COMPUTATIONWORLD 2021

On the Use of Code Generation Patterns for the Creation of Evolvable Documents and Runtime Artifacts

HERWIG MANNAERT, GILLES OORTS,
KOEN DE COCK, PETER UHNAK

APRIL, 2021

Universiteit Antwerpen

Presenter

- Herwig Mannaert
  - Electronics engineer, PhD Computer vision (1993)
  - Full professor University of Antwerp
    - https://www.uantwerpen.be/en/staff/herwig-mannaert/
  - Research on evolvable software engineering
  - Co-author Normalized Systems Theory (NST)
    - Theory to design evolvable modular structures
  - Co-founder NSX bv
    - Making code generators and software based on NST
  - Co-founder Cast4All NV
    - Making software for energy monitoring, e.g., solar installations
  - Husband and father of 4 children

Overview

- Modular and Evolvable Document Creation
- Expansion of Evolvable Modular Structures
- Exploring Runtime Expansion of Artifacts
- Conclusion

Overview

- Modular and Evolvable Document Creation
  - Document Creation and Single Sourcing
  - Modular and Parametrized Document Generation
- Expansion of Evolvable Modular Structures
- Exploring Runtime Expansion of Artifacts
- Conclusion

Document Creation and Single Sourcing

- An example of modularity in document management and creation is Component Content Management (CCM):
  - Separation of content and layout
  - Granularity based on smallest unit of information
- Related is the fundamental concept of Single Sourcing:
  - Make content reusable by separating it from format
  - Write content in stand-alone modules
  - Modules are linked to generate documents
- Document creation distinguishes:
  - Repurposing
  - Reassembly

Modular and Parametrized Document Creation

- CCM highly reminiscent of similar concepts in software codebases
  - Separate concerns in singular source files
  - Assemble source modules in quest to reuse
  - Evolvability hampered by rippling of changes
  - Successive versions in time and variations in content
- Concurrent variants highly reminiscent of software code generation
  - Parameter-based or model-based instantiation
    - Procedures and descriptions of various technical installation manuals
    - Many variants of simple administrative documents and letters

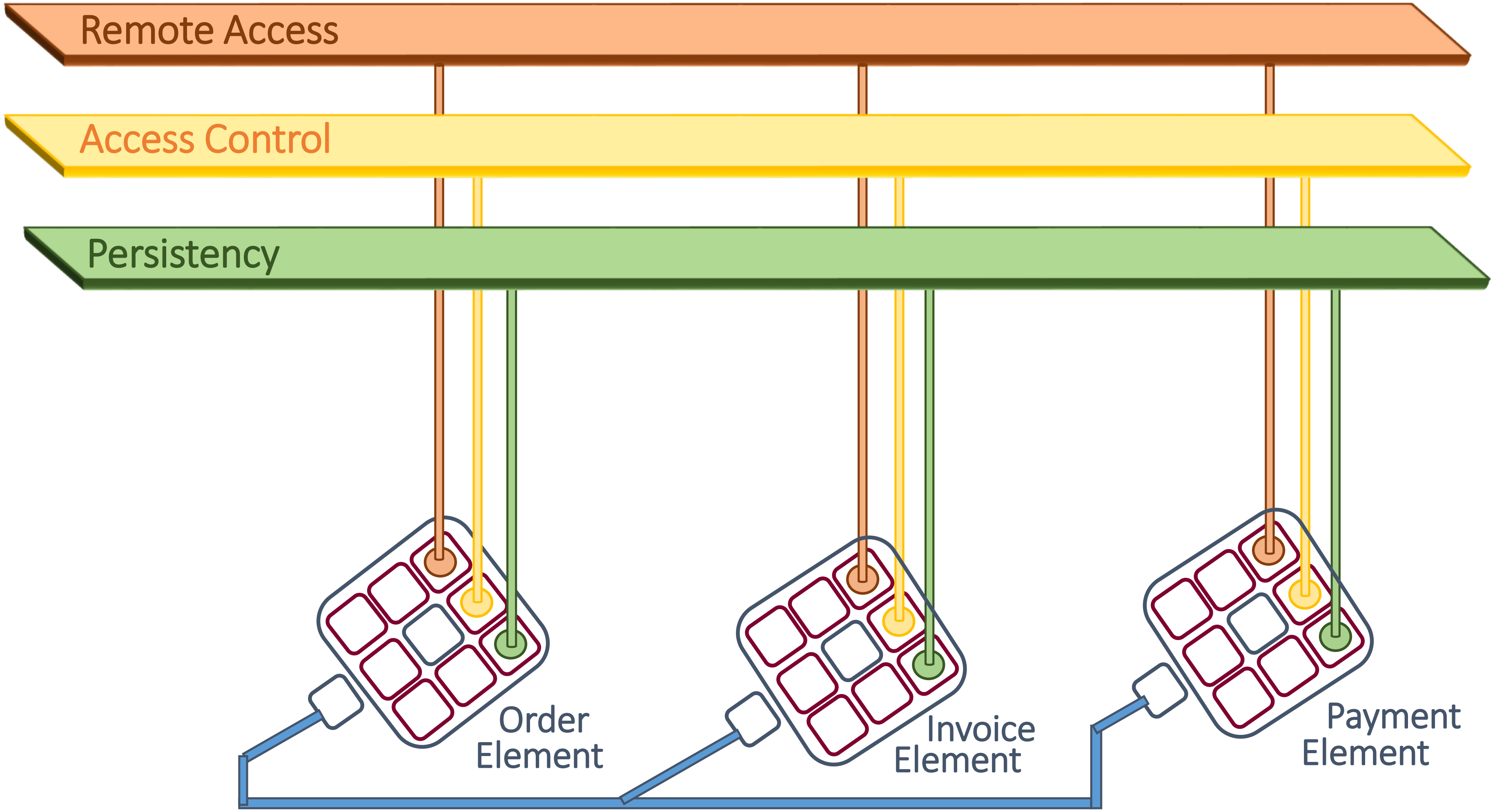
Overview

- Modular and Evolvable Document Creation
- Expansion of Evolvable Modular Structures
  - Normalized Systems Theory and Evolvable Structures
  - Meta-Circular Code Generation or Artifact Expansion
- Exploring Runtime Expansion of Artifacts
- Conclusion

