Improved Model Based approach to address the Architecture, Design and Specification of complex systems.

Matlab Expo 2018
19 June | Paris

Jean Duprez, Airbus Operations SAS
19 June, 2018
How? → Easy & efficient use.

“Make things simple & easy!"

"True simplicity is derived from so much more than just the absence of clutter and ornamentation. It's about bringing order to complexity... You have to deeply understand the essence of a product in order to be able to get rid of the parts that are not essential.”

Jony Ive (Chief Design Officer at Apple)

www.apple.com
How?

1. Optimize model structure by **Factorizing Model** elements.
   - Make modeling of re-usable elements simple & easy.
   - Promote genericity of model elements by better managing specificities and variability.

2. Improve design quality & capitalization, by **Formalizing Design Expectations** into the model.
   - Cascade design expectations into the model structure.
   - Use model elements to formalize Requirements.
   - Ensure full traceability between requirements and with associated model elements

3. Represent the System through **Multiple Views**.
   - Reduce model complexity
   - Better consider expectations
   - Better address the information
   - Using most adapted graphical representations to address each concerns in the most efficient way.
   - Using several abstraction levels
   - Considering only relevant information
   - Using the more adapted description formalism
How ? ➔ **Complexity** challenge.

How to address **Complexity** ?

➔ Break complexity into sets of simpler parts.
How ? \(\rightarrow\) **Complexity** challenge.

**How to address Complexity ?**

\(\rightarrow\) **Break complexity** into sets of simpler parts.

\(\rightarrow\) **Design** each elementary part …

\(\rightarrow\) **manage integration.**

Allow to get Deep understanding of each sub-set.

---

19 June, 2018

Improved Model Based approach to address the Architecture, Design and Specification of complex systems.

- Jean Duprez, Airbus Operations SAS
How to address Complexity?

- **Break complexity** into sets of simpler parts.
- **Design** each elementary part...
- **manage integration**.

Optimize the breakdown by **improving** the design model structure.

Allow to get Deep understanding of each sub-set.
Factorizing the design by: generating re-usable components.

Example of a simplistic Side-Sticks design:

2 side sticks, for the captain & first officer,
• with 2 axis: longitudinal & lateral,
• with a potentiometer on each axis,
  • with a track to measure the angular position,
• and a track for monitoring.
How? \(\rightarrow\) **Complexity** challenge. 1 \(\rightarrow\) **Optimize model structure.**

Use of **Model Referencing.**
How? ➔ **Complexity challenge.** ➔ **Optimize model structure.**

**Use of Model Referencing.**

From 14 sub-systems to 14 instantiations of 3 models.
How? ➔ **Complexity** challenge. 1 ➔ Optimize model **structure**.

**Factorizing** the design by: generating re-usable components.

**No duplication** of re-usable components.

➔ referencing instead of copy-pasting.

⚠️ But, model referencing is not as simple and easy to use as subsystem blocks.

➔ Make it **simple & easy** to use.

*(thanks to specific customizations)*

- Make modeling through models and model blocks as easy as with Sub-Systems.

- Make elements re-using as simple as a Copy / Paste.

- Automatically **manage interfaces** and busses *(management of data types, creation and modification of Bus-Objects, propagation of buses modifications to Bus-Selectors, etc…)*.

- Make it robust to renaming.

Factorizing the design by: generating re-usable components.

Promote Genericity by advance management of specificities.

→ managing variability at the most relevant level.
Example of the Side-Sticks Acquisition:

1. Optimize model structure.

Example of the Side-Sticks Acquisition:

Factorized thanks to Model Referencing.
Example of the Side-Sticks Acquisition:

Gains for Pitch orders and Roll orders definitions are different.

Factorized thanks to Model Referencing.

• Use of Model variants.

• Use of Variant Subsystem.
  ➔ Describe an “abstract envelope” representing several variants.
How? \(\rightarrow\) **Complexity challenge.** \(\textcircled{1}\) \(\rightarrow\) **Optimize model structure.**

- Use of model arguments to model specificities while keeping the model generic.
How? → **Complexity challenge.**

1. **Optimize model structure.**

- Use of model arguments to model specificities while keeping the model generic.

 ➔ Make it simple & easy to use.

Thanks to customizations to create and manage enumerates to:

• Manage variants configuration.
• Manage local specificities by passing enumerates as Model Argument.

By capturing and formalizing,

• the context of use of each element,
• design configurations.
How to address Complexity?

- Break complexity into sets of simpler parts.
- Design each elementary part …
- manage integration.

How to ensure that this giant puzzle will answer need?
How to address Complexity?

