DREAM Types
A Domain Specific Type System for Component-Based Message-Oriented Middleware

Philippe Bidinger¹, Matthieu Leclercq¹, Vivien Quéma¹,², Alan Schmitt¹, Jean-Bernard Stefani¹

¹INRIA, France
²Institut National Polytechnique de Grenoble, France

SAVCBS 2005
Outline

1. Motivations
   - Component-based programming
   - The DREAM framework
   - Problem statement

2. DREAM types
   - Overview
   - Message types
   - Component types
   - Checking a configuration

3. Use case

4. Conclusion
Motivations

Component-based programming

The DREAM framework

Problem statement

DREAM types

Overview

Message types

Component types

Checking a configuration

Use case

Conclusion
Component-based frameworks have emerged in the past two decades:

- applications (EJB, CCM)
- middleware (dynamicTAO, OpenORB)
- operating systems (OSKit, THINK)

A component:

- is independently **deployable**
- is **configurable** (attributes)
- has **interfaces** (client, server)
- communicate through **bindings** between interfaces
Outline

1 Motivations
   - Component-based programming
   - The DREAM framework
   - Problem statement

2 DREAM types
   - Overview
   - Message types
   - Component types
   - Checking a configuration

3 Use case

4 Conclusion
The DREAM framework

- Component framework for constructing message-oriented middleware (MOM)
  - General component model
  - Component library
    - Message queues, serializer, channels, routers, ...
  - Tools for the description, configuration and deployment of MOMs

- Various MOMs can be built:
  - Publish/Subscribe, Event/Reaction, Group communication protocols, ...
A simplistic DREAM MOM

- **producer**
- **serializer**
- **addIP**
- **channelOut**
- **consumer**
- **deserializer**
- **removeIP**
- **channelIn**
**DREAM messages**

- **DREAM components exchange messages**
  - Messages are Java objects that encapsulate named chunks
  - Each chunk implements an interface that defines its type

- Basic operations over messages
  - read, add, remove, or update a chunk of a given name
Three kinds of run-time errors
- A chunk is absent when it should be present
- A chunk is present when it should be absent
- A chunk does not have the expected type

But... all messages in DREAM have the same type: the Message Java interface
Motivations

Component-based programming
The DREAM framework
Problem statement

DREAM types

Overview
Message types
Component types
Checking a configuration

Use case

Conclusion
Goals
Catching configurations errors early on, when writing the architecture description of a Dream MOM

How?
By defining a richer type system allowing the description of:
- the internal structure of messages
- the behavior of components
**Overview**

**Goals**
Catching configurations errors early on, when writing the architecture description of a DREAM MOM

**How?**
By defining a richer type system allowing the description of:
- the internal structure of messages
- the behavior of components
Adaption of existing work on type systems for extensible records for ML (D. Rémy, 1993)

Definition
An extensible record is a finite set of associations, called fields, between labels and values.

- DREAM messages can be seen as records, where each chunk correspond to a field of the record
- DREAM components can be seen as polymorphic functions
Adaption of existing work on type systems for **extensible records** for ML (D. Rémy, 1993)

**Definition**

An extensible record is a finite set of associations, called *fields*, between labels and values.

- DREAM messages can be seen as records, where each chunk corresponds to a field of the record.
- DREAM components can be seen as polymorphic functions.
Adaption of existing work on type systems for extensible records for ML (D. Rémy, 1993)

**Definition**

An extensible record is a finite set of associations, called *fields*, between labels and values.

- DREAM messages can be seen as records, where each chunk correspond to a field of the record.
- DREAM components can be seen as polymorphic functions.
Motivations

Component-based programming
The DREAM framework
Problem statement

DREAM types

Overview
Message types
Component types
Checking a configuration

Use case

Conclusion
Message types

- A message type consists of:
  - a list of pairwise distinct labels together with
    - the type of the corresponding value
    - a special tag `abs` if the message does not contain the given label

- Includes `type`, `field`, and `row (record)` variables

- `ser`, an ad-hoc type constructor
  - if $\tau$ is an arbitrary type, $\text{ser}(\tau)$ is the type of serialized values of type $\tau$
Examples

\[\mu_1 = \{a: \text{pre}(A); b: \text{pre}(B); \text{abs}\}\]
\[\mu_2 = \{a: \text{pre}(A); b: \text{pre}(B); c: \text{abs}; \text{abs}\}\]
\[\mu_3 = \{a: \text{pre}(X); \text{abs}\}\]
\[\mu_4 = \{a: Y; \text{abs}\}\]
\[\mu_5 = \{a: \text{pre}(A); Z\}\]
\[\mu_6 = \{a: \text{pre}(A); b: Z'; Z''\}\]
\[\mu_7 = \{a: \text{pre}(A); a: \text{pre}(B); \text{abs}\}\]
\[\mu_8 = \{a: X; b: \text{abs}; X\}\]
Outline