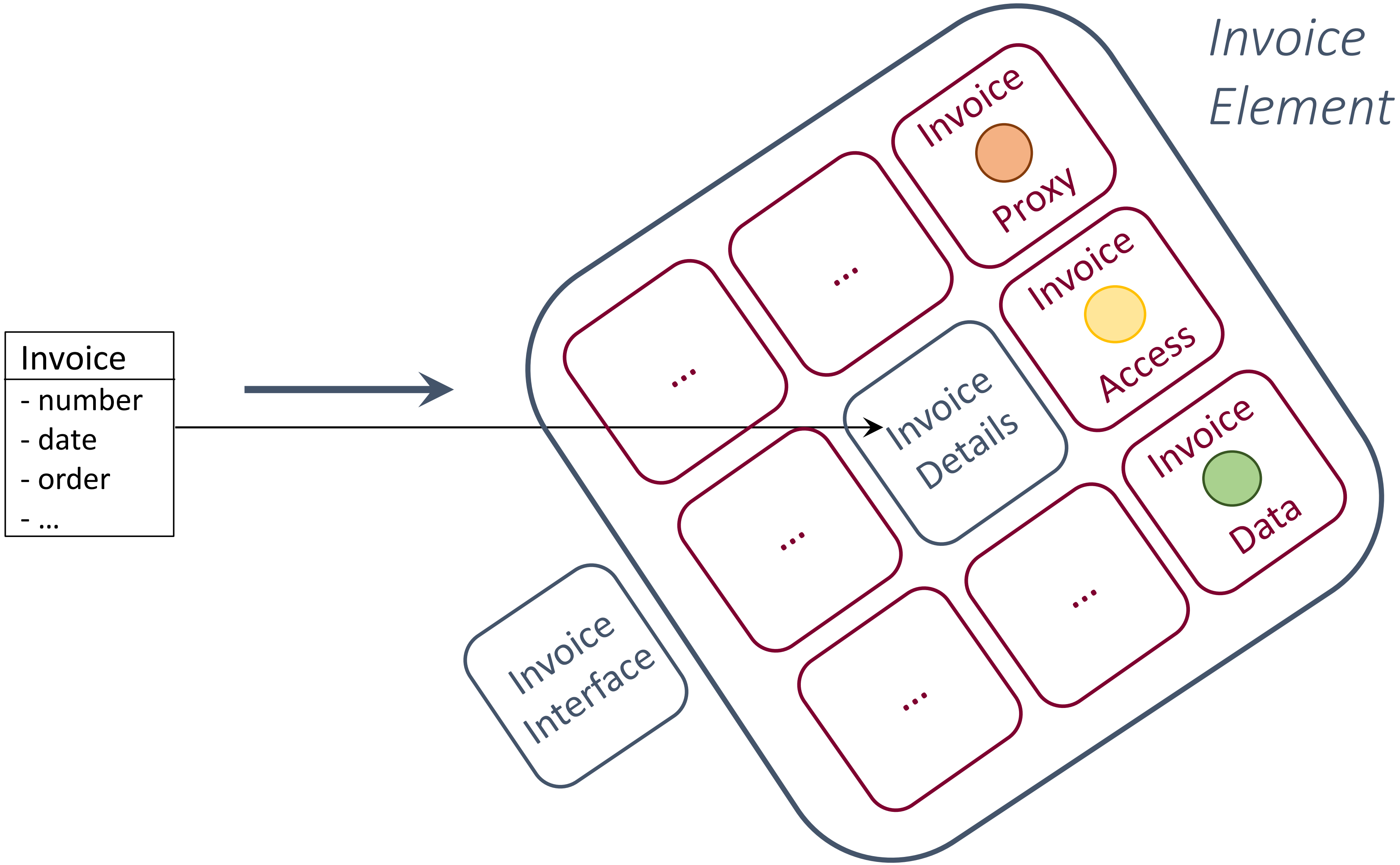
Normalized Systems Theory and Evolvable Structures

- In order to avoid dynamic instabilities in the software design cycle, the *rippling of changes needs to be depleted or damped: a = 0*
- As these ripples create *combinations of multiple changes* for every functional change, we call these instabilities ***combinatorial effects***
- Demanding systems theoretic stability for the software transformation, leads to the derivation of ***four principles*** in line with existing heuristics
- Adhering to these principles avoids dynamic instabilities, meaning that these *principles are necessary, not sufficient for systems stability*
    - *Separation of Concerns*
    - *Action Version Transparency*
    - *Data Version Transparency*
    - *Separation of States*

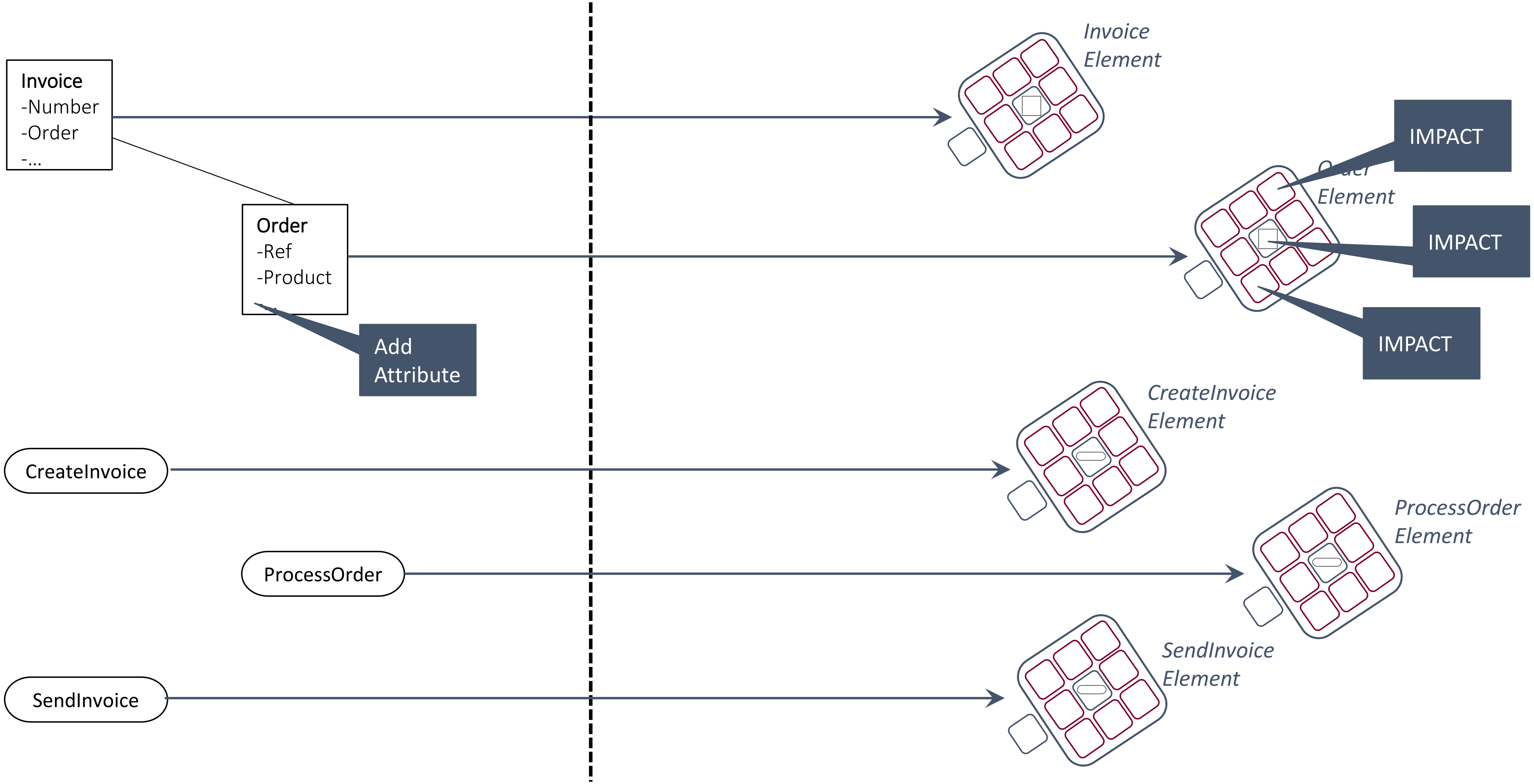
Applying Design Theorems, Software Elements Emerge