- **Break complexity** into sets of simpler parts.
- **Design** each elementary part ...
- **manage integration.**

Early Testing & Corrective loops
How ? ➔ **Analysis & Simulation.**

**How to address Complexity ?**

- Allow to manage complexity but not to master it.

- Break complexity into sets of simpler parts.
- Design each elementary part …
- Manage integration.

Early Testing & Corrective loops

---

19 June, 2018

Improved Model Based approach to address the Architecture, Design and Specification of complex systems.

- Jean Duprez, Airbus Operations SAS
How to address Complexity?

1. Use of **Requirements Engineering**.

2. Use of **Requirement based System Engineering**.

   - **Use of Modelling**, to Structure the Specification,

   - Formalizing expectations at each level.
How? \( \Rightarrow \textbf{Requirements Engineering.} \ (2) \ \Rightarrow \textbf{Use of Requirement.} \)

Traceability between models and requirements.

Vertical traceability between requirements.
How to address Complexity?

Use of Requirement based System Engineering.

Use of Modelling,
• to Structure the Specification,
• to Formalize Requirements.

Formalizing expectations at each level.
The function shall inhibit the order when a failure is detected.
How? \(\rightarrow\) **Requirements Engineering.** \(\rightarrow\) **Use of Requirement.**

**Statement:**
The function shall inhibit the order when a failure is detected.

**Rationale:**
To avoid potential impact of failures on the aircraft motion.

**Additional information:**
The function shall inhibit the order when a failure is detected.
To avoid potential impact of failures on the aircraft motion.
How to address Complexity?

Allow to well manage expectations cascading, but not to master the design.

Formalizing expectations at each level.

How? ➔ Requirements Engineering.
How to address Complexity?

How to get deep & full understanding of the overall system?

3 Use of Systemic approach.

Considering the system as a whole.

Focusing on specific aspects the overall system, thanks to:

- Extraction of relevant data
- Usage of adapted view points

Abstraction
How to address Complexity?

Global Model is a set of consistent diagrams or models.

to address
• different scope,
• different abstraction levels,
• different point of views.

How? → Adapted Graphical representation → Systemic approach.

Abstraction

Extraction of relevant data

Usage of adapted view points.
How? ➔ Adapted Graphical representation ➔ Systemic approach.

The use of multiple views allows to efficiently address each concern.

Data Centric approach:

Considering models as means to
• visualize, edit and analyze design data.
• To simulate associated design behavior
Conclusion

1. Optimize model structure by Factorizing Model elements.
   - Make modeling of re-usable elements simple & easy.
   - Promote genericity of model elements by better managing specificities and variability.

2. Improve design quality & capitalization, by Formalizing Design Expectations into the model.
   - Cascade design expectations into the model structure.
   - Use model elements to formalize Requirements.
   - Ensure full traceability between requirements and with associated model elements

3. Represent the System through Multiple Views.
   - Using most adapted graphical representations to address each concerns in the most efficient way.
   - Using several abstraction levels
   - Considering only relevant information
   - Using the more adapted description formalism

- Reduce model complexity
- Better consider expectations
- Better address the information
Conclusion

1. Optimize model structure by **Factorizing Model** elements.

2. Improve design quality & capitalization, by **Formalizing Design Expectations** into the model.

3. Represent the System through **Multiple Views**.

- Reduce model complexity
- Better consider expectations
- Better address the information

Deployed in a **collaborative framework**

Based on **Simulink customizations** (done in collaboration with the MathWorks consulting team).

Ensuring **full scalability**
Conclusion

Presented example:
12 models.
17 model blocks.
37 instances.
Maximum interface decomposition:
6 “sub-interfaces” through 2 levels.

Typical real example:
> 50 models.
> 100 model blocks.
> 500 instances.
Through ~10 levels.
Maximum interface decomposition:
> 4 levels.
> 25 sub-interfaces.

Expected global size:
> 3 000 models.
> 6 000 model blocks.
> 30 000 instances.
Through >10 levels.
> 10 000 requirements.
> 10 parallel active users editing same scope.

Ensuring full scalability
Perspectives

1. Optimize model structure by Factorizing Model elements.

   - Initiate the functional analysis and structure the deriving functional description.
   - Allow checking the design definition towards expected operational deployment.
   - Use scenarios to support testing and automatic testing.

2. Improve design quality & capitalization, by Formalizing Design Expectations into the model.

   - Use Scenarios modeling to model operational use cases.
   - Ease the capture of functional expectations.
   - Additional viewpoint to address the system from a procedural point of view.

3. Represent the System through Multiple Views.

4. Use Scenarios modeling to model operational use cases.

   - Ease the capture of functional expectations.
Thank you