An NMF solution to the Families to Persons case at the TTC 2017

Georg Hinkel

FZI FORSCHUNGSZENTRUM INFORMATIK

Sparse adoption of MDE in industry

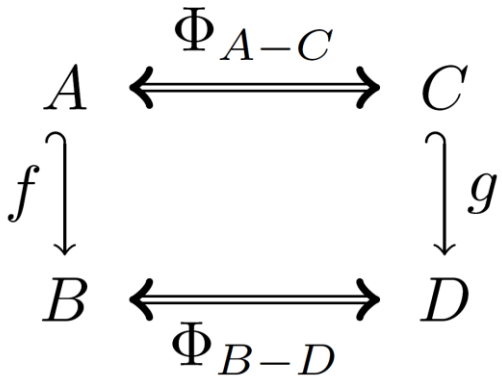
- Tool support perceived insufficient [Sta06,Mo+13]
  - Much less manpower in tool development than IDEs such as Visual Studio, IntelliJ, …
- Developers hardly change their primary language [MR13]
  - Project requirements or code reuse

.NET Modeling Framework (NMF)

- Repository management in .NET
  - Generate code for metamodels
  - Load models
  - Save models
  - (Mostly) Compatible to EMF
- Multimode Model Synchronization
  - Incremental
  - Bidirectional
  - Internal Language in C#
- Open source: http://github.com/NMFCode/NMF

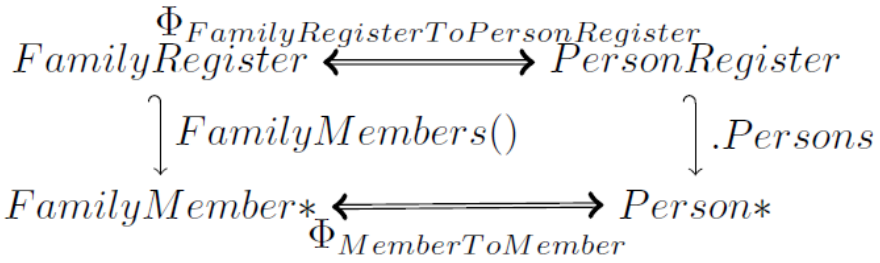
NMF Synchronizations

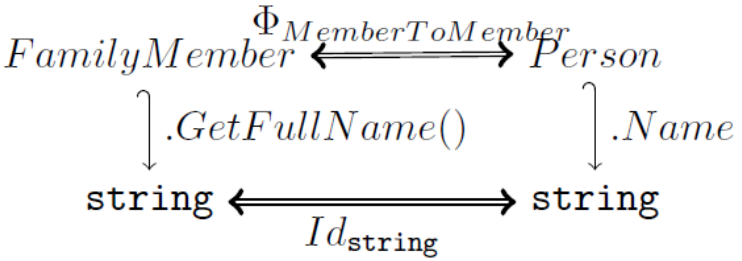
- Formal basis: Synchronization blocks



- Bidirectional, incremental model synchronization
  - Lenses used for writing values
  - Incrementalization system used for incremental updates
  - Updates can be constructed directly for all types of changes
  - Synchronization is hippocratic

Synchronization in the Families2Persons Case






```
1  public class FamilyRegisterToPersonRegister : SynchronizationRule<FamilyRegister, PersonRegister> {
2    public override void DeclareSynchronization() {
3      SynchronizeMany(SyncRule<MemberToMember>(),
4                      fam => new FamilyMemberCollection(fam),
5                      persons => persons.Persons);
6    }
7  }
8  public class MemberToMember : SynchronizationRule<IFamilyMember, IPerson> {
9    public override void DeclareSynchronization() {
10     Synchronize(m => m.GetFullName(), p => p.Name);
11   }
12 }
```

Custom Lenses: GetFullName

- Lenses provided through functions + annotated lens put + incrementalization
  - To incrementalize a function ➔ Custom incrementalization
  - To invert a function ➔ provide lens put
  - Reuse NMF incrementalization system for incrementalization

```
1  private static ObservingFunc<IFamilyMember, string> fullName =
2    new ObservingFunc<IFamilyMember, string>(m =>
3        m.Name == null ? null : ((IFamily)m.Parent).Name + ",␣" + m.Name);
4
5  [LensPut(typeof(Helpers), "SetFullName")]
6  [ObservableProxy(typeof(Helpers), "GetFullNameInc")]
7  public static string GetFullName(this IFamilyMember member) {
8    return fullName.Evaluate(member);
9  }
10 public static INotifyValue<string> GetFullNameInc(this IFamilyMember member) {
11   return fullName.Observe(member);
12 }
13 public static void SetFullName(this IFamilyMember member, string newName) {
14   ...
15 }
```