Applying Design Theorems, Software Elements Emerge



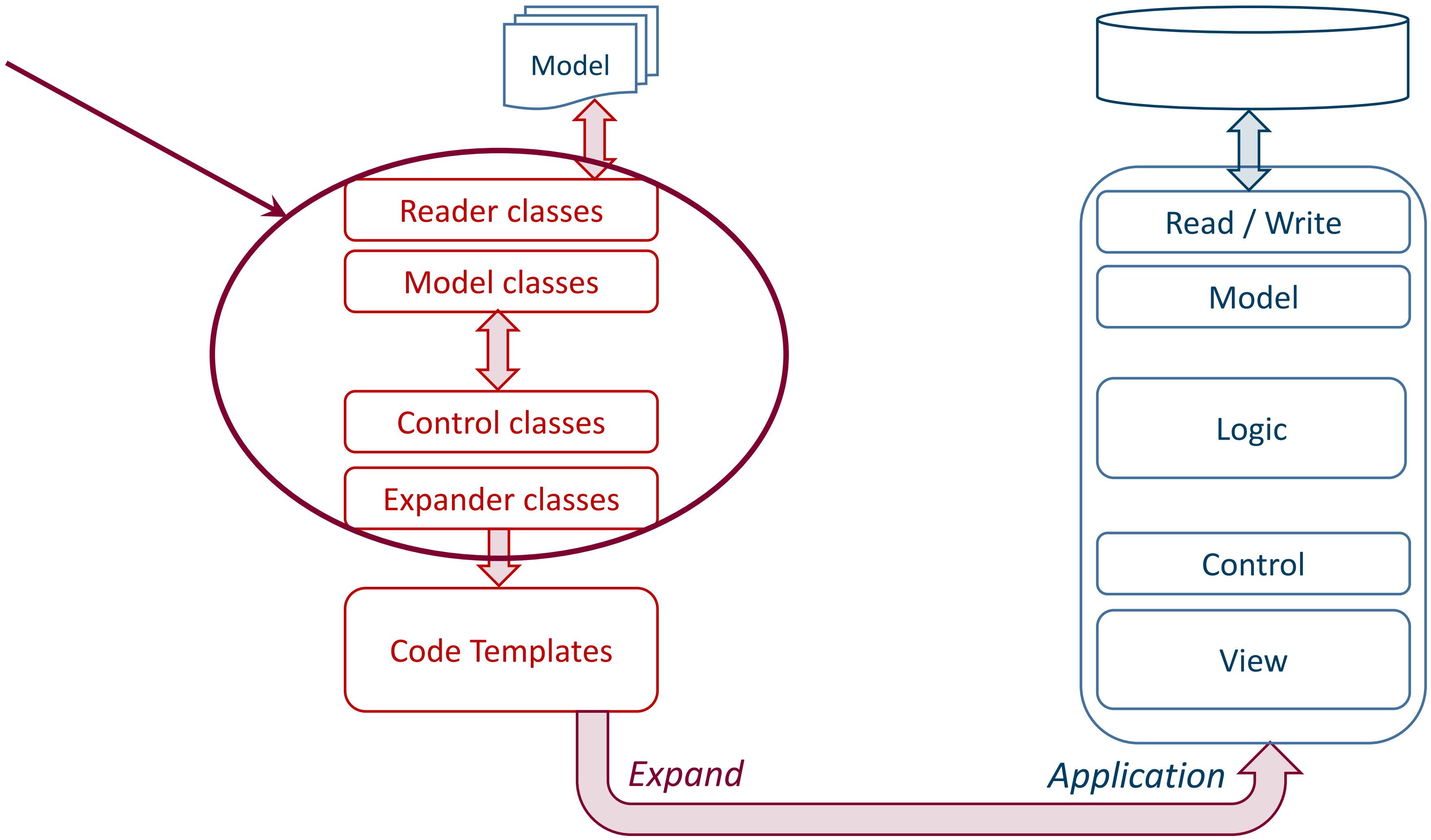
Software Element Instances Create Stable Skeletons



