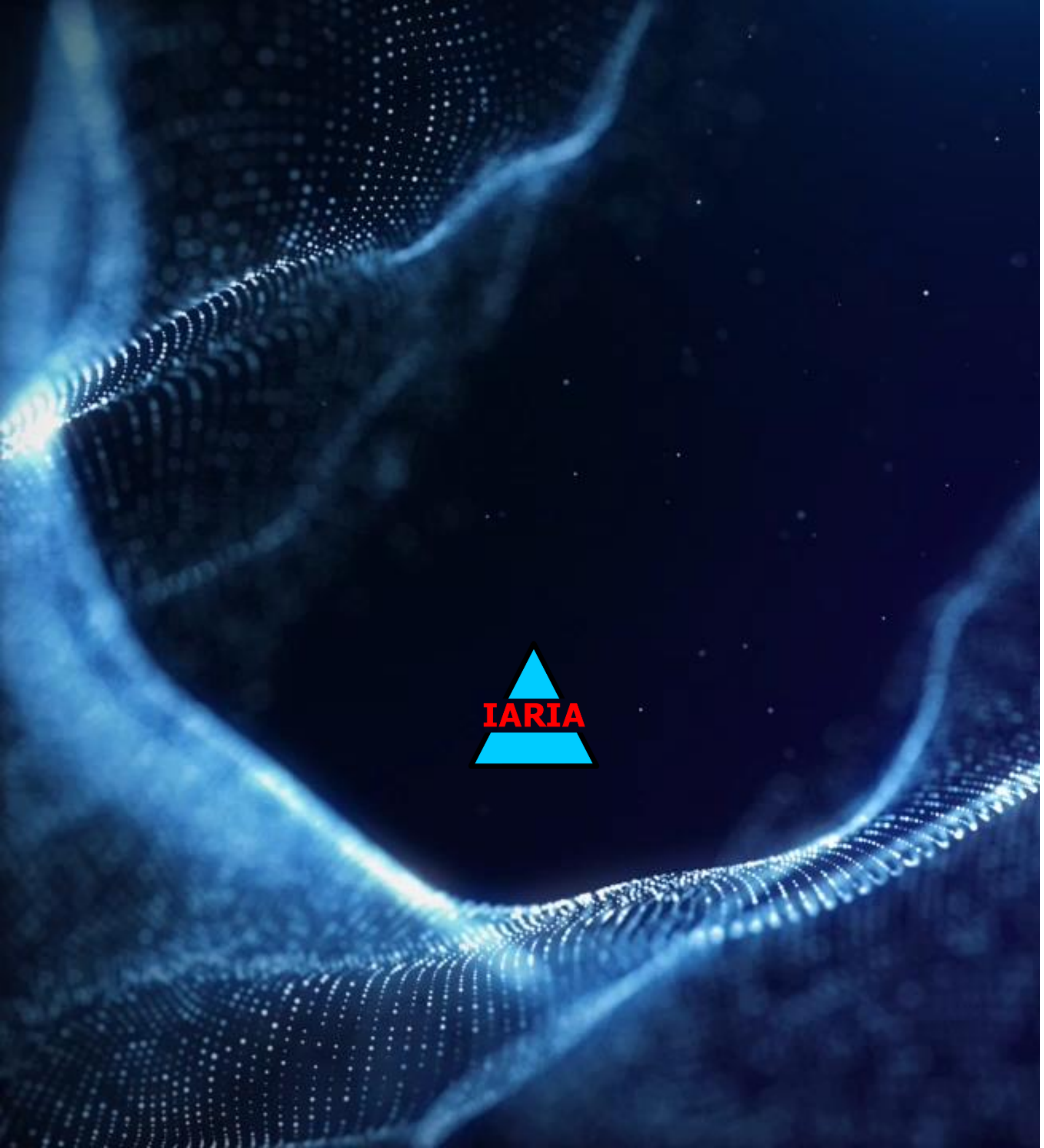


DEVCOM SOLDIER CENTER

U.S.ARMY

SOFTNET 2020

Integrating two Metaprogramming Environments: An Explorative Case Study

HERWIG MANNAERT, CHRIS MCGROARTY. SCOTT GALLANT, KOEN DE COCK

October 22, 2020

Universiteit Antwerpen

Overview

- Metaprogramming and Related Concepts
- Toward Scalable Collaborative Metaprogramming
- Structure of Metaprogramming Environments
- Toward Integrating the Environments
- Conclusion

Overview

- Metaprogramming and Related Concepts
  - Metaprogramming
  - Meta-Circularity
  - Systems Integration
- Toward Scalable Collaborative Metaprogramming
- Structure of Metaprogramming Environments
- Toward Integrating the Environments
- Conclusion

Metaprogramming

- Automatic programming:
  - The act of automatically generating source code from a model or template
- Generative programming
  - To manufacture software components in an automated way
- Metaprogramming
  - Computer programs have the ability to treat other programs as their data
- It is as old as programming itself:

```
System.out.println(“Hello world.”);
```

