FRESCO

Foundational Research on Service Composition

Modelling Support for Service Compounds

Thomas Plümpe
Henning Brandt

8 July 2002

OVERVIEW

- CONTEXT
- OBJECTIVES
- METHOD OF PROCEEDING
- RELATED WORK
  - Overview
  - Γ-language / Chemical Abstract Machine (CHAM)
  - PICCOLA language / πL-calculus

CONTEXT

The FRESCO framework

- Purpose: enable service providers to model, design and execute composite services
- Conceptual elements:
  - Models for service composition, aggregation, coordination
  - Methodology for using the framework
  - ...
- Technology elements:
  - Integrated Development Environment (IDE)
  - Integrated Runtime Environment (IRE)

CONTEXT

Basic principles for service creation and provision:

- Composition
  - The capabilites of a (composite) service S are based entirely on other services SC_1, SC_2, ... , referred to as its service components.
  - specify causal and temporal relations between components
  - special focus on dynamic nature of composition
- Aggregation
  - (dynamic) acquisition of instances of service components by the provider of the composite service
- Coordination
  - management of cooperation between service components during service provision

OBJECTIVES

- Develop a conceptual service composition model
  - capturing the FRESCO composition approach
  - providing an intuitive metaphor (opt.)
    candidate: chemical metaphor (due to Banatre/Le Metayer)

- Derive a formal service composition model
  - expressing a feasible subset of properties
  - providing proof mechanisms for composition specifications
  - preserving the metaphor

- Implement a service IDE prototype
  - based on the formal composition model
  - allowing compound services to be assembled from service components
  - for verifying compositions / detecting architectural mismatch

OBJECTIVES IN CONTEXT

The FRESCO framework

- Purpose: enable service providers to model, design and execute composite services

- Conceptual elements:
  - Models for service composition, aggregation, coordination
  - Methodology for using the framework
  - ...

- Technology elements:
  - Integrated Development Environment (IDE)
  - Integrated Runtime Environment (IRE)

METHOD OF PROCEEDING

Investigate ...

... requirements imposed by the FRESCO service composition approach	... existing service models (as used in web services-related standards and in related research)	... relevant formal models (architectural, process calculi)

Define conceptual service composition model

Develop a formal model from the conceptual model

Design and implement service IDE

Write diploma thesis and report

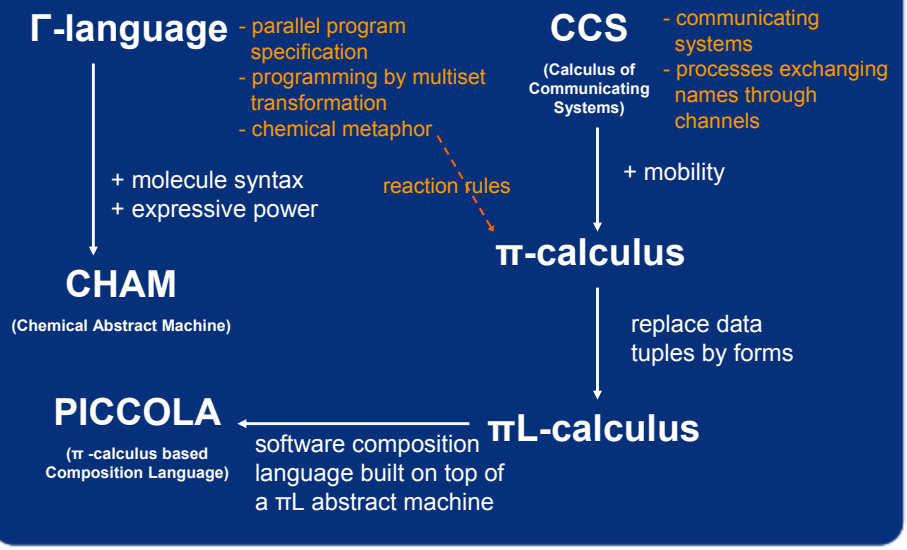
RELATED WORK - Overview

	approach formal ?	targets		who ?	category
		service composition	component architectures		
WSFL		✓		IBM	industry standard specs
XLANG		✓		Microsoft	
DySCo		✓		HP Labs	research projects
E-Flow		✓		HP Labs	
ICARIS		✓		Carleton Uni Toronto	
DynamiCS			✓	VSIS, Uni HH	
PICCOLA	✓		✓	SCG, Uni Bern	applied formal methods
π-calculus	✓		✓	R. Milner, Cambridge Uni	
CHAM	✓		✓	Inverardi, Wolf	