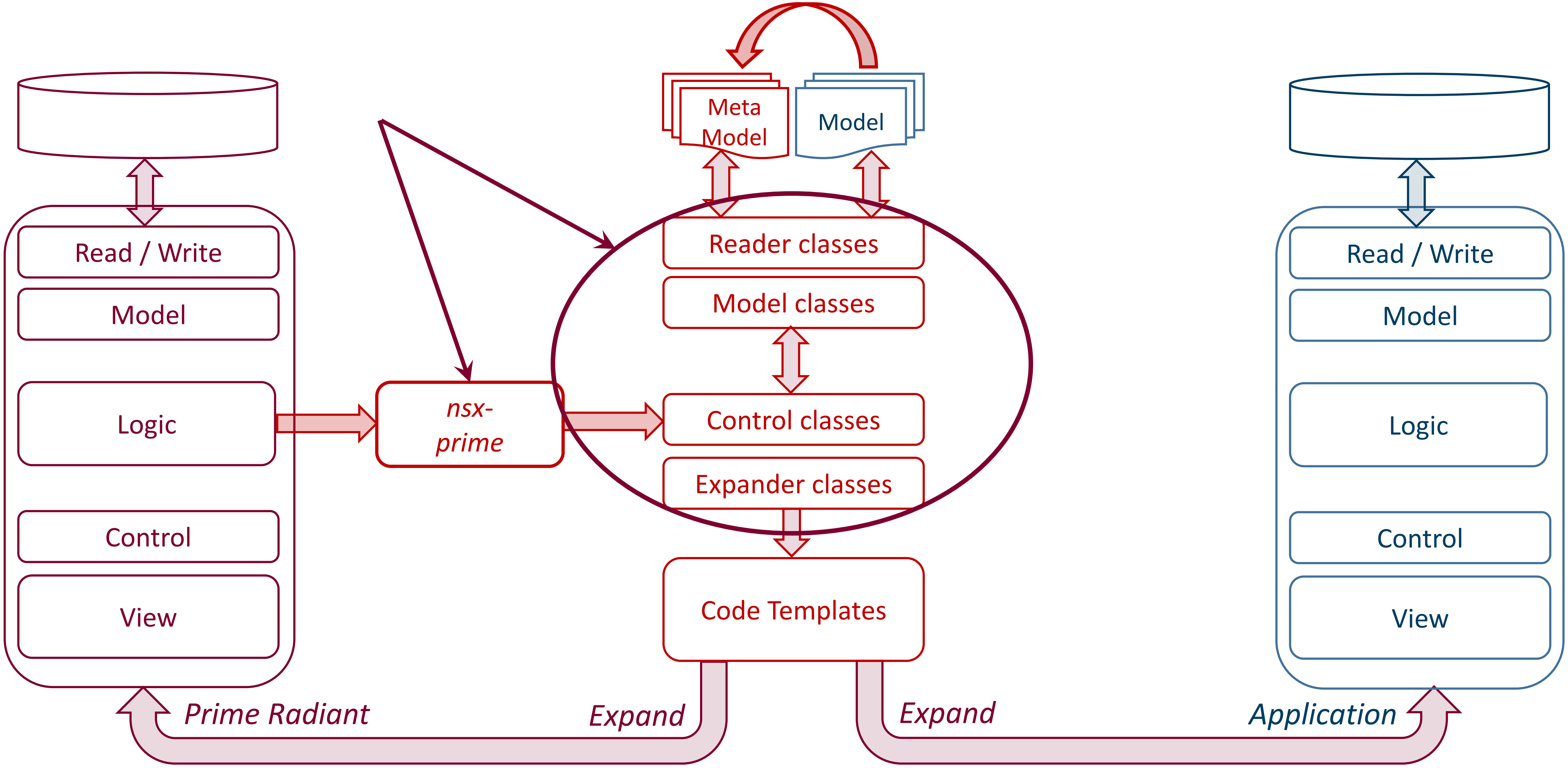
Software Element Instances Create Stable Skeletons

- Element structures are needed *to interconnect with CCC solutions*
- NS defines 5 types of elements, aligned with basic software concepts:
  - ***Data elements***, to represent data variables and structures
  - ***Task elements***, to represent instructions and/or functions
  - ***Flow elements***, to handle control flow and orchestrations
  - ***Connector elements***, to allow for input/output commands
  - ***Trigger elements***, to offer periodic clock-like control
- It seems obvious to use code generation techniques to create instances of these recurrent element structures
- Due to its simple and deterministic nature, we refer to this process as *expansion*, and to the generators as *expanders*

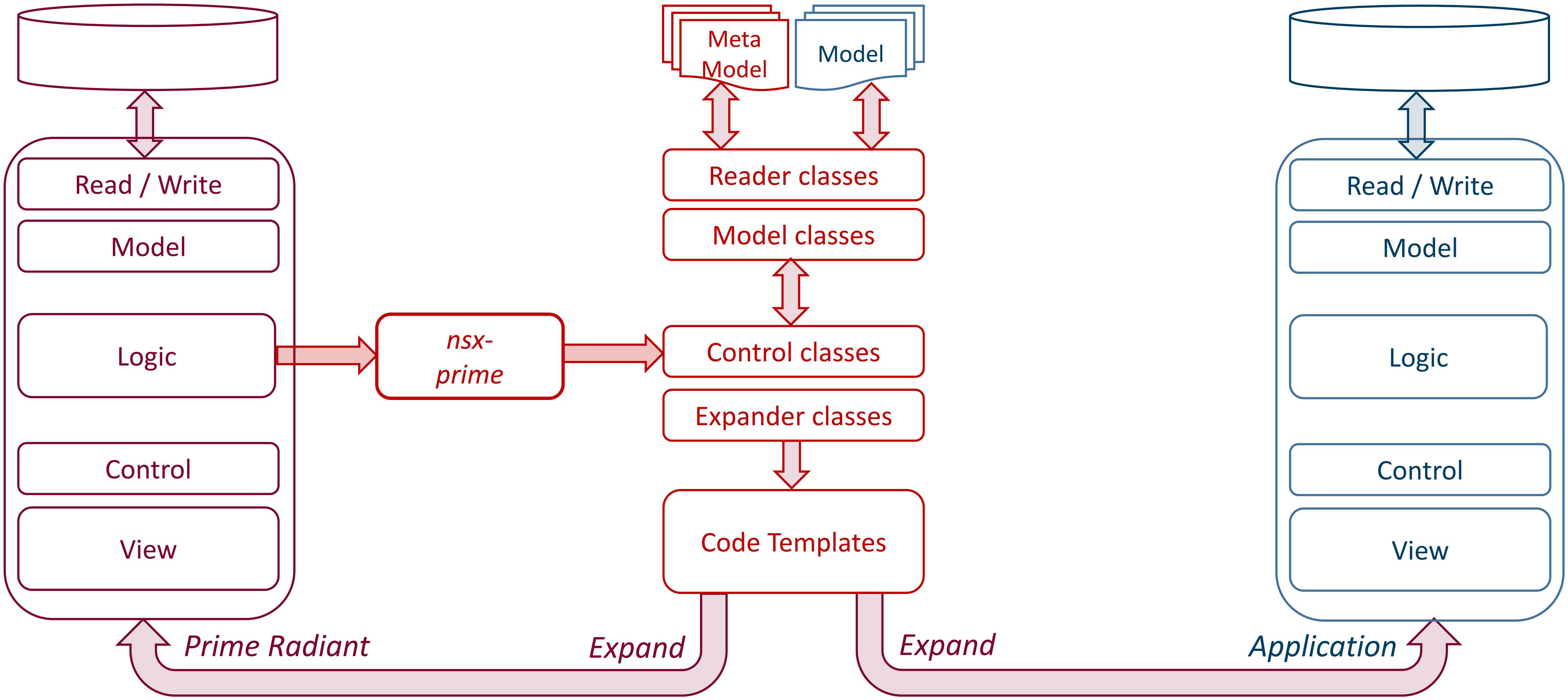
Meta-Circular Code Generation or Artifact Expansion