⇩

```
System.out.println(“System.out.println(\“Hello world.\”);”);
```

DEVCOM SOLDIER CENTER

The Field of Metaprogramming

- Better known through names/trends like:
  - Model-Driven Architecture (MDA) / Model-Driven Engineering (MDE)
  - Model-Driven Software Development (MDSD)
  - Low-Code Development Programs (LCDP)
- The field is still evolving and facing challenges and criticisms:
  - Suitability for large-scale and mission-critical enterprise systems
  - Lack of intermediate representation, pervasive concepts for DSL reuse
- Both the need and potential benefits are real
  - Increase programming productivity
  - Consolidate programming knowledge
  - Valuable for systems engineering, modeling, simulation

DEVCOM SOLDIER CENTER

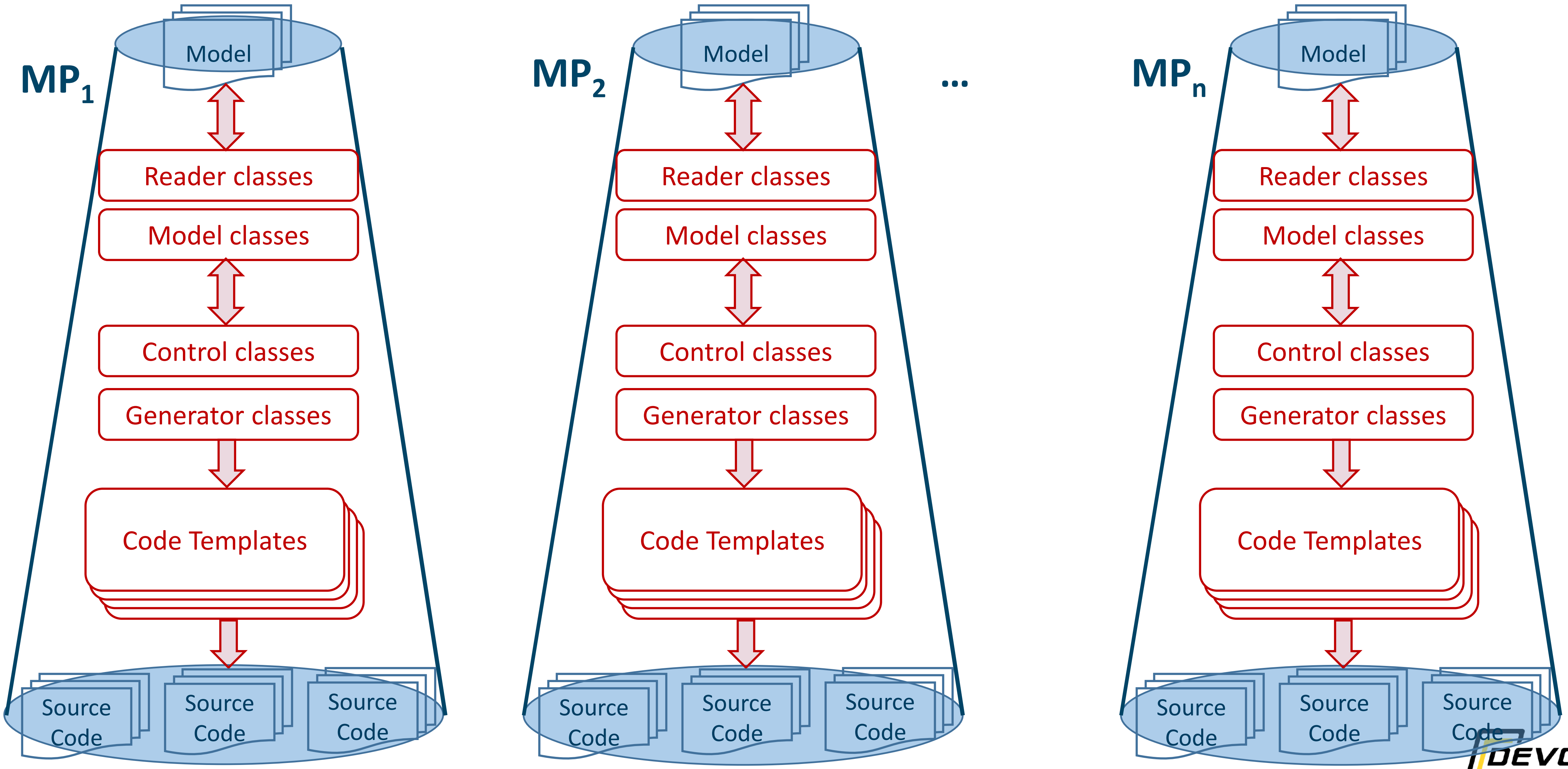
Concepts Relevant for Metaprogramming

- *Meta-Circularity* can enable unified view on code and meta-code :
  - *Homoiconicity* is specifically associated with a language that can be manipulated as data using that language
  - *Meta-circularity* expresses the fact that there is a connection or feedback loop between the meta-level, the internal model of the language, and the actual models or code expressed in the language
- *Systems Integration* can foster collaboration and provide value :
  - Systems integration in information technology refers to the process of linking together different computing systems and software applications, to act as a coordinated whole
  - Due to the many, often disparate, metaprogramming environments and tools, there is a need for systems integration in metaprogramming

