Christopher Werner
Institute of Software and Multimedia Technology, Software Technology Group

Model Synchronization with the Role-oriented Single Underlying Model

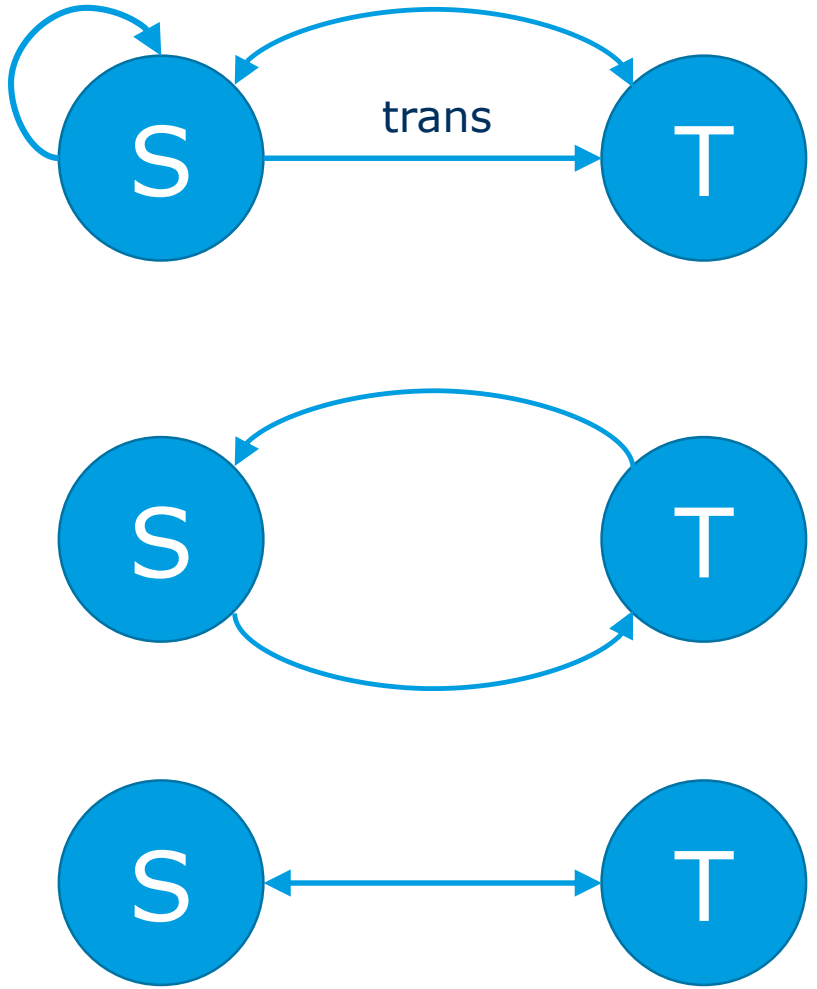
Christopher Werner and Uwe Aßmann

Copenhagen

October 14th 2018

Model Synchronization

- Multiple models express different concerns of interrelated concepts
- Interrelated concepts lead to much related models
- Related models:
  - Contain redundant information
  - Independently editing leads to inconsistencies
- Solution: Defining model synchronizations between interrelated models
- Unidirectional synchronization
- Bidirectional synchronization