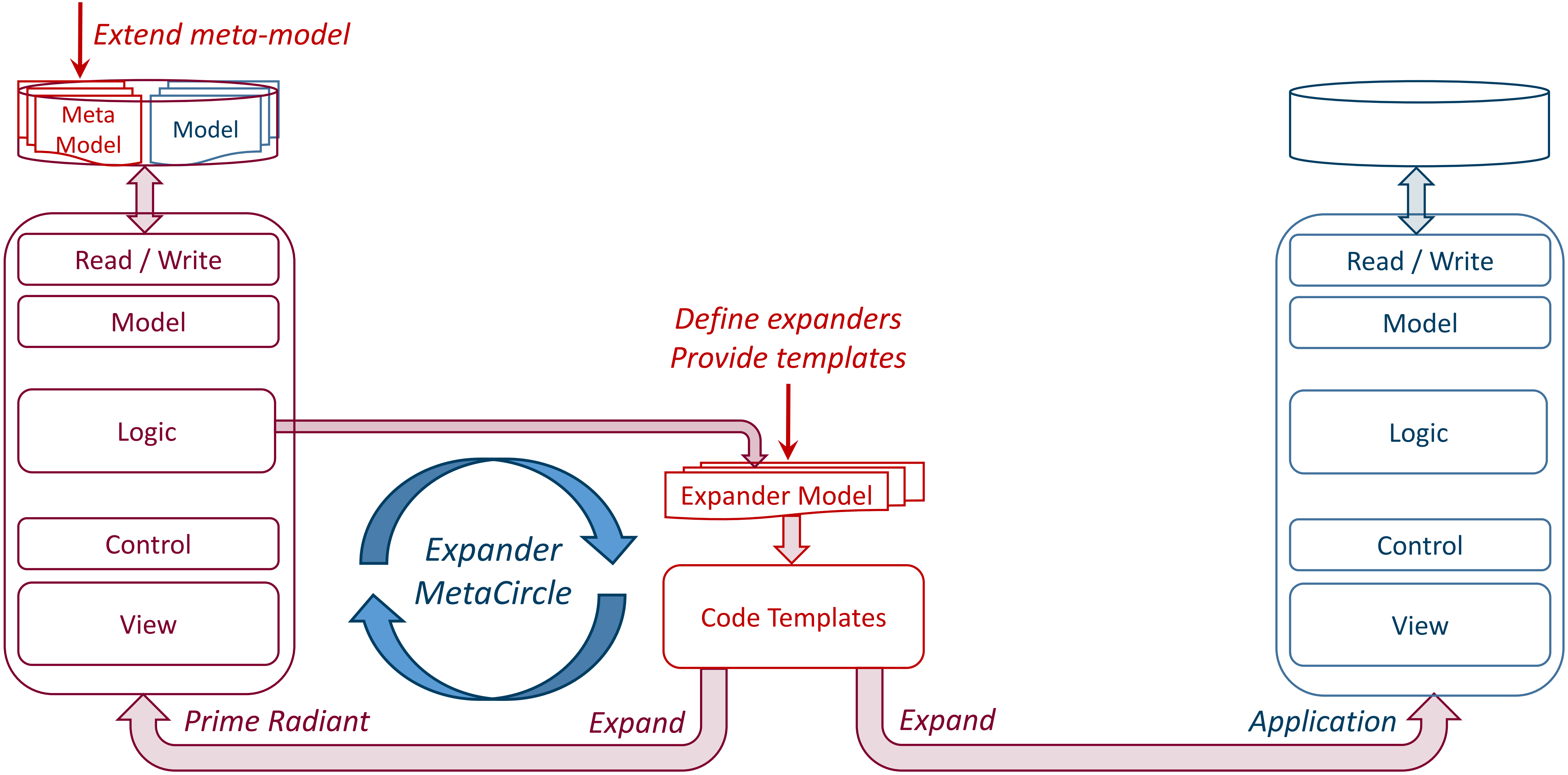
Meta-Circular Code Generation or Artifact Expansion



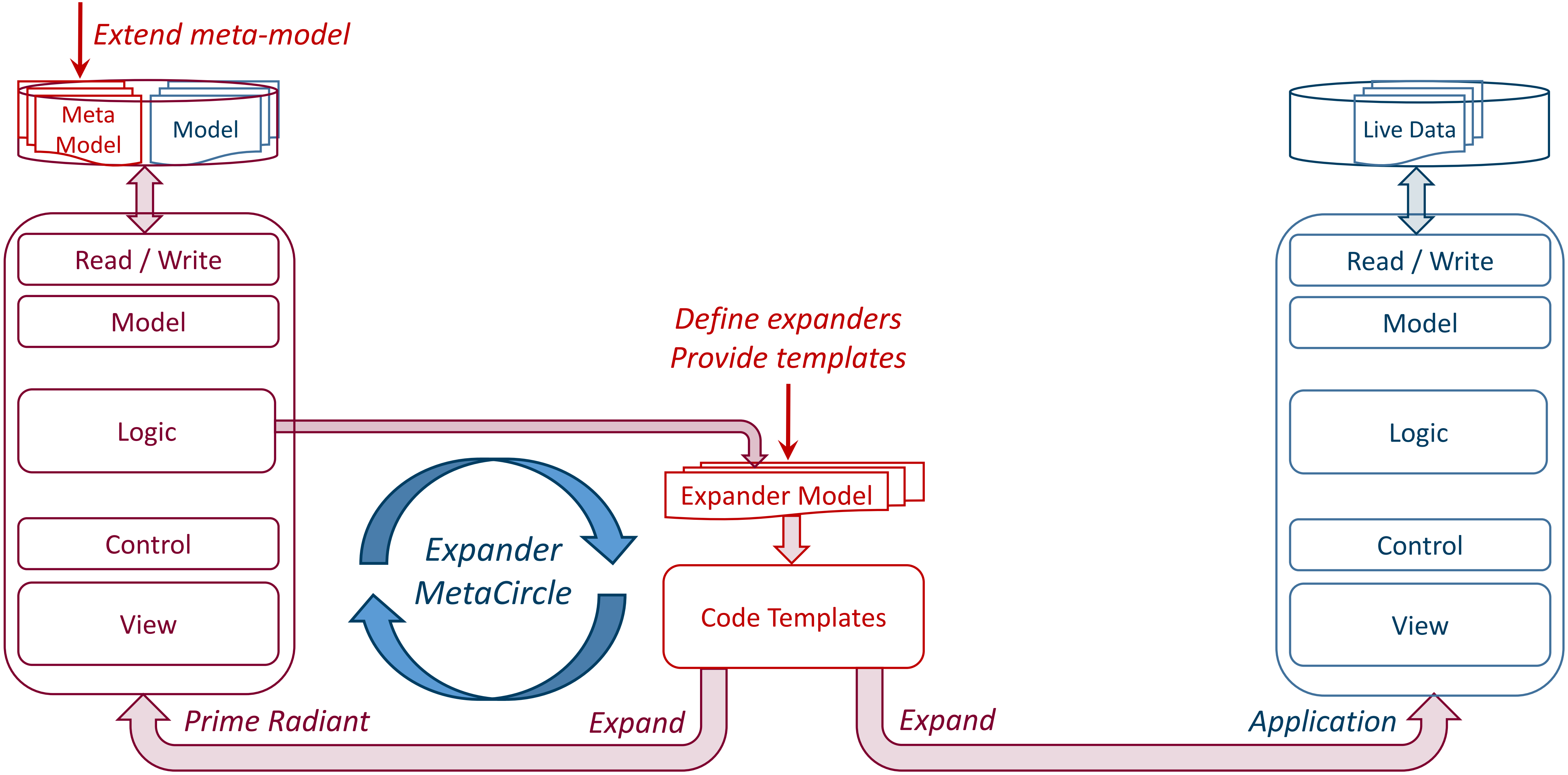
Meta-Circular Code Generation or Artifact Expansion



