Digitalization as enabler of successful system integration

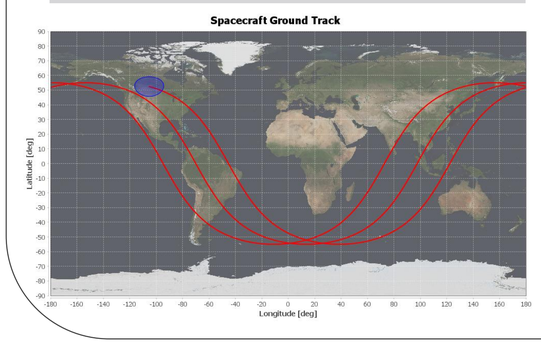
Increasing design quality while reducing development time? An example FireSAT mission.

Felix Löser MSc [1], Dr. Marius Riestenpatt gen. Richter [1], Dr. Stephan Rudolph [2]
[1] IILS mbH, Leinfelden, Germany, felix.loeser@iils.de, [2] Institut für Flugzeugbau, Universität Stuttgart, Stuttgart, Germany, rudolph@ifb.uni-stuttgart.de

Integrating satellite subsystems into a consistent system design is one of the biggest bottlenecks in current day-to-day engineering. As system design and mission scope mature over the development cycle, individual subsystems or components have to be redesigned to fulfill the more defined needs of the system. These redesigns are time-intensive and expensive. To overcome this challenge, graph-based design languages enable engineers to formalize their design knowledge into a machine executable form. Each system, subsystem and component can be automatically designed, meaning that a re-design is just an automated re-execution of a given design language. Thus, system integration is transformed into a process of automatically updating subsystems until they align with the overall system requirements.

Mission Definition

Mission Definition
- Goal → Fire detection
- Payload → Infrared Camera
- Lifetime → 5 year mission



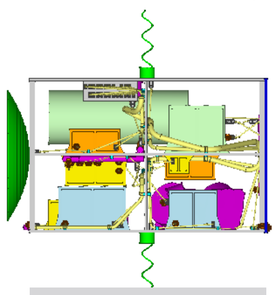
Payload Design
- Accuracy → Detection within 1 km
- Resolution → 20 to 30 m

Orbit
- Altitude → 700 km
- Inclination → 55°

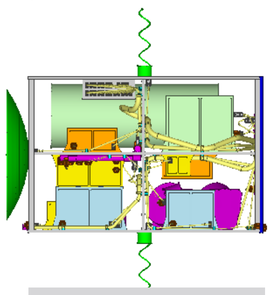
Mission requirements mainly impact the payload design and orbit choice. They are interconnected, e.g. a camera's focal length depends on the resolution and distance to target. Further requirements such as de-orbit maneuvers and mission life-time impact further subsystems.

Wire Harness

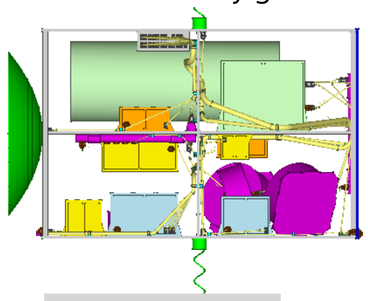
Each component has a power demand and generates data to be processed. The wire harness provides the necessary connection between components and power supply / data processing. Utilizing a generic routing design language, the connectors, fixings and cables can be dynamically defined and the wire harness automatically generated.



Resolution: 30 m
Total cable length: 97.1 m



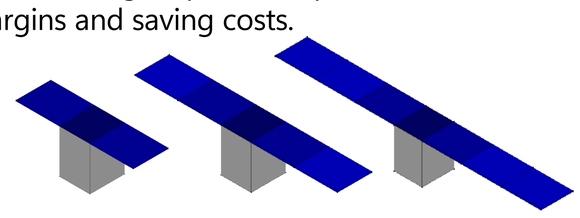
Resolution: 25 m
Total cable length: 92.4 m



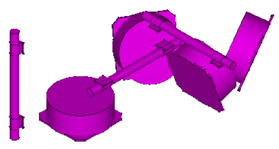
Resolution: 20 m
Total cable length: 116.8 m

System and Component Design

System design is a highly interdependent process, resulting in many redesigns. While traditional manual redesigns are time-intensive and costly, automated system designs provide quick results with consistent quality, reducing design margins and saving costs.



