Aranea
Message Transport

Using Language-Independent JSON Format

July 2009

B. Herlig, bherlig@student.ethz.ch

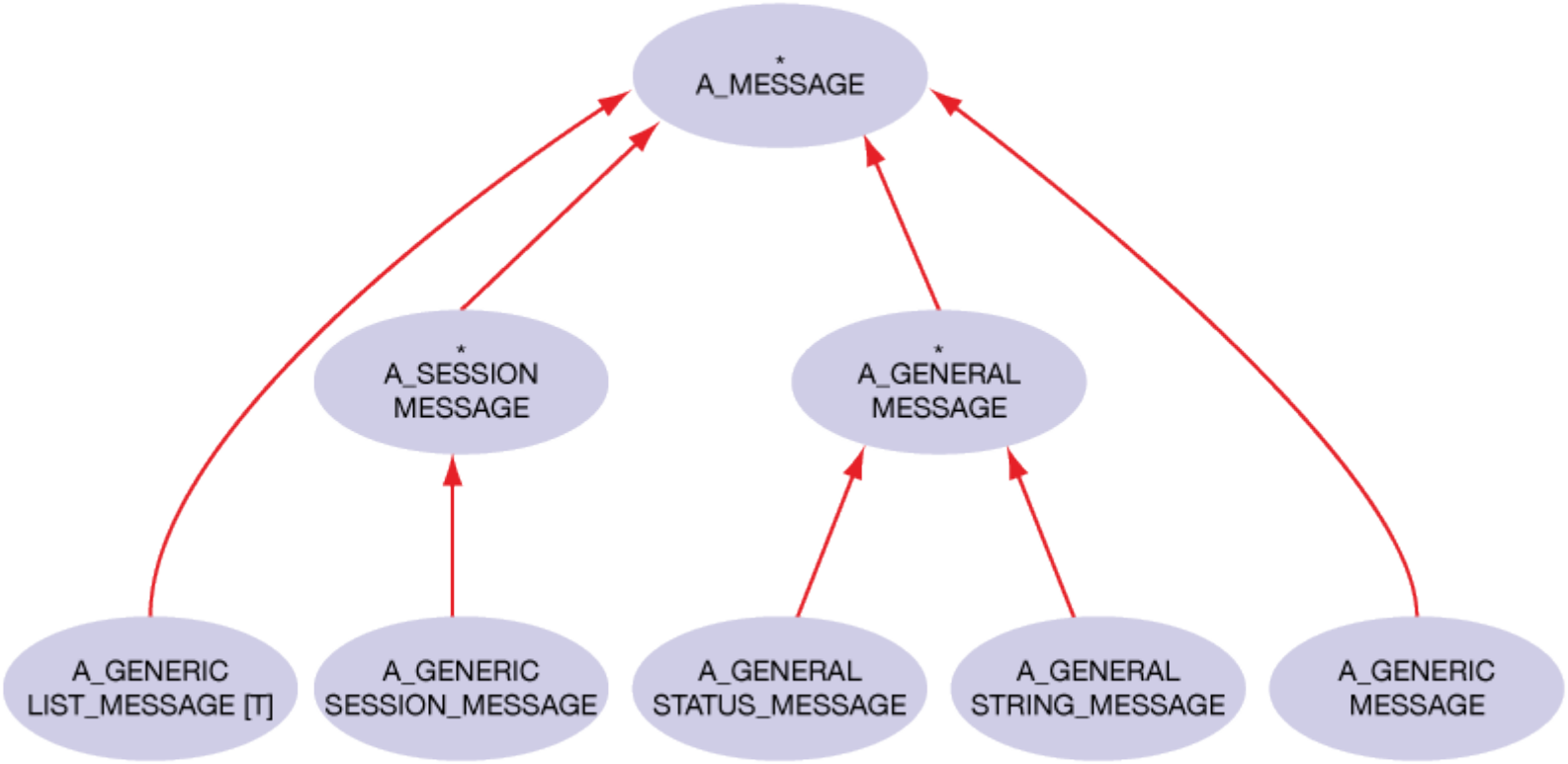
Overview

- Status Quo
- New Message Library
- Message Generation
- Migration

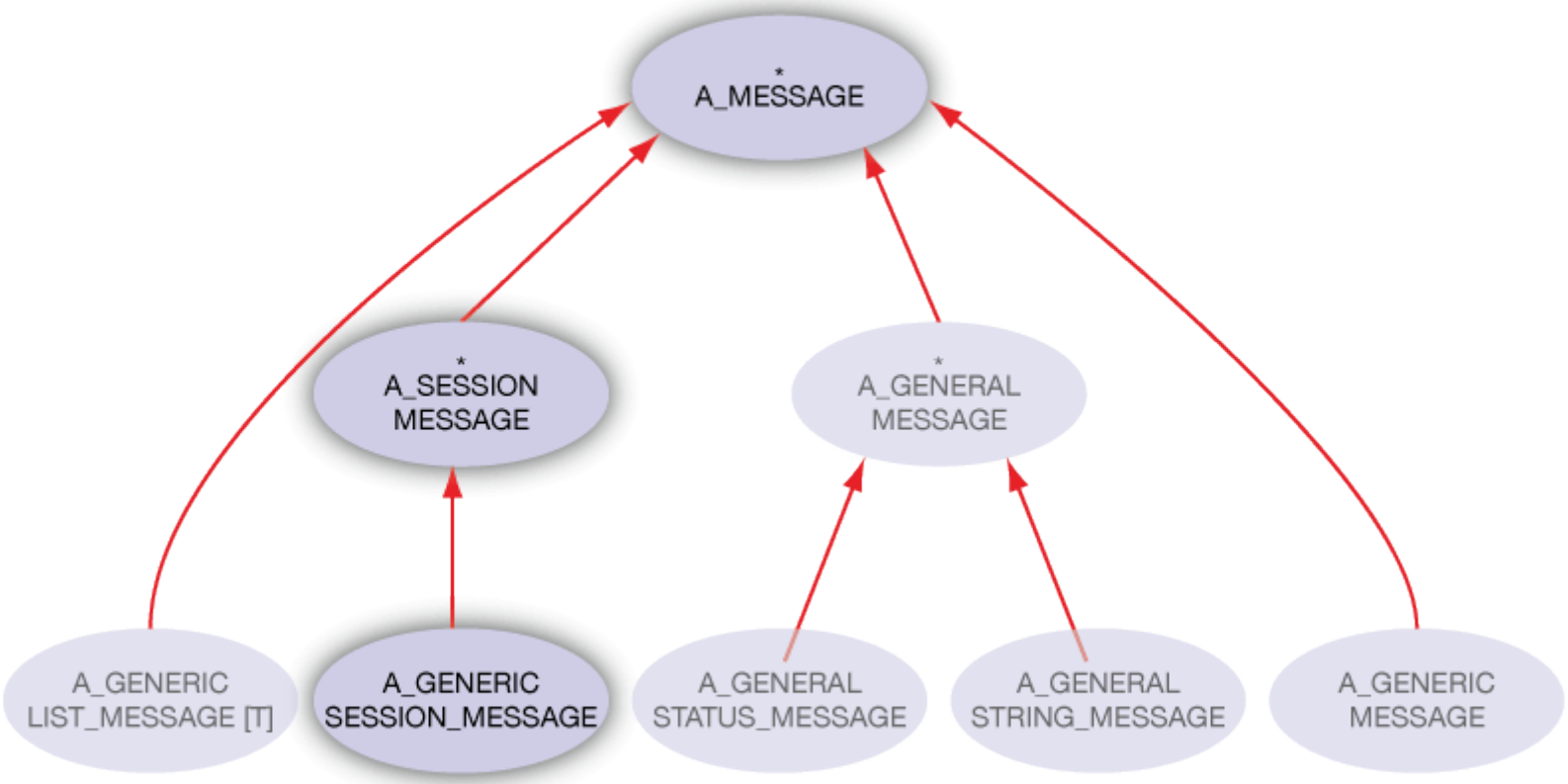
Status Quo

- «It's oh so Eiffel»
- Exclusively Eiffel Nodes
- Serialization:
  - Base Types: `STRING`, `INTEGER`, ...
  - Complex Types:
    - `HASH_TABLE [ANY, STRING]`
    - → Eiffel Serialization to bytestream
    - → Transport as ApacheMQ ByteMessage