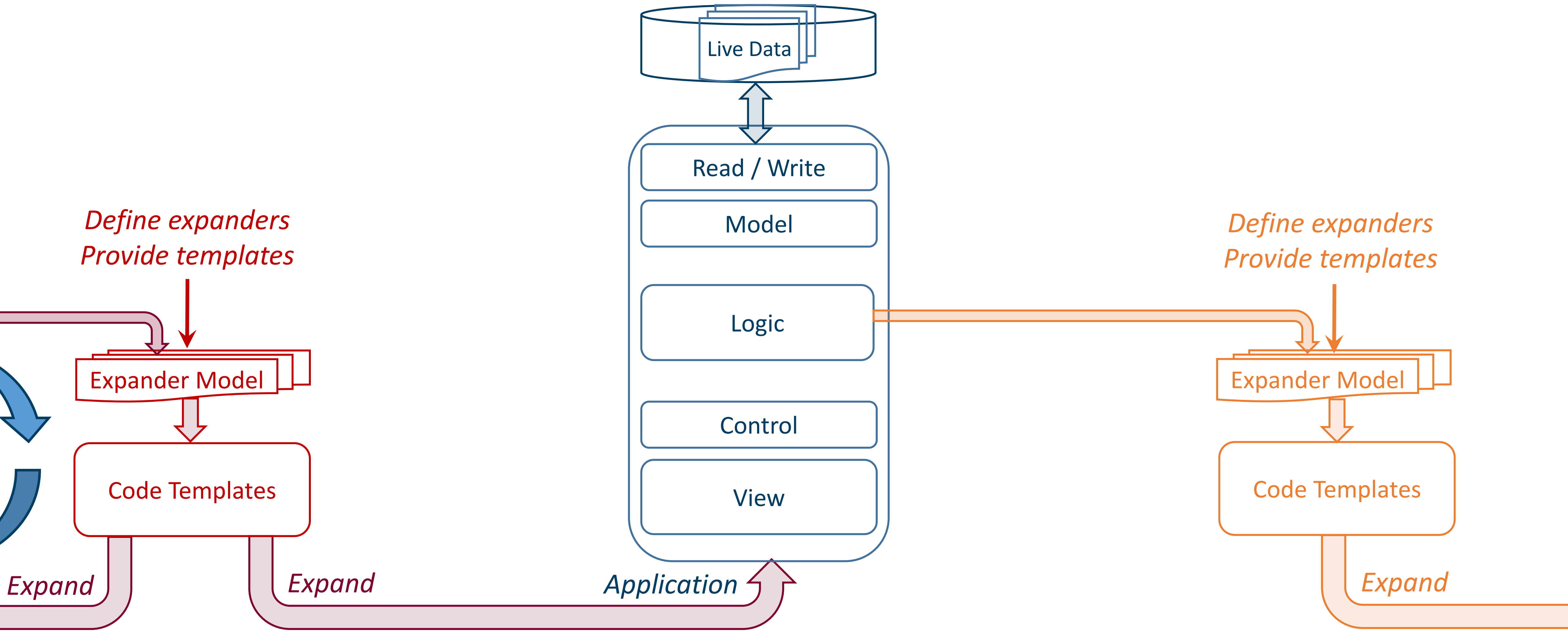
Meta-Circular Code Generation or Artifact Expansion



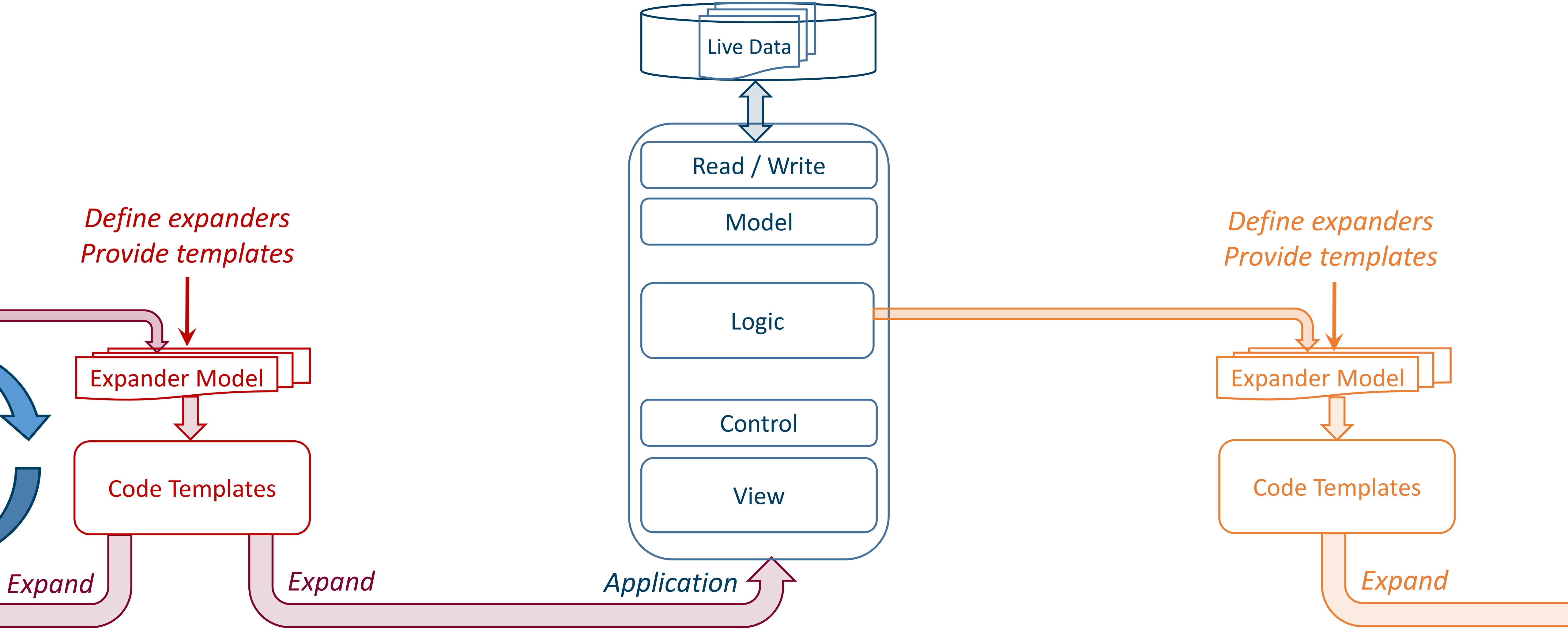
Meta-Circular Code Generation or Artifact Expansion