Varying number of solar panels depending on required power

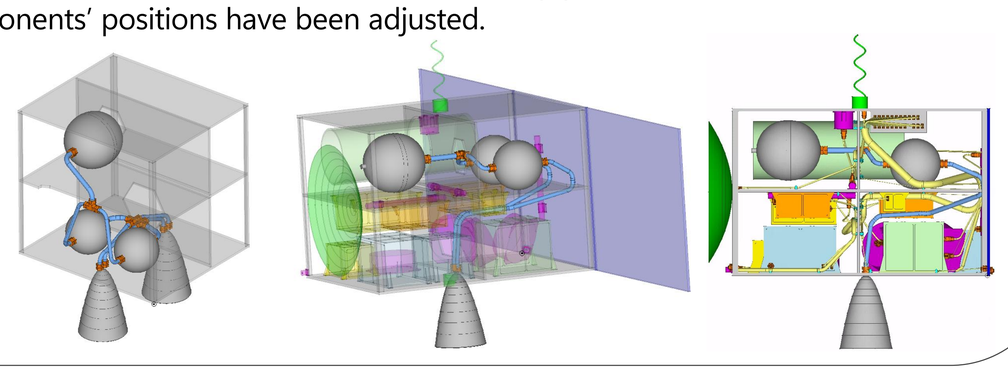


Reaction wheels and Magnetorquers

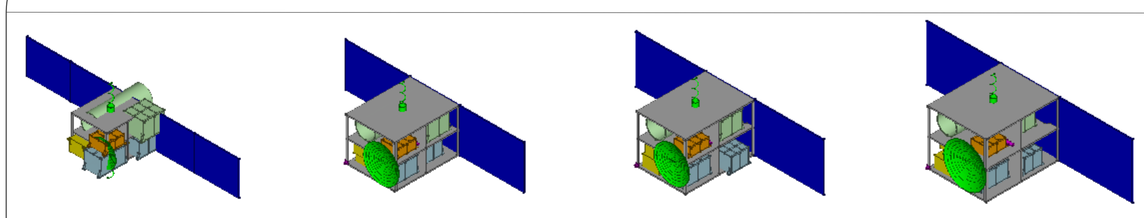
Components are either chosen from a predefined list or sized based on symbolic equations. Automated component design enables quick processing of updated system requirements.

Piping and Propulsion System

The propulsion system provides the necessary Δv to perform orbital maneuvers, e.g. the required de-orbit maneuver at mission end-of-life. The system shown below is a blow-down bipropellant system with helium as pressurant, hydrazine as fuel and nitric oxide as oxidizer. To fit the tanks and pipes within the satellite bus, some components' positions have been adjusted.

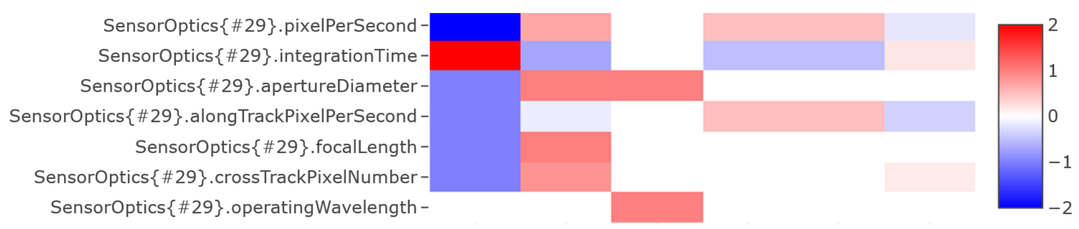


Structural Integration



Integrating components into the structure leads to regular subsystem redesigns. As the size of the satellite bus increases to accommodate the other systems' components, the amount of solar panels decreases as each individual panel increases in size. The larger satellite bus also requires more powerful reaction wheels, resulting in further automatic redesigns.

