TNO ESI

Powered by industry
and academia

AUTOMATIC SYNTHESIS AND DEPLOYMENT OF COMPONENT-BASED SOFTWARE SYSTEMS

Jeroen Kouwer (Thales) and Benny Akesson (TNO-ESI)

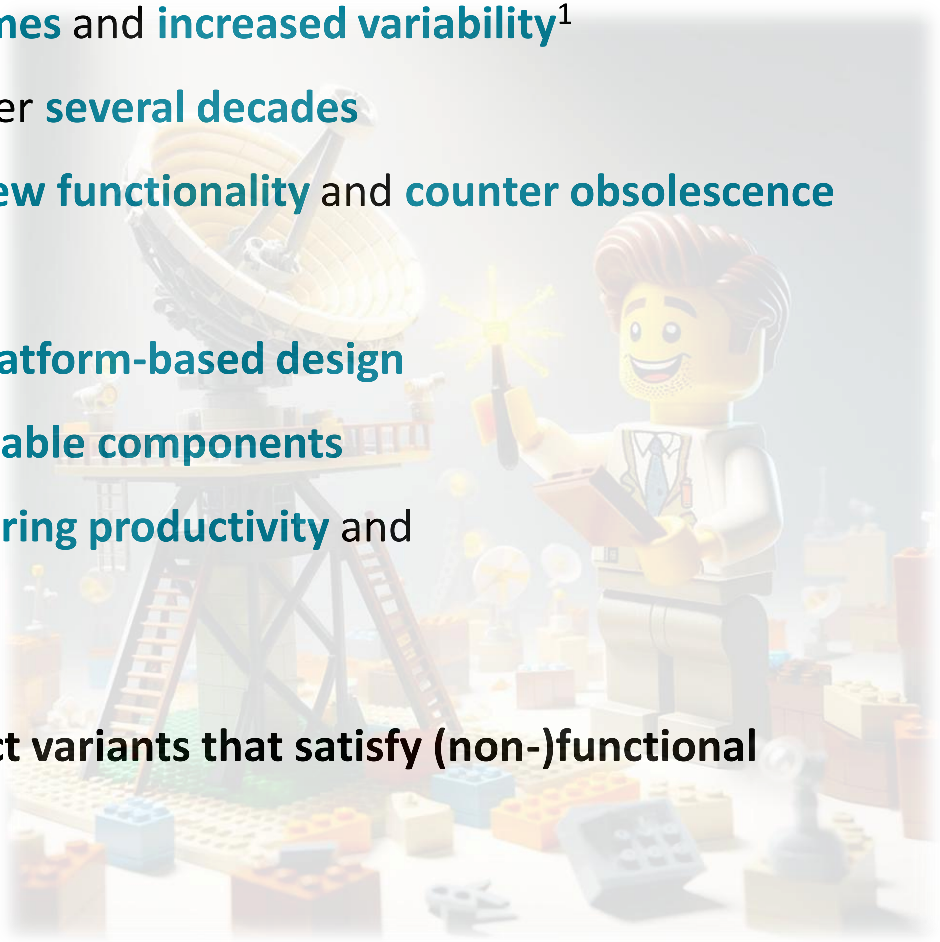
INDUSTRY CHALLENGES

The high-tech equipment industry is challenged by **long life-times** and **increased variability**[1]

- Many **product variants** must be developed and evolved over **several decades**
- Upgrades of hardware and software components to **add new functionality** and **counter obsolescence**

The industry is addressing this challenge by moving towards **platform-based design**

- Developing and evolving systems from a **repository of reusable components**
- Additional innovation required to further **increase engineering productivity** and address **scarcity of skilled engineers**[2]



How can we effectively develop and evolve (software) product variants that satisfy (non-)functional requirements as technology evolves?

[1] Hendriks, T. and Azur, S. "Vision and Outlook for Systems Architecting and Systems Engineering in the High-tech Equipment Industry" TNO Report 2024 R10542, 2024
[2] A. van der Werf, et al. "Made in NL: The value of the Dutch high-tech manufacturing industry," PricewaterhouseCoopers, 2024

KEY INGREDIENTS

1. Specifying the software components to integrate requires considerable **expertise**
   - Need for **democratization of engineering** to address scarcity of engineers and **optimization** to improve competitiveness of system

2. Software component integration takes a lot of **time and effort**
   - Need for **automation** to increase engineering productivity

3. Updating system configurations and software components as technology changes is **time-consuming**
   - Need for technology-agnostic **abstractions** and **automation**