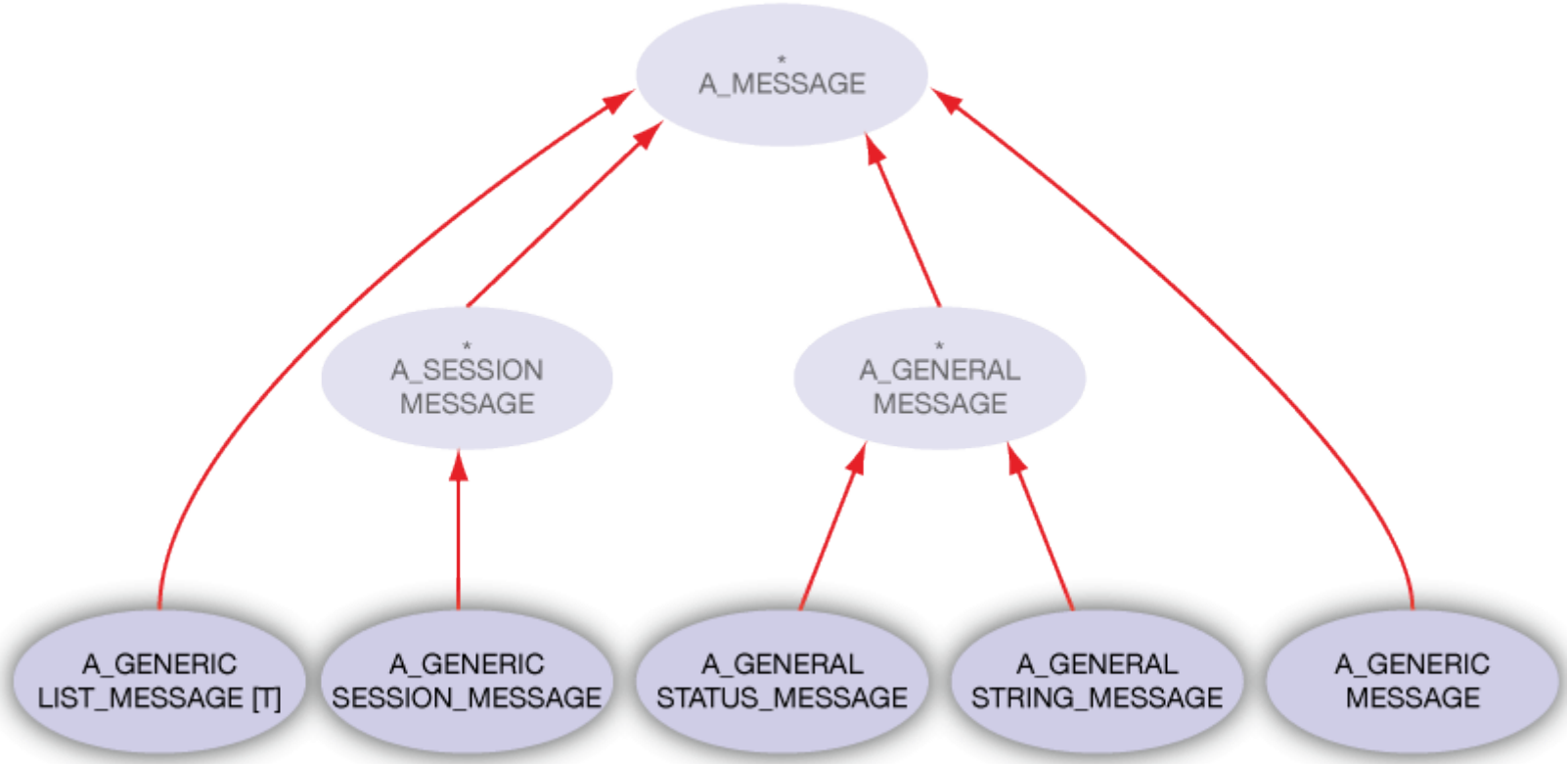
Status Quo


A_MESSAGE
A_SESSION MESSAGE
A_GENERAL MESSAGE
A_GENERIC LIST_MESSAGE [T]
A_GENERIC SESSION_MESSAGE
A_GENERAL STATUS_MESSAGE
A_GENERAL STRING_MESSAGE
A_GENERIC MESSAGE

Status Quo


A_MESSAGE
A_SESSION MESSAGE
A_GENERAL MESSAGE
A_GENERIC LIST_MESSAGE [T]
A_GENERIC SESSION_MESSAGE
A_GENERAL STATUS_MESSAGE
A_GENERAL STRING_MESSAGE
A_GENERIC MESSAGE

Status Quo


A_MESSAGE
A_SESSION MESSAGE
A_GENERAL MESSAGE
A_GENERIC LIST_MESSAGE [T]
A_GENERIC SESSION_MESSAGE
A_GENERAL STATUS_MESSAGE
A_GENERAL STRING_MESSAGE
A_GENERIC MESSAGE

Status Quo

- Why change it?

- New Nodes without existing Eiffel Libraries?
- Faster native implementations in other languages?
- Flexibility -
  Use any Language, on any Platform!!