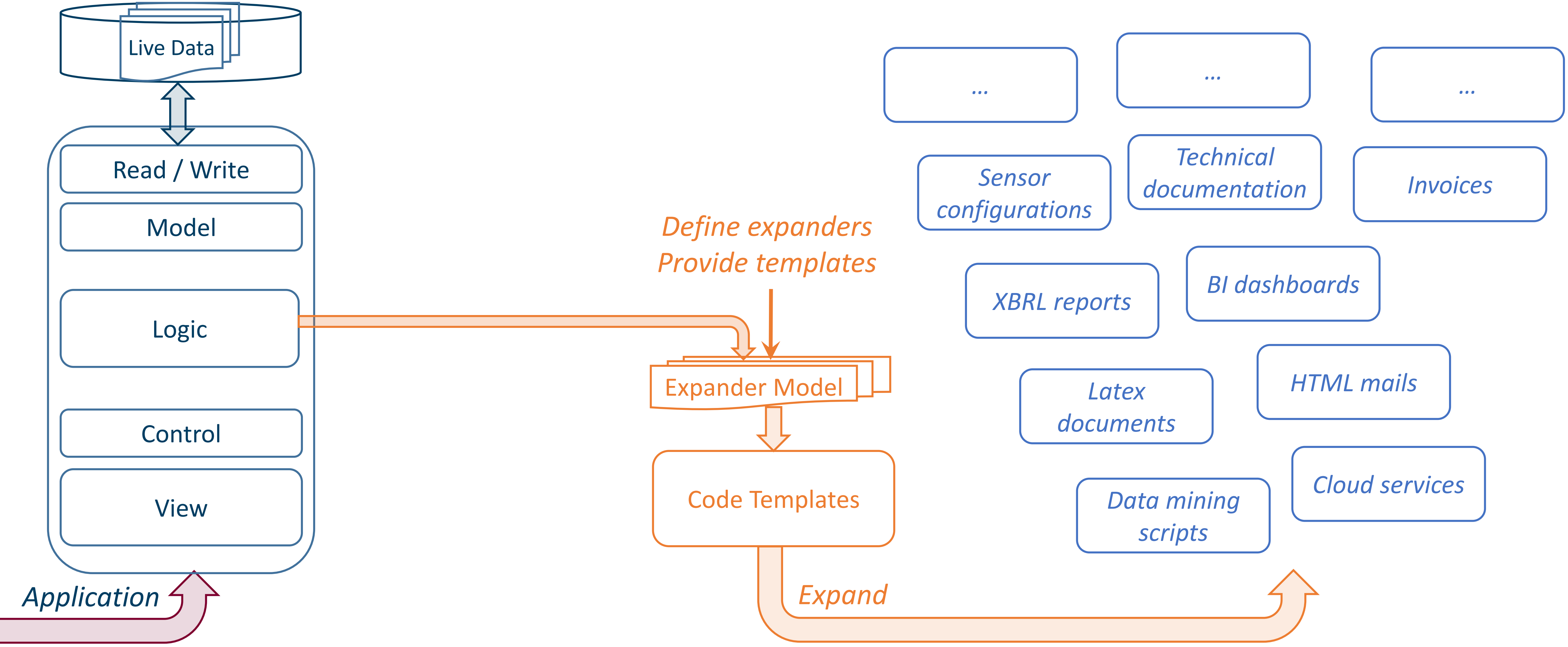
Meta-Circular Code Generation or Artifact Expansion



Meta-Circular Code Generation or Artifact Expansion



Meta-Circular Code Generation or Artifact Expansion



Overview

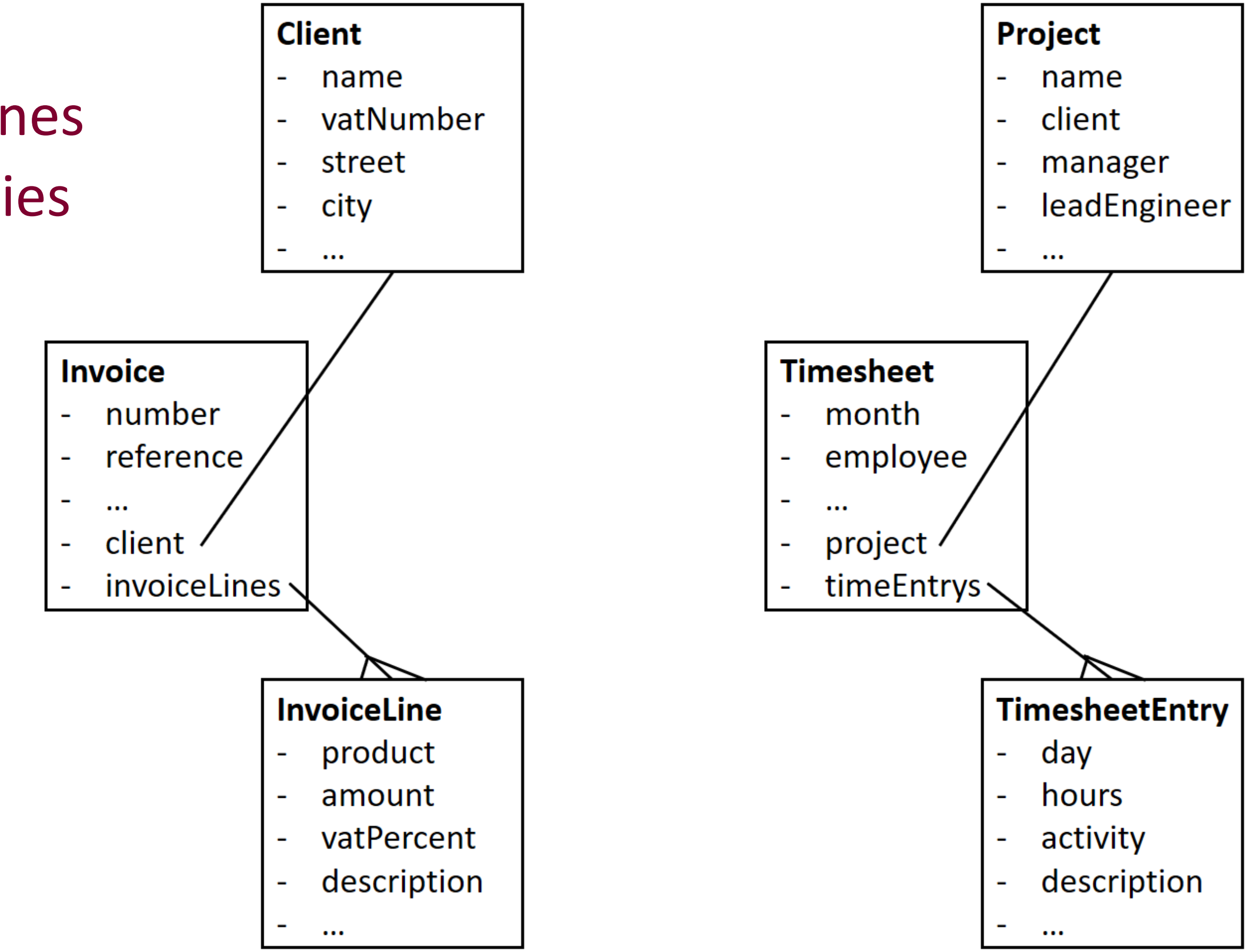
- Modular and Evolvable Document Creation
- Expansion of Evolvable Modular Structures
- Exploring Runtime Expansion of Artifacts
  - Document Creation and Information Systems
  - Declarative Control and Runtime Expansion
- Conclusion

