Customisation of Relationships between Types

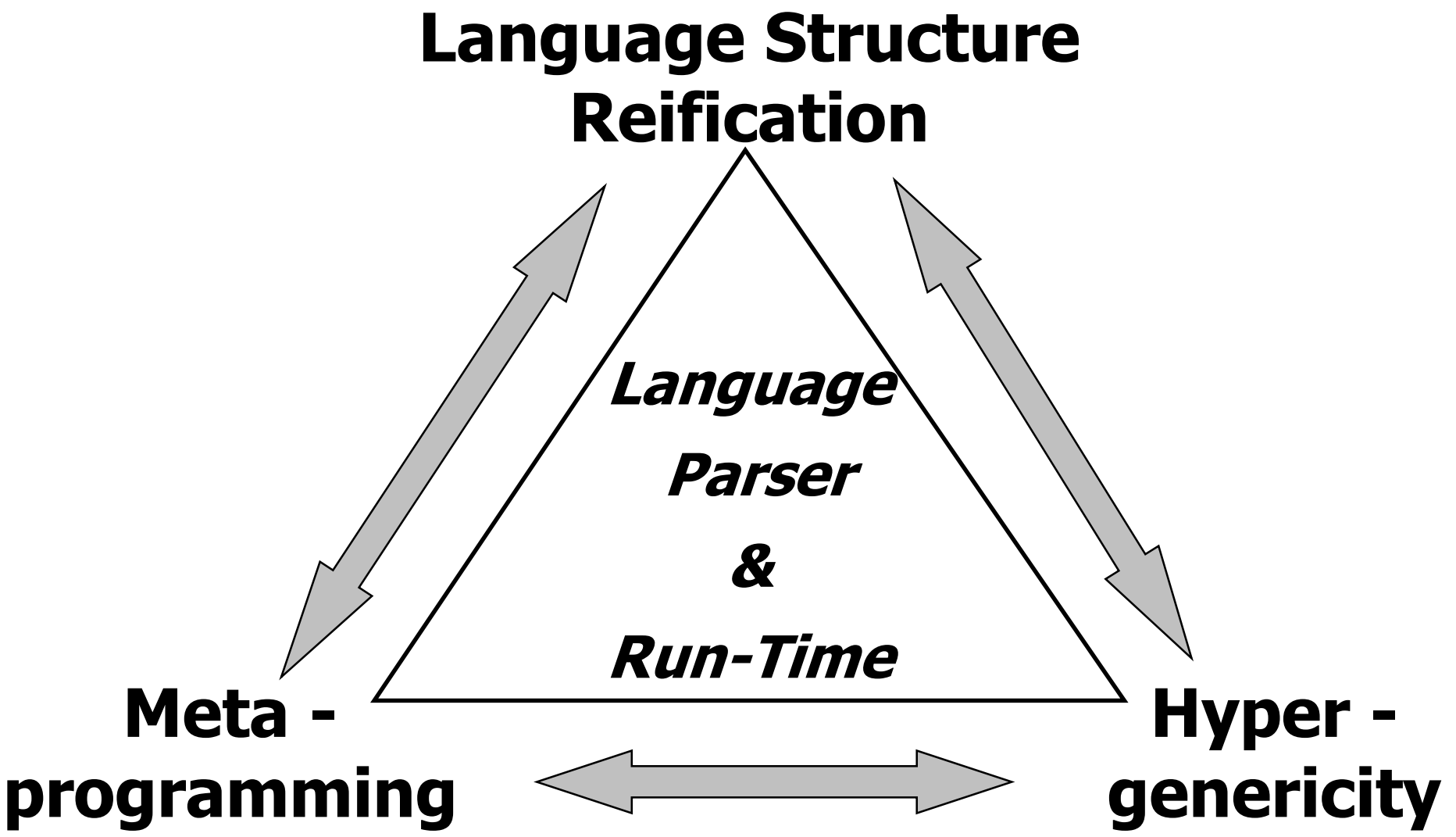
A. Capouillez, R. Chignoli, P. Crescenzo and Ph. Lahire

http://www.i3s.unice.fr/~ocl/

ECOOP 2000 - June 12-16, 2000

Posters Session

OFL Overview



OFL Overview

We intend to use meta-programming technology and facilities provided by hypergenericity in order to build an open language which allows people to better define the role of the relations between the application components.