Custom Collection Lenses: FamilyMembers

- NMF provides a base class for custom collections: `CustomCollection`
  - Takes a query as argument
  - User has to specify collection manipulation methods `Add`, `Remove`, `Clear`

```
1  private class FamilyMemberCollection : CustomCollection<IFamilyMember> {
2    public FamilyRegister Register { get; private set; }
3    public FamilyMemberCollection(FamilyRegister register)
4      : base(register.Families.SelectMany(fam => fam.Children.OfType<IFamilyMember>()))
5    { Register = register; }
6
7    public override void Add(IFamilyMember item) {
8      ...
9    }
10   public override bool Remove(IFamilyMember item) {
11     ...
12   }
13   public override void Clear() {
14     ...
15   }
16 }
```

Temporary Stereotypes

- Problem: Last name and gender of a person is encoded in containment hierarchy ➔ unavailable before new family member is added
- Solution: Save this information in a temporary stereotype

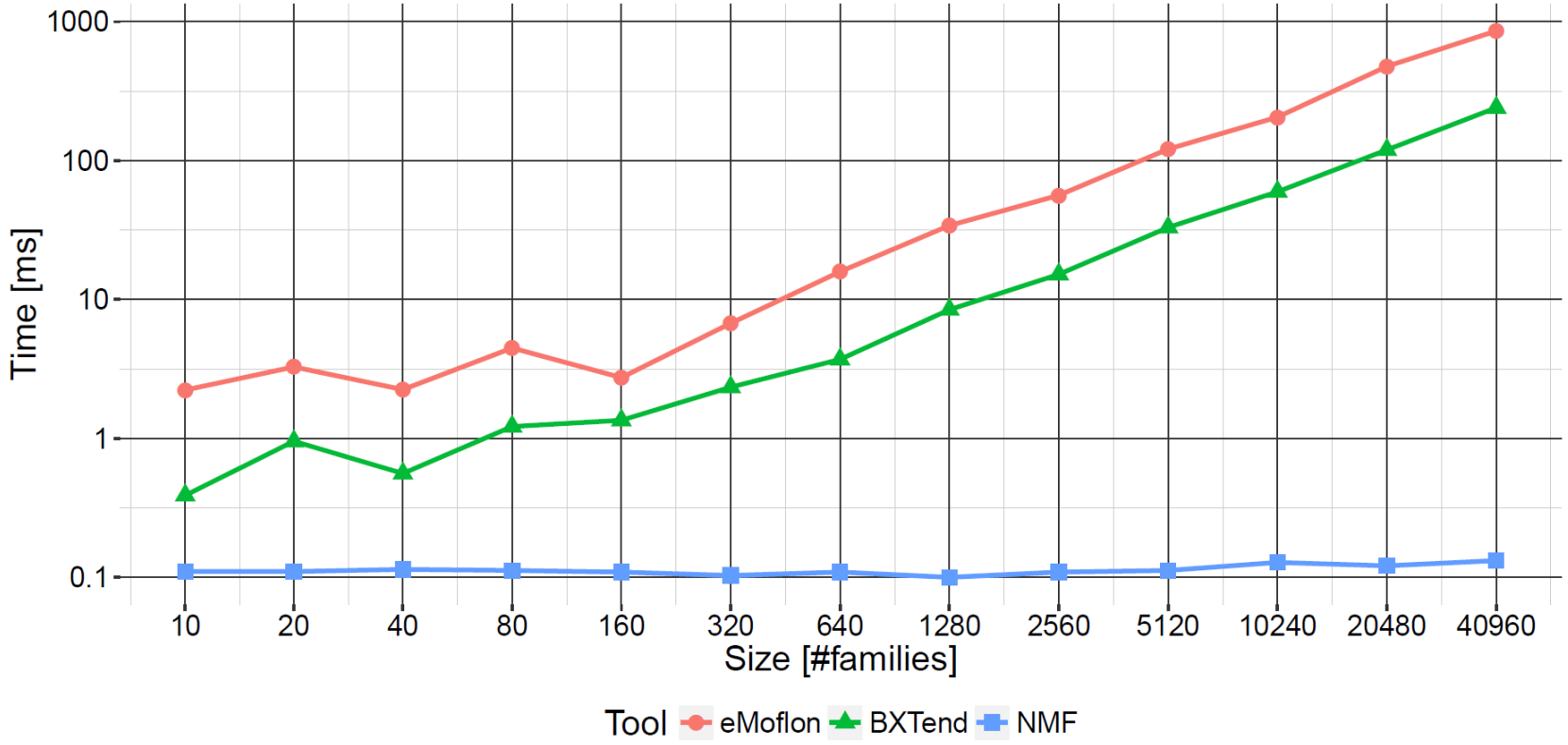
```
1  public class MemberToMale : SynchronizationRule<IFamilyMember, IMale> {
2    public override void DeclareSynchronization() {
3      MarkInstantiatingFor(SyncRule<MemberToMember>(),
4            leftPredicate: m => m.FatherInverse != null || m.SonsInverse != null);
5     }
6    protected override IFamilyMember CreateLeftOutput(IMale input, ...) {
7      var member = base.CreateLeftOutput(input, candidates, context, out existing);
8      member.Extensions.Add(new TemporaryStereotype(member) {
9        IsMale = true,
10       LastName = input.Name.Substring(0, input.Name.IndexOf(','))
11     });
12     return member;
13   }
14 }
```