Formal Methods – Concepts

- Calculi
  - formal (low-level, minimalist) ways of specifying computing concepts
  - tools for rigorous analysis of computing systems and formal proof
  - basis for deriving higher-level languages by adding higher-level features (data structures, objects, functions, ...)
- Abstract Machines
  - executional models of computing systems
  - provide implementions of calculi
  - serve the analysis of dynamic aspects of a computing system
- Examples:
  - λ-calculus: investigation of computable functions, basis for LISP
  - Turing machines, RAMs: models of sequential machines, e.g. to study computational complexity

Formal Methods – Overview



The Γ programming language

- **G**eneral **A**bstract **M**odel for **M**ultiset M**a**nipulation
- Main Advantage: focus on logical parallelism, i.e. without specifying more sequentiality than necessary
- Contrasts with traditional imperative/formal programming languages:

  (1) ma := max_array(a);
  (2) mb := max_array(b);
  (3) m := max(a,b);
- first (1), then (2): unnecessary sequentiality (may run in any order)
- first (1) + (2), then (3): a necessary sequentiality

- fact(N) = GAMMA((R,A)) ({1, ..., n}) where

  R(x, y) = true

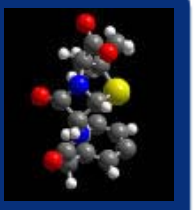
  A(x, y) = {x*y}

Γ example

Example: fact(5)

1. Jointly reacting elements x,y are removed from the multiset
2. The result of the function A(x, y) is added to the multiset
3. Program terminates when the predicate holds for no combination of values

The Chemical Metaphor



- The execution of GAMMA programs is often compared to a chemical reaction
- The multiset then corresponds to a chemical solution
- The control structure corresponds to the stirring mechanism,
- heating (turning a molecule into many), cooling (turning many molecules into one) and simple reaction rules (preserving the multiset's cardinality)
- A solution is inert when no transformation rule is active

The Chemical Abstract Machine

- based on the GAMMA programming style
- adds
  - a syntax for molecules,
  - a classification of transformation rules
  - a membrane/airlock construct, thus achieving
    - the expressive power of classic process calculi, and
    - a mechanism for describing modules and interfaces
- An example rule:

  i(char)◆o(tok) ◆ lexer, o(char) ◆ text
  → o(tok) ◆ lexer ◆ i(char), text ◆ o(char)

CHAM applications in software architecture description

- GAMMA/CHAM programs are executable, but slow
- main application: precise specification of functions/systems
- specification of a multiphase compiler
  - Inverardi, Wolf [1995]
  - description as a monolithic software system
  - with a focus on membrane/airlock use
- specification of a compressing proxy server
  - Inverardi, Wolf, Yankelevich [2000]
  - description as a set of interacting components

    ➔ basis for a formal service architecture description?

PICCOLA

- Pi-calculus based composition language
- Conceptual framework:
  applications = components + scripts
- generalized approach to composition not biased towards special component models or architectural styles

- The architectural style of components is determined by
  - the connectors used to connect them
    (events channels, pipes, method invocations, ...)
  - rules governing their composition
    (example: Stream >> File → File)

PICCOLA

- Modelling primitives:
  - agents – communicating entities performing calculations
  - forms – extensible immutable records containing mappings from labels to values
  - channels – shared communication channels used by agents to exchange forms
- Formal foundation:
  - πL-calculus (variant of the polyadic πL-calculus, modified to work on forms instead of tuples)
  - forms do not alter the expressive power of the calculus but it makes it much simpler to express higher-level abstractions in Piccola

PICCOLA – System Layers

Application	components + scripts
Architectural styles	streams, events, GUI composition
Core libraries	basic control abstractions (if-then-else, try-catch, ...), basic object model, basic coordination abstractions, interface to Java
Piccola language	built-in types (numbers, strings, booleans), operator syntax, nested forms, „services“ (= functions)
πL abstract machine	agents, channels, forms

PICCOLA – Highlights and Impulses

Appealing features

- communication of forms
  - versatile data structure, suitable for representing interfaces, complex arguments, contexts, ...
- layered approach
  - keep formal underlyings simple
  - retain ability to explain higher-level concepts in terms of the formal foundations
  - eases extensibility and substitutability of elements
- elaborate abstractions built on the π-calculus

Thanks for listening

Comments and questions welcome.

Henning Brandt 5brandt@informatik.uni-hamburg.de
Thomas Plümpe 5pluempe@informatik.uni-hamburg.de