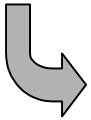
The concepts of *description*, *relationship* and *language* will be reified in order to:

- Allow a parser to get all necessary information for the generation of the appropriate application object-code.
- Ensure that the application run-time will be able to retrieve what is needed for carrying out the execution according to the semantics associated to reified concepts.

Some actions associated to the semantics of relationships will be performed at compile time whereas some will be handled at run-time.

OFL: *Open Flexible Language*

- Framework
  - ✓ Libraries of components for real applications
  - ✓ Persistent objects
- Objectives
  - ✓ A better handling of persistency (evolution, access, etc.)
  - ✓ A better sharing of objects (application, language, etc.)
- Means: Relationship and class customisation



OFL Model:

- Concept-relationship
- Concept-description
- Concept-language

OFL: *Open Flexible Language*

We think that it is important to provide to programmers better tools for the design of library of components and the integration of those libraries into an application (which can address persistent occurrences of the software components).

We propose a model based on three concepts (language, description and relationship) that may be customised in order to define a more accurate semantics for the relationships between classes including their co-ordination.

One of the objectives of **OFL model** is to provide means for a better handling of the evolution of persistent objects (version, view, etc.).

OFL Model: Fundamentals

- Hypergenericity
  - ✓ Specific and general parameters
  - ✓ Actions and controls
  - ✓ Assertions
- Meta-programming
  - ✓ New parameters
  - ✓ Modification of action and control semantics

