool supported development process for the ICDs can be set-up.

The tool itself does not impose a particular process; several ways can be followed. In particular, the System Engineering development process can rely of several refinement layers, e.g. functional, logical and physical, that are domain and company dependent.

The overall process can be applied in the same way at any system development level; it consists in:

1. Definition of the data in items of the system and use a traceability tool to establish a link between these model elements and the requirements defined in textual documents.
2. Specify the connections between the components, and propagate the data along this topology.
3. Verify the consistency of the model with automated rule checker such as SCADE System checker.
4. Extract information from the consistent model: ICDs tables, reports, and SCADE Suite operators interfaces for critical software components to be developed with SCADE Suite [10, 11, 12].

Step 1 can be mainly manual in particular for a totally new system, or heavily rely on the SCADE System import feature when a large part of the data pre-exists from a previously developed similar system. Traceability is supported by the SCADE LifeCycle Requirements Management gateway tool. It allows straightforward impact analysis when requirements are modified.

Steps 2 and 3 are detailed in the previous section. The important thing is that the checking rules are customized to verify that the methodology defined by the method team has been followed by the designers.

Step 4 is now detailed in the two sections below.

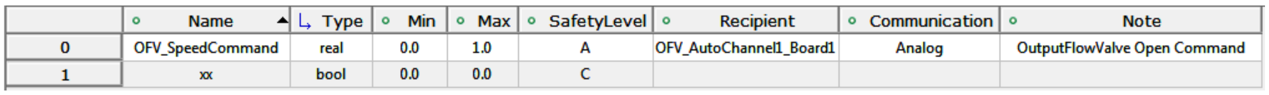
When the system development process is set-up in several refinement layers a possible usage of the technics presented is the following:

- As part of the functional analysis of the considered subsystem, the functional data exchanged between the functions are defined and allocated on the connectors representing the flow dependencies between the functions,
- When the architecture of the system is defined and the functions are allocated to the architecture components, one has to define how the functional data shall be implemented. This is done by defining, using the same tool feature, logical data. The functional data are allocated on the logical data, which in turn are allocated on the architecture topological elements, connectors or ports,
- In complex architectures where data are packed/unpacked at several architecture levels, the mechanism can be used again, modeling that several logical data can be transported by e.g. a message, itself defined as a data allocated to topological elements. This later step involving binary encoding and communication protocols has not been yet investigated; we consider for now in the paper the functional or logical architecture only.

Customizable ICDs tables

Once the data are properly managed in the model, generating ICDs becomes easy as all the required information is consistently managed in the tool. The tool offers an Applicative Programing Interface (API) to traverse, verify and extract any information for documentation generation from a program or script.

The SCADE System tool goes a bit further in providing customizable tables: in any block, table of its data can be displayed, showing in columns the properties set in annotations attached to the data. Query columns can also be specified: through simple user defined scripts interacting with the model API, these columns can be filled with e.g. any information extracted from the model. For example the table display below can be easy specified:



	Name	Type	Min	Max	SafetyLevel	Recipient	Communication	Note
0	OFV_SpeedCommand	real	0.0	1.0	A	OFV_AutoChannel1_Board1	Analog	OutputFlowValve Open Command
1	xx	bool	0.0	0.0	C			

In addition to the name and datatype of the data, its user defined properties “Min”, “Max”, “SafetyLevel” and “Note” are displayed. The content of the user defined “Recipient” column is the result of a query providing the name of the final block owning a port onto which the data is allocated. The blocks can be defined as “final” with e.g. another annotation. In a similar way, the content of the “Communication” column is the result of a query providing the content of a specific annotation set to one of theconnectors through which the data “circulate” (i.e. data allocated to its two ends).