New Message Library

New Message Library

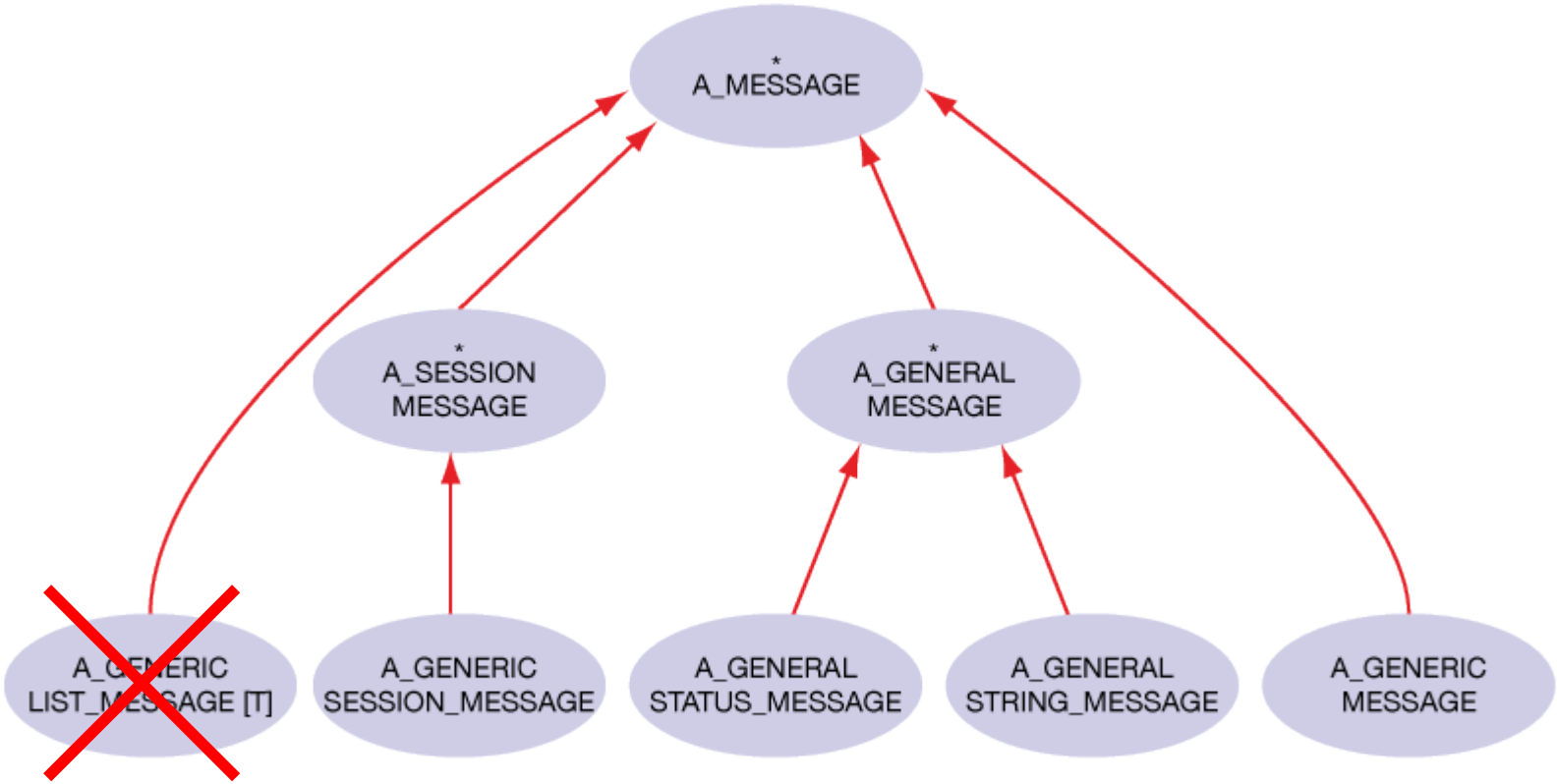
- The Key: Serialization
- XML, YAML, Thrift, Protobuf, JSON
- Speed vs. Message Size
- Cross-Language
- Spread / Acceptance
- Ease of Use
- Eiffel Binding: Workflow & Code Generation
- Documentation

➔ The Winner: JSON

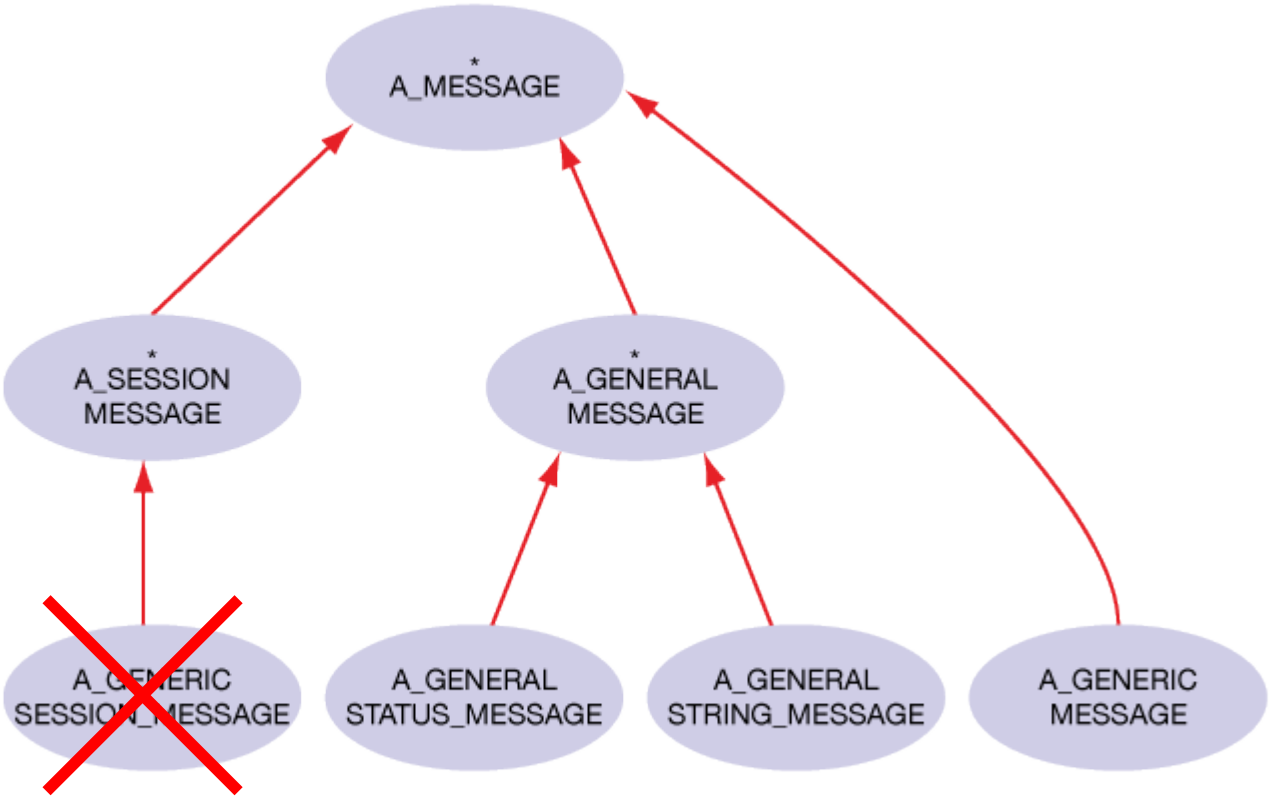
New Message Library

- De-/ Serialization of all possible Values
  - Transparent for Aranea's client-applications
- Code generatable (Origo: 170 Messages)
- Strong Typed
- Nice Code :)