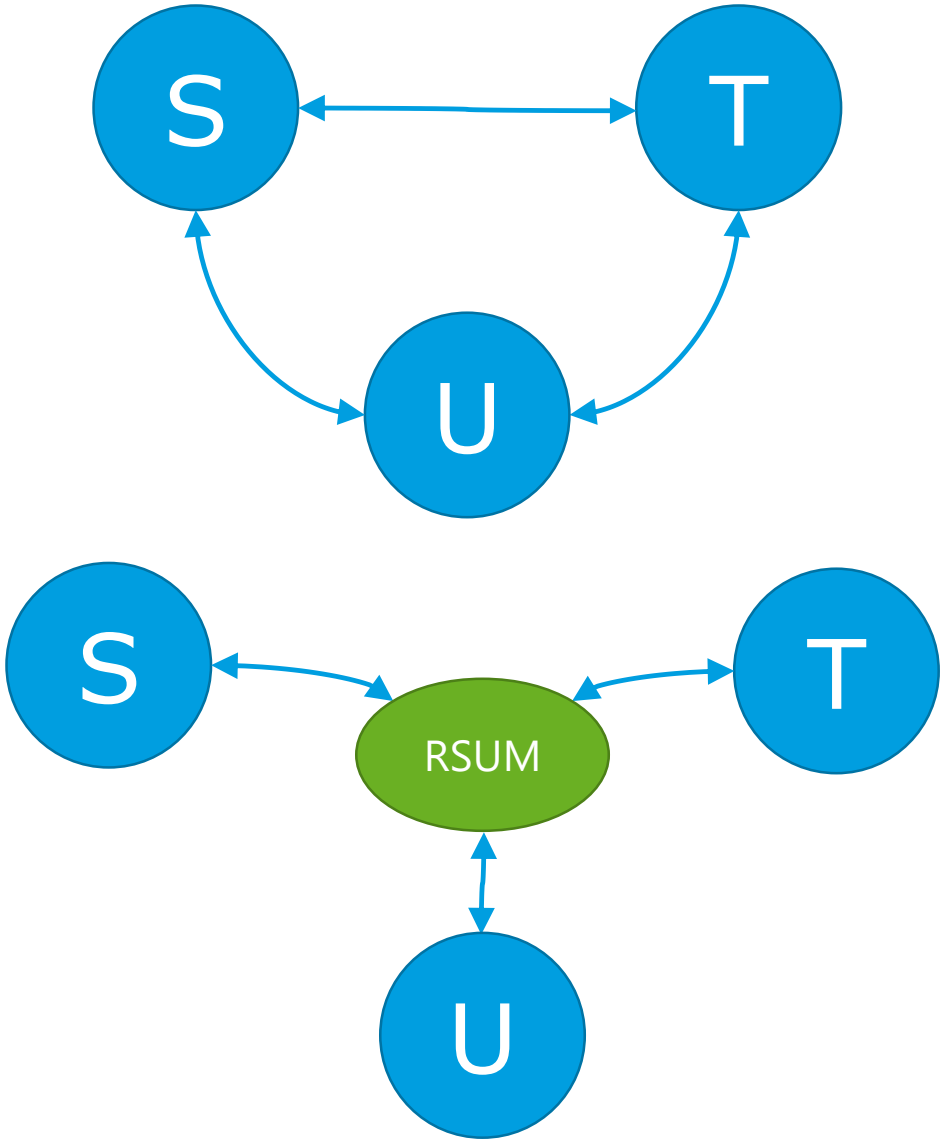
Model Synchronization at Runtime

Currently often used:

- Manually triggering of the synchronization process
- Batch updates of the whole model
- Specification of Synchronization rules at design time
- Synchronization of two models

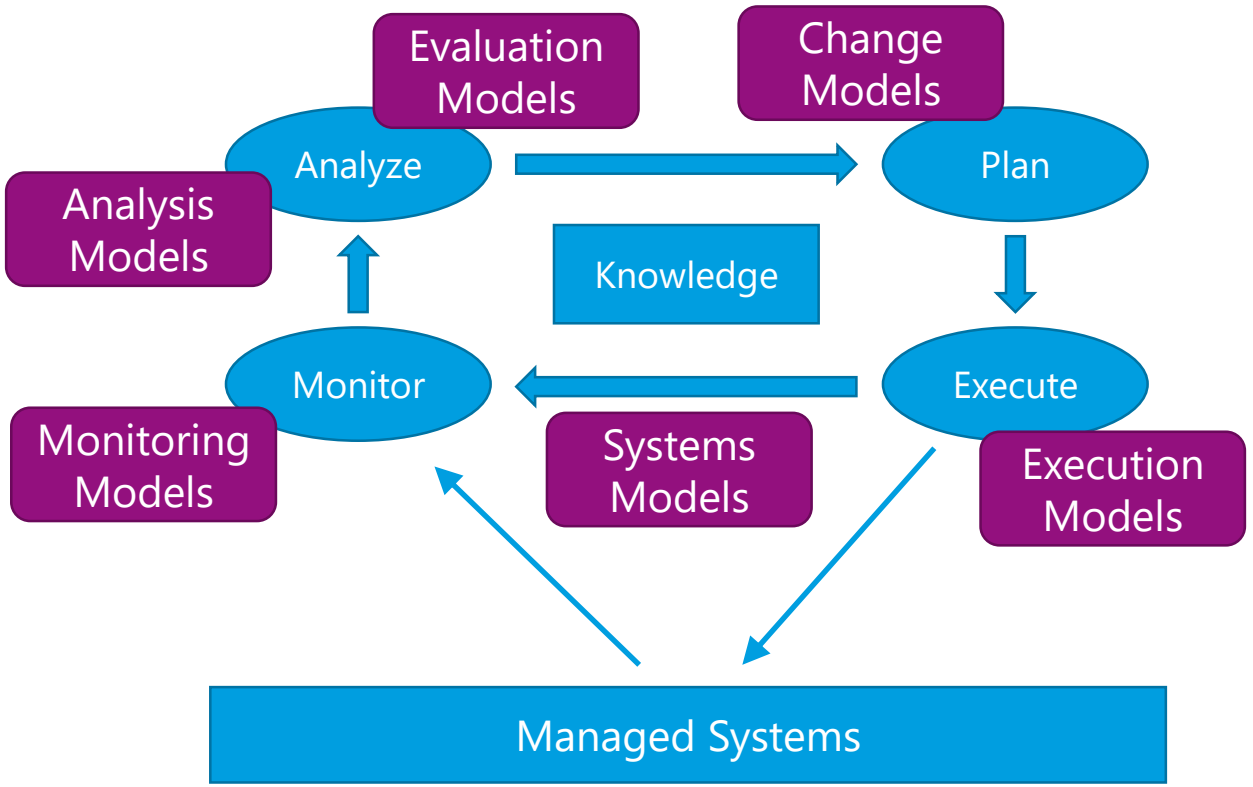
Runtime model synchronization:

- Automatically triggering of the synchronization process (immediate and continuous updates at runtime)
- Incremental updates of small changes
- Specification of models/views from the single underlying model (SUM) at runtime (add and remove models/views at runtime)
- Multiple models are views of underlying model