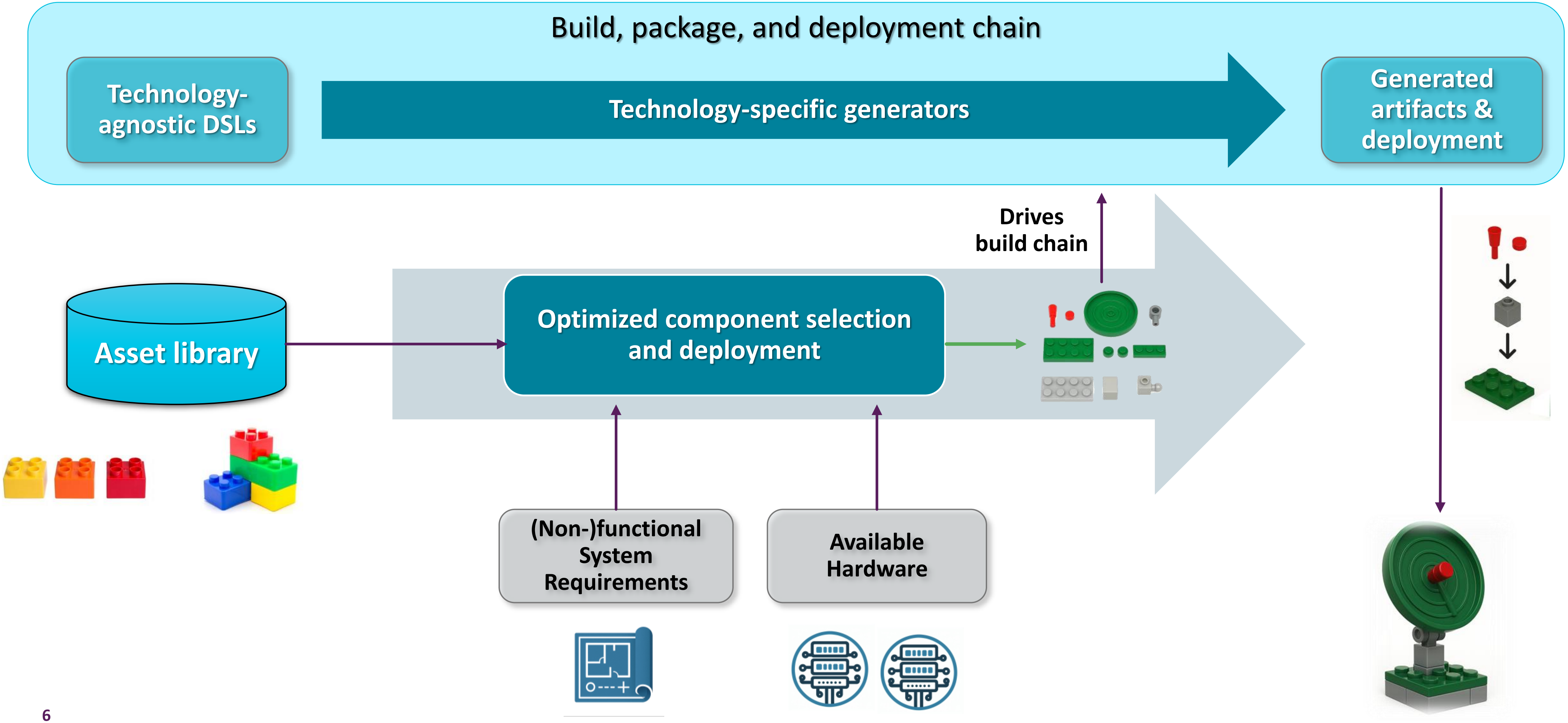
TECHFLEX APPROACH

TechFlex addresses the challenge of **increasing engineering productivity** associated with **system diversity and evolution** at the level of the **software platform**

We propose a **model-based approach to specification and deployment** with three steps:

1. **Technology-agnostic specification** of reusable building blocks (**abstraction**)
2. **Technology-specific generation** of build, packaging, and deployment artifacts (**automation**)
3. **Synthesis of component compositions and deployment** (**democratization, automation, optimization**)