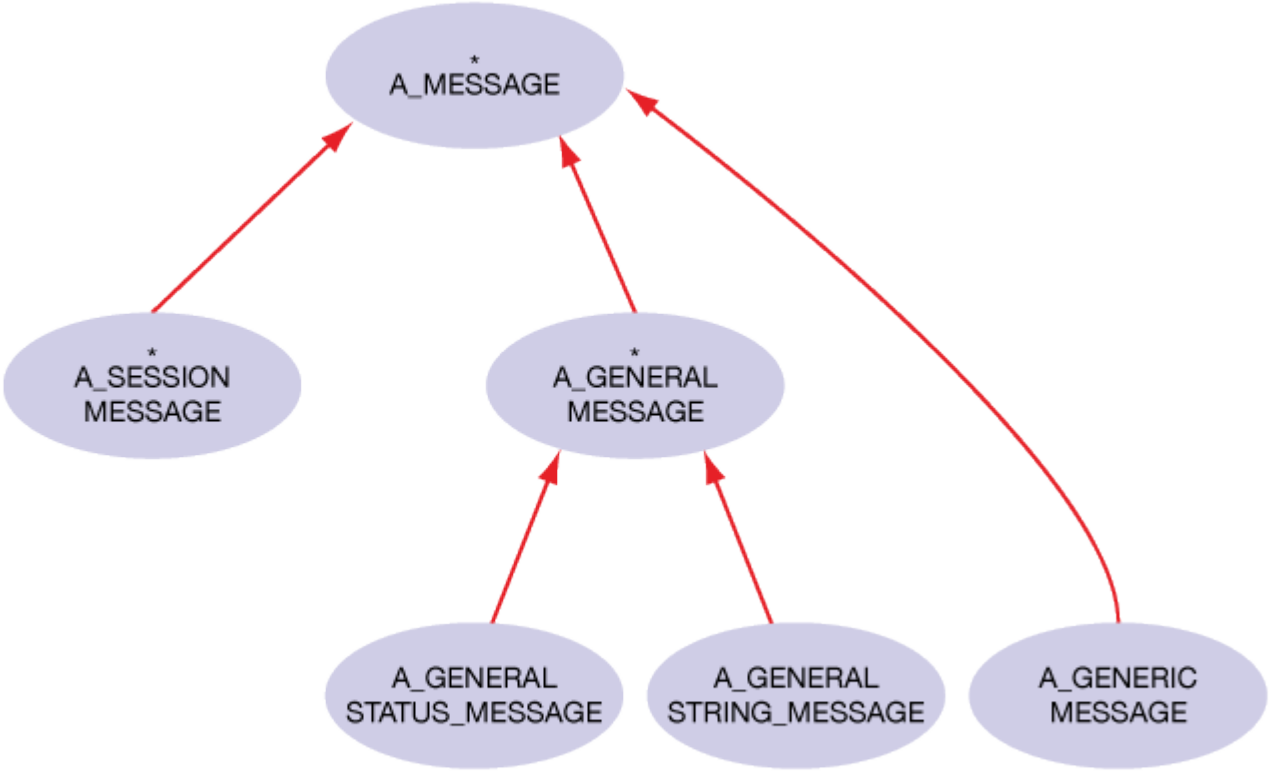
New Message Library



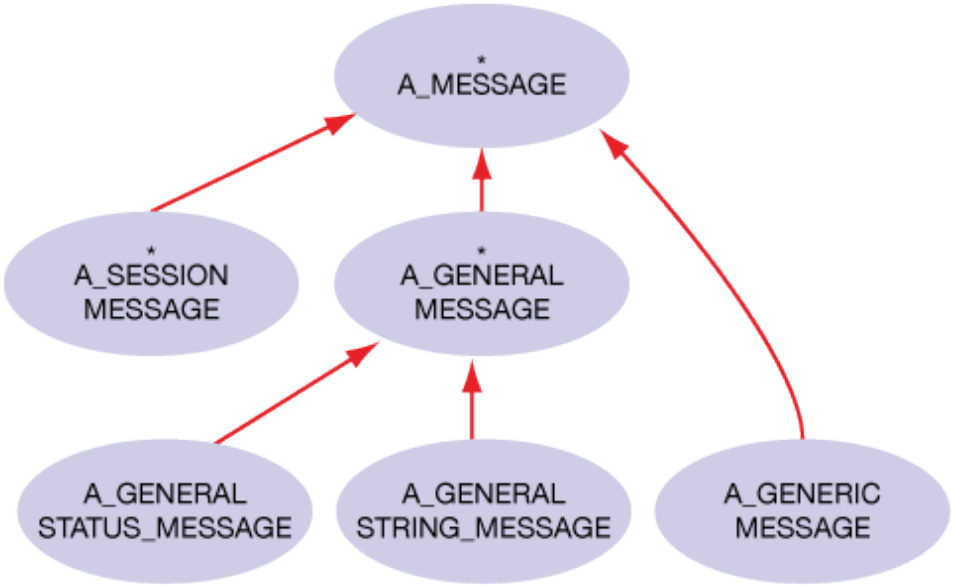
New Message Library



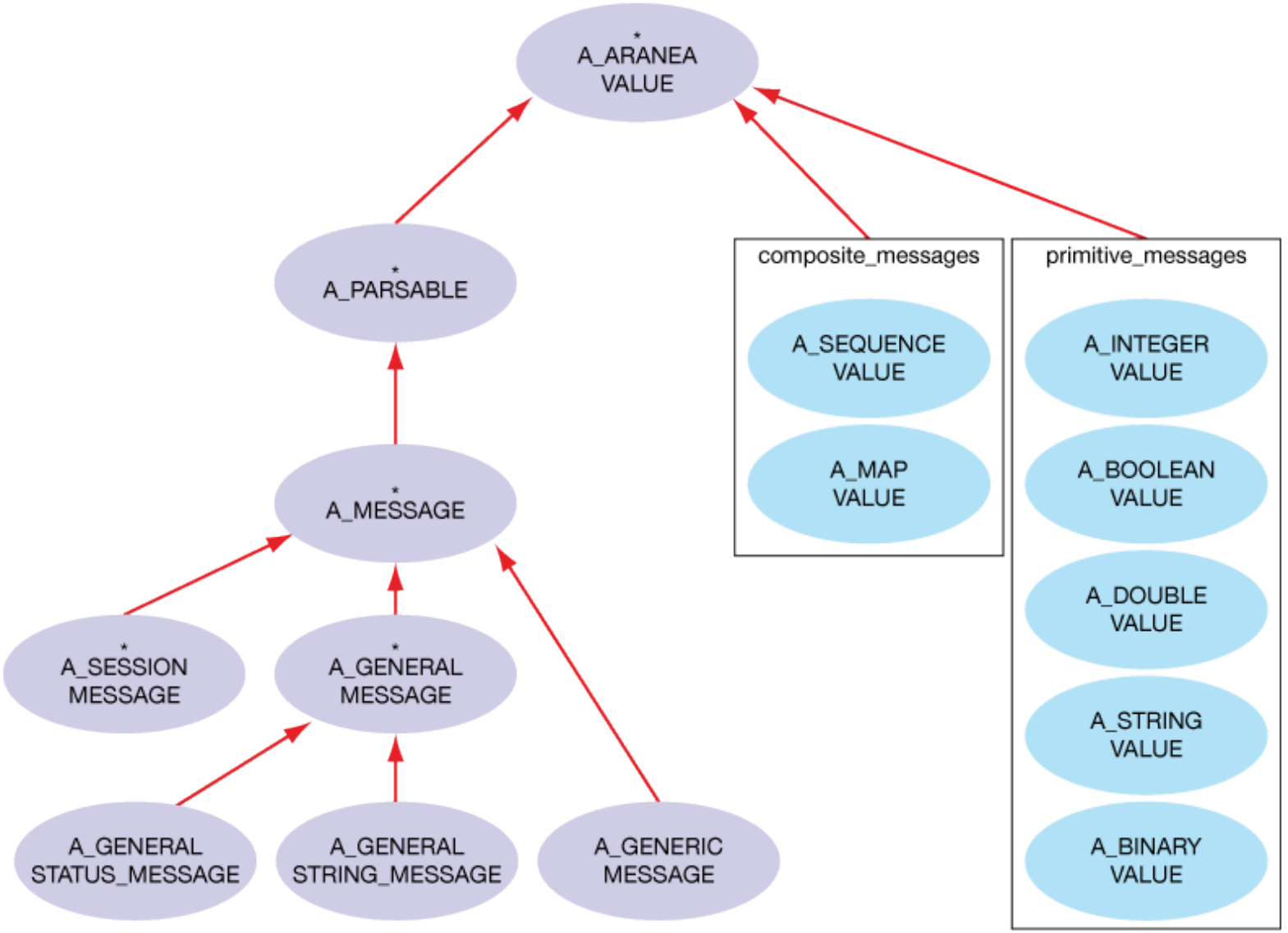
New Message Library



New Message Library



New Message Library



New Message Library

(Informal) Grammar:

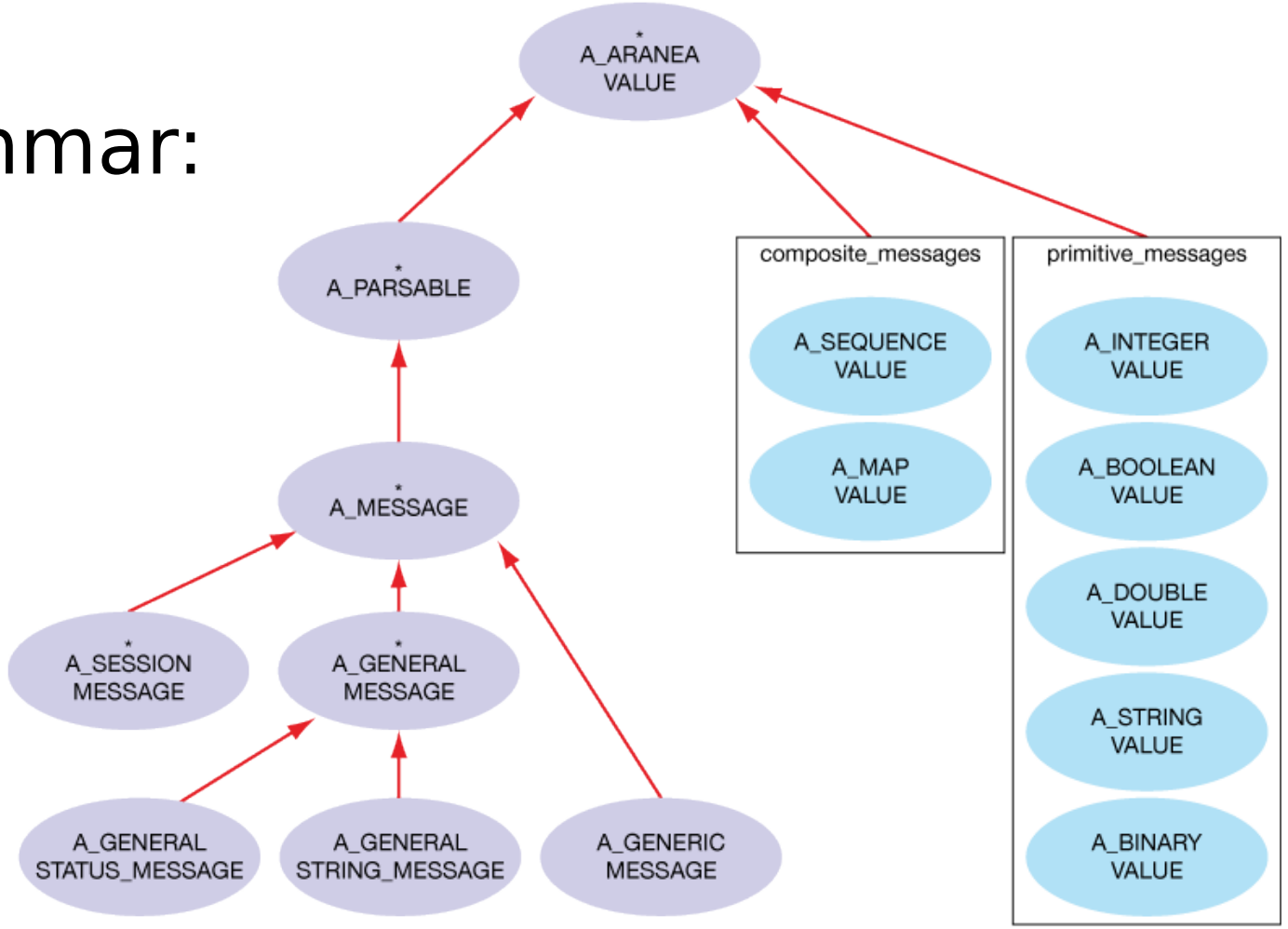
```
Message: (Field)*

Field -> I : T
I -> identifier

T -> integer
  -> double
  -> boolean
  -> string
  -> binary

  -> map [T]
  -> sequence [T]

  -> record [ I1 : T1, I2 : T2, … ]
```



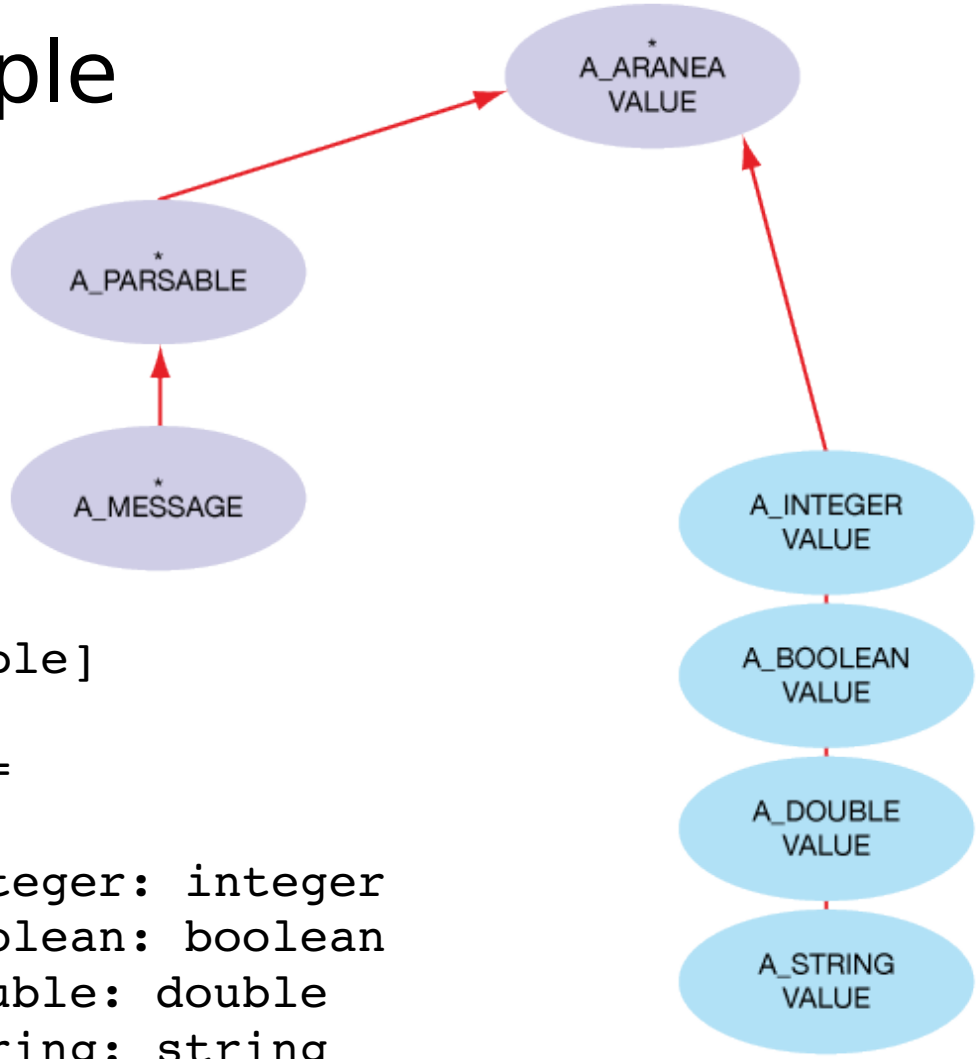
New Message Library

- Example

```
[message Simple]
{
  properties=
  {
    field_integer: integer
    field_boolean: boolean
    field_double: double
    field_string: string
  }
}
```