90% hypergenericity / 10% meta-programming

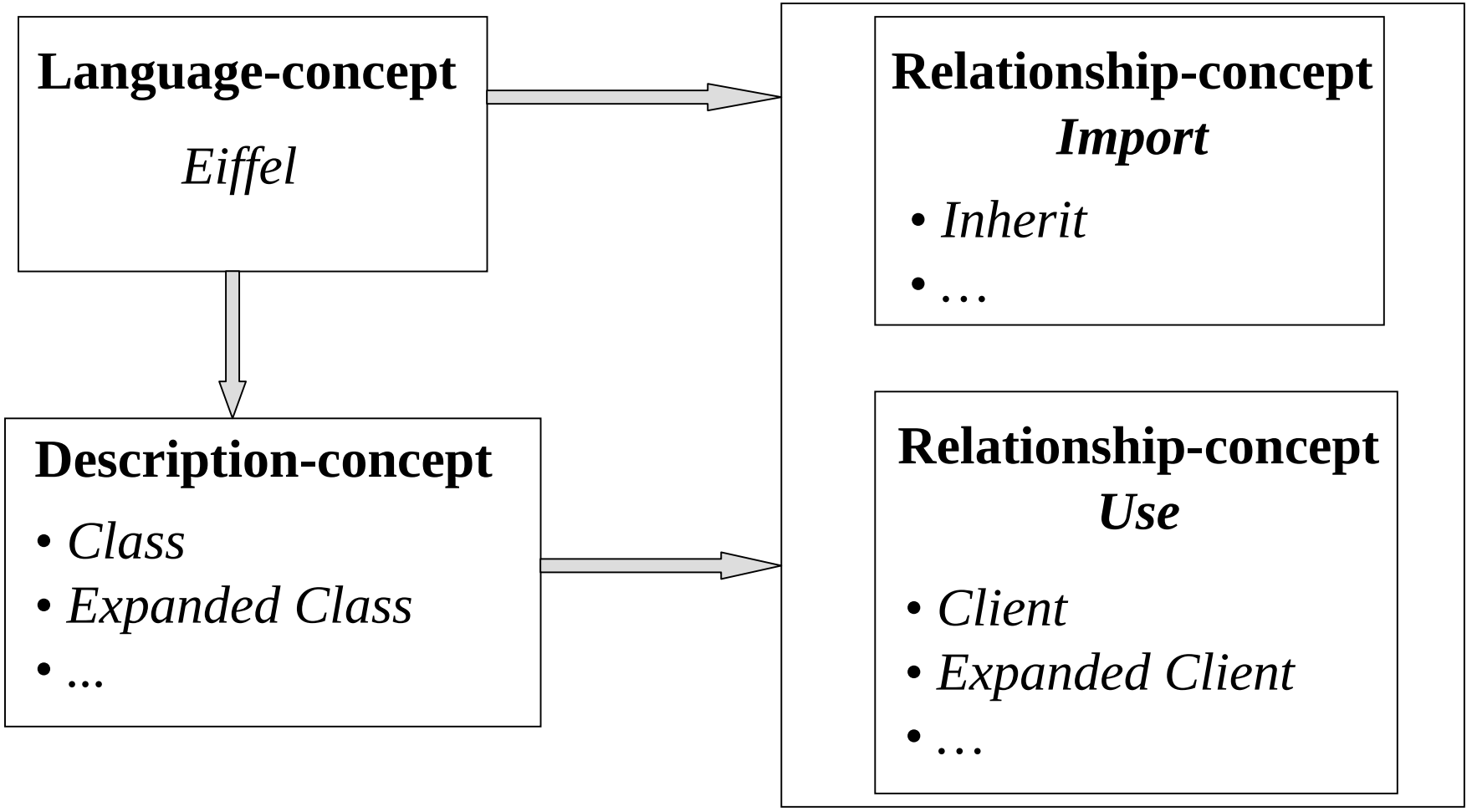
OFL Model: Fundamentals

In OFL we identified a set of actions, controls and assertions that may be found in a parser or a virtual