TECHFLEX WORKFLOW



PRESENTATION OUTLINE

Introduction

Technology-agnostic Specification and Deployment

Synthesis of Component-based Systems

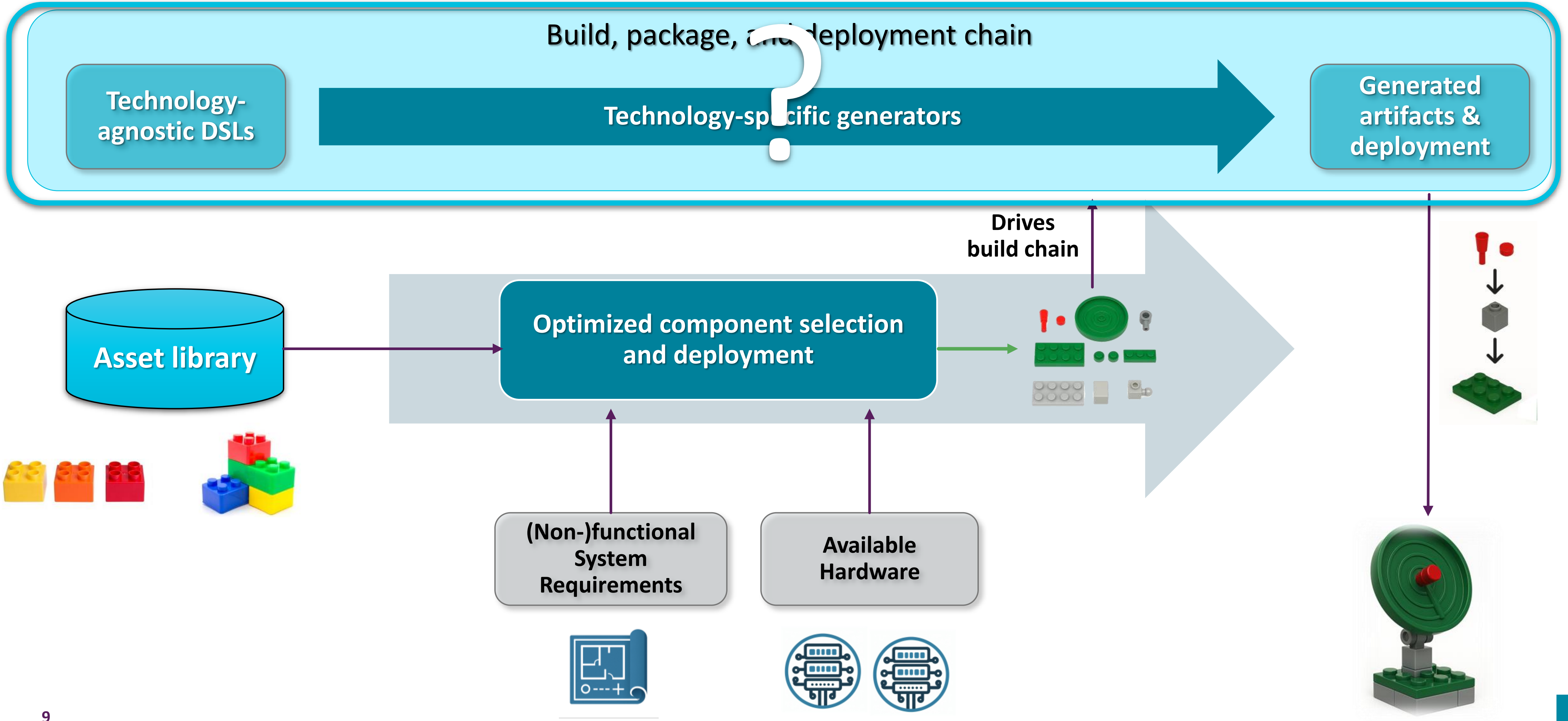
Validation

Conclusions

2

TECHNOLOGY-AGNOSTIC SPECIFICATION AND DEPLOYMENT

TECHFLEX WORKFLOW



SOFTWARE LIFECYCLE MODEL

To enable **technology-agnostic specification**, we started with an analysis using a **Software Lifecycle Model**

- Distinguishes **different phases** in the software lifecycle
- Captures all **artifacts**, e.g. components, interfaces, container images, their **relations** and **technology dependencies**
- Structuring software development in this way ensures that changes **only propagate downwards** through phases

Specified

Artifacts:
Specifications of

- Interfaces
- Components
- Modules
- Product instance

Implement →

Implemented

Artifacts:

- Source files
- Header files

Technology dependency:

- Implementation language, e.g. Java or C++

Generate →

Generated

Artifacts:

- Bundles
- Applications
- Runnable Units

Technology dependency:

- Platform
- Containerization technology, e.g. Docker or Podman

Deploy →

Instantiated

Artifacts:

- Deployment files

Technology dependency:

- Orchestration technology, e.g. Kubernetes

TECHNOLOGY-AGNOSTIC SPECIFICATION

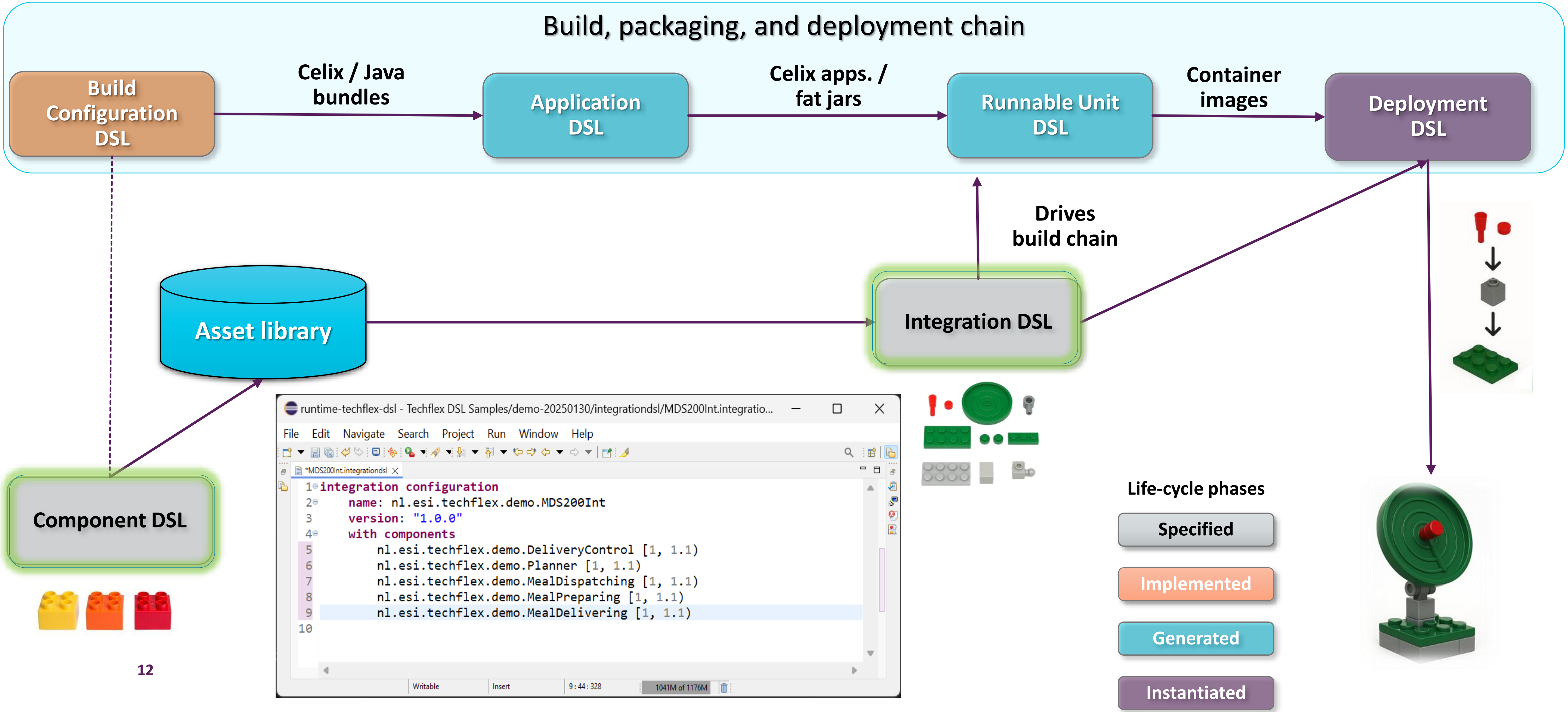
We defined a new family of **technology-agnostic DSLs** to specify elements and relations in all life cycle phases

- **Abstract** from the identified technology dependencies
- Variants are specified as instances of DSLs **minimizing manual intervention**
- Used existing Component DSL and added **additional DSLs** tailored to existing development processes

Create **technology-specific generators** to automatically create relevant development artifacts

- **Automation reduces effort** of repetitive tasks
- Technology-specific generators are **phased in and out** as software technologies evolve
- Address technology changes at the level of the **product family** instead of the individual variant

TECHFLEX DSL WORKFLOW



SYNTHESIS-BASED ENGINEERING OF SOFTWARE-INTENSIVE SYSTEMS

Engineering approach → / ↓ Development step	Traditional Engineering	Model-Based Engineering	Verification-Based Engineering	Synthesis-Based Engineering
Requirements design	Document-based	**Technology-agnostic specification and deployment**	**Model-based (formal)**	Model-based (formal)
System design	Document-based		Model-based (formal)	**Computer-aided (formal, synthesized)**
Realization in software (implementation code)	Traditional software engineering (coding)		Code generation (fault-free code)	Code generation (fault-free code)
Verification (against requirements)	Testing		**Formal verification (model checking)**	**Correct-by-construction (guaranteed)**
Validation (of requirements)	Testing		Testing + Simulation	Testing + Simulation

