Hochschule RheinMain

AN NMF SOLUTIONS TO THE TTC2023 CONTAINERS TO MINIYAML CASE

Prof. Dr. Georg Hinkel
20.07.2023

NMF
.NET Modeling Framework

Hochschule RheinMain

- Modelling platform for the .NET platform
  - Generate code from NMeta or Ecore metamodels
  - Load and save XMI models
  - Integrated incrementalization capabilities

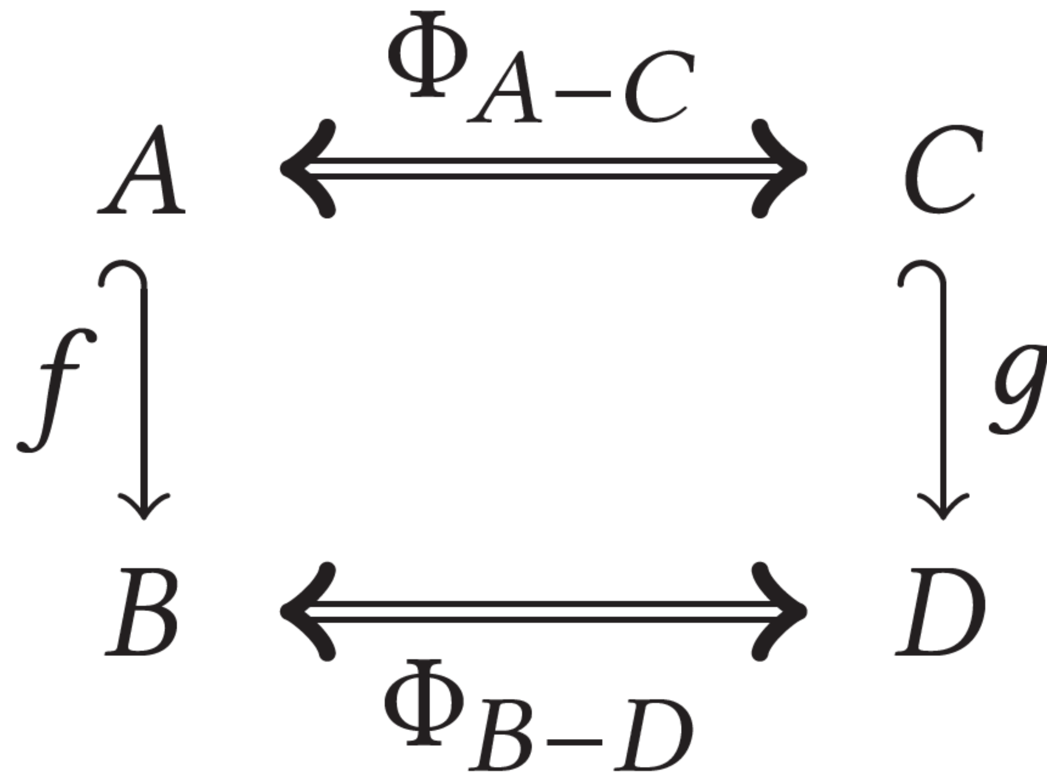
- Open Source
  - Apache 2.0
  - https://github.com/NMFCode/NMF

NMF-Expressions by: georg.hinkel

429.825 total downloads last updated 21 days ago Latest version: 2.0.188

NMF SYNCHRONIZATIONS

Hochschule RheinMain



- Model synchronization language and framework
- Algebraic model of synchronization blocks
  - Proved correctness
  - Proved hippocraticness
- Declarative
  - Uni- or bidirectional execution
  - Optional change propagation
  - Check-only mode
- Internal DSL in C#

ARCHITECTURE

Hochschule RheinMain

- NMF is written in C#, therefore difficult to call from Java

- Solution approach
  1. Start companion process in .NET
  2. Use Ecore Notification API to record changes made to the models
  3. Persist changes to NMeta Changes format
  4. Load change model in companion process
  5. Deserialize change model
  6. Apply change model
  7. Serialize result model (unless changes are done in idle mode)

- Architecture causes a lot of overhead ➔ separate time measurement to ensure only change propagation counts

TROUBLE: STABILITY OF ECLIPSE

- My Eclipse Modeling Tools 2022-12 had stability issues
  - BXTool suddenly could not be found, problem only resolved through reboot (no guarantee!)
- Consequence: Original solution did not pass all tests