1. Motivations
   - Component-based programming
   - The DREAM framework
   - Problem statement

2. DREAM types
   - Overview
   - Message types
   - Component types
   - Checking a configuration

3. Use case

4. Conclusion
Component types

- A component has a set of *server ports* and *client ports*
- Each port is characterized by:
  - its name
  - the type of the values it can carry
- The type of a component is polymorphic, mapping client port types to server port types
- Polymorphism is important for two reasons:
  - the same component can be used in different contexts with different types
  - it expresses explicit dependencies between client and server port types
Examples

\begin{align*}
\text{id} & : \forall X.\{i : \{X\}\} \rightarrow \{o : \{X\}\} \\
\text{dup} & : \forall X.\{i : \{X\}\} \rightarrow \{o_1 : \{X\}; o_2 : \{X\}\} \\
\text{add}_a & : \forall X.\{i : \{a : \text{abs}; X\}\} \rightarrow \{o : \{a : \text{pre}(A); X\}\} \\
\text{remove}_a & : \forall X, Y.\{i : \{a : Y; X\}\} \rightarrow \{o : \{a : \text{abs}; X\}\} \\
\text{reset} & : \forall X.\{i : \{a : \text{pre}(A); X\}\} \rightarrow \{o : \{a : \text{pre}(A); X\}\} \\
\text{serialize} & : \forall X.\{i : \{X\}\} \rightarrow \{o : \{s : \text{ser}({\{X\}}); \text{abs}\}\} \\
\text{deserialize} & : \forall X.\{i : \{s : \text{ser}({\{X\}}); \text{abs}\}\} \rightarrow \{o : \{X\}\}
\end{align*}
Motivations

- Component-based programming
- The DREAM framework
- Problem statement

DREAM types

- Overview
- Message types
- Component types
- Checking a configuration

Use case

Conclusion
Type checking using equational theory and unification algorithm (D. Rémy, 1993)

Configuration well-typed iff we can solve the equations:

\[
\begin{align*}
\{X\} &= \{ts : \text{pre}(A); Y\} \\
\{X\} &= \{ts : \text{abs}; Z\}
\end{align*}
\]
Use case

Motivations
DREAM types
Use case
Conclusion

(a)

(b)

P. Bidinger et al.
DREAM types
From bindings, we deduce the following equations:

\[
\begin{align*}
\{ \text{tc} : \text{pre}(& \text{TestChunk}); \text{abs} \} &= \{ U \} \\
\{ \text{sc} : \text{pre}(& \text{ser}(U)); \text{abs} \} &= \{ \text{ipc} : \text{abs}; Z \} \\
\{ \text{ipc} : \text{pre}(& \text{IPChunk}); T \} &= \{ \text{ipc} : \text{pre}(& \text{IPChunk}); Z \} \\
\{ \text{ipc} : \text{pre}(& \text{IPChunk}); Z \} &= \{ Y \} \\
\{ Y \} &= \{ \text{ipc} : \text{pre}(& \text{IPChunk}); X \} \\
\{ \text{ipc} : \text{abs}; X \} &= \{ \text{tc} : \text{pre}(& \text{TestChunk}); \text{abs} \}
\end{align*}
\]
From 6, we deduce that

\[ X = \{ tc : \text{pre}(\text{TestChunk}); \text{abs} \} \]

Then from 5, we have

\[ Y = \{ ipc : \text{pre}(\text{IPChunk}); tc : \text{pre}(\text{TestChunk}); \text{abs} \} \]

It follows from 4 and 3 that

\[ T = Z = \{ tc : \text{pre}(\text{TestChunk}); \text{abs} \} \]

Besides, we deduce from 2 that

\[ Z = \{ sc : \text{pre}(\text{ser}(U)); \text{abs} \} \]

\[ tc : \text{pre}(\text{TestChunk}); \text{abs} \text{ and } sc : \text{pre}(\text{ser}(U)); \text{abs} \text{ are not unifiable} \Rightarrow \text{the configuration is not correct} \]
Conclusion

- Domain specific type system for messages and components
  - Based on existing work on extensible records
  - Rich enough to address component assemblages such as protocol stacks

- FFS: type system is too restrictive to type DREAM components that exhibit different behavior depending on the presence of a given label in a message (e.g. routers)
  - DREAM operational semantics
  - Intersection types
M. Leclercq, V. Quéma and J.-B. Stefani.

**DREAM**: a *Component Framework for Constructing Resource-Aware, Configurable Middleware*.

IEEE Distributed Systems Online, vol. 6 no. 9, 2005.

E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma and J.-B. Stefani.

**FRACTAL**: an *Open Component Model and its Support in Java*.

For Further Reading II

P. Bidinger, A. Schmitt and Jean-Bernard Stefani.
An Abstract Machine for the Kell Calculus.

Component-Oriented Programming with Sharing: Containment is not Ownership.
Questions

- http://dream.objectweb.org – DREAM implementation and documentation

- http://sardes.inrialpes.fr/kells – Kell calculus papers and implementation