CHAIMS

Compiling High-level Access Interfaces for Multi-Site Software

Objectives:

- Investigate revolutionary approach to large-scale software composition
- Define a purely compositional megaprogramming language
- Develop prototype software system to test and proof concepts

Principal investigator: Gio Wiederhold, Stanford University

http://www-db.stanford.edu/CHAIMS

Challenges in Composing Large, Heterogeneous, Distributed Modules

Heterogeneous distribution protocols:

- DCE, CORBA, DCOM, RMI...
- composing megamodules residing in different distribution protocols

Remote services:

- heavy weight services, long execution times
- potential of concurrent execution, parallel invocation of several services
- pre-invocation estimates (run-time optimization)

Domain experts as megaprogrammer:

- "simple" programming language
- optimizations and scheduling of invocations done by compiler
- automatic generation of stubs for various distribution protocols

⇒ CHAIMS megaprogramming language

CHAIMS language primitives

Pre-invocation:

SETUP, setting up connection to megamodule

SET-, GETATTRIBUTES, setting global parameters in megamodule

ESTIMATE, getting estimate of execution time for optimization

Invocation and result gathering

INVOKE, invoke a specific method

EXAMINE, test status of an invoked method

EXTRACT, extract results from an invoked method

Termination:

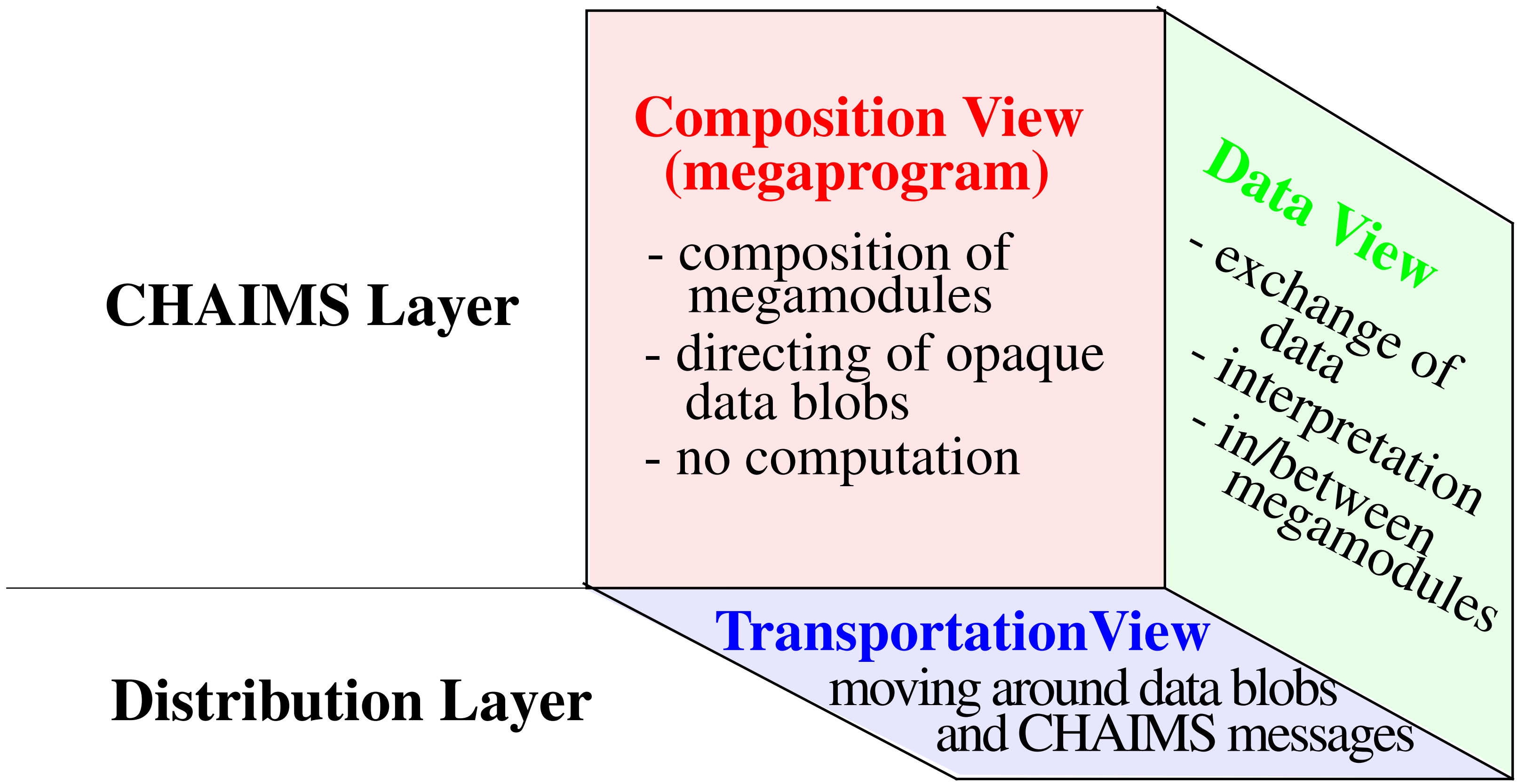
TERMINATE, terminate method invocation or connection to megamodule