machine depending on the static or dynamic point of view.

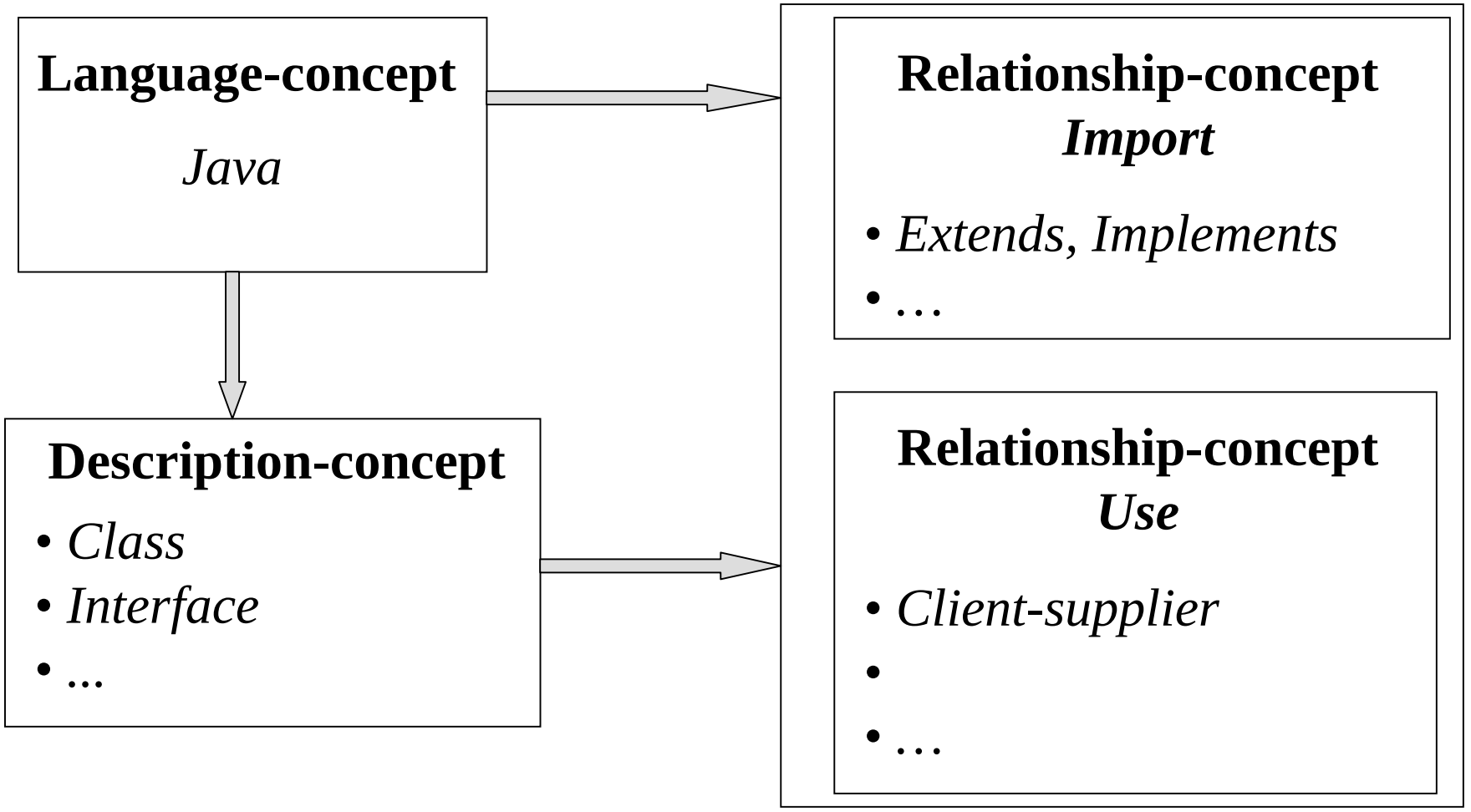
Those actions, controls and assertions are driven by the values associated to sets of general or specific parameters