New Message Library

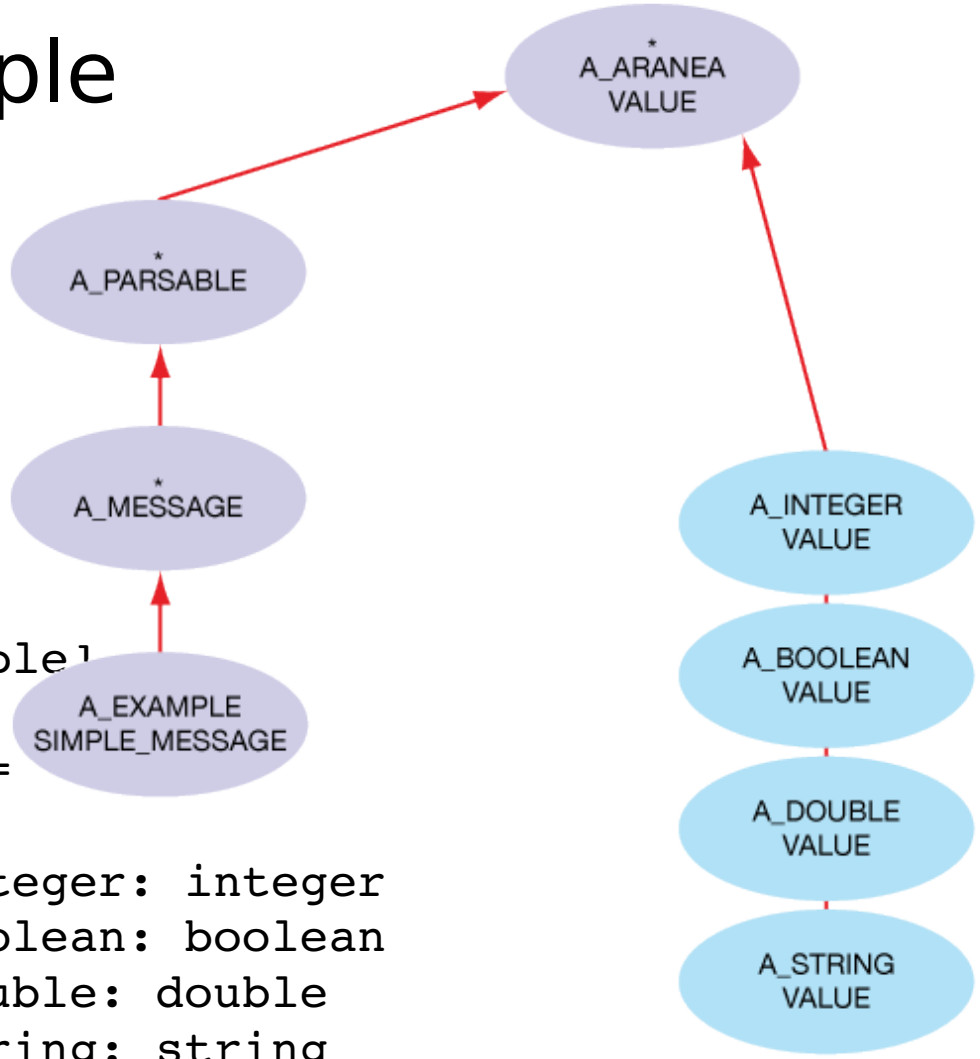
- Example




```
[message Simple]
{
  properties=
  {
    field_integer: integer
    field_boolean: boolean
    field_double: double
    field_string: string
  }
}
```

New Message Library

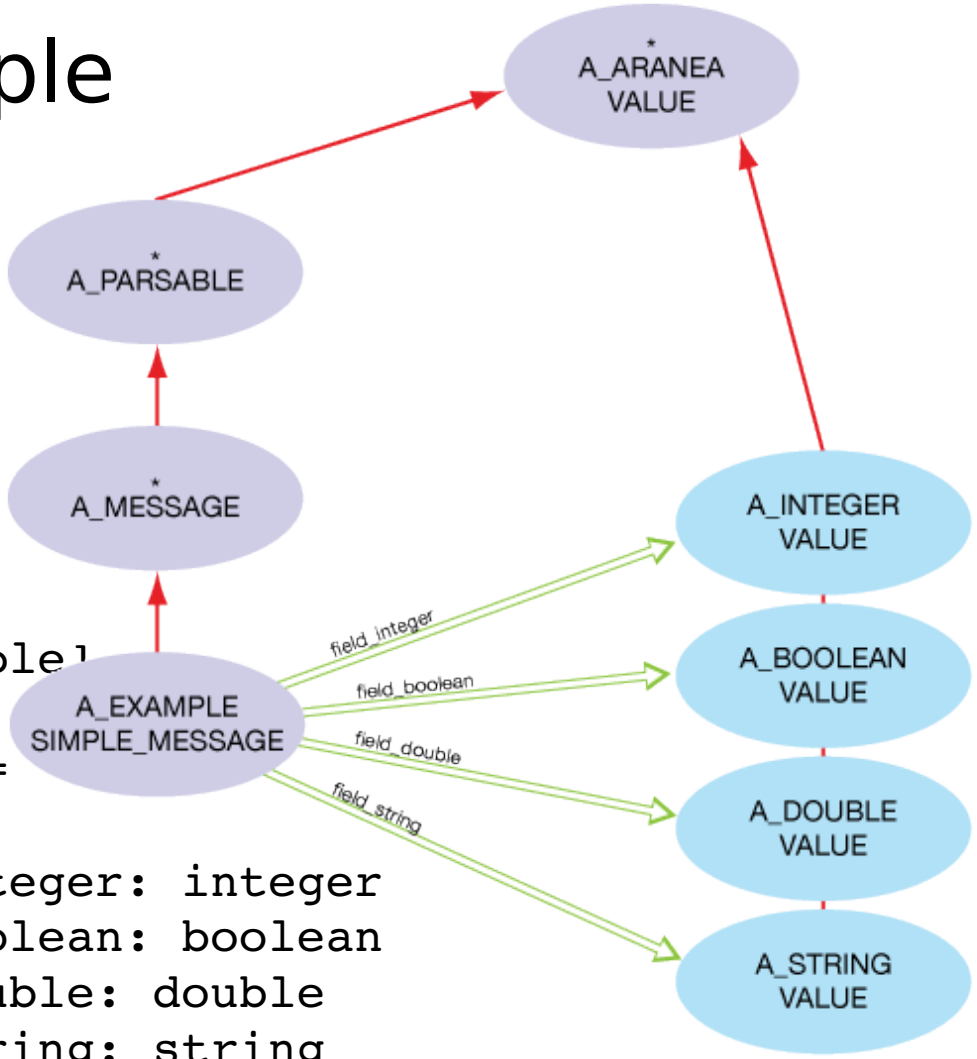
- Example




```
[message Simple]
{
  properties=
  {
    field_integer: integer
    field_boolean: boolean
    field_double: double
    field_string: string
  }
}
```

