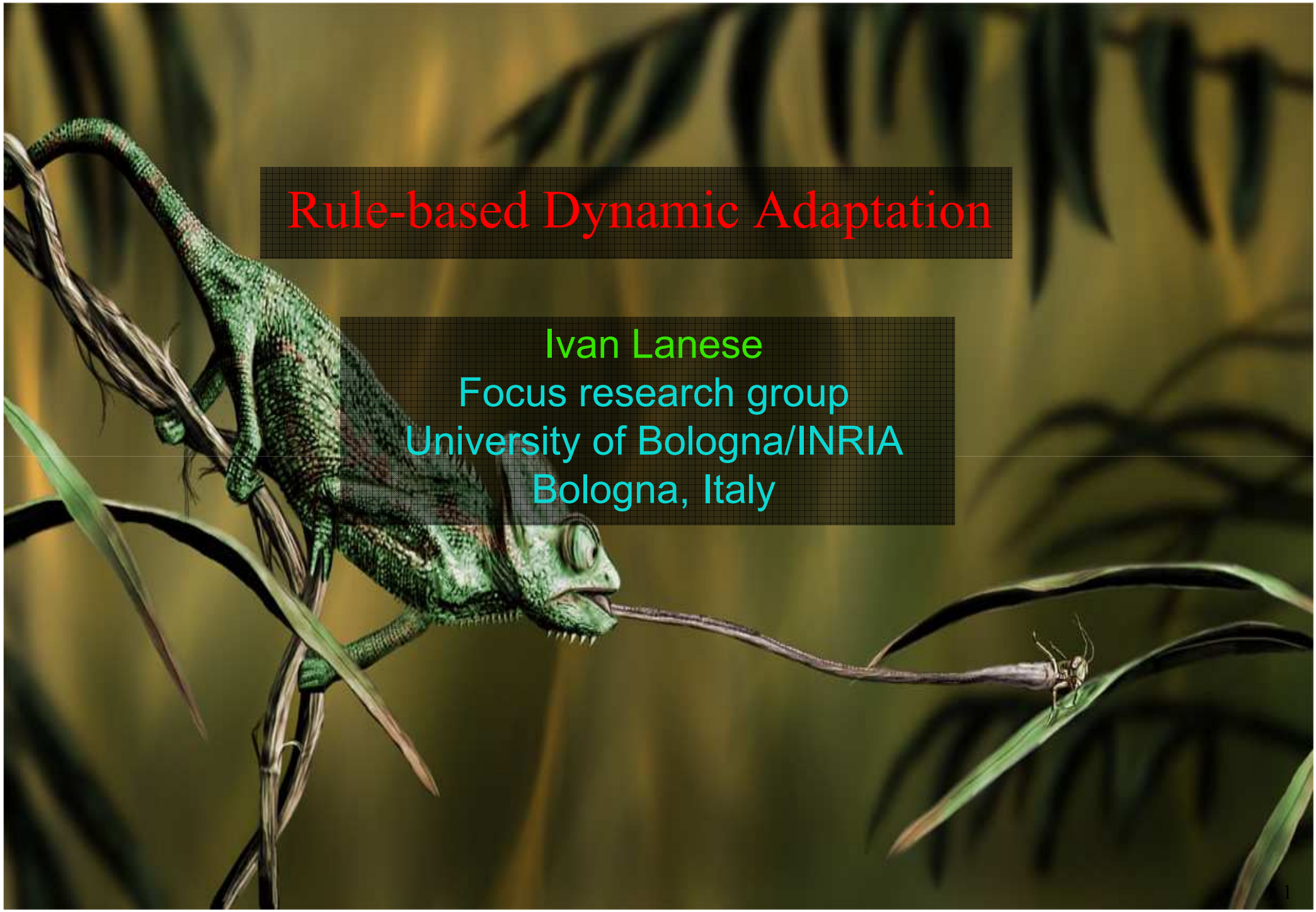
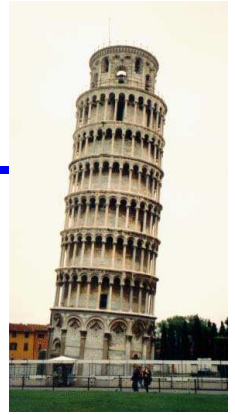


Rule-based Dynamic Adaptation

Ivan Lanese
Focus research group
University of Bologna/INRIA
Bologna, Italy

About my work



- Ph.D. in Computer Science at University of Pisa
  - Under the supervision of Prof. Ugo Montanari
  - Concurrency theory, graph transformations, formal methods, global computing

- Post-doc at University of Bologna
  - In the group of Prof. Sangiorgi and Prof. Zavattaro
  - Working inside projects Sensoria and HATS
  - Concurrency theory, formal methods, service oriented computing, adaptation
- Collaborations with Lisbon, Leicester, INRIA Grenoble, IRST Trento, ItalianaSoftware s.r.l.