In most case, the modification of parameter values is the only action to be performed by the meta-programmer

OFL: Approach by Example - Eiffel Language



OFL: Approach by Example - Java Language



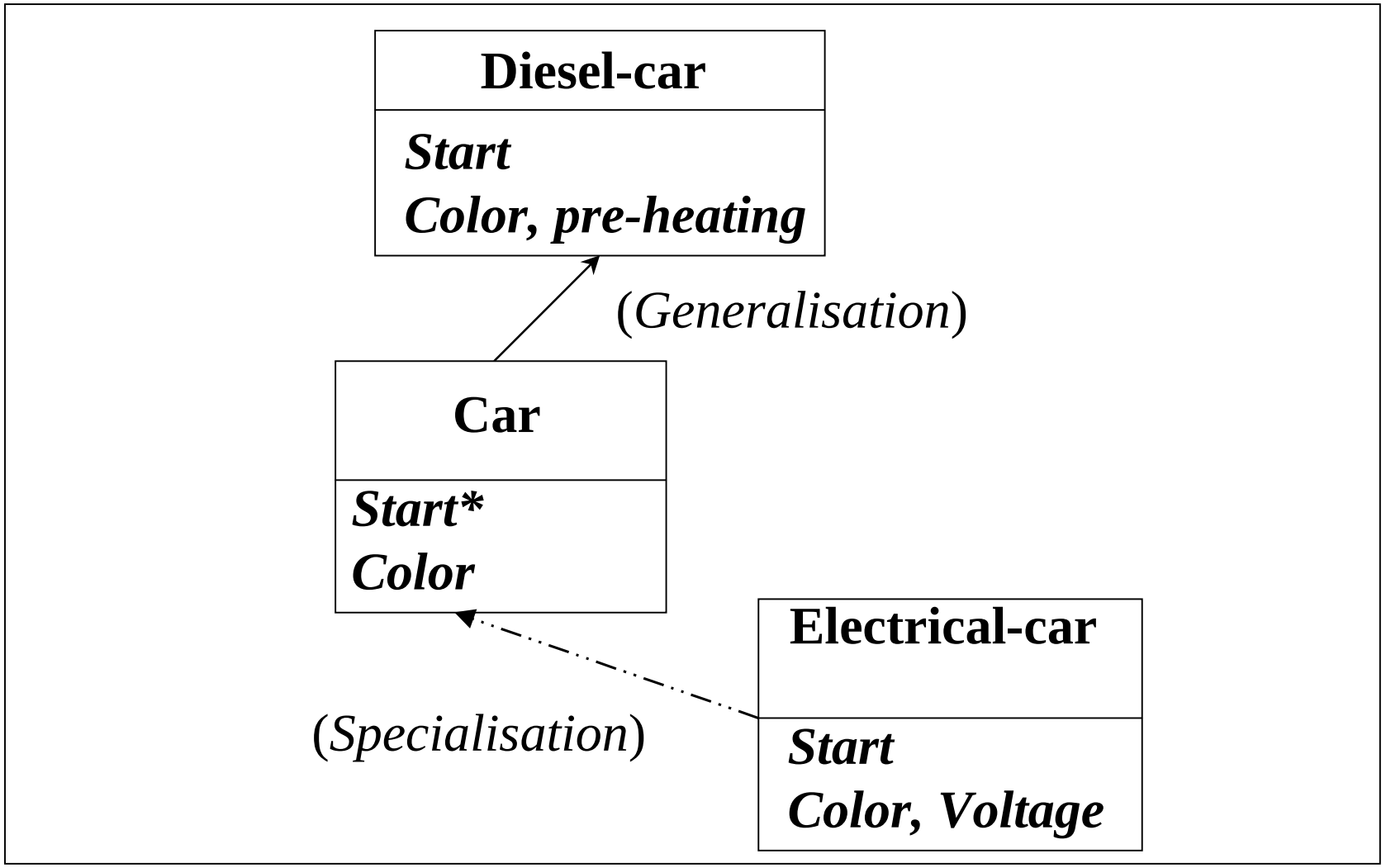
OFL: Approach by Example

We intend to describe the operational semantics of a language like Eiffel or Java. It means to define and store the parameters of a description-concept for each kind of class and the parameters of a relation-concept for each kind of relationships.

We distinguish three kinds of relationship: the **import relationships** (for example *inheritance* in Java), the **use relationships** (for instance the *expanded client-supplier* relationship in Eiffel), the **object-to-class relationships** (like *instanciation* in Eiffel or Java).

Each of these concepts may be defined independently from a language and then put in a library of concepts. The language-concept allows meta-programmer to adapt the use of the concept in order to fit into language semantics requirements