Document Creation and Information Systems

- Interesting duality or synergy:
  - document creation requires an information system
  - information systems typically create documents
- Document creation using NST meta-circular code generation:
  - start from a regular NST information system
  - target simple administrative documents, e.g., *Invoices*, *Timesheets*
  - use model classes to feed runtime-data to the instantiation of documents
- Dedicated development is limited to:
  - Declare document generator or expander, e.g., *TexInvoiceExpander*
  - Define parameters using OGNL expressions, e.g., *TexInvoiceExpanderMapping*
  - Write LaTeX template containing the defined instance parameters

Declarative Control and Runtime Expansion

- Take two simple examples:
  - **Invoice** for Client with InvoiceLines
  - **Timesheet** for Project with Entries



Declarative Control and Runtime Expansion

TexInvoiceExpander.xml

```
<expander name=“TexInvoiceExpander"
          xmlns="http://nsx.normalizedsystems.org/20201/expander">
  <packageName>net.palver.expander.latex.invoice</packageName>
  <elementTypeName>Invoice</elementTypeName>
  <layerType name=“ROOT"/>
  <technology name="COMMON"/>
  <sourceType name=“TEX"/>
  <artifactName>Invoice-$invoice.number$.tex</artifactName>
  <artifactPath>$expansion.directory$/$artifactSubFolders$/</artifactPath>
  <isApplicable>true</isApplicable>
  <active value="true"/>
  <anchors/>
</expander>
```

TexInvoiceExpanderMapping.xml

```
<mapping xmlns="http://nsx.normalizedsystems.org/202001/expanders/mapping">

  <value name=“info" eval=“invoice.info"/>
  <value name=“number" eval=“invoice.number"/>
  <value name=“client" eval=“invoice.client.name"/>
  <value name=“vatNr" eval=“invoice.client.vatNr"/>
  <value name=“street" eval=“invoice.client.street"/>
  <value name=“city" eval=“invoice.client.city"/>
  <value name=“isForeign" eval=“!invoice.client.country.equals(‘Belgium’)"/>

  <list name=“invoiceLines” eval=“invoice.invoiceLines” param=“invoiceLine”>
    <value name=“info" eval=“invoiceLine.info"/>
    <value name=“product" eval=“invoiceLine.product"/>
    <value name=“amount" eval=“invoiceLine.amount"/>
  </list>
</mapping>
```

Overview

- Modular and Evolvable Document Creation
- Expansion of Evolvable Modular Structures
- Exploring Runtime Expansion of Artifacts
- Conclusion

Conclusions

- We adapted the NST metaprogramming environment:
  - to produce sources for documents or files
  - based on runtime data in regular NS applications
  - through declarative definitions and OGNL evaluations
- Such a document generation environment:
  - adheres to concepts like single sourcing and CCM
  - takes advantage of existing flexibility ad robustness
- Implementation is an architectural pathfinder:
  - creating currently only simple documents

Some References

- Mannaert Herwig, McGroarty Chris, Gallant Scott, De Cock Koen, **Integrating Two Metaprogramming Environments : An Explorative Case Study :** ICSEA 2020 - ISSN 2308-4235 - IARIA, 2020, p. 166-172
- Mannaert Herwig, De Cock Koen, Uhnak Peter, **On the realization of meta-circular code generation : the case of the normalized systems expanders**, ICSEA 2019 - ISSN 2308-4235 - IARIA, 2019, p. 171-176
- De Bruyn Peter, Mannaert Herwig, Verelst Jan, Huysmans Philip, **Enabling normalized systems in practice : exploring a modeling approach**, Business & information systems engineering - ISSN 1867-0202 - 60:1(2018), p. 55-67.
- Mannaert Herwig, Verelst Jan, De Bruyn Peter, **Normalized systems theory : from foundations for evolvable software toward a general theory for evolvable design**, ISBN 978-90-77160-09-1 - Koppa, 2016, 507 p.
- Mannaert Herwig, Verelst Jan, Ven Kris, **Towards evolvable software architectures based on systems theoretic stability**, Software practice and experience - ISSN 0038-0644 - 42:1(2012), p. 89-116
- Mannaert Herwig, Verelst Jan, Ven Kris, **The transformation of requirements into software primitives : studying evolvability based on systems theoretic stability**, Science of computer programming - ISSN 0167-6423 - 76:12(2011), p. 1210-1222
- Mannaert Herwig, De Bruyn Peter, Verelst Jan, **On the Interconnection of Cross-Cutting Concerns within Hierarchical Architectures**, IEEE Transactions on Engineering Management - ISSN 0018-9391 - (2020), p. 1-16.
- **Normalized Systems Foundation Lectures** : https://www.youtube.com/channel/UCc8P1LREJogSlhwmAvdkq2A
- **Normalized Systems Prime Radiant Online:** https://foundation.stars-end.net **and** https://exchange.stars-end.net

QUESTIONS ?

herwig.mannaert@uantwerp.be