- Metaprogramming and Related Concepts
- Toward Scalable Collaborative Metaprogramming
- Structure of Metaprogramming Environments
- Toward Integrating the Environments
- Conclusion

Overview

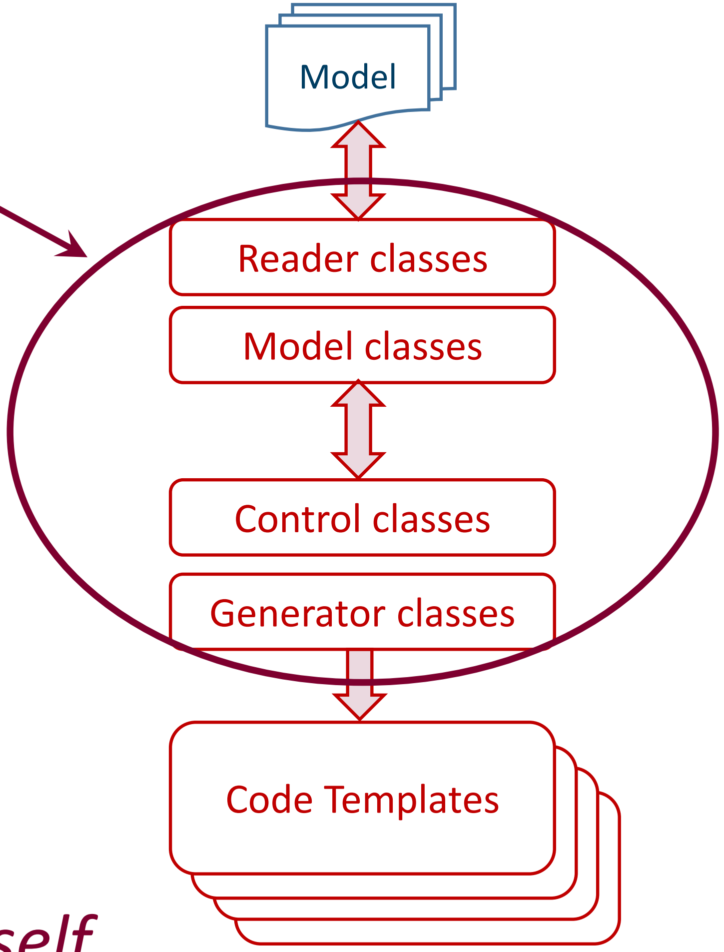
Vertical Integration or Metaprogramming Silos