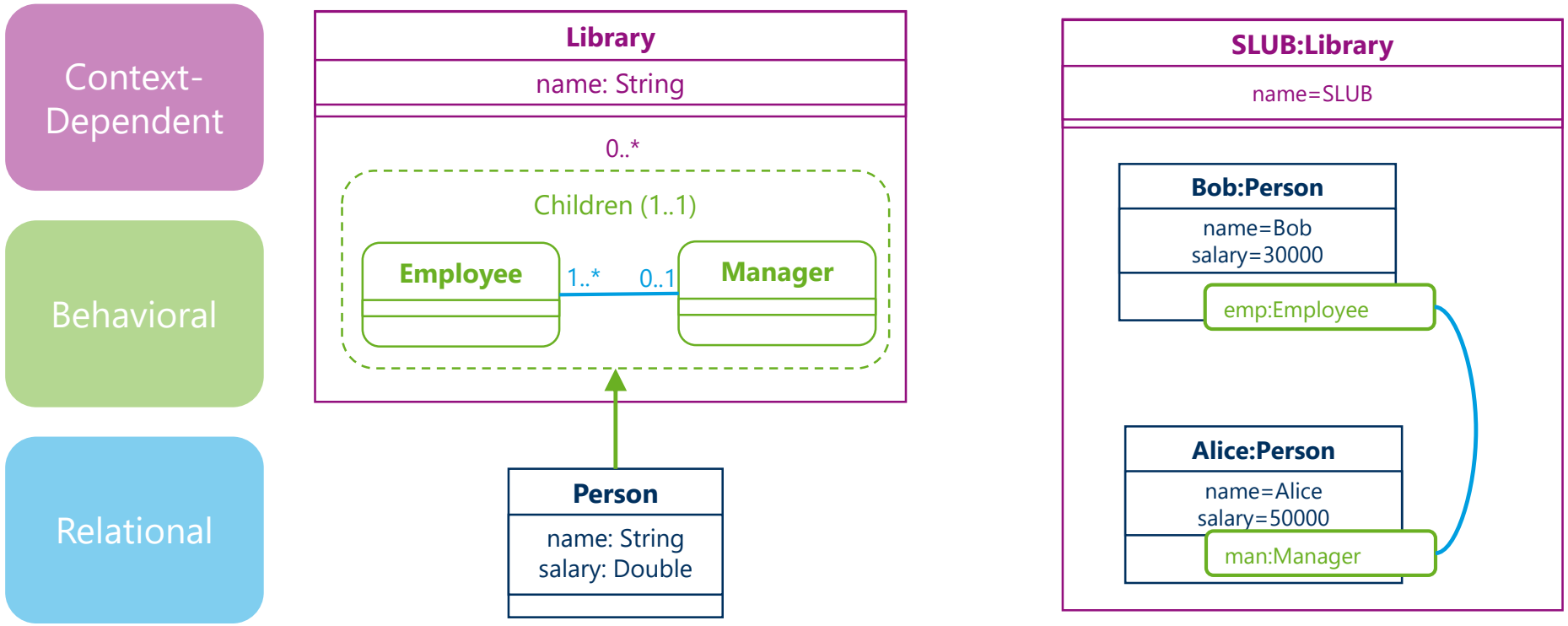
Why Model Synchronization at Runtime?

- Multiple related models in a self-adaptive software system
- This models must be hold consistent over time
- Needs an efficient runtime synchronization mechanism



Roles in a Nutshell

Library Example *[Kühn2014]*



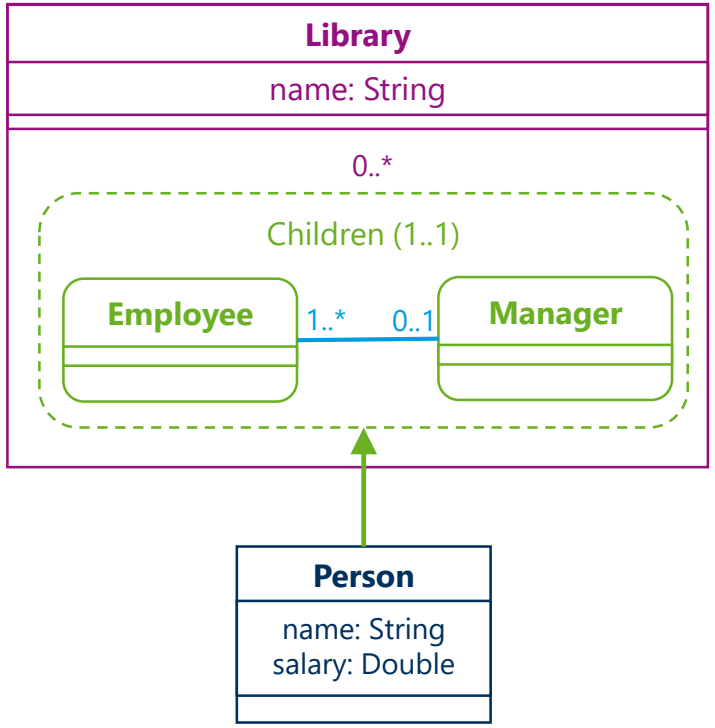
Research Questions

Goals:

- Simplify model synchronization between multiple runtime models with a single underlying model
- Flexible and extensible single underlying model approach with runtime view creation mechanisms

Requirements:

1. Incremental model synchronization
2. Runtime model synchronization
3. Runtime creation of views
4. Synchronization of multiple models
5. Integration of runtime models as views and elements in the single underlying model



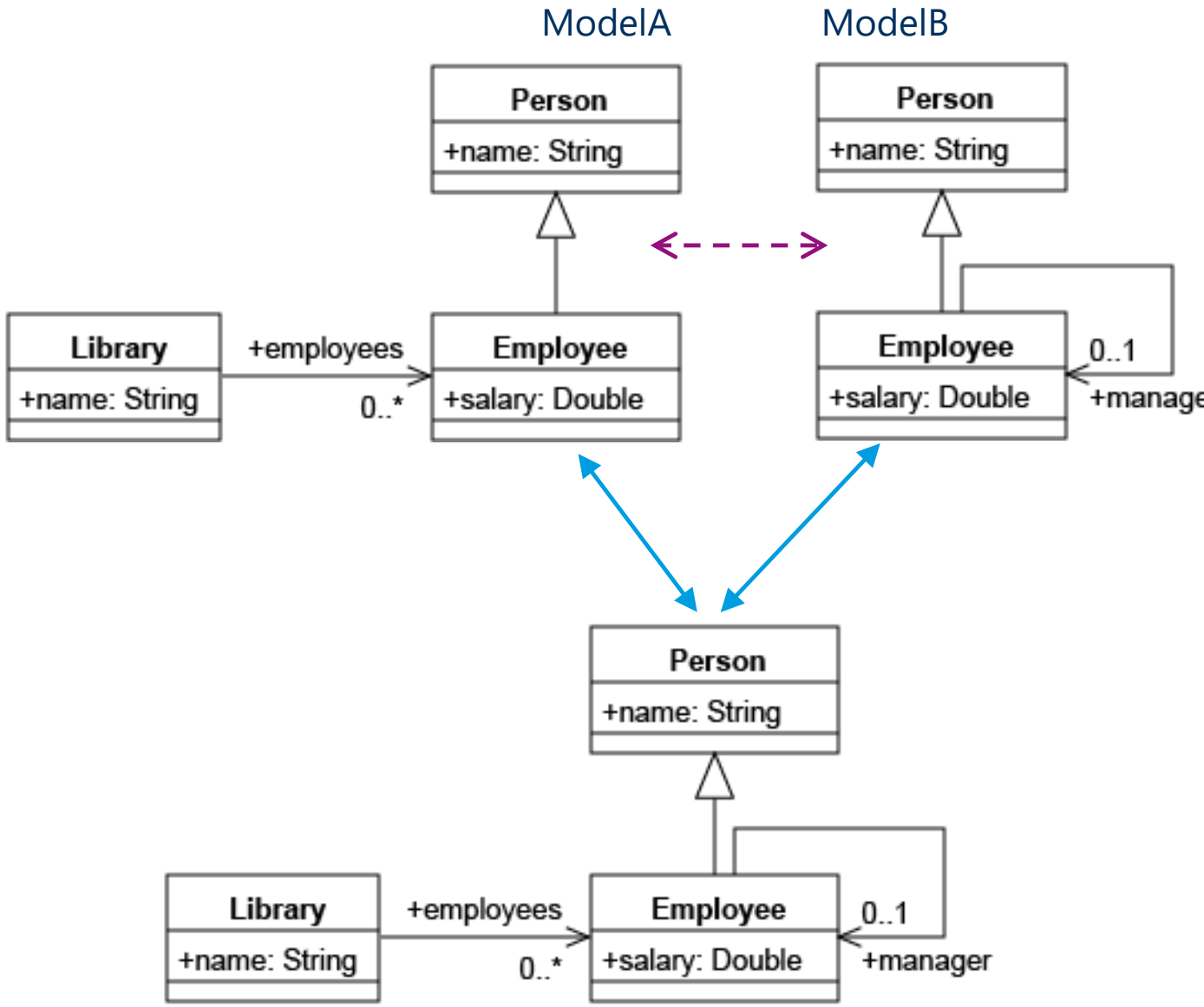
Running Example

- Two related models in the library and employee context
- Combination to one single underlying models
- Related models are now only views and are automatically synchronized with the underlying model

Combination:

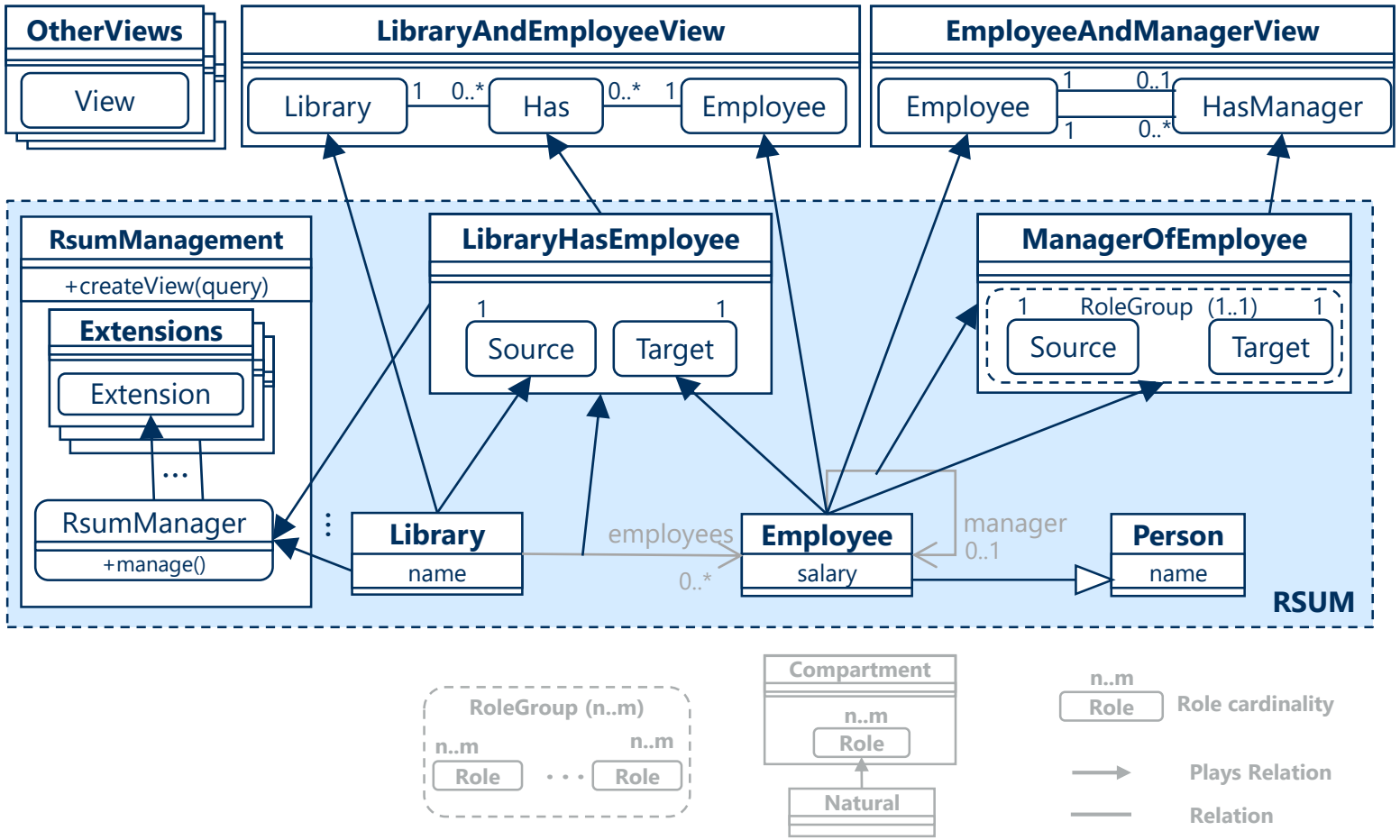
ModelA.Person = ModelB.Person

ModelA.Employee = ModelB.Employee



Overall Concept

- RSUM contains naturals, relational compartments and management compartment
- *RSUMManagement* manages the instances in the RSUM and the extensions
- *RSUMManager* coordinates the role bindings
- Runtime integration of new naturals, relational compartments and extensions possible
- Traceability information over the play relations



Concept of Relational Compartments

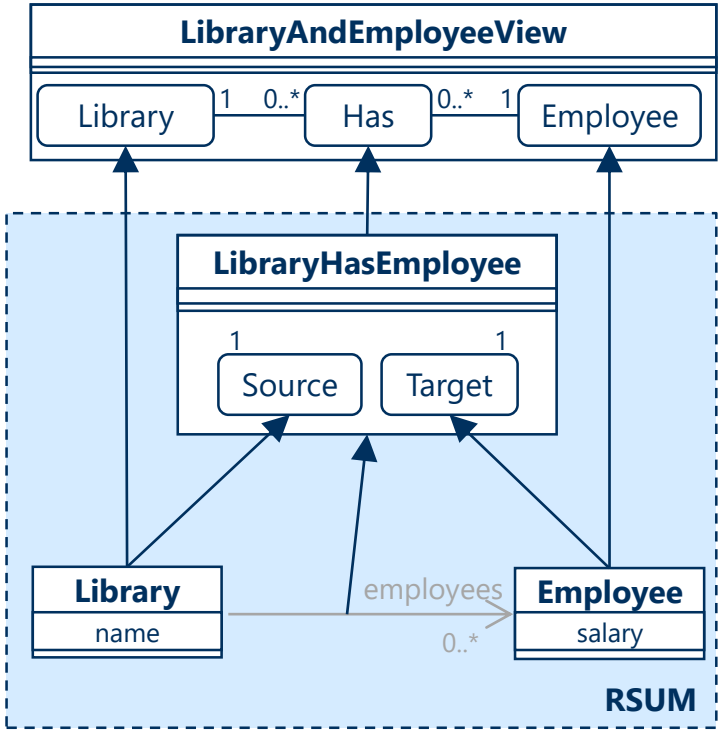
Relational compartments represent relations in the underlying model