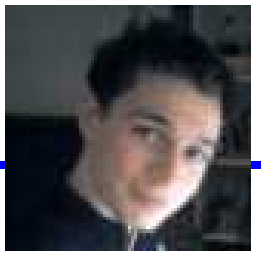
The HATS project

- European FET FP7 project
- HATS stands for "*Highly Adaptable and Trustworthy Software using Formal Models*"
- 8 academic partners, 2 industrial research institutes, 1 SME
- Partners from 7 different countries
- European contribution of 5,27 M€ over 48 months
- Started in March 2009

HATS aim

- Software has to last for decades to justify huge investments, and to adapt to different environment conditions
- Software has to be trustworthy
- We want to achieve this goal using formal methods
- Approach based on an Abstract Behavioral Specification language (ABS)
  - Object oriented language
- Try to give formal basis to the Software Product Lines approach
  - Also families of products have to evolve

People to thank



- Antonio Bucchiarone
  - Fondazione Bruno Kessler - IRST, Trento
  - Adaptation expert, software engineer
- Fabrizio Montesi
  - IT University of Copenhagen - italianaSoftware s.r.l.
  - Jolie developer and guru, JoRBA developer
- Many people from University of Bologna and from the HATS project

Roadmap

- Adaptation
- Dynamic adaptation
- A rule-based approach
- An introduction to Jolie
- The JoRBA prototype
- Conclusions

Roadmap

- Adaptation
- Dynamic adaptation
- A rule-based approach
- An introduction to Jolie
- The JoRBA prototype
- Conclusions

Adaptation

- An adaptable application changes so to adapt itself to new operating conditions
- The study of adaptation involves
  - Programming languages
  - Artificial intelligence
  - Software engineering
  - Formal methods
  - Operating systems
  - …

Adaptation is hot

- Huge push to study adaptation both in industry and academia
- Conferences
  - IEEE International Conference on Autonomic Computing
  - Workshop on Software Engineering for Adaptive and Self-Managing Systems
- Research initiatives
  - ADAPT research focus (we are here! ☺)
  - HATS project
  - S-Cube network of eccellence on “Software Services and Systems Network”
  - ALLOW project on “Adaptable Pervasive Flows”

Adaptation vs evolution

- No universally accepted definition of adaptation or evolution
- Adaptation usually related to environment conditions
- Adaptation usually does not change static definitions
  - Objects but not classes in object oriented systems
- Evolution usually related to long-term improvement
- Evolution usually changes future instances
  - Class definitions are changed, and new objects created accordingly
- We will consider both the aspects
- We use the two terms almost as synonims

Aims for evolution

- Corrective evolution: removing faults
- Perfective evolution: improving the system
  - Non functional properties
  - Adding new features
- Adaptive evolution: adapting the system to the environment
  - Adaptation to local conditions
  - Adaptation to surrounding applications
  - Adaptation to workload
  - Adaptation to user preferences

Causes of adaptation

- Completely new environment
  - Configurable applications (static)
  - Mobile applications (dynamic)
- Environment that evolves
  - Old services no more available
  - Evolution of partner applications
  - Exploiting new services for improvement
- New requirements
  - New user desires
  - Improving non functional properties

Why adaptation? Code reuse

- Developing new applications from scratch is more expensive (both time and money)
  - Parts of the old applications can be reused
  - More stable since already tested
    - » Regression test needed for use in the new application
- Not only code, but also
  - Documentation
  - Test cases
  - Models
  - Tools
- No need for data migration

Why adaptation? Continuous availability

- May avoid service interruption when moving from old to new application
- Service interruption dangerous in safety critical systems…
- … but also in commonly used systems
  - Hosted email services (Gmail, Yahoo mail, Hotmail)
  - Online banking systems
  - Cellular networks
    - » Never tried to phone during New Year's Eve?
- Continuous availability requires to apply adaptation at runtime