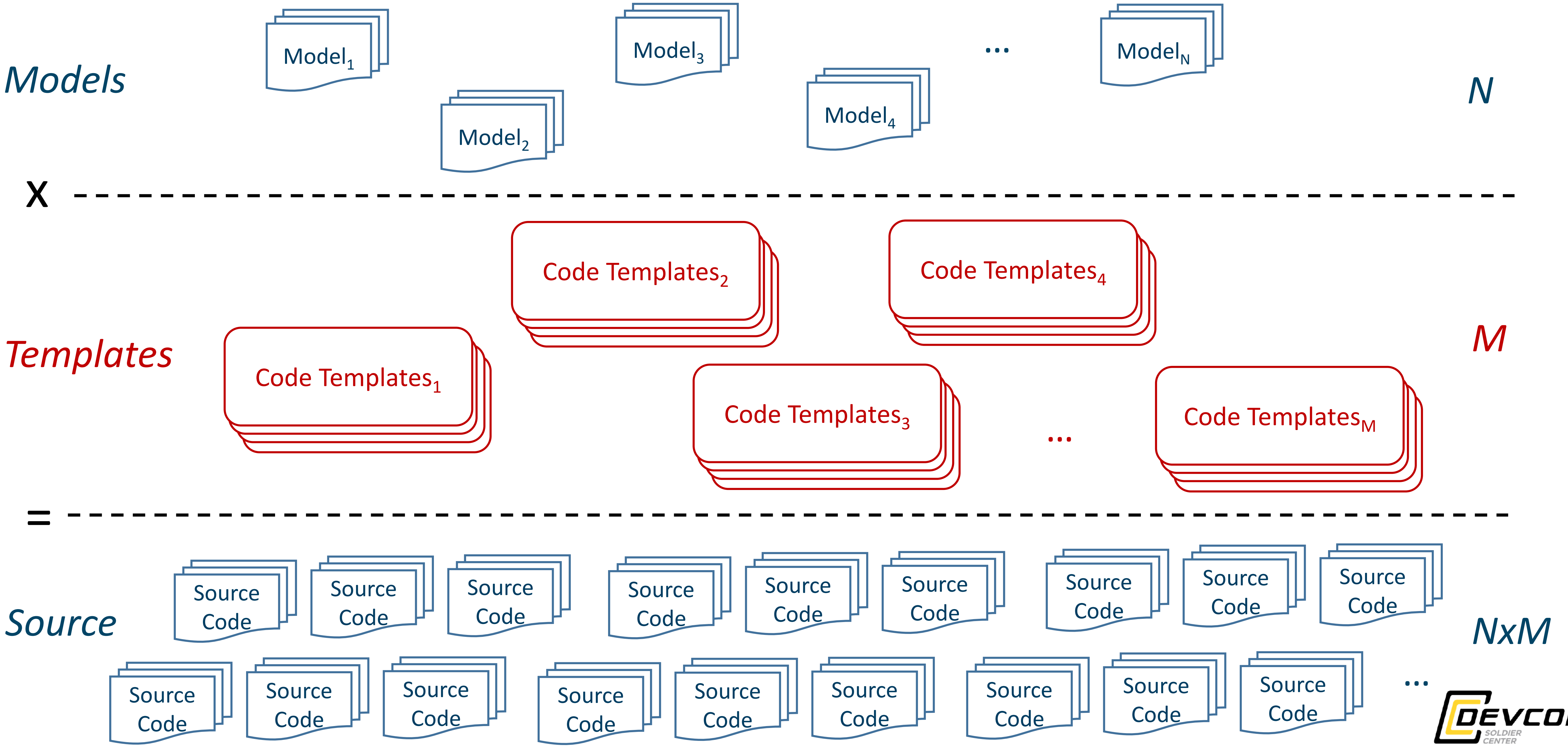
DEVCOM SOLDIER CENTER

On Meta-Circularity in Meta-Programming

- You also have to maintain the meta-code
  - Consists of several modules
  - Is in general not trivial to write
- Will face growing number of implementations:
  - Different versions
  - Multiple variants
  - Various technology stacks
- Will have to adapt itself to:
  - Evolutions of its underlying technology
    - Which even may become obsolete
- *<u>Meta-Circularity</u>: meta-code that (re)generates itself*



On Horizontal Integration in Metaprogramming



Overview

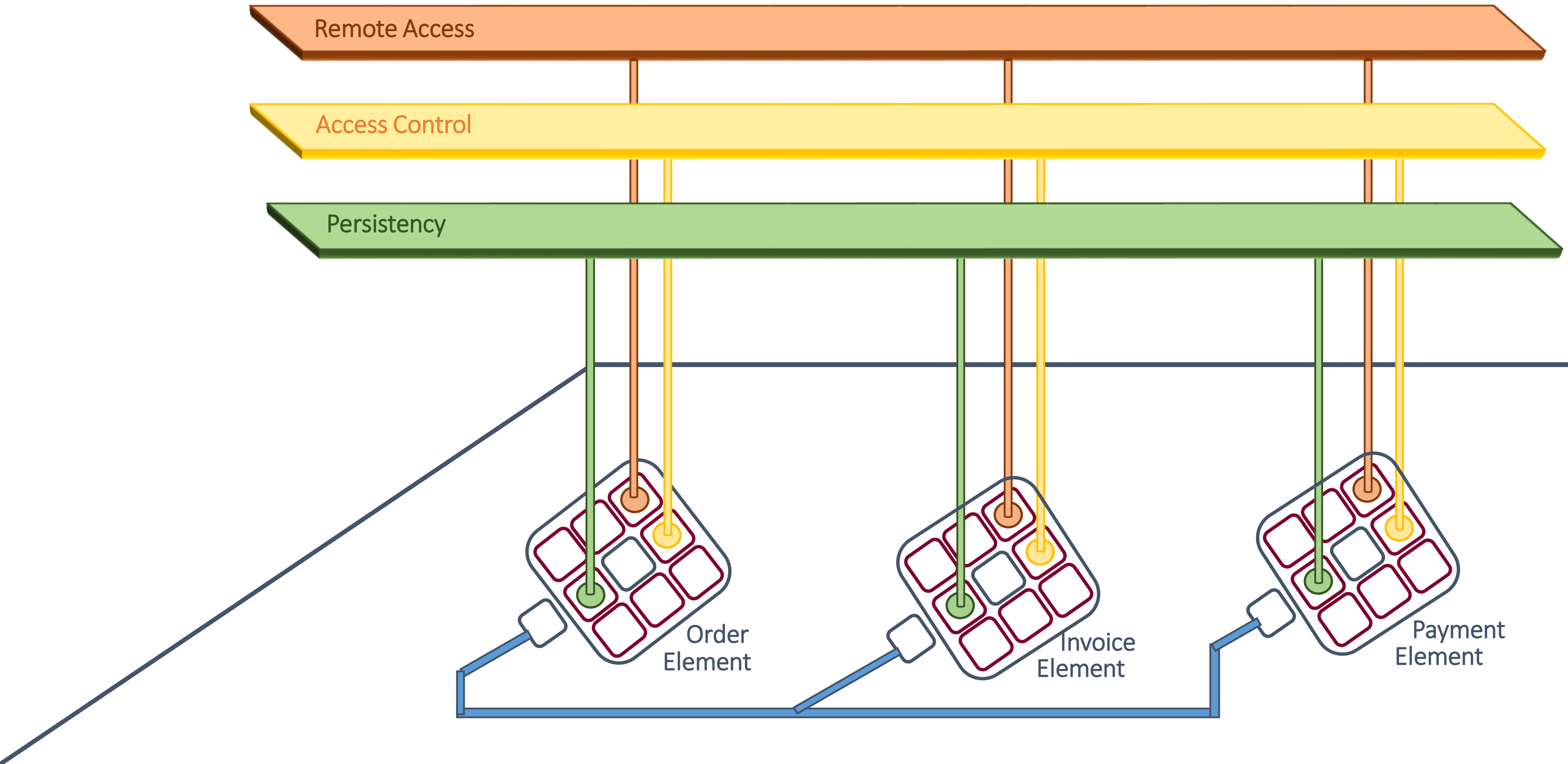
- Metaprogramming and Related Concepts
- Toward Scalable Collaborative Metaprogramming
- Structure of Metaprogramming Environments
  - Normalized Systems Theory (NST) Metaprogramming Environment
  - Generative Environment Simulation Models
- Toward Integrating the Environments
- Conclusion