Advantages:

1. Loading relations at runtime (using class loader functionality from programming languages like Scala and Java)
2. Extending naturals with new relations at runtime (playing of roles in relational compartments)
3. Add behavior and states to relations (methods and attributes in relational compartments)
4. n-ary relations (more role types in the relational compartment)

Limitations:

1. One relational compartment for each instance pair
2. Management of play relations of instances

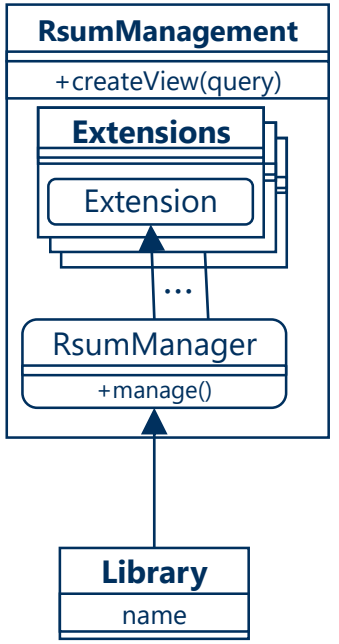


Extension Mechanism

Extension mechanism allows adding of new functionalities to the RSUM

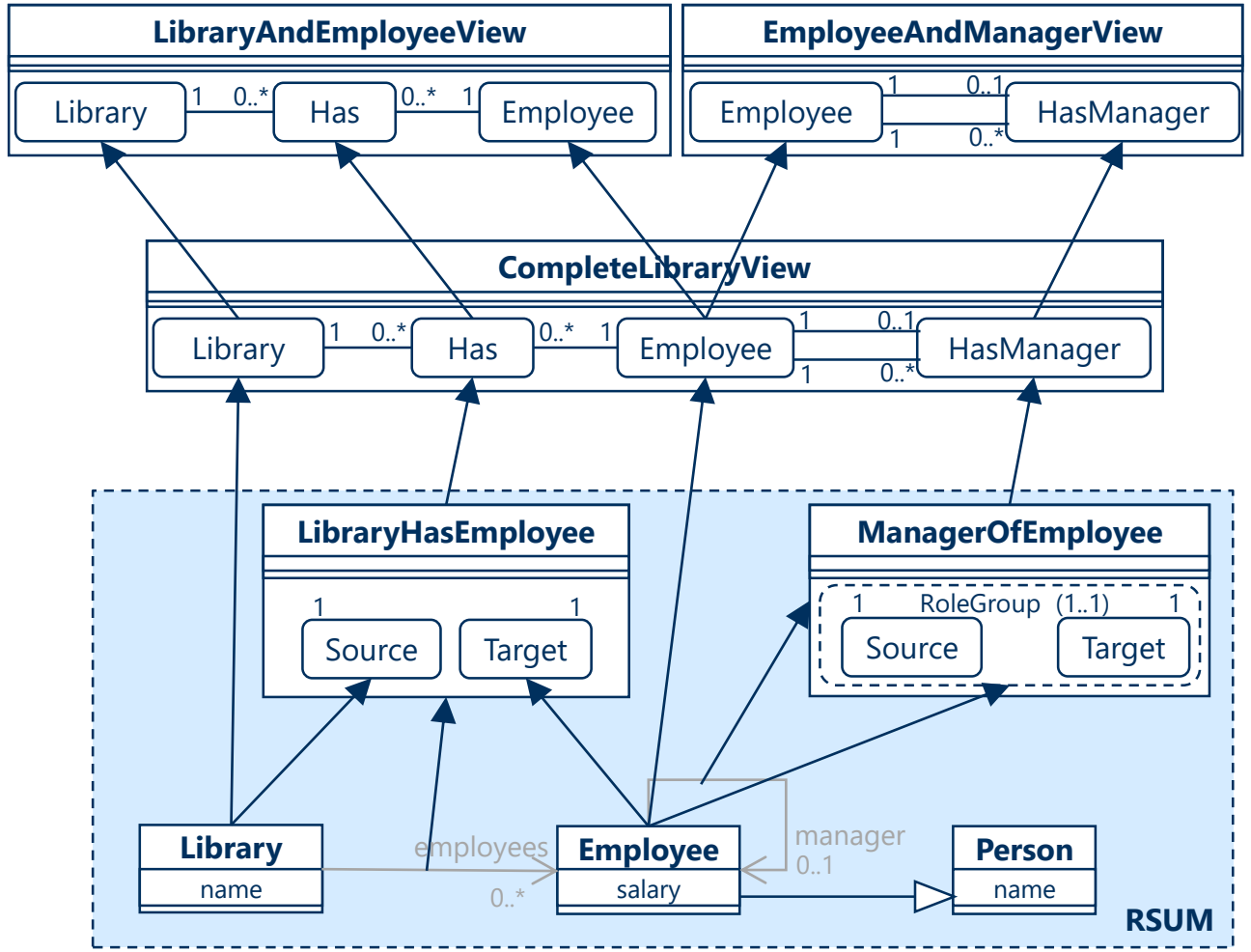
Use cases:

- History mechanism
  - Save changes over a period of time on one element
  - Allows logging on specific elements
- Versioning mechanism
  - Allows UNDO and REDO operations on single model elements



Deep Views

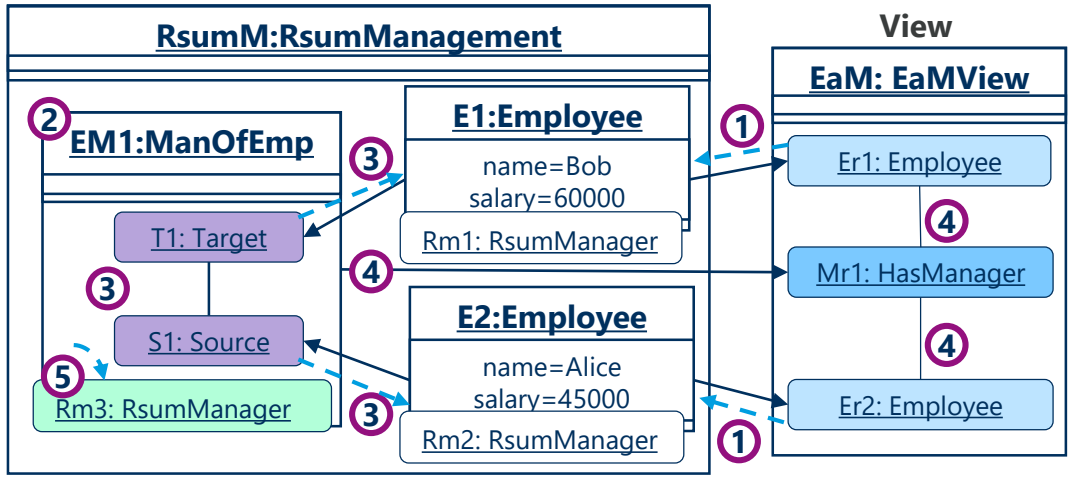
- Runtime models that depend on other runtime models
- Create a structure of deep views
- Destruction of *CompleteLibrary* view also destroys the two other views
- Create a structure for the views
- Insert a new abstraction layer



Creation Process

Process of adding a manager relation between two employee

1. Get the naturals of the employees from the RSUM
2. Create a new *ManagerOfEmployee* compartment in the RSUM
3. Bind the source and target roles to the used employees
4. Create a new *HasManager* role in the view and bind it to the new relational compartment
5. Add a new *RsumManager* role for the relational compartment in the *RSUMManagement*



Implementation with SCROLL

SCala ROLes Language (SCROLL) *[Leuthäuser2015]*:

- Role-based programming language
- Open source Scala library
- Implements most role features *[Kühn2014]*
- Flexible, lightweight, and easily extensible
- Compartments contain role graphs

Important operators:

- "Play" binds a role to a player
- "+" operator before a method call performs a dynamic dispatch to a suitable role played by the receiver

Scala allows class loading for runtime integration of new relational compartments, views, and extensions

```
class Library ( _name : String ) {
    private var name : String = _name
    def getName (): String = name
    def setName (n: String): Unit = {
        name = n
    }
}

class LibraryRole ( name : String ) extends IViewRole {
    def getNameView (): String = {
        return + this getName ()
    }
    def setNameView ( name : String ): Unit = {
        + this setName (name)
        + this changeTrigger ()
    }
}
```

Limitations of Implementation

- Currently hand written prototype
- Runtime models as compartments with roles as new objects
- Limitations from the use of relational compartments
- No dynamic loading of roles into compartments
- No domain specific language to describe the models as views from the RSUM
- Remove relations from the classes and use them as naturals
- Sometimes complex mechanism to combine all models in one RSUM

```
object RsumManagement extends MultiCompartment {
    protected var extensions   = ListBuffer [..]()
    protected var activeViews  = ListBuffer [..]()
    protected var allViews     = ListBuffer [..]()
    protected var allRelations = ListBuffer [..]()
    protected var allNaturals  = ListBuffer [..]()
    /* Insertion, creation, and deletion of views */
    class RsumManager () {
        def manageRsum (input : Object): Unit = {
            if ( input.isInstanceOf [IRelationCompartment])
                allRelations = allRelations :+ input
            else
                allNaturals = allNaturals :+ input
            input play roles in activeViews and extenstions
        }
    }
}
```

Related Work

MORSE Approach *[Holmes2009]*

- Model aware service environment consisting of a model repository and model-aware services
- Repository manages model projects and artifacts
- Using unique ids for all elements

SM@RT Tool *[Song2010]*

- Synchronization between running models and a MOF-compliant
- Synchronizations are triggered before and after write operations

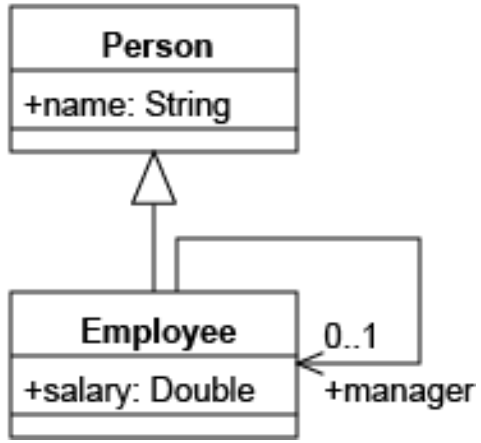
TGGs *[Vogel2010]*

- Synchronize and generate runtime models with triple graph grammars
- Incremental updates at runtime