New Message Library

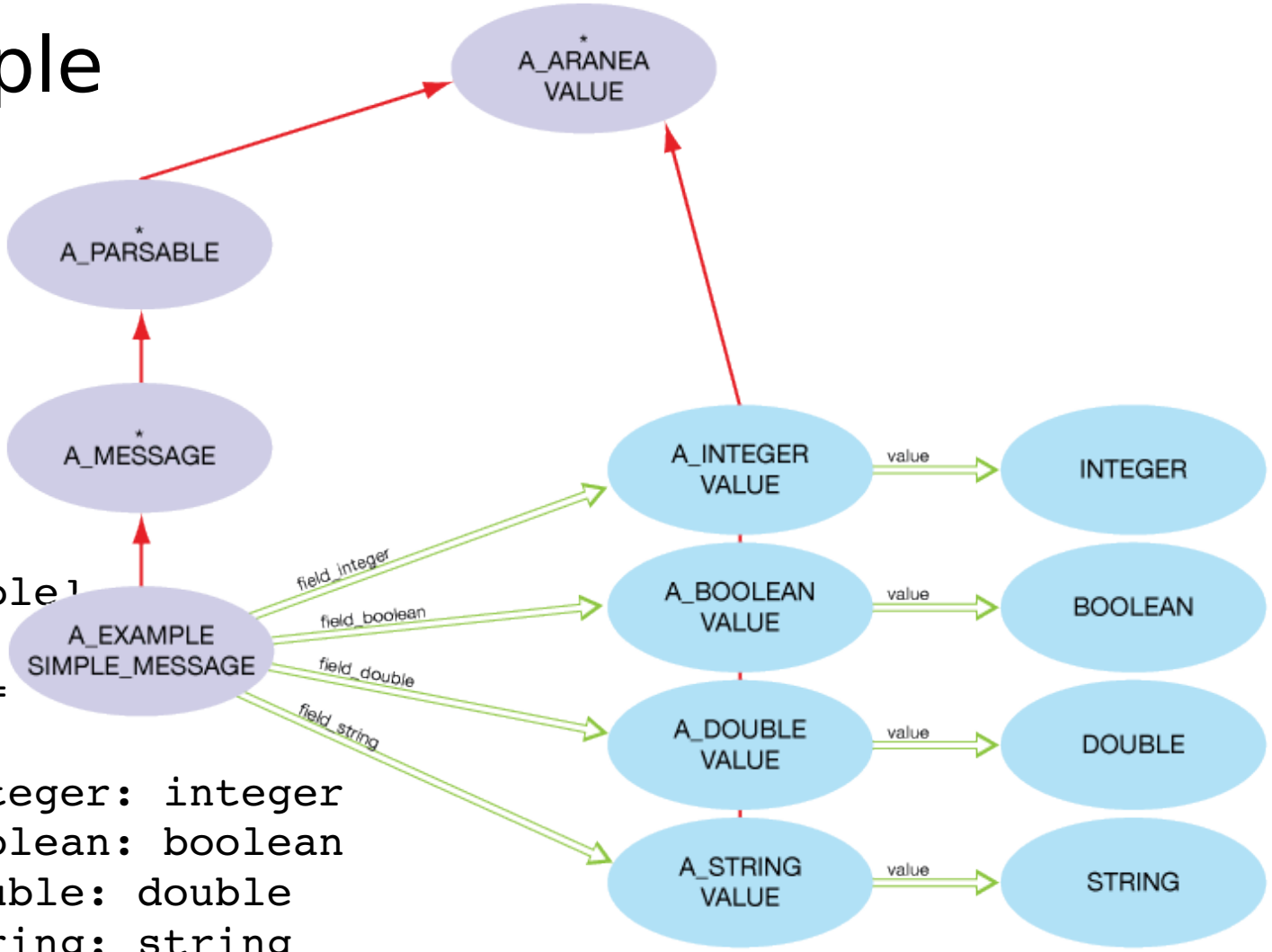
- Example




```
[message Simple]
{
  properties=
  {
    field_integer: integer
    field_boolean: boolean
    field_double: double
    field_string: string
  }
}
```

New Message Library

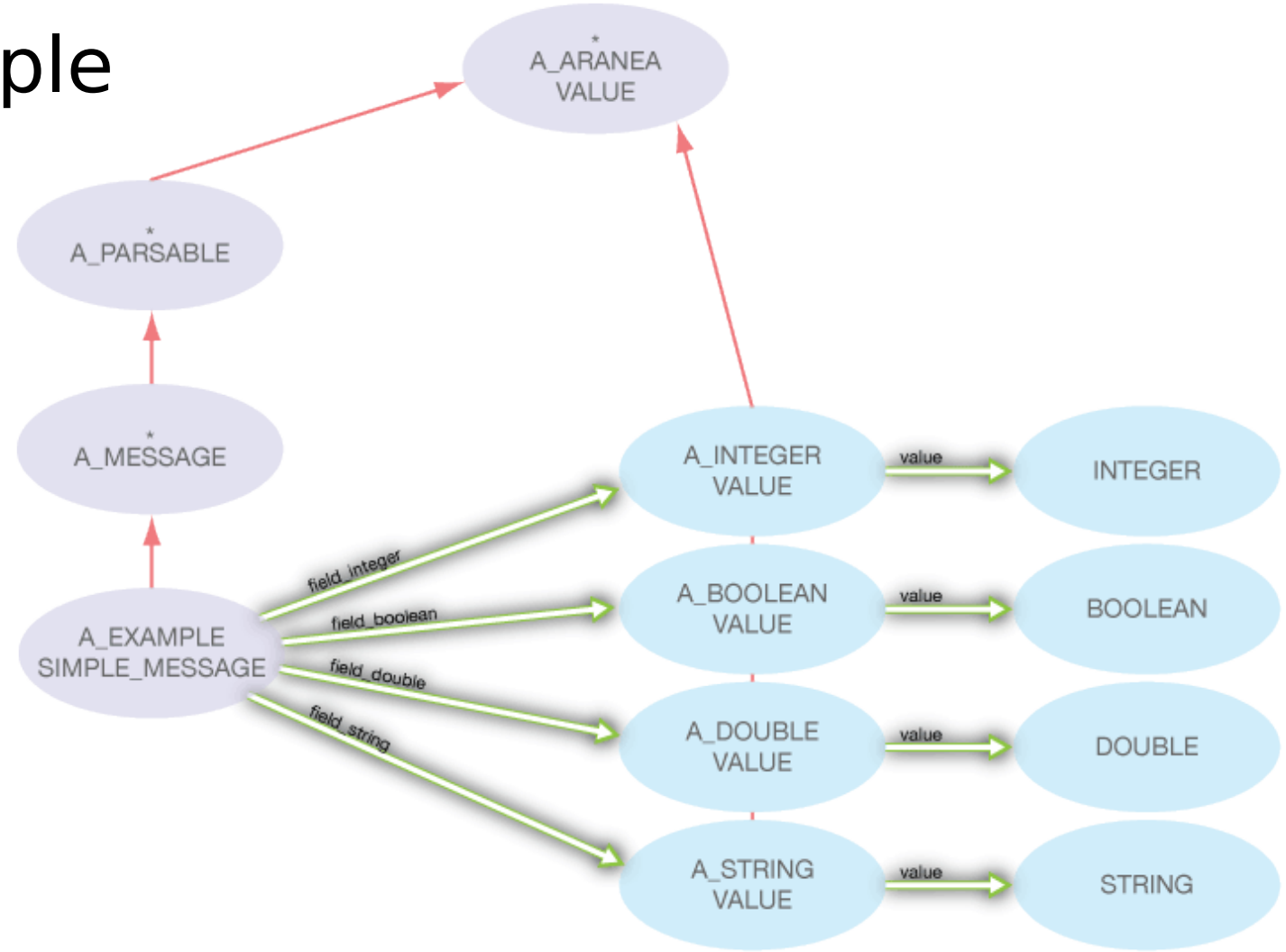
- Example




```
[message Simple]
{
  properties=
  {
    field_integer: integer
    field_boolean: boolean
    field_double: double
    field_string: string
  }
}
```