Metaprogramming Normalized Systems – Essentials

- *Evolvability* of information systems is crucial for organizations
- Normalized Systems Theory:
  - Seeks to provide ex-ante proven approach to build *evolvable software*
  - Founded on systems theoretic stability (Bounded-Input, Bounded-Output, or BIBO), for the impact of changes
- NST proves a set of principles, that are necessary conditions to avoid *instabilities or combinatorial effects*:
  - Separation of Concerns
  - Action Version Transparency
  - Data Version Transparency
  - Separation of States
- This implies *fine-grained modular structure*

DEVCOM SOLDIER CENTER

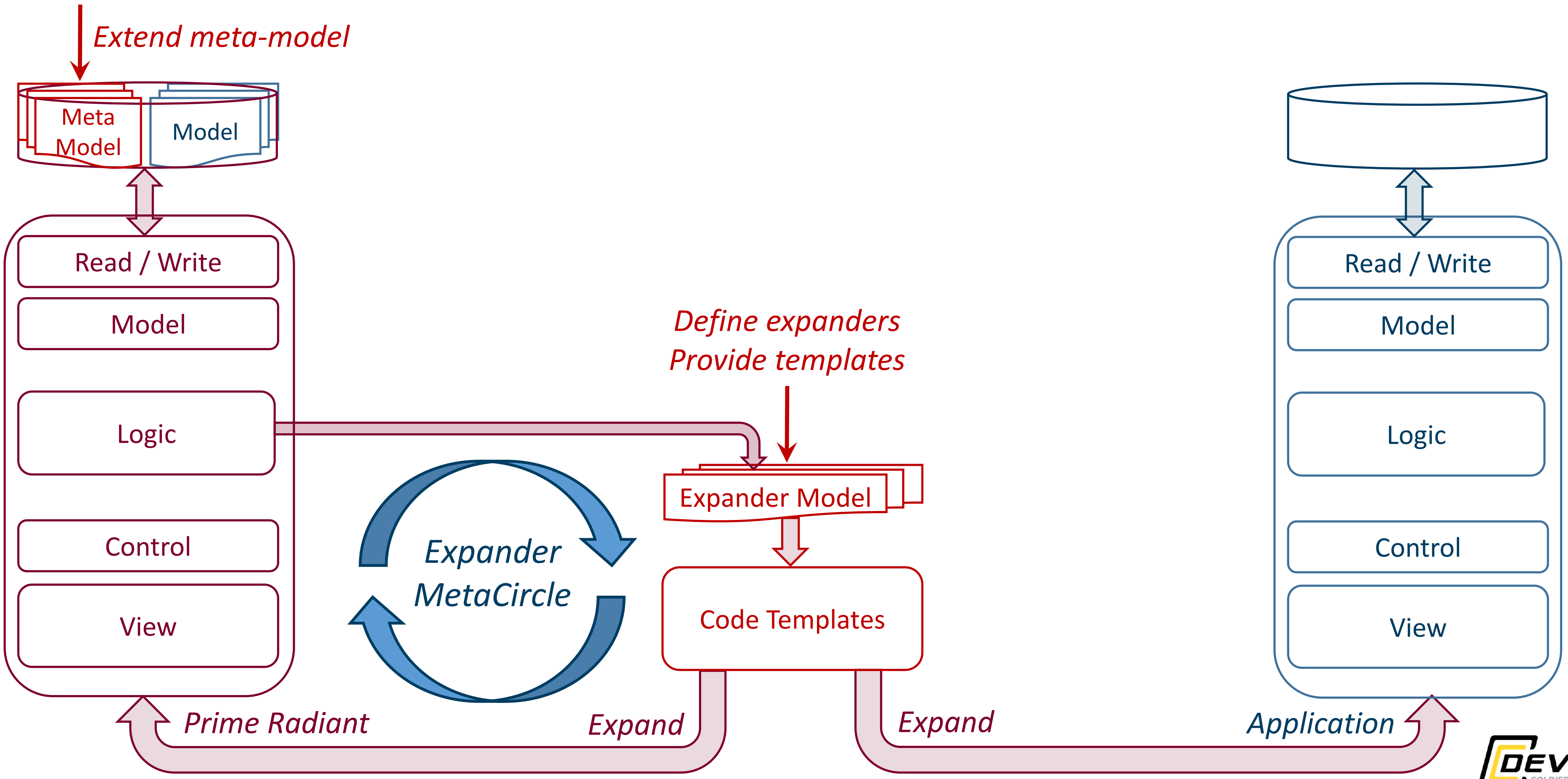
Metaprogramming Normalized Systems – Elements