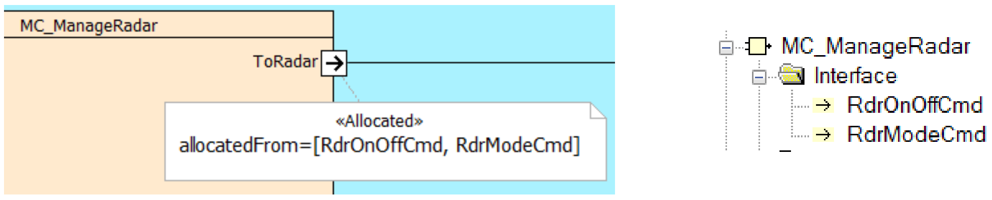
All the information required in ICDs tables can be customized with a few scripts, allowing an always up-to-date representation of the ICDs. And as SCADE System tables can be exported in a click to .csv files, there’ is no need to develop dedicated model reporter or gateway to external data bases.

Synchronization with SCADE Suite software development tool

Once the system description is complete and checked, the individual software blocks in the system can be refined in the form of models in SCADE Suite. Automatic and DO-178B Level A-qualified code generation can be applied to the SCADE Suite models.

A mapping between SysML block interface and SCADE Suite operator interface has been formally defined and implemented in SCADE System Synchronizer [7].

The SCADE System Synchronizer tool has been extended to support this new way of specifying block interfaces. Instead of synchronizing the block ports with SCADE Suite operator inputs and outputs, it relies on the data allocated to these ports. The following figure shows the result of the synchronization between such SCADE System block and SCADE Suite. The datatype themselves are translated on user request as detailed in [13].



Conclusion

Keeping ICDs consistent with architecture design drawn with Visio-like tools is not easy and not efficiently done with reviews. Managing models make things much more efficient.

The SCADE System tool proposes a method and tool features to manage Interface Control Document (ICDs) in a model based way that is compliant to the SysML standard.

It relies on block values and annotations transparently implemented in terms of UML profiles. Data dictionaries are managed with tables that can be customizable to show any information required with the various ICDs. The key feature for a friendly IDE is to manage in a proper way the multi-instantiation and propagation of data; these features are implemented in the SCADE System product.

Finally, the interface of software critical components can be generated automatically from the system model.

The solution reconciles the benefits of the model based technology with the deployed system engineering processes based on ICDs. Evaluations are engaged in large aeronautic projects at the time this paper is released.

References

1. “Systems Engineering Handbook, a Guide for System Life Cycle Processes and Activities”, SE Handbook Working Group, INCOSE, January 2010.
2. “ARP4754 Rev A. Guidelines for Development of Civil Aircraft and Systems”, SAE Aerospace, Revised 2010-12
3. “NASA Systems Engineering Handbook”, NASA/SP-2007-6105 Rev1
4. “Training Manual for Elements of Interface Definition and Control”. Vincent R. Lalli, Robert E. Kastner, and Henry N. Hartt, NASA Reference Publication 1370, January 1997.
5. “OMG Systems Modeling Language (OMG SysML)”, OMG, Version 1.2, June 2010
6. Papyrus, http://www.eclipse.org/papyrus
7. Esterel technologies SCADE products, http://www.esterel-technologies.com
8. “SCADE System, a comprehensive toolset for smooth transition from Model-Based System Engineering to certified embedded control and display software”, Thierry Le Sergent, Alain Le Guennec, François Terrier, Yann Tanguy, Sébastien Gérard. ERTS 2012
9. “Integrating System and Software Engineering Activities for Integrated Modular Avionics Applications”, Thierry Le Sergent, Frederic Roméas, Olivier Tourillion. 2012 SAE Aerospace Electronics and Avionics Systems Conference, Phoenix Arizona, USA.
10. “The Synchronous Dataflow Programming Language LUSTRE”, N. Halbwachs, P. Caspi, P. Raymond, D. Pilaud. Proceeding of the IEEE, September 1991.
11. “A Conservative Extension of Synchronous Dataflow with State Machines”, J.L. Colaço, B. Pagano, M. Pouzet. EMSOFT’05
12. “SCADE 6: A Model Based Solution For Safety Critical Software Development”, François-Xavier Dormoy. ERTS 2008
13. “Bridging UML and Safety-Critical Software Development Environments”, Alain Le Guennec, Bernard Dion. ERTS 2006
14. Methodology Handbook “Efficient Avionics Systems Engineering with ARP-4754A Objectives Using SCADE System”, http://www.esterel-technologies.com