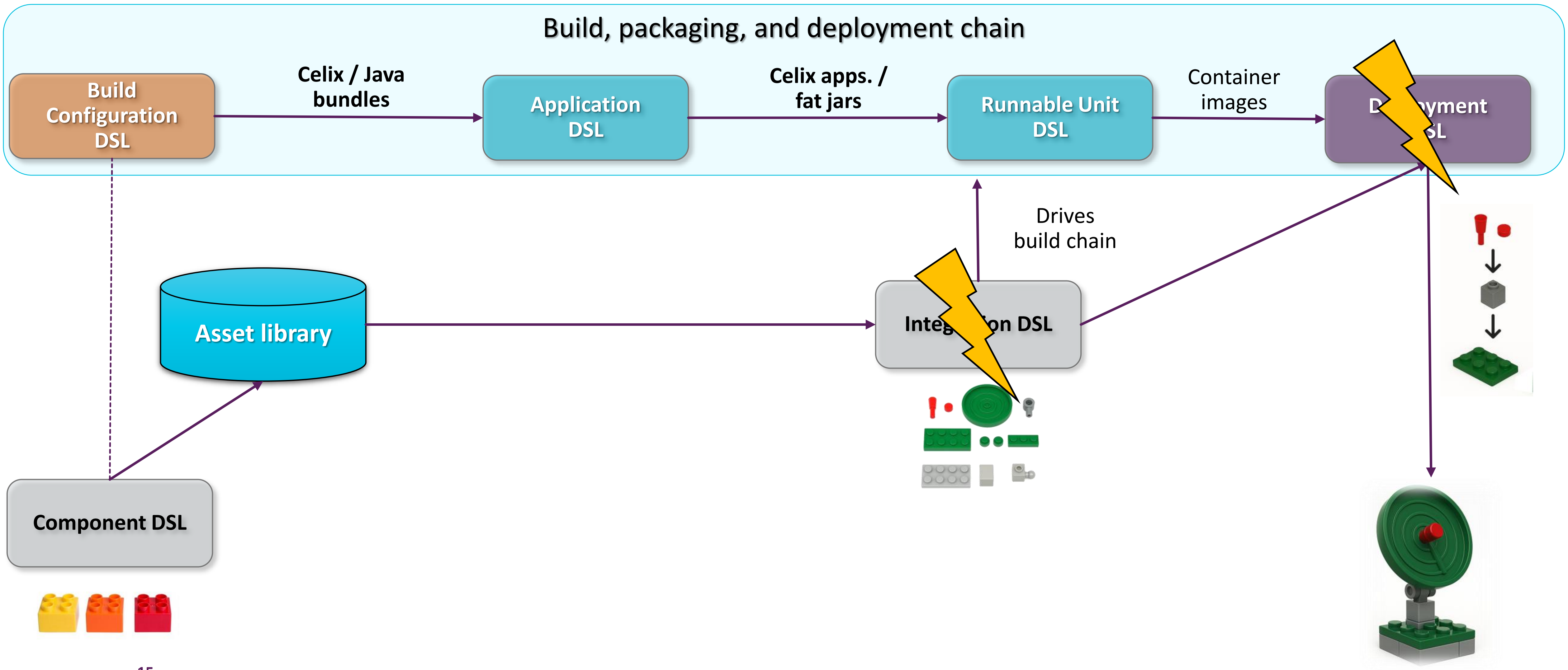
Legend: Manual work / Focus (Semi-)automatic

(Simplified overview.)

3

SYNTHESIS OF COMPONENT-BASED SYSTEMS

TECHFLEX DSL WORKFLOW



SYNTHESIS OF COMPONENT-BASED SYSTEMS

Is it really necessary to manually specify the composition of components and their deployment?

- Specifying the components to integrate and where to deploy them requires **significant effort** and **expertise**

Synthesis **automatically configures a system** from a **repository of reusable components**

- Uses **models of components** and their provided/required **capabilities**, **resource budgets**, and **non-functional behavior**, as well as **optimization criteria**

Potential benefits of system synthesis

- **Automation increases engineering productivity**
- Synthesis **democratizes** system integration, **reducing the dependency on experts**
- Design **optimization** may **increase quality/performance** while **reducing cost**

BUILDING ON EXISTING WORK

Components are modelled according to the **OSGi capability-requirement model**[1]

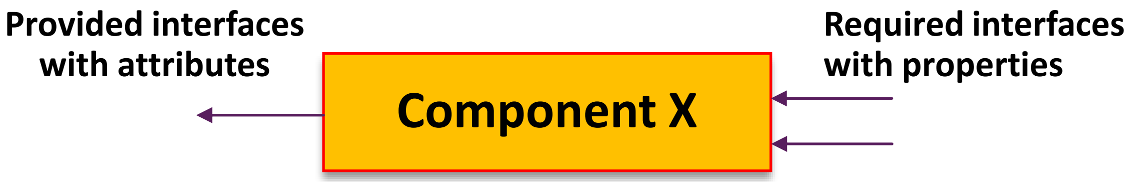
- Provided and required **interfaces** and **capabilities**
- Provided **attributes** and required **properties** of an interface (like quality)

Resource model considers **compute cores** and **memory** as **consumables**

Optimization using **Genetic Algorithm**[2]

- **Tunable** computation/accuracy trade-off
- Provides good results in a **few seconds**

Proof-of-concept tool developed by Thales and used as starting point in this project



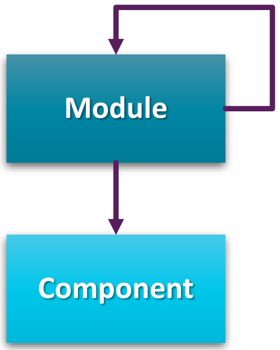
[1] "Using Requirements and Capabilities" [Online]. Available: https://blog.osgi.org/2015/12/using-requirements-and-capabilities.html