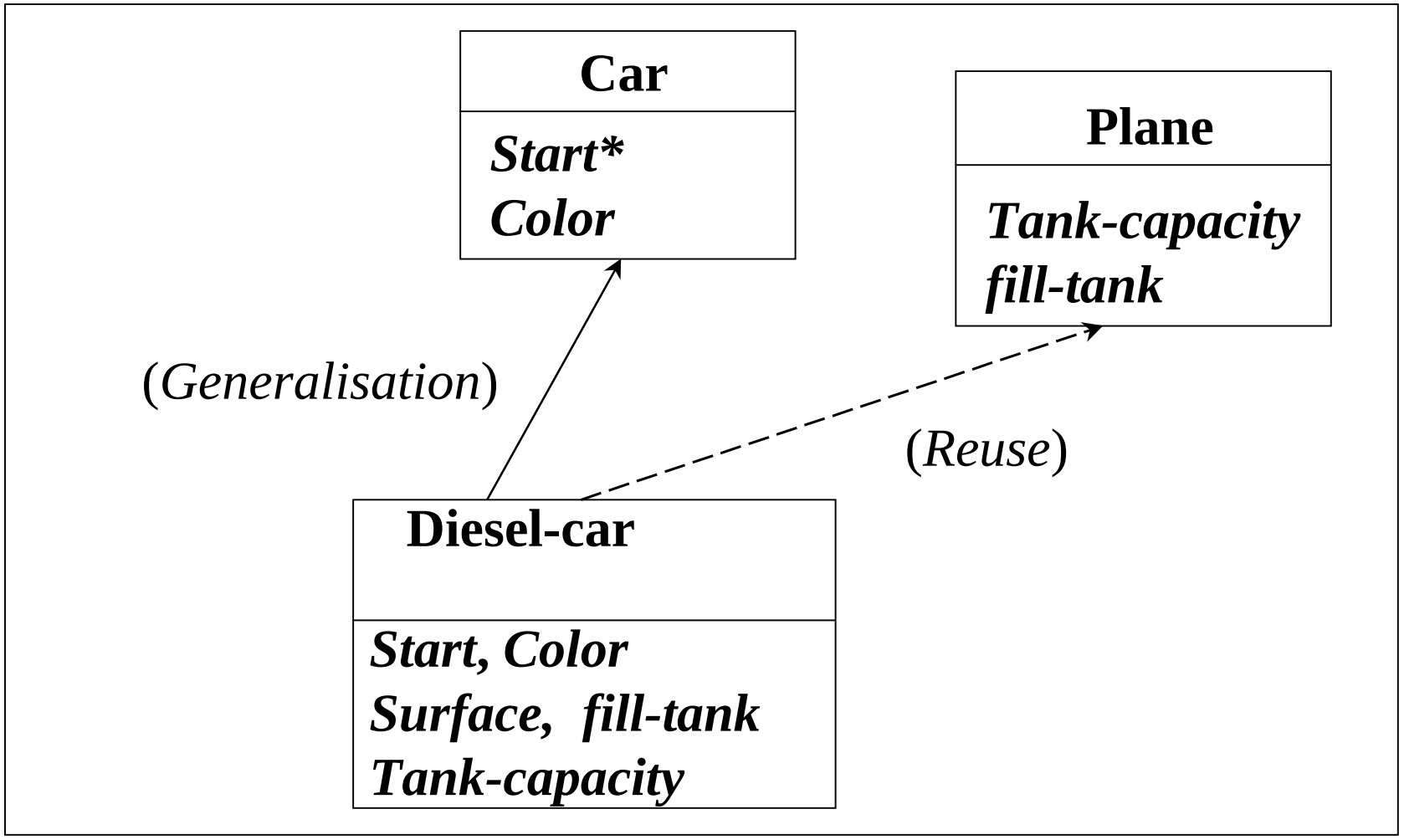
Example: Generalisation (Import Relationship)



Example: Generalisation (Import Relationship)

General parameters	**Specific parameters**
• Name: *generalisation* • Kind: *inter-type-import* • Cardinality: *1-∞* • Circularity: *False* • Repetition: *true* • Symmetry: *false* • Opposite: *none*	*None*
	Assertions *Covariance for generalisation relationship*