Possible adaptation scenarios

- A travel organizing application adapts so to exploit new train/bus/teleport connections
- An agenda application adapts so to interact with new peripherals (smartphone, …)
- A car driving application adapts to the traffic laws of the country it is travelling in
- A workflow for business activities automatically exploits new, more efficient services that become available on the net

Adaptation features

- Adaptation trigger: what makes the adaptation start?
  - User event, environment event, programmed adaptation
- Adaptation manager: who controls adaptation?
  - Human manager, adaptation middleware
- Adaptation goal: which is the specific aim of the adaptation?
  - Satisfying a performance bound, making a new functionality available, ensuring interoperability with a new device,…
- Adaptation strategy: which approach is taken for reaching the adaptation goal?
  - Which parts of the application are changed? When?
- Adaptation mechanisms: which algorithms/techniques are used for enacting adaptation?

Time of adaptation

- Adaptation mechanisms can be applied both statically and dynamically
- Four interesting scenarios
  - Static evolution
  - Static configuration
  - Built-in adaptation
  - Dynamic adaptation

Static evolution

- Aims at evolving a software application so to satisfy new requirements
- The development process of the application is restarted, exploiting old artifacts as far as possible
  - Can be seen as a new cycle in a cyclic development process
- Main advantage: code reuse
  - And documentation reuse, test case reuse, …
- Strong emphasis in regression testing
- Manual evolution (clearly, tools may help)
- The application may be reinstalled or patched
  - With dynamic patch service availability may be preserved

An example: source compatibility for Java

- Ongoing research in the HATS project
- If a Java package evolves, can I find out whether all the programs that compiled using the old version still compile?
  - Necessary condition for regression test
- Syntactic conditions can be found
- A tool for checking source compatibility has been built
- http://softech.informatik.uni-kl.de/Homepage/SourceCompatibility

Static configuration

- Aims at configuring applications so to deploy them in different environments (cfr. Püschel talk)
- Many configurations of the application are developed together, sharing a common core
- Main advantage: code reuse (and the like)
- Strong emphasis in modular development and analysis

Example: software product line

- Describes a family of software products as different compositions of a set of features
- Like for cars in automotive industry
  - Examples of features: diesel/gasoline, red/black/white color, airbag/no airbag
- Different configurations can be built by choosing the desired features
- Not all the feature combinations may be allowed
- How to build the actual product?

Built-in adaptation

- Adaptation needs are foreseen when the application is designed
- Mechanisms for triggering and enacting adaptation are included in the application
  - Mechanisms for monitoring the environment conditions
  - Mechanisms for triggering adaptation when needed
  - Adaptation logic to find the right adaptation strategy
  - Mechanisms for enacting adaptation
- Adaptation is applied automatically, without intervention from technicians and without service disruption

Example: Adaptive Pervasive Flows

- Ongoing research in the ALLOW project
- Flows are sets of actions glued together by an execution plan
- Pervasive flows live in the real world, attached to real objects and modelling their behavior
- Adaptive pervasive flows should adapt to the environment they move through
- They include built-in facilities for reacting to environment conditions
  - Environment-triggered events
  - Conditions on environment state

Built-in adaptation challenges

- Built-in adaptation exploits methods from artificial intelligence, software engineering and control theory
- Related to autonomic computing and self-* systems (self-healing, self-correcting, …)
- Possible triggers and strategies are known statically, but
  - Triggers are detected at runtime
  - The strategy is chosen and applied dynamically

Built-in adaptation: shortcomings

- All the possible adaptation situations should be foreseen during application development
  - Which event may trigger the adaptation
  - Which new behavior will be required
  - Which adaptation strategies will be needed
  - How to choose the best adaptation strategy
  - How to enact it
- Strong responsibility on designers
- Not always possible
  - Ok for configurable applications (for a known set of possible environments)
  - Extremely difficult for long term evolution

Roadmap

- Adaptation
- Dynamic adaptation
- A rule-based approach
- An introduction to Jolie
- The JoRBA prototype
- Conclusions

Dynamic adaptation

- Aims at adapting applications to changes that were not predictable at application development time
  - Adapting to unforeseen environment
  - Adapting to new technologies
    - » Multiprocessors in Püschel talk
  - Adapting to new user needs
- Allows applications to survive for long periods of time, evolving so to remain suitable
- Applicable also when built-in adaptation is not