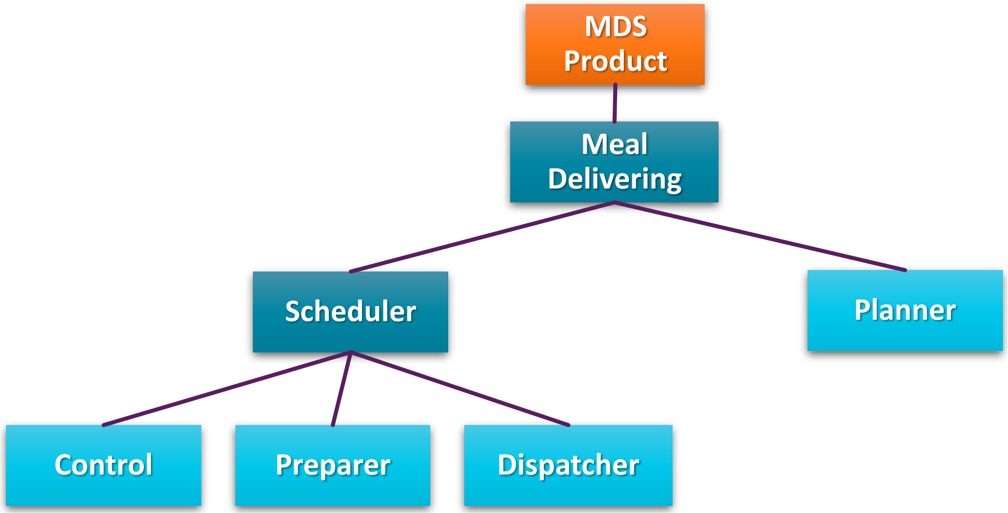
[2] Distributed Evolutionary Algorithms in Python [online], https://github.com/DEAP/deap

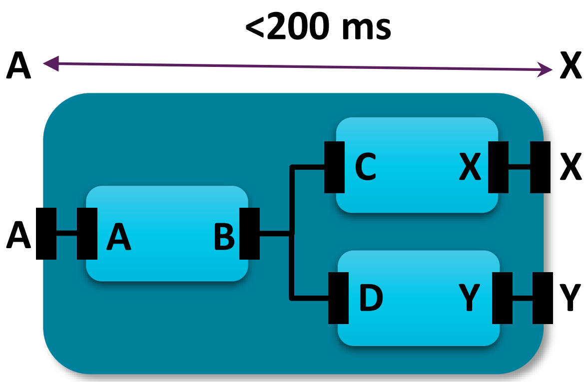
HIERARCHICAL DECOMPOSITION

Introducing **modules** into component model helps **managing the complexity** of complex systems

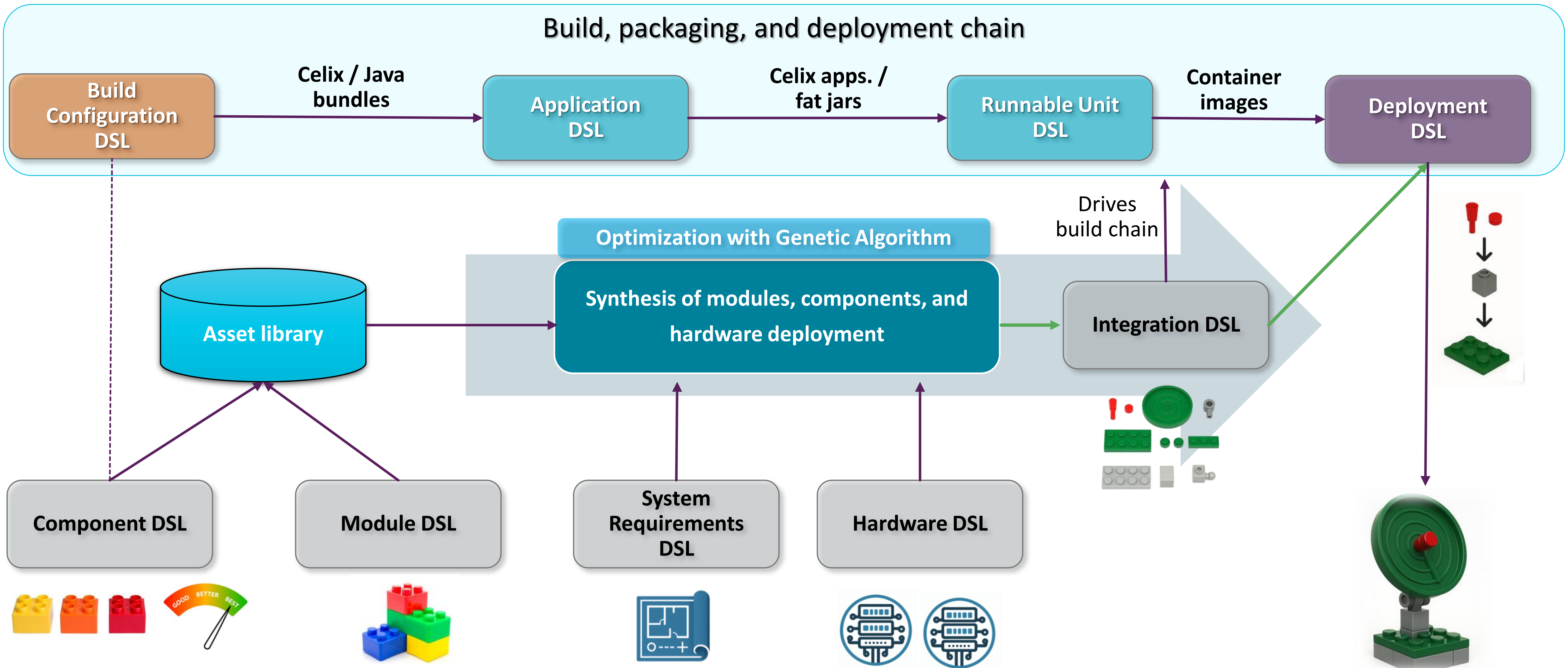
- Allow **structured decomposition** of system in **multiple steps**, helping understanding
- Allows **specification** of provided/required capabilities at **higher level** than components
- Allows (timing) **requirements** to be specified before constituent components are known