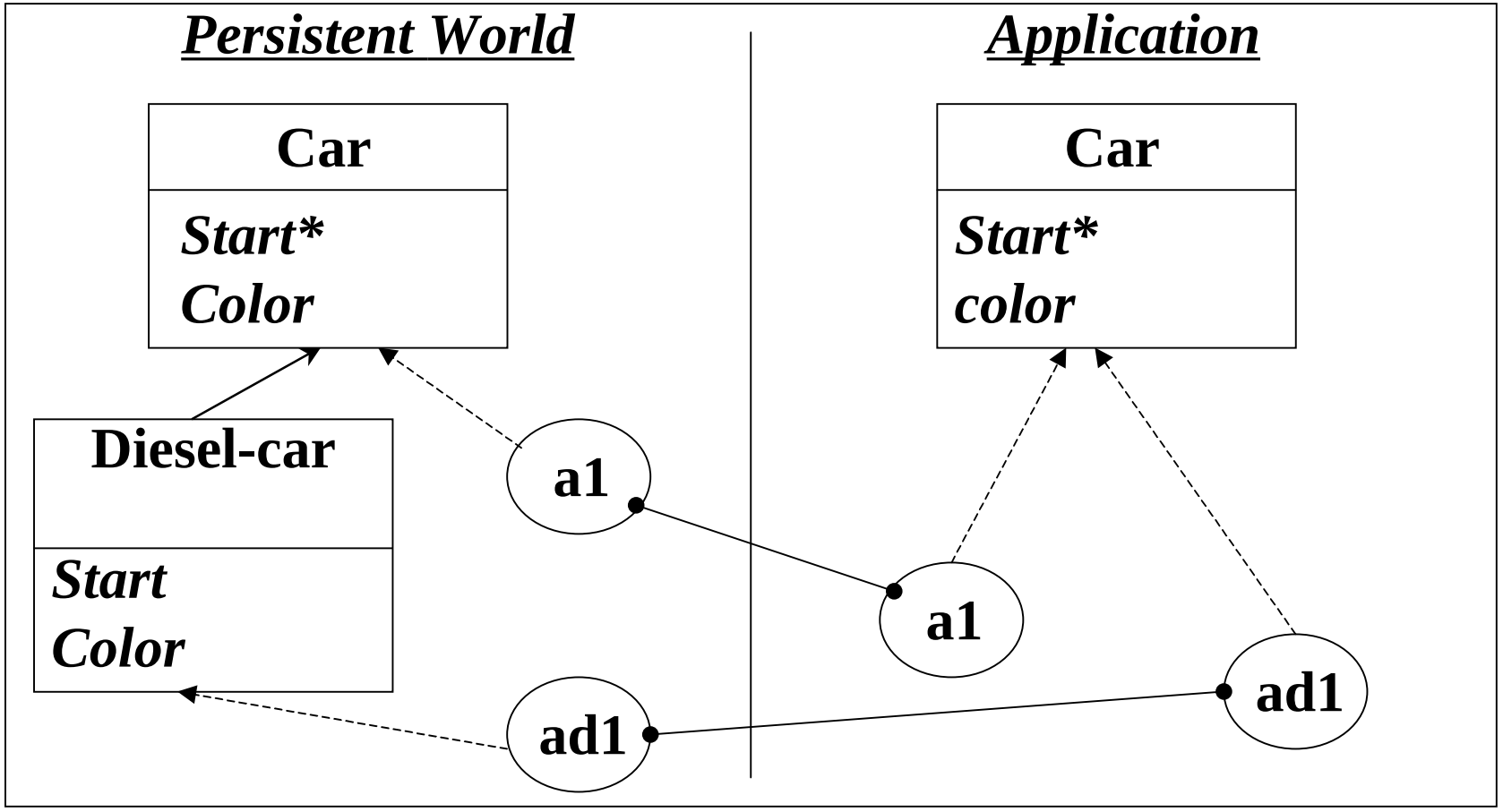
Example: Reuse (Import Relationship)



Example: Reuse (Import Relation)

General parameters	**Specific parameters**
• Name: *reuse* • Kind: *inter-type-import* • Cardinality: *1-∞* • Circularity: *true* • Repetition: *true* • Symmetry: *false* • Opposite: *none*	*None*
	Assertions *No feature redefinition*

Example: Aggregation (Use) / Persistency



Example: Aggregation (Use) / Persistency

General parameters	**Specific parameters**
• Name: *aggregation* • Kind: *inter-type-use* • Cardinality: *1-∞* • Circularity: *true* • Repetition: *true* • Symmetry: *false* • Opposite: *none*	• Migration-control • Loading-control • Transaction-start • Adaptor-policy • etc.
	Assertions *Control of the mechanism for contagious persistency*

Example: Aggregation (Use) / Persistency

We describe **use relationships** in the framework of applications which access to persistent instances of classes belonging to libraries of highly reusable components. It means that it is necessary:

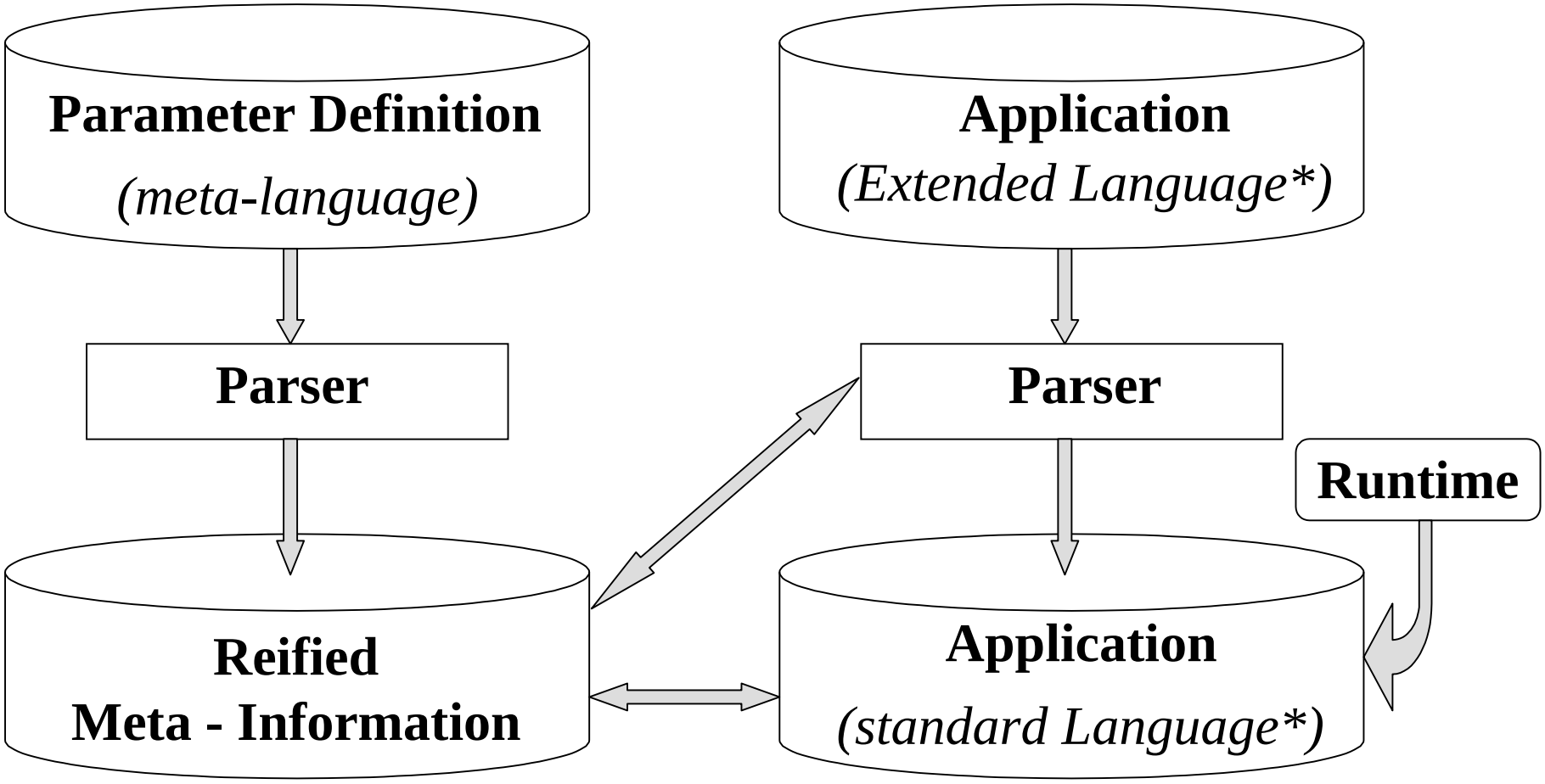
Once (for this kind of application):

- To define a set of parameters which is **specific** to persistency and
- To adapt algorithms associated to actions and controls in order to integrate the influence of new parameters into the behaviour of actions and controls.

For each new type of relationship (same kind of application)

- To set the parameter (general / specific) values for the relationships
- To define the assertions which constraints the relationship

A Possible Implementation



(*) *IDL*, etc.

A Possible Implementation

To design an open language starting from IDL (OMG) which integrates the ability to define (meta-programmer) and use (programmer) new types of relationship, description and language.

To build a parser which:

- Uses meta-information associated to OFL-concepts in order to take into account of actions and controls that may be checked statically.
- Generates standard IDL code and some reified information for the main structures of the language (*class*, *routine*, *etc.*).
- Links a specific run-time for actions and controls that should be performed when the application is running.