Dynamic adaptation requirements

- Adaptation should change the code of the application, not only its state
  - New protocols, new algorithms
- The new protocols/algorithms and even the kind of adaptation required are not known
  - When the application is developed
  - When the application is deployed
  - When the application is started
- Adaptation should be as automatic and user-transparent as possible
  - No or minimal intervention from technicians
  - No service disruption

Challenges of dynamic adaptation

- Scarce information at application development time
- Some other information (and code!) should be obtained at runtime
- The adaptation logic cannot be part of the application
  - Requires interaction with some adaptation manager
- The application should provide some hooks for adaptation
  - Difficult to adapt a black box
  - How much information is needed for predisposing good hooks?

A different perspective

- Is there any relation with software update?
  - Linux modules
  - Automatic updates of applications
- Those kinds of updates are requested either by the user or by automatically looking for a newer version on the net
- Not aimed at adapting to different environment conditions
- No or trivial logic controlling the adaptation
  - If a newer version is available, then install it
- Quite ad hoc

Our aim

- A general strategy/framework for developing adaptable applications
- Requiring minimum information at application development time
- Able to tackle different kinds of adaptation
  - Performance enhancement
  - Evolution of functionalities
  - Adaptation to different environments
- With minimal service disruption
  - No reboot
- With minimal intervention from technicians

Travelling scenario

- Bob is travelling from Bologna to University of Koblenz-Landau for the ADAPT summer school
- Bob organizes all its travels with a Travelling organization application
  - Finding the good train/bus/airplane connections for the travel
  - Buying the tickets using Bob credit card (after Bob confirmation)
  - Instructing Bob about which train/bus/airplane to take
- The travelling application has to interact with the information systems of train stations/airports/bus stations

Travelling scenario

- For this journey the application has to
  - Find suitable means of transport for the journey
    - » We assume train+bus
  - Book the train ticket from Bologna to Koblenz
  - Instruct Bob to take the booked train
  - Book the bus from Koblenz train station to Koblenz University
  - Instruct Bob to take the bus

High speed trains

- Assume a new high speed connection has been created between Bologna and Koblenz
- Applications involving taking trains may require adaptation
  - For instance, our Travelling application
- When the Travelling application connects to Bologna train station information system, Bologna train station adaptation middleware
  - Discovers that the application may be adapted to exploit the new high speed connection
  - Checks if adaptation is possible
    - » The application provides the needed adaptation interface
  - Checks if adaptation is desirable according to Bob's preferences
    - » The new connection is not too expensive
  - Sends the updated code for booking the new high speed train to Bob's application

A few observations

- The creation of an high speed connection was not expected when the travelling application has been developed (or even when it has been started)
- Booking the high speed train may exploit a different protocol than booking other trains
- If the old connection still exists, Bob may still want to use it (e.g., for price reasons)

Other possible scenarios

- Bob's train may be late and the application should adapt to satisfy the scheduling
  - Finding new train connections
  - Using taxi instead of bus in Koblenz to save time
- A teleport connection has been created and can be exploited
- Bob's train has been suppressed, and a completely new journey should be planned
- The train station information system has been changed, and the application should adapt to the new protocols

Variety of adaptation scenarios

- Adaptation scenarios differ on many respects
- Different aims
  - Improvement of results (high-speed train)
  - Management of problems (suppressed train)
- May require temporary or permanent adaptation
- May influence different parts of the application
- We want an approach able to deal with all these scenarios

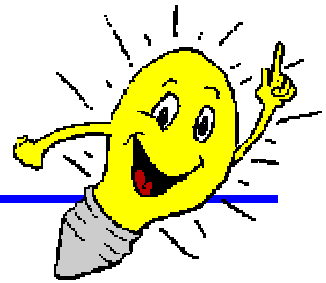
How to implement these scenarios?

- Which kind of information about adaptation is required at application development time?
  - Difficult to adapt a black box (wrappers, …)
- How can this information be specified?
- Which information can be retrieved at runtime?
- How can this information be exploited?
- How can this procedure be implemented in practice?