COMPLETE TECHFLEX WORKFLOW



SYNTHESIS-BASED ENGINEERING OF SOFTWARE-INTENSIVE SYSTEMS

Engineering approach → / ↓ Development step	Traditional Engineering	Model-Based Engineering	Verification-Based Engineering	Synthesis-Based Engineering
Requirements design	Document-based	**Technology-agnostic specification and deployment**	**Model-based (formal)**	**Synthesis of component-based systems**
System design	Document-based		Model-based (formal)	
Realization in software (implementation code)	Traditional software engineering (coding)		Code generation (fault-free code)	
Verification (against requirements)	Testing		**Formal verification (model checking)**	
Validation (of requirements)	Testing		Testing + Simulation	

Legend: Manual work / Focus (Semi-)automatic

(Simplified overview.)

4

VALIDATION

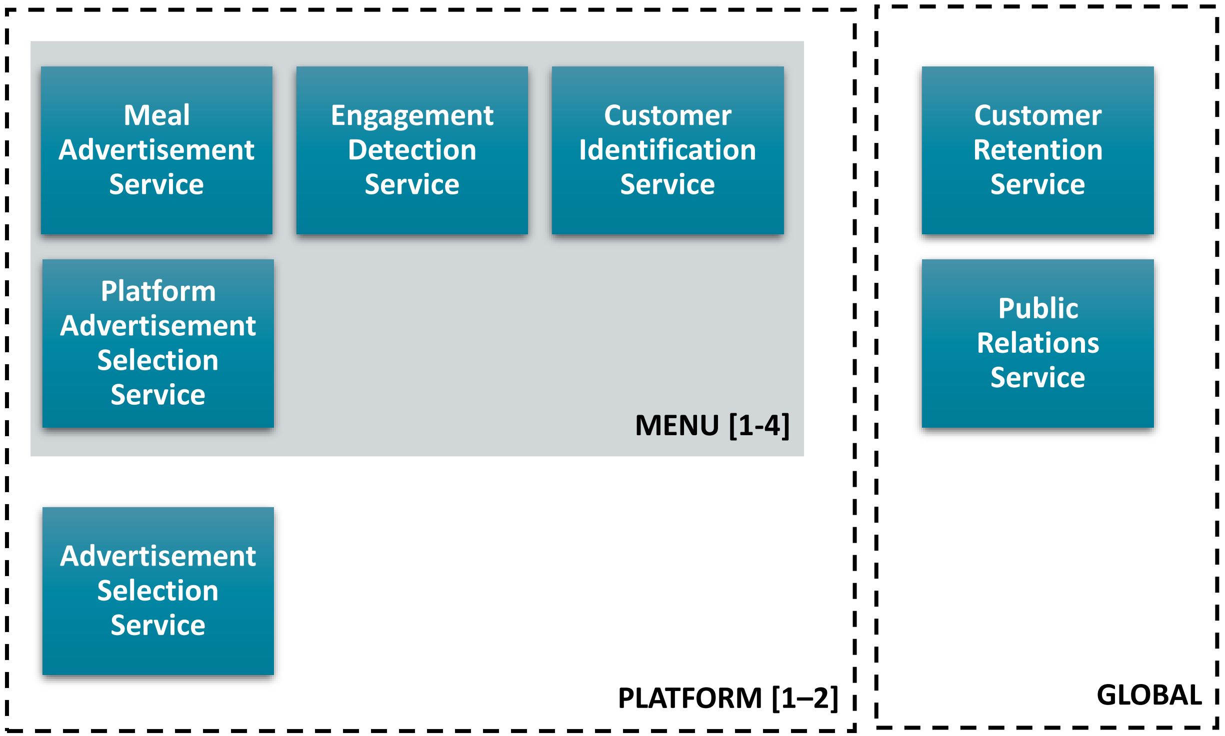
VALIDATION OF METHODOLOGY

We validate our methodology using an (anonymized) **industrial case study** from the **defense domain** comprising

- 70 interface specifications
- 57 component specifications
- 34 module specification

The total instantiated case has a **few hundred elements** in it

These must be deployed on **154 computing nodes**



RESULTS

System resolved in **< 10 seconds**

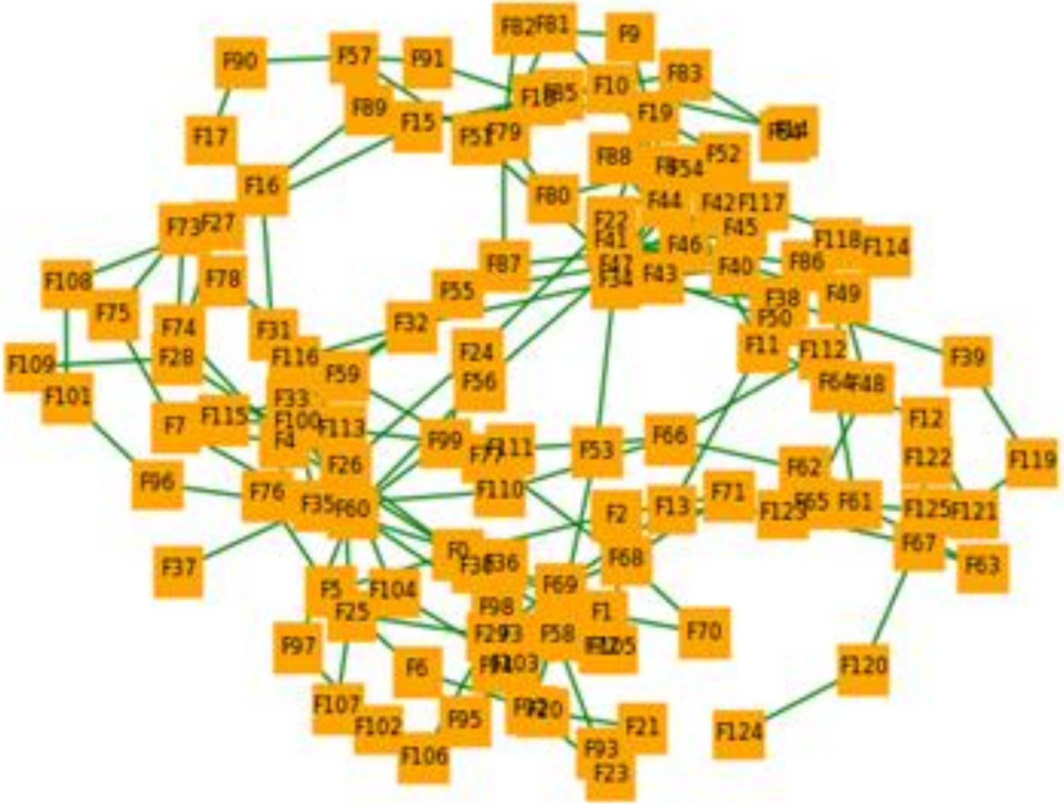
- Manual resolution would take several weeks
- Up front investment in modelling is required

Multiple **technology-agnostic** generators

- K8s generators
- Thales specific generators

Need for various **visualizations**:

- Functional relations
- Structural
- Deployment



Hardware

Function

Hardware dependency

Functional dependency

5

CONCLUSIONS

CONCLUSIONS

We demonstrated how automated generation and synthesis from models **increases engineering productivity** and **democratizes engineering** on a case study from the defense domain

TechFlex addressed this using a **model-based approach to specification and deployment** with three steps:

1. **Technology-agnostic specification** of modules and components as reusable building blocks
2. **Technology-specific generation** of build, packaging, and deployment artifacts
3. **Automatic synthesis of component compositions and deployment** based on (non-)functional requirements

AUTOMATED SYNTHESIS AND DEPLOYMENT OF SOFTWARE INTEGRATIONS

Variability and evolution are key drivers of complexity in high-tech equipment. Every product has many variation points, resulting in a large number of unique system configurations. These systems must be maintained throughout the lifetime of the system, which may last several decades. During this time, digital technologies, e.g., containerization and orchestration technologies, may become deprecated or evolve many times. Manually configuring, updating, and deploying the software of each product variant, such that both functional and performance requirements of customers are satisfied, is both expensive and time-consuming.

This presentation describes a synthesis-based approach that aims to increase engineering productivity by automating the configuration, integration, and deployment of software for product variants, while considering the performance requirements of critical system flows.