Control:

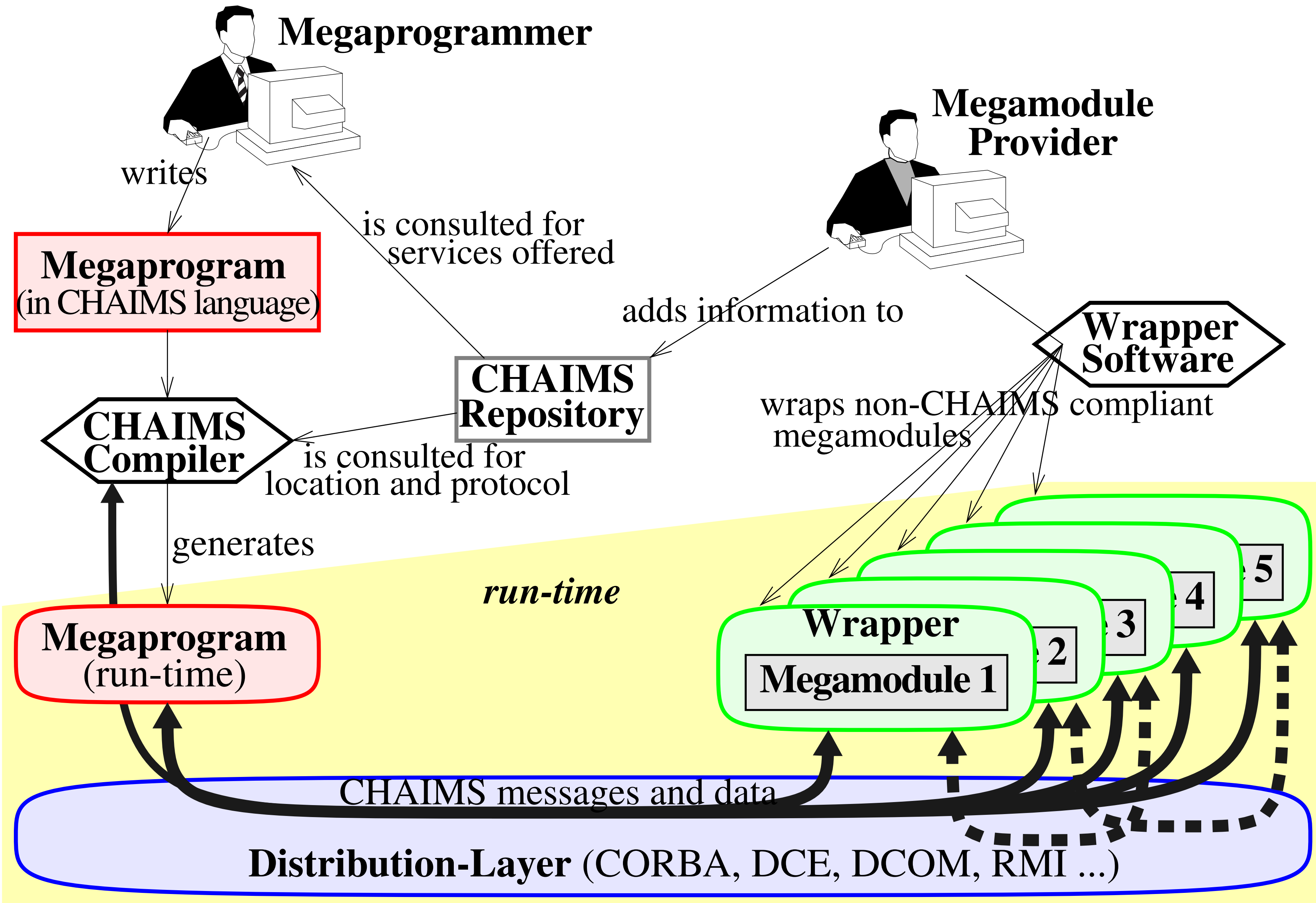
WHILE, IF

breaking up CALL-statement and adding control

Composition, Data, Transportation Views



Architecture and Composition Process



Progress so far

- Design of CHAIMS megaprogramming language
- CHAIMS compiler for DCE and CORBA
- CHAIMS wrapper template for CORBA
- Examples, documentation and publications

Next Steps

- Building further examples based on existing CHAIMS infrastructure
- Extending compiler for RMI
- Adapting wrappers to DCE and RMI
- Repository
- Optimization and scheduling of invocations
- Optimization of data flows

Conclusions

We explore new ways of composing large megamodules

➡ reuse of existing and new megamodules

We use a purely compositional megaprogramming language

➡ focus on architectural and compositional issues

We generate the protocol specific client code and have templates for the server wrappers

➡ work and knowledge avoidance

We address optimization and estimation issues

➡ scaleability of composition to large data volumes and computation intensive services