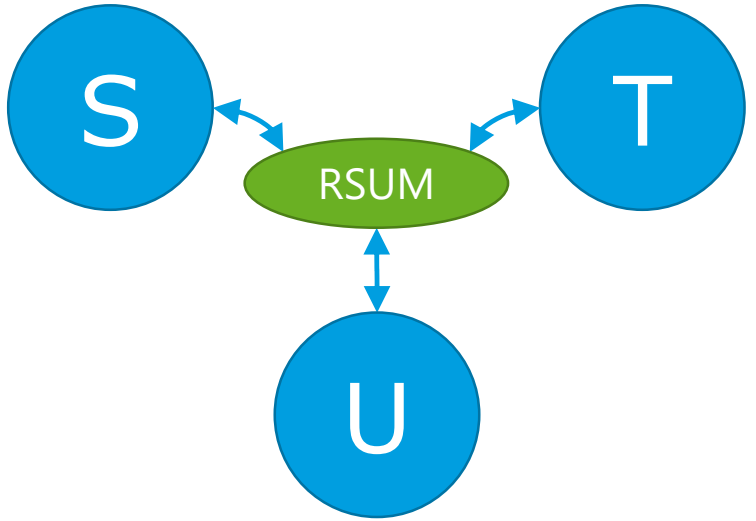
Comparison with View-based Approaches

	EMF Profiles *[Langer2012]*	mVTGG *[Anjorin2014]*	OSM *[Atkinson2010]*	OpenFlexo *[Golra2016]*	VIATRA viewers *[Depreceni 2014]*	RSUM
Bidirectional updates	●	●	●	●	●	●
Immediate updates	●	●	●	●	●	●
Incremental updates	●	●	●	●	●	●
Virtual views	○	○	●	●	●	●
Deep views	n.a.	●	○	○	●	●
No object schizophrenia	○	○	○	○	○	●

Conclusion & Future Work



- Advantages of roles as the foundation for a runtime model synchronization approach using a single underlying model
- Runtime models as views over a single underlying model that manages the information
- Feasibility of the role-oriented single underlying model approach by prototypically splitting a model into two views and synchronize them
- Implementation with SCROLL



- Extend the implementation to a framework
- Use a domain specific language to create the views form the RSUM in a simple way
- Extend the example and show the benefits on a bigger case study

References (1)

[Anjorin2014] A. Anjorin, S. Rose, F. Deckwerth, and A. Schürr. 2014. “Efficient Model Synchronization with View Triple Graph Grammars”. In *Modelling Foundations and Applications*. Springer International Publishing, Cham, 1–17.

[Atkinson2010] C. Atkinson, D. Stoll, and P. Bostan. 2010. Orthographic Software Modeling: A Practical Approach to View-Based Development. In *Communications in Computer and Information Science*. Springer Science & Business Media, 206–219.

[Debreceni2014] C. Debreceni, Á. Horváth, Á. Hegedüs, Z. Ujhelyi, I. Ráth, and D. Varró. 2014. “Query-driven Incremental Synchronization of View Models”. In *2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*. ACM.

[Golra2016] F. R. Golra, A. Beugnard, F. Dagnat, S. Guerin, and C. Guychard. 2016. “Addressing Modularity for Heterogeneous Multi-model Systems Using Model Federation”. In *Companion Proceedings of the 15th International Conference on Modularity*. ACM, 206–211.

[Holmes2009] T. Holmes, U. Zdun, and S. Dustdar. 2009. “MORSE: A Model-Aware service environment”. In *2009 IEEE Asia-Pacific Services Computing Conference (APSCC)*. 470–477.

[Kühn2014] T. Kühn, M. Leuthäuser, S. Götz, C. Seidl, and U. Aßmann, "A Metamodel Family for Role-Based Modeling and Programming Languages," Cham: Springer International Publishing, 2014, pp. 141–160.

References (2)

[Langer2012] P. Langer, K. Wieland, M. Wimmer, J. Cabot, et al. 2012. “EMF Profiles: A Lightweight Extension Approach for EMF Models”. *Journal of Object Technology* 11, 1 (2012), 1–29.

[Leuthäuser2015] M. Leuthäuser and U. Aßmann, "Enabling view-based programming with scroll: Using roles and dynamic dispatch for etablishing view-based programming," in Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering, New York, NY, USA: ACM, 2015, pp. 25–33.

[Song2010] H. Song, Y. Xiong, F. Chauvel, G. Huang, Z. Hu, and H. Mei. 2010. “Generating Synchronization Engines between Running Systems and Their Model-Based Views”. In *Models in Software Engineering*. Springer Berlin Heidelberg, 140–154.

[Vogel2010] T. Vogel, S. Neumann, S. Hildebrandt, H. Giese, and B. Becker. 2010. “Incremental Model Synchronization for Efficient Run-Time Monitoring”. In *Models in Software Engineering*. Springer Berlin Heidelberg, 124–139.