Integration into the benchmark framework

- NMF runs on a different platform and has its own model representation
- Therefore, solution runs as a separate continuous process that communicates with the benchmark framework through stdin/stdout
  1. Record model changes and export to NMF model change format
  2. Read the changes from NMF solution
  3. Propagate
  4. Serialize target (or source) model
  5. Deserialize target model to perform assertions in the benchmark framework
     - Set Update Policy: Write commands to stdin
- Model change recorder does not support moves ➔ NMF solution does not supports move
- Custom time measurement for scalability measurements

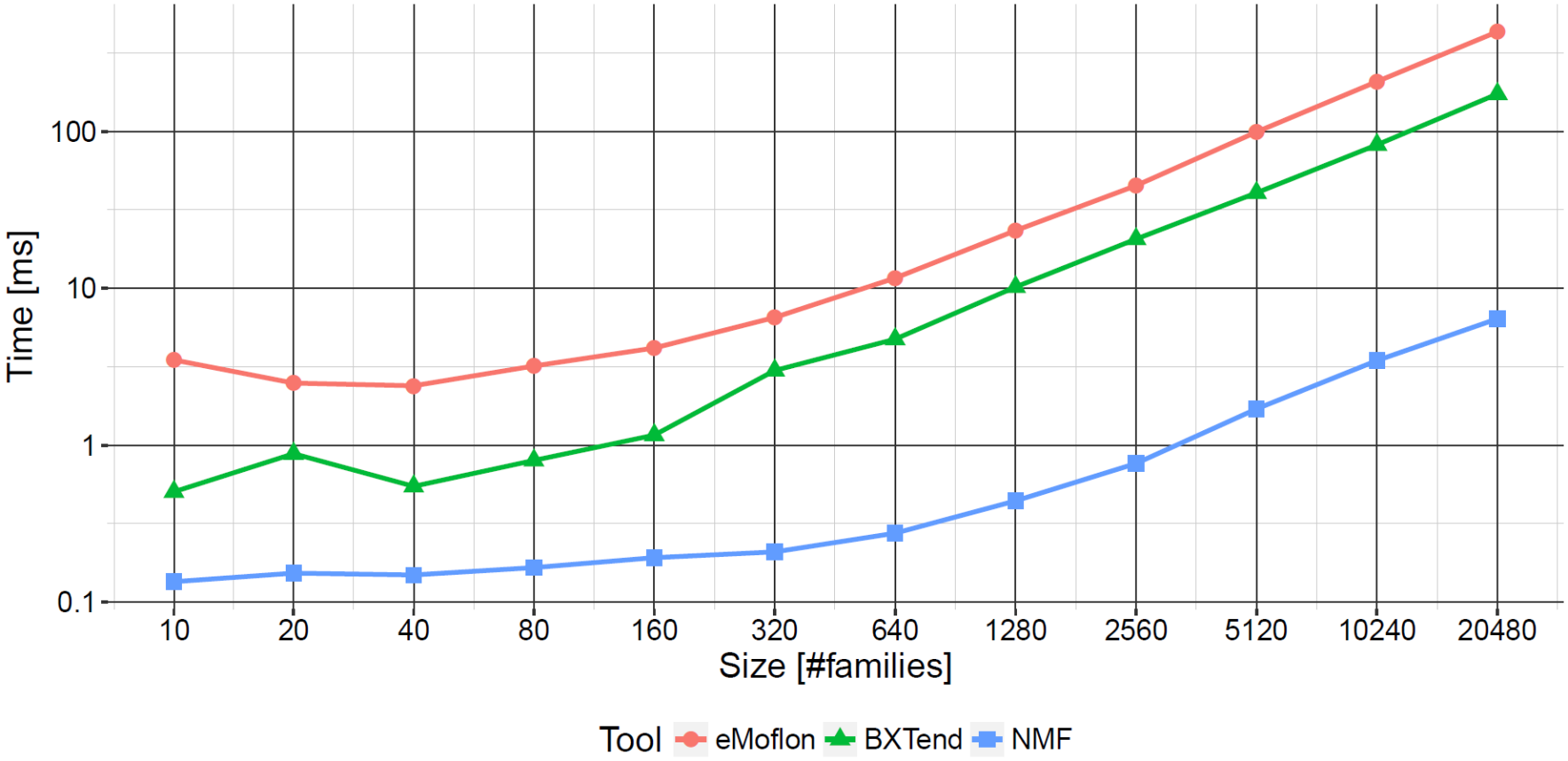
Evaluation

- Conciseness
  - 196 lines (73 of them empty or braces) + „benchmark overhead“
- Performance: Incremental Forward depicted below ($\Omega(1)$)