Roadmap

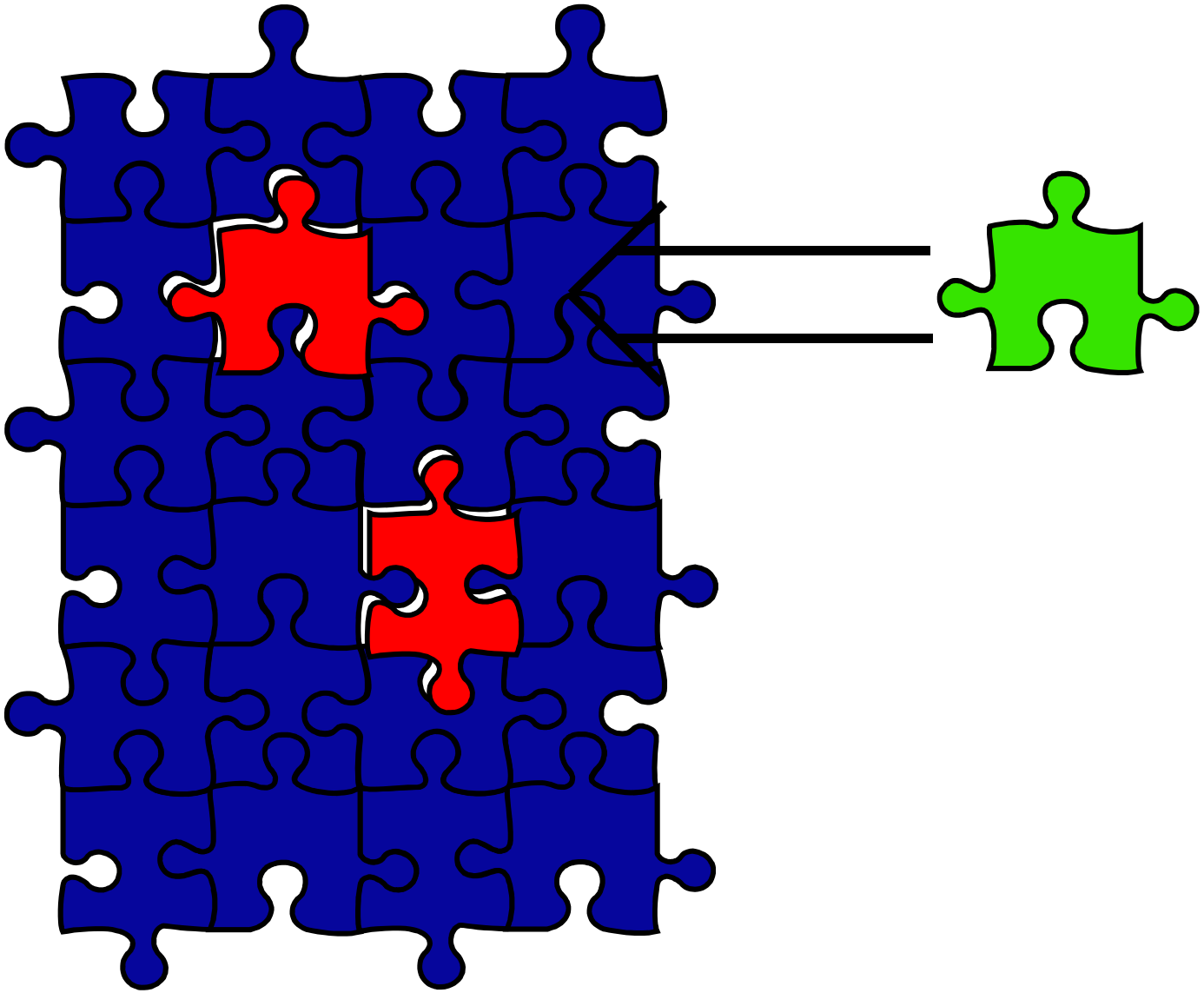
- Adaptation
- Dynamic adaptation
- A rule-based approach
- An introduction to Jolie
- The JoRBA prototype
- Conclusions

