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

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
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
• 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

Remote Access

Access Control

Persistency

Order Element

Invoice Element

Payment Element
Applying Design Theorems, Software Elements Emerge

Invoice
- number
- date
- order
- ...

Invoice Details

Invoice Element

Invoice Interface

Invoice
- Proxy
- Access
- Data
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

Reader classes
Model classes
Control classes
Expander classes
Code Templates

Expand
Application

Read / Write
Model
Logic
Control
View
Meta-Circular Code Generation or Artifact Expansion

- **Prime Radiant**
- **Expand**

**Pathway:**
- **Read / Write** → **Model** → **Logic** → **Control** → **View** → **nsx-prime** → **Code Templates** → **Model classes** → **Control classes** → **Expander classes** → **Model** → **Meta Model** → **Read / Write** → **Application**
Meta-Circular Code Generation or Artifact Expansion

Prime Radiant

nsx-prime

Read / Write
Model
Logic
Control
View

Model classes
Control classes
Expander classes
Code Templates

Reader classes
Model

Read / Write
Model
Logic
Control
View

Application

Expand

Expand
Meta-Circular Code Generation or Artifact Expansion

- Extend meta-model
  - Meta Model
  - Model
  - Read / Write
  - Model
  - Logic
  - Control
  - View

- Define expanders
  - Provide templates
  - Expander Model
  - Code Templates
  - Expander
  - MetaCircle
  - Prime Radiant
  - Expand

- Application
  - Read / Write
  - Model
  - Logic
  - Control
  - View
Meta-Circular Code Generation or Artifact Expansion

Extend meta-model

Define expanders
Provide templates

Expander Model

Prime Radiant
Expand

MetaModel
Model
Read / Write
Model
Logic
Control
View

Expander Code Templates

Live Data
Read / Write
Model
Logic
Control
View
Application

Expand
Expand

MetaCircle
Meta-Circular Code Generation or Artifact Expansion

Define expanders
Provide templates

Expander Model
Code Templates

Read / Write
Model
Logic
Control
View

Live Data

Define expanders
Provide templates

Expander Model
Code Templates

Expand
Expand
Application
Meta-Circular Code Generation or Artifact Expansion

Define expanders
Provide templates

Define expanders
Provide templates

Expander Model
Code Templates
Expand

Expander Model
Code Templates
Expand

Expand

Read / Write
Model
Logic
Control
View
Live Data
Application
Meta-Circular Code Generation or Artifact Expansion

Live Data

- Read / Write
- Model
- Logic
- Control
- View

Define expanders
Provide templates

Code Templates

Expand

- Expander Model

- Sensor configurations
- Technical documentation
- Invoices
- BI dashboards
- XBRL reports
- HTML mails
- Data mining scripts
- Cloud services
- Latex documents
- Data mining scripts
On the Use of Code Generation Patterns for the Creation of Evolvable Documents and Runtime Artifacts

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

```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>

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

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 and 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


• 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


• Normalized Systems Foundation Lectures: https://www.youtube.com/channel/UCc8P1LREJogSlhwmAvdqk2A
• Normalized Systems Prime Radiant Online: https://foundation.stars-end.net and https://exchange.stars-end.net
QUESTIONS?
herwig.mannaert@uantwerp.be