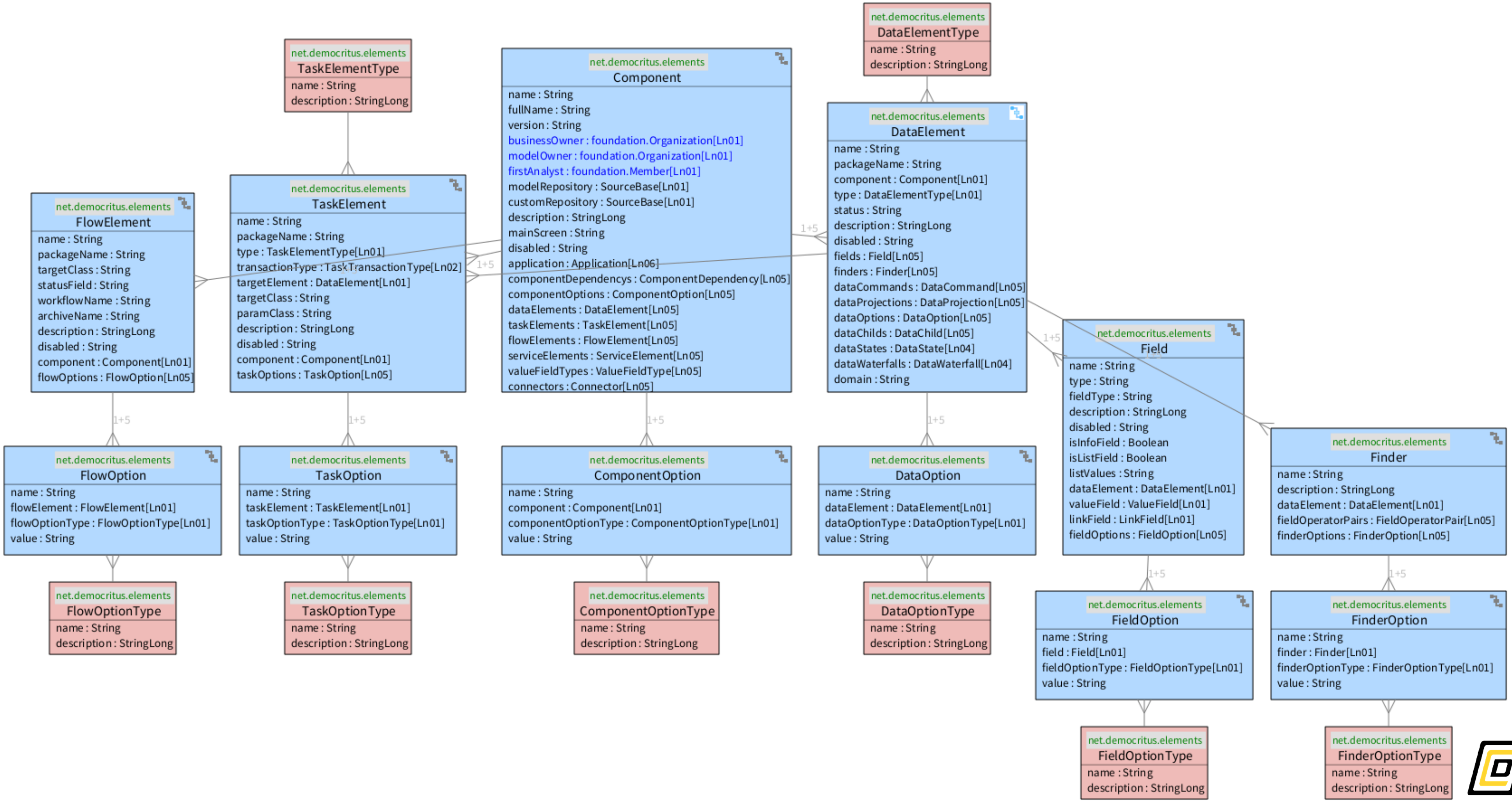
Metaprogramming Normalized Systems – Expansion

- Element structures are needed to interconnect with CCC solutions
- Normalized Systems (NS) defines 5 types of elements, aligned with basic software concepts:
  - Data element
  - Task element
  - Flow element
  - Connector element
  - Trigger element
- Code generation is used to create instances of these elements
- Due to its simple and deterministic nature, we refer to this process as *expansion*, and to the generators as *expanders*