New Message Library

- Example



Message Generation

- Goal: The code of all Message-related classes should be generated.
- Use OAW, with
  - Aranea's custom language (DSL, Domain Specific Language)
  - Aranea's custom Eclipse-Editor
  - Aranea's custom Code-Generator

Message Generation

Workflow

(Build Eclipse plugins)

1. Write Message definition in DSL file
2. Run OAW-Workflow to generate code
3. Copy generated files to your project.

Message Generation

1. Write Message definition in DSL file

Datatypes

```
string
integer
double
boolean
binary

sequence<…>
map<…>

record(name, […; …])
```

Restrictions

```
notVoid
notEmpty
greaterZero
greaterEqualZero
```

Usage:

```
Property:
   name=ID ":" DataType (Restriction)* (description=STRING)+;
```

Message Generation

Records: `record(name, [i1: T1; i2: T2; …])`

- Define inline datatype
- (Almost) Regular Message will be generated
  - Inherits `A_PARSABLE`, but not `A_MESSAGE`
  - Use as field, but not as standalone message
- Replaces usage of `TUPLE`s

```
revisions: custom ["
  ARRAYED_LIST [
    TUPLE [
      creation_time: INTEGER;
      user: STRING;
      text: STRING;
      tags: STRING]]"]
```

```
revisions: sequence <
    record(revision, [
      creation_time: integer;
      user: string;
      text: string;
      tags: string])
  >
```

Message Generation

A complex definition

```
[message ListReply]
{
  properties=
  {
    release_list: map<record(release, [
      ...
      files: sequence<
          record(list_file, [ name: string; platform: string ])
      >
    ])> notVoid "Release list"
  }
}
```

Message Generation

Restrictions

- Each property may have restictions
- Translate to Eiffel contracts
- Also for properties of records!

```
revisions: sequence<
    record(revision, [
      creation_time: integer greaterEqualZero;
      user: string notEmpty;
      text: string notEmpty;
      tags: string notVoid ])
  >
```

Message Generation

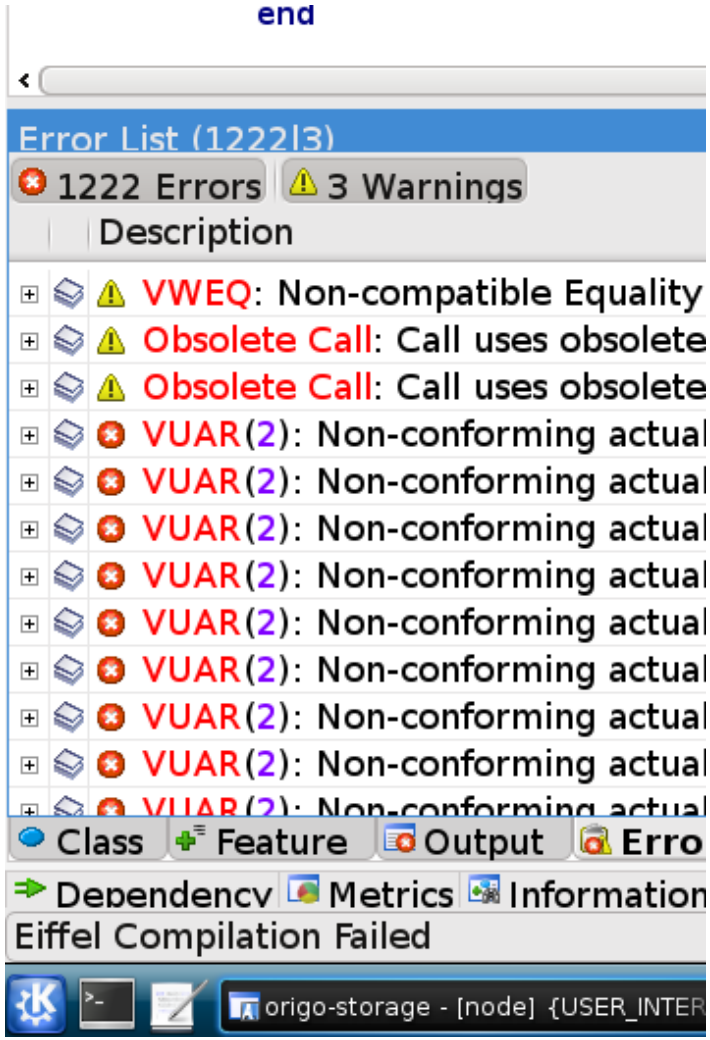
Pitfalls

- Vastly impoved runtime checks
  - But not complete
- Known Bug:
  - Uniqueness of Identifiers not enforced
    - Namespaces
    - Record Names
    - Inheritance
- What did/will you find?

Migration

Migration

1. Upgrade
2. Compile
3. Fix errors :)



Migration

Preparation

- Update Aranea
- (Re-) Build OAW-plugins

DSL-Migration

- `longstring` → use `binary`
- `Longinteger` → `integer` (is `INTEGER_64`)
- `stringlist`, `integerlist`, `longintegerlist` → use `sequence<…>`
- `custom` → use `record`, `sequence` and `map`

Migration

Eiffel code Migration I : Library part

- `A_GENERIC_LIST_MESSAGE [T]` → Use separate message with a field
  `A_SEQUENCE_VALUE [T]`
- `A_GENERIC_SESSION_MESSAGE` → Use separate message inheriting from `A_SESSION_MESSAGE` with appropriate fields
- Hint (optional): replace use of `A_GENERIC_MESSAGE` with separate message (advantage: no modification to generated classes)

Eiffel code Migration II

- Fix Message creation
  - Use Aranea's primitive types

Before:

```
make (a_session_string)
```

Now:

```
make (
   create {A_STRING_VALUE}.make (a_session_string)
)
```

Eiffel code Migration II

- Fix Message creation
  - Create complex types from appropriate container

```
Before: create mylist.make; mylist.add (mystring)
Now:
ds_list: DS_ARRAYED_LIST [A_STRING_VALUE]

create ds_list.make (1)
ds_list.force_last (
   create {A_STRING_VALUE}.make (mystring))

mylist.make (create
{A_SEQUENCE_VALUE[A_STRING_VALUE]}.make (ds_list))
```

Eiffel code Migration III

- Fix Message value-access

```
Before:
   my_string := a_msg.string_field
   my_array ?= a_generic_msg.arguments (“my_array”)

Now:
   my_string := a_msg.string_field.value

   ds_list: DS_ARRAYED_LIST [A_INTEGER_VALUE]
   ds_list := a_msg.array_field.sequence

   loop
      my_int := ds_list.item_for_iteration.value
   end
```

Migration

- Pay good attention to weak-typed containers
  - Goanna (e.g. node for XML-RPC) uses a weak type system for transformation to XML
  - `HASH_TABLE [ANY, STRING]`
  - The rest should be checked by the Typesystem
- Run your testcases

?uestions

Links

- Aranea: http://aranea.origo.ethz.ch
  - Aranea-Messages: http://aranea.origo.ethz.ch/wiki/aranea_message
- Origo: http://www.origo.ethz.ch
- OAW: http://www.openarchitectureware.org
- JSON: http://www.json.org