ISOMORPHISMS

Hochschule RheinMain

- First step: Identify isomorphisms (aka correspondences)
  - Entire composition model corresponds to entire YAML model
  - A container corresponds to a YAML node under services
  - A volume corresponds to a YAML node under databases
- Isomorphisms are reflected in rules

```
internal class ContainersToMiniYaml :
  ReflectiveSynchronization
{
  public class MainMap :
    SynchronizationRule<Composition, Map> …

  public class Container2MapEntry :
  SynchronizationRule<IContainer, IMapEntry>
  …

  public class Volume2MapEntry :
    SynchronizationRule<IVolume, IMapEntry>
  …
}
```

TROUBLE: THINGS THAT ARE NOT ISOMORPHIC

Hochschule **RheinMain**

- Composition model has explicit image model elements
- Images in YAML are implicit, given by map entry called image

Custom code necessary to deal with Image model elements

- Explicit model elements to denote volume mounts
- Volume mounts are implicit, combined reference to volume and mount directory

Custom code necessary to deal with volume mounts

SYNCHRONIZATION BLOCKS

MainMap

Hochschule RheinMain

```
SynchronizeLeftToRightOnly(_ => "2.4", m => m.Scalar<string>("version"));

SynchronizeMany(SyncRule<Container2MapEntry>(),
    c => new ServicesCollection(c),
    m => m.ForceEntries("services"));

SynchronizeMany(SyncRule<Volume2MapEntry>(),
    c => c.Nodes.OfType<INode, IVolume>(),
    m => m.ForceEntries("volumes"));
```

Custom lens to read/write scalar with given key

Custom collection that takes images from containers and adds them as root elements

Custom lens to read/write elements with given key

SYNCHRONIZATION BLOCKS

Container2MapEntry

Hochschule RheinMain

```
Synchronize(c => c.Name, me => me.Key);

Synchronize(c => GetImage(c), me => me.Scalar<string>("image"));
Synchronize(c => c.Replicas.WithDefault(1), me => me.Scalar<int>("replicas"));

SynchronizeMany(
    c => new VolumeMountCollection(c),
    me => new ScalarCollection(me, "volumes"));

SynchronizeMany(
    c => new DependsOnNameCollection(c),
    me => new ScalarCollection(me, "depends_on"));
```

CUSTOM LENSES

DependsOnNameCollection

Hochschule **RheinMain**

```
private class DependsOnNameCollection : CustomCollection<string> {
    private readonly IContainer _container;
    public DependsOnNameCollection(IContainer container)
      :  base(container.DependsOn.Select(c => c.Name)) {_container = container;}

    public override void Add(string item)
    {
        var composition = (IComposition)_container.[illegible]nt;
        var container = composition.Nodes.….First(c => [illegible]e == item);
        _container.DependsOn.Add(container);
    }

    public override void Clear() { … }
    public override bool Remove(string item) { … }
}
```

Updates are processed automatically, but developer needs to specify Add/Clear/Remove

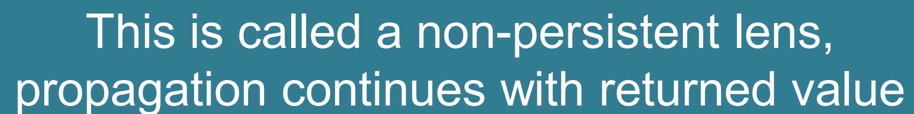
CUSTOM LENSES

WithDefault

Hochschule RheinMain

```
[LensPut(typeof(YamlHelpers), nameof(ApplyDefault))]
public static T? WithDefault<T>(this T? value, T defaultValue) { … }

public static T? ApplyDefault<T>(this T? value, T defaultValue, T? newValue) { … }
```



CUSTOM LENSES

GetImage

Hochschule RheinMain

```
private static readonly ObservingFunc<IContainer, string?> _getImage =
    ….FromExpression(c => c.Image != null ? c.Image.Image_ : null);

[ObservableProxy(typeof(Container2MapEntry), nameof(GetImageIncremental))]
[LensPut(typeof(Container2MapEntry), nameof(SetImage))]
public static string? GetImage(IContainer container)
    => _getImage.Evaluate(container);

public static INotifyValue<string?> GetImageIncremental(IContainer container)
    => _getImage.Observe(container);

public static void SetImage(IContainer container, string image) { … }
```

Persistent lens, i.e. propagation stops

CONCLUSIONS

- NMF Synchronizations allows to solve the case with one coherent transformation

- Developers have to resolve ambiguities manually
  - Custom lenses

- Probably, smaller lenses are easier to test in isolation than entire model transformations
  - Some lenses are very generic, especially on the YAML side
  - Maybe theorem provers could test/proof lens properties?

Hochschule RheinMain

THANKS FOR YOUR ATTENTION

Prof. Dr. Georg Hinkel, georg.hinkel@hs-rm.de