Our approach: main ideas

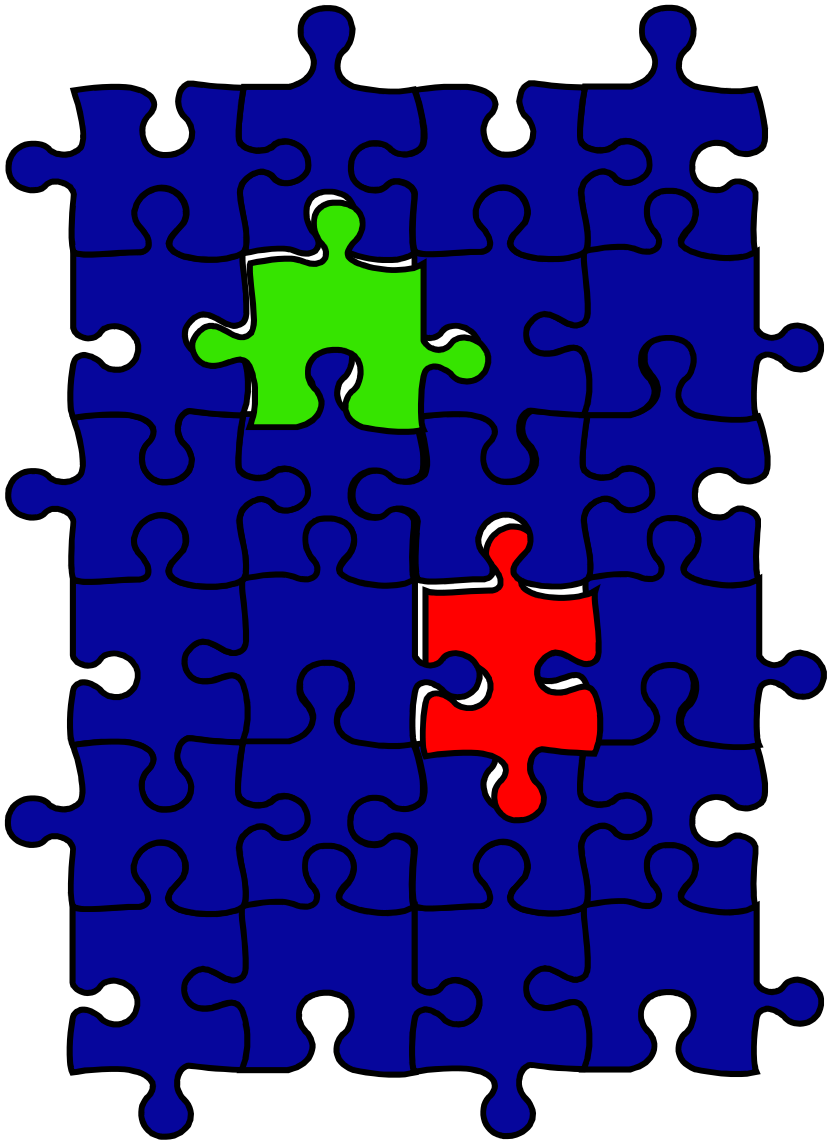


- The application should provide an adaptation interface
  - A list of adaptable activities
  - A few pieces of information about them
  - Ports to interact with the adaptation manager
- Adaptation is specified as a set of adaptation rules
  - One for each possible adaptation
  - External to the application
  - That can be created/modified after the application has been fully developed
  - Managed by an adaptation manager
    - » Possibly implemented as a set of distributed adaptation servers

Our approach: an intuition



Our approach: an intuition



Adaptation interface

- For each activity the adaptation interface should provide
  - A description of the activity, for identifying it
    - » At least its name
    - » Possibly its goal, a description according to some ontology, …
  - The set of variables that the activity uses to interact with the rest of the application
    - » We assume a shared-memory interaction model
  - A description of the non functional properties guarenteed by the current implementation
  - A comparator function allowing to compare two sets of non functional properties according to the user preferences

The Take Train activity

- Description: **Take Train**
- Variables: **source** (Bologna Train Station), **destination** (Koblenz Train Station), **number** (IC2356)
- Non functional properties: **time = 9h41m**, **cost = 80€**
- Comparator function: new implementation is better if requires less time and each saved minute costs less than 0,3€

Adaptation rules

- An adaptation rule specifies when, where and how adaptation will be performed
  - Implements the adaptation logic
- Each rule includes:
  - A description of the activity to be adapted
  - An applicability condition (involving the state of the environment and the public state of the application)
  - The updated code for the activity
  - The set of variables needed by the new code to interact with the rest of the application
  - A state update
  - The non functional properties provided by the new activity

High speed train rule

- Description of the activity: **Take Train**
- Applicability condition