Component-Based Software Engineering	**Exercise Sheet No. 1**
Dipl.-Medieninf. Christian Piechnick	Software Technology Group
INF 2080	Institute for SMT
christian.piechnick@tu-dresden.de	Department of Computer Science
	Technische Universität Dresden
	01062 Dresden

Composition Systems and Metamodelling

Task 1: Composition Systems Basics

Composition systems simplify the development of large systems by focusing on loose coupling, separation of concerns, reuse, reducibility and standardization. This tasks repeats the terminology and the fundamentals of composition systems.

1a) **Task:** Clemens Szyperski provided one of the well-established definitions of a component [1]. How did he define a component? Try to summarize the key features of a component in your own words!

Solution: Definition: *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition.*

- Unit of composition (can be composed with other components)
- Contractually specified interfaces (explicit interface descriptions that act as an contract)
- Explicit context dependencies only (all dependencies are explicitly described)
- Can be deployed independently (components do not rely on concrete other components)
- Are subject to third-party composition (Can be reused others)

1b) **Task:** What are the three elements of composition systems? Try to explaint each part in your own words!

Solution:

Component Model: A component model describes the shape of components. It describes what components are, how they are structured, what features they have and how their ports look like.

Composition Technique: The composition technique describes how components can be composed and decomposed.

Composition Language: The composition language is a language for defining composition programs. It uses a component model and a composition technique.

1c) **Task:** Is the UNIX Pipes and Filters approach a composition system? Explain the three parts!

Solution: Yes. The components are filters with 3 standardized untyped ports (stdin, stdout, stderr). The connectors are pipes. The composition technique is to compose filters by connecting the stdout or stderr port with the stdin port of another filter. The composition language is a shell script.

1d) **Task:** Can LEGO (`http://www.lego.com/en-us/default.aspx`) be considered as composition system? Explain your conclusion.

Solution: Yes. The components are LEGO bricks with provided (the studs on top) and required ports (the holes on the bottom). They can be composed by plugging a stud of one brick in the holes of other bricks. The way LEGO describes its instructions (visually) can be considered a composition language.

Bibliography

1. Clemens Szyperski, *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley Longman Publishing Co., Inc., 2002 *The book is available in the SLUB.*

Task 2: Metamodelling Basics

Metamodelling is one central discipline in today's software engineering landscape. Especially for safety-critical systems, models are used to formally describe the structure and behaviuor of systems. Thus, certain properties can be proven. But not only for safety-critical systems, models are used. For example, one important engineering tool today are Domain Specific Languages (DSLs).
This tasks repeats the basic terminology and concepts of metamodelling.

2a) **Task:** Explain the terms *Model*, *Metamodel* and *Metametamodel*. What are the relations across those elements?

Solution:

Model: A model is an abstraction of some öriginal∴ Usually models refelect a part of the real or virtual world, with a certain degree of abstraction.
Metamodel: A metamodel describes types of model elements and their relations.
Metametamodel: A metametamodel describes the types of metamodels and their relations.

2b) **Task:** The *Meta Object Facility (MOF)* standard of the OMG emphazises 4 layers of metamodelling (M3 - M0). In the lecture we called this *Meta-Pyramid*. How are the terms of task `2a)` aligned with those layers?

Solution:

M3: Metametamodels
M2: Metamodels
M1: Models

2c) **Task:** Why is there no 5^{th} layer M4?

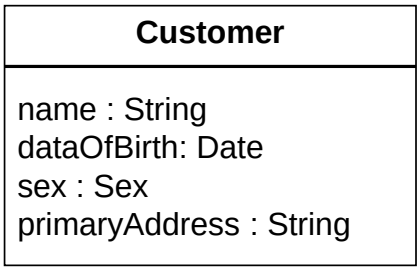
Solution: The metametamodel can be used to describe itself. Therefore, a layer M4 is not needed.

2d) **Task:** What is a Domain-Specific-Language (DSL)? Give an example.

Solution: A DSL is a language in Software Engineering specifically designed for a certain domain. A dsl uses concepts and keywords from this domain, and thus, can be used by domain experts which are non-programmers.

An example could be a language for describing business workflows within a company:

Metamodel:



Language Syntax Specification:

```
Customer := 'customer' name=String '{'
        'data of birth' ':' dateOfBirth = DateString
        'sex' ':' sex = Sex
        'primary address' ':' primaryAddress = String
'}'

Sex := 'female' | 'male'
```

Example Instance:

```
customer John Doe{
        date of birth: 01.01.1980
        sex: male
        primary address: Sample Street 1, 12345 Samplecity
}
```

2e) **Task:** How can your example be aligned with the Meta-Pyramid?

Solution:

Metamodel: M2
Language Syntax Specification: M2
Example Instance: M1

2f) **Task:** Explain the terms *reflection*, *introspection* and *meta-object protocol (MOP)*.

Solution:

Reflection is the ability of a model to reason about and change its own metamodel.
Introspection is Read-Only reflection.
MOP is the implementation of a language semantics, using the language itself.

2g) **Task:** What happens when the MOP is changed?

Solution: The semantics of the language is changed.

Task 3: Metamodelling in Practice

We are going to design a composition system for linear board games. The components are fields on the board, which have a color (red, white or black) and a shape (circle or rectangle). A board can be composed of fields by connecting the fields with a directed path. Each field has exactly one predecessor field and exactly one successor field. There are two special kinds of fields: a *start field*, which has no predecessor and an *end field*, which has no successor. The board must have exactly one start and exactly one end field. There must be a path from the start to the end field. Furthermore the board game can have at least two and at most 4 playing pieces. Each playing piece is assigned to exactly one field. At beginning of the game every playing piece is set to the start field.

1. Download the latest version of the Eclipse Modeling Tools (`http://eclipse.org/downloads/`)
2. Install the latest version of xText (`http://eclipse.org/Xtext/`)
3. Install the latest version of Xpand (`https://eclipse.org/modeling/m2t/?project=xpand`)

3a) **Task:** Create an EMF-Metamodel (Ecore Model) for the board game structure!

Solution: The solution can be downloaded on the website.

3b) **Task:** Create a DSL using xText that let you design board games!

Solution: The solution can be downloaded on the website.

3c) **Task:** Use the xText validation engine to validate at least 3 rules that must be enforced in the static semantics of the language! (Look at the xText help section for further information)

Solution: The solution can be downloaded on the website.

3d) **Task:** Use Xpand to generate HTML5 markup that visualizes your board game!

Solution: The solution can be downloaded on the website.