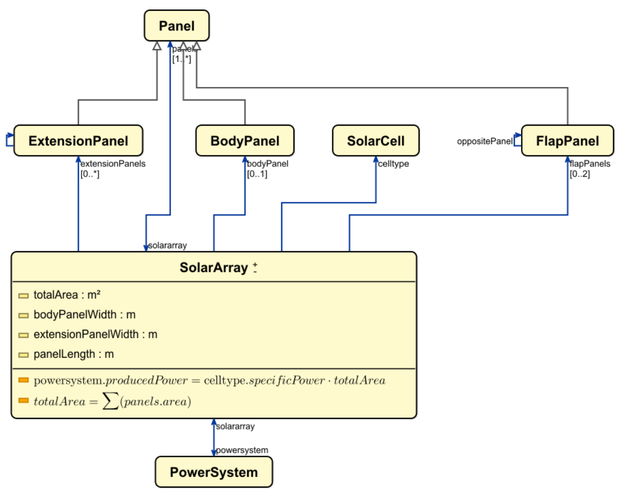
Evaluation



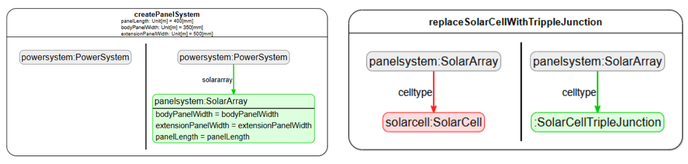
Evaluating design decisions enables the improvement of future designs and providing higher quality results. Heatmaps are a visual form of sensitivity analysis, allowing a comprehensive visual analysis of interacting design parameters.

Graph-based Design Languages

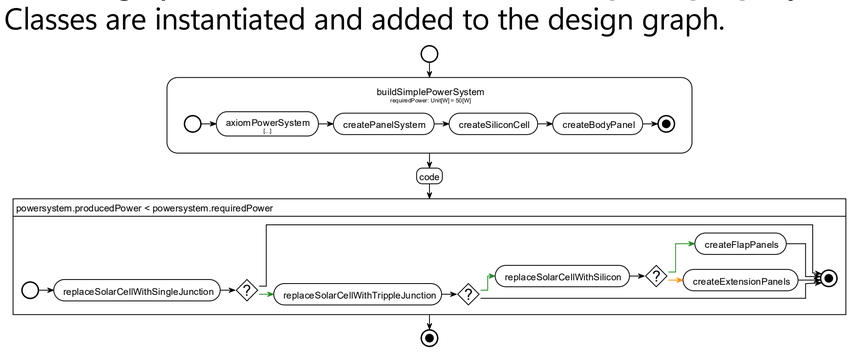
The results presented above have been achieved with graph-based design languages, developed and executed with the Design Compiler DC43.



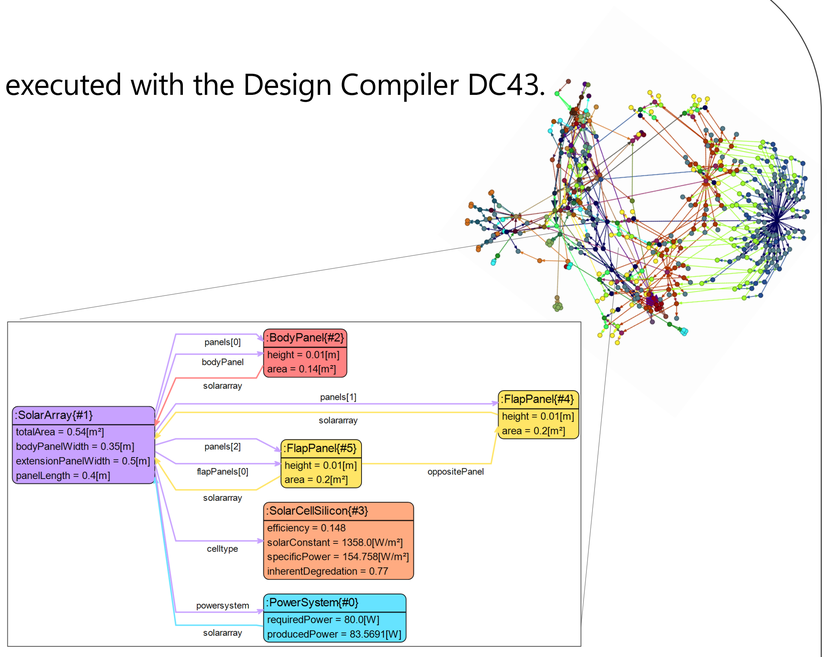
Ontologies as **Class Diagrams** provide design **vocabulary.**



Rules as **graph transformations** define the design language **syntax**. Classes are instantiated and added to the design graph.



Production Systems as **Activity Diagrams** define **design logic** and order of executed rules. **Control structures** enable design **decisions**.



Design Graph represents a specific generated model. All design knowledge is available within the graph nodes. Filtering and mapping yields domain specific models such as CAD, FEM and more.