Incremental Backward

- Update of a new person has to traverse all families ➔ $\Omega(n)$



Conclusion

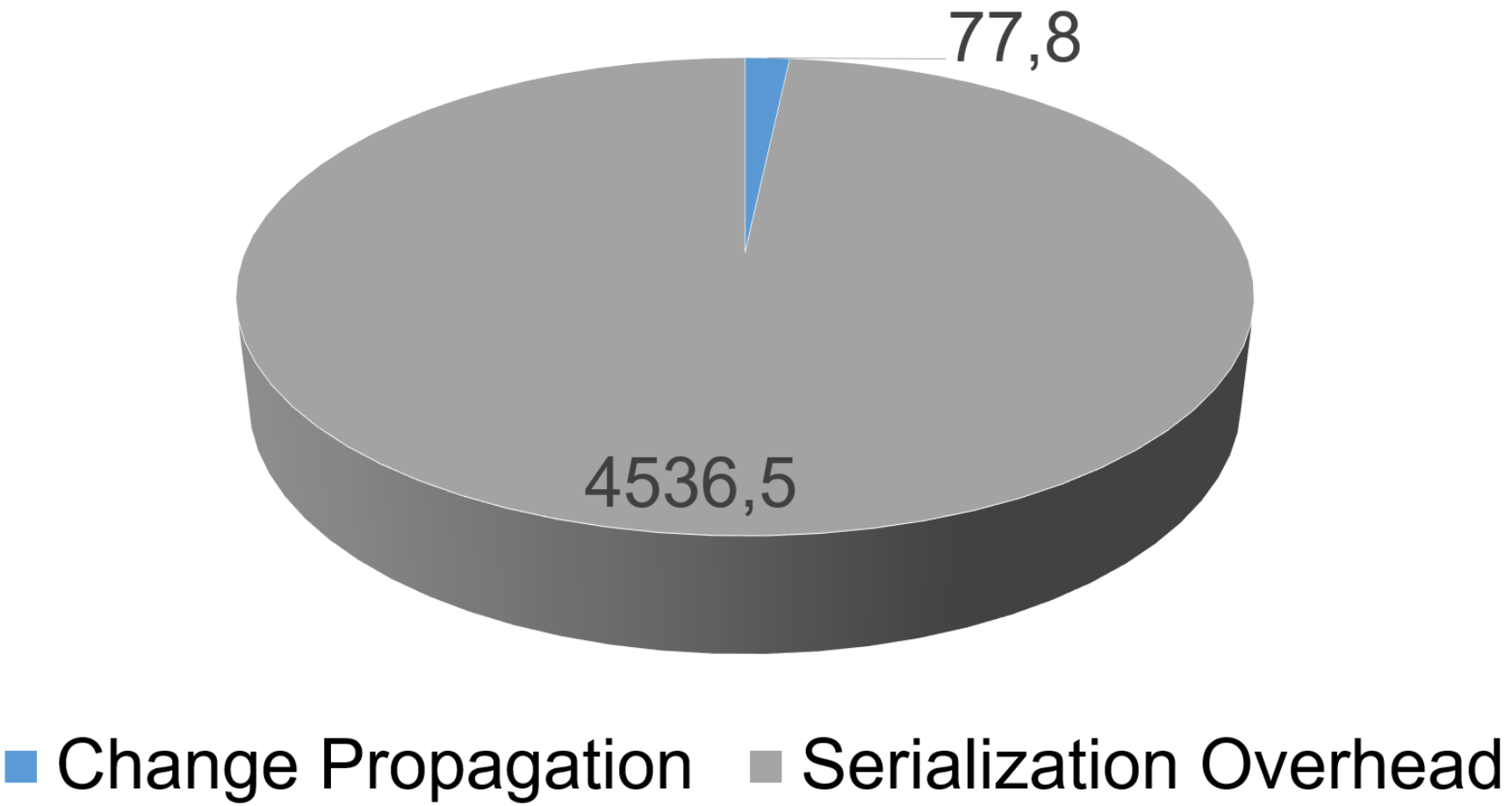
- Key advantages of the solution
  - Concise (about as concise as external languages)
  - Declarative incrementality
  - Declarative bidirectionality
  - Easily extensible
  - Correctness of the synchronization engine formally proven
  - Synchronization is hippocratic
  - Solution easily integrates into C# ➔ good tool support

- Drawbacks
  - Move not implemented in benchmark solution
  - Serialization overhead leads to large total runtime of the solution

hinkel@fzi.de

THANK YOU FOR YOUR ATTENTION

Serialization Overhead (Batch Forward Tests)



Incremental Forward

Size	eMoflon	BXtend	NMF
222	0.0	0.0	0.0
442	0.0	0.0	0.0
882	0.0	0.0	0.0
1,762	0.0	0.0	0.0
3,522	0.0	0.0	0.0
7,042	0.0	0.0	0.0