Metaprogramming Normalized Systems – Architecture



Metaprogramming Normalized Systems – Meta-Model



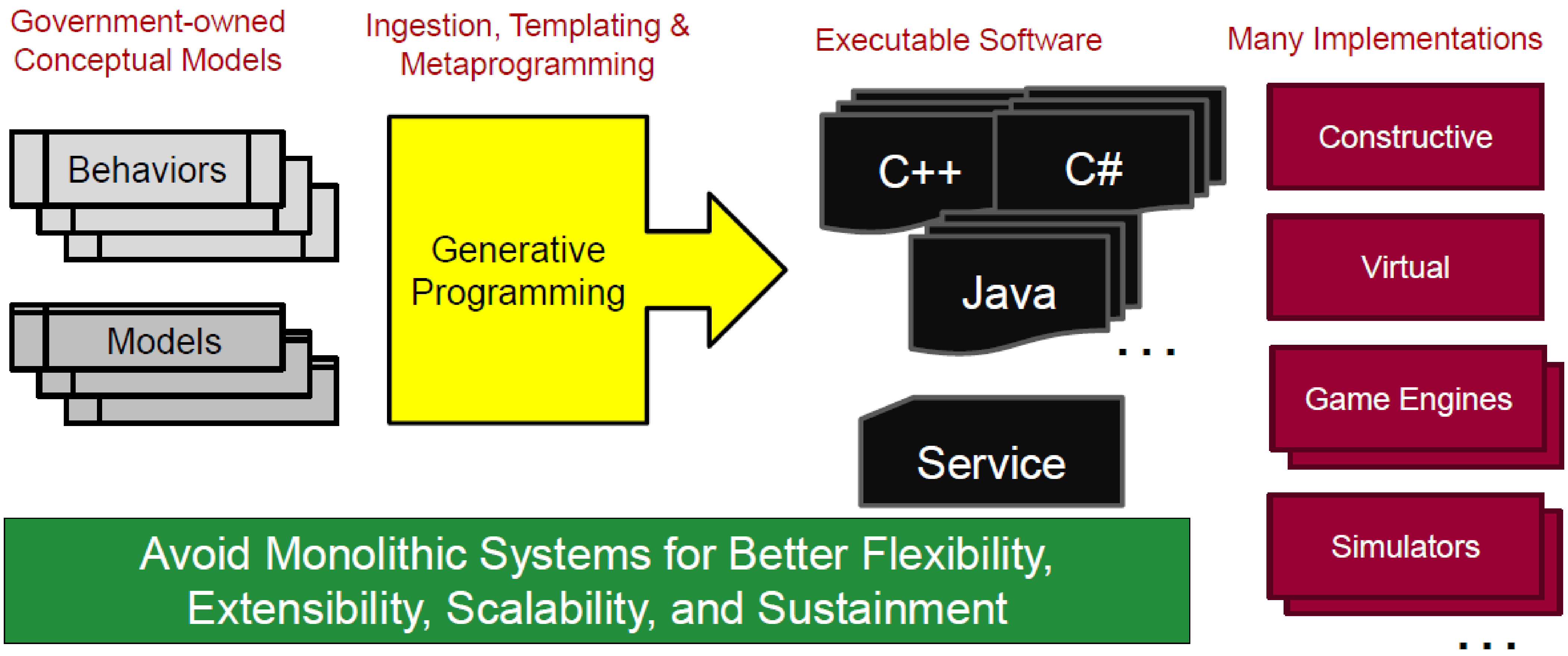
DEVCOM SOLDIER CENTER

Metaprogramming Simulation Models – Essentials

- The United States Army has developed and documented hundreds of approved *models for representing behaviors and systems*.
- Manual translation of these models leads to:
  - implementation errors and verification difficulties
  - workload of incorporating these models into other environments
- A generative programming approach is being examined to
  - capture models within an executable systems engineering format
  - facilitate authoritative models to operate within multiple platforms
  - generate software to implement those representations and behaviors
  - integrate into multiple simulations regardless of programming language

DEVCOM SOLDIER CENTER

Metaprogramming Simulation Models – Architecture



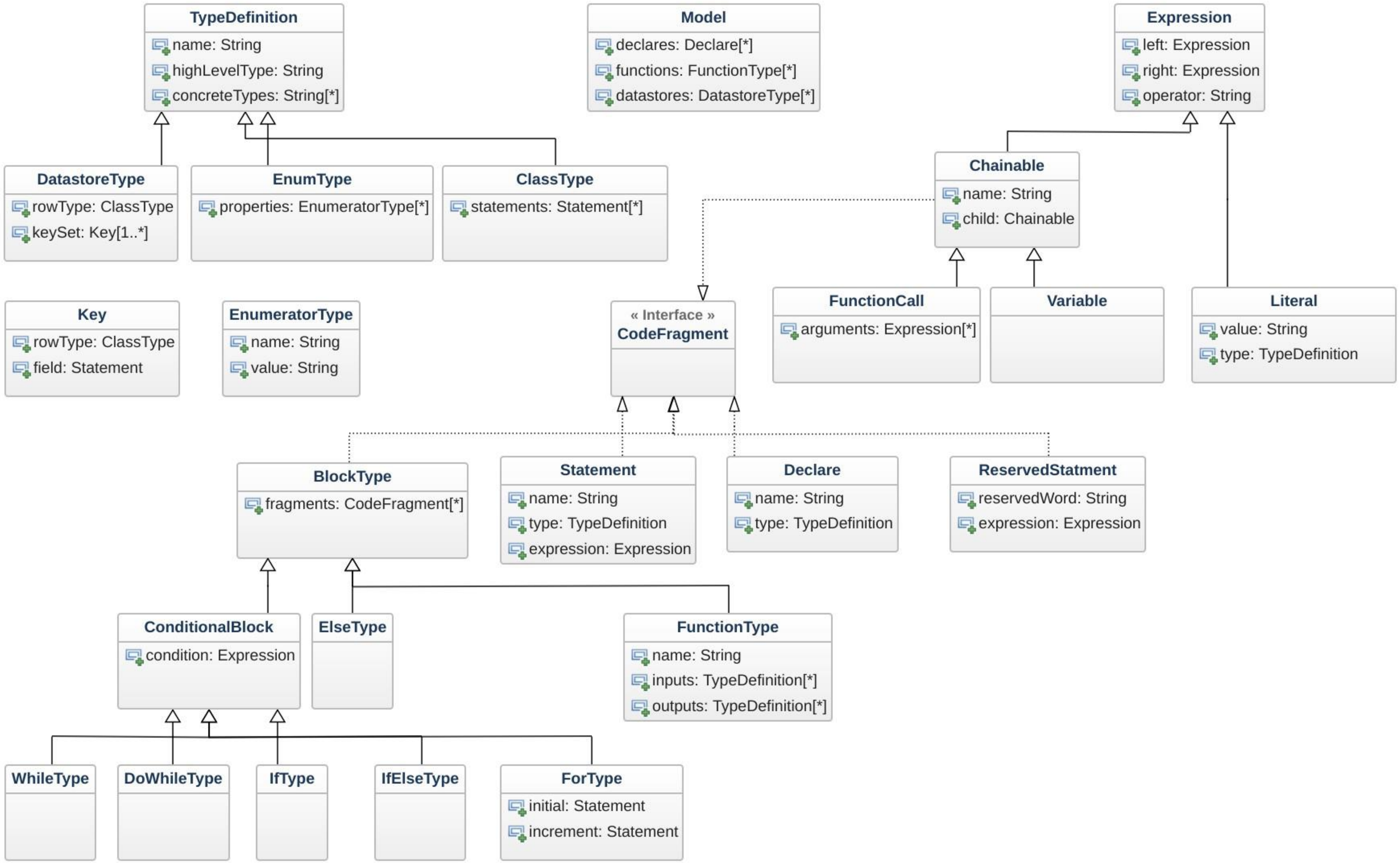
Metaprogramming Simulation Models – Interchange

- To decouple front-end and back-end, an *Interchange Format (IF)* :
  - allows to record models in various front-ends
  - to pass these models from front-end to back-end
  - enables a more horizontal integration architecture

- *Synthetic Training Environment (STE) Canonical Universal Format (SCUF)*
  - is based on XML documents
  - defined by an XML Schema Definition or XSD
  - focuses on the domain elements used within the U.S. Army's canonical descriptions of the simulation models.

DEVCOM SOLDIER CENTER

Metaprogramming Simulation Models – Meta-model



Overview

- Metaprogramming and Related Concepts
- Toward Scalable Collaborative Metaprogramming
- Structure of Metaprogramming Environments
- Toward Integrating the Environments
  - Embracing the SCUF Meta-Model
  - Supporting the Templating Engine
- Conclusion

Integrating two Environments – Meta-Models

- Both metaprogramming environments:
  - exhibit an horizontal integration architecture
  - use XML to exchange between models and templates
- Normalized Systems environment allows to:
  - define any Entity Relationship Diagram (ERD), including entities of SCUF meta-model, e.g., *Statement*
  - generate the meta-circular stack for these entities, including:
    - XML readers and writers, e.g., *StatementXmlReader*, *StatementXmlWriter*
    - classes representing model instances, e.g., *StatementDetails*, *StatementComposite*
    - view and control classes for create and manipulate models in a user interface
  - make the models available to the templates through Object-Graph Navigation Language (OGNL) expressions
    - e.g., *statement.type.name*, *statement.expression.operator*

DEVCOM SOLDIER CENTER

Integrating two Environments – Templates

- Normalized Systems environment
  - allows to activate every coding template by:
    - declaring the template in an *XML expander file*
    - defining the OGNL expressions in an *XML mapping file*
  - provides a connector for *StringTemplate* templating engine
- *Velocity templates* of simulation models:
  - can be converted to StringTemplate, but:
    - effort *proportional to the amount and size* of templates
    - additional templates would continue to create workload
  - require a connector for *Velocity* templating engine
    - seems a manageable effort
      - straightforward as Velocity allows more logic

- Metaprogramming and Related Concepts
- Toward Scalable Collaborative Metaprogramming
- Structure of Metaprogramming Environments
- Toward Integrating the Environments
- Conclusion

Overview

Conclusions

- We have explored the horizontal integration between two different metaprogramming environments
- Contributions:
  - We have shown that the meta-model of another metaprogramming environment can be embraced by our previously proposed meta-circular architecture
  - We have shown that models based on other meta-model can be made available to the coding templates without the additional development of meta-code
- Limitations:
  - Not yet operational due to lack of template engine connector
  - Horizontal integration needs to be deepened and broadened

DEVCOM SOLDIER CENTER


QUESTIONS ?
herwig.mannaert@uantwerp.be