14,082	0.015	0.0	0.0
28,162	0.031	0.016	0.0
56,322	0.062	0.016	0.0
112,642	0.11	0.031	0.0

Incremental Backward

Size	eMoflon	BXtend	NMF
222	0.0	0.0	0.0
442	0.0	0.0	0.0
882	0.0	0.0	0.0
1,762	0.0	0.0	0.0
3,522	0.0	0.0	0.0
7,042	0.0	0.0	0.0
14,082	0.016	0.0	0.0
28,162	0.015	0.016	0.0
56,322	0.047	0.016	0.0
112,642	0.094	0.046	0.001

Batch Forward

Size	eMoflon	BXtend	NMF
222	0.0	0.0	0.062
442	0.016	0.0	0.063
882	0.015	0.0	0.063
1,762	0.046	0.0	0.08
3,522	0.063	0.0	0.087
7,042	0.125	0.0	0.144
14,082	0.266	0.0	0.342
28,162	0.547	0.016	0.649
56,322	1.172	0.031	1.29
112,642	2.407	0.062	2.475

Batch Backward

Size	eMoflon	Bxtend	NMF
222	0.015	0.0	0.062
442	0.016	0.0	0.066
882	0.031	0.0	0.066
1,762	0.125	0.0	0.084
3,522	0.406	0.0	0.103
7,042	1.453	0.015	0.178
14,082	5.437	0.047	0.455
28,162	20.891	0.281	1.132
56,322	83.733	1.125	3.074
112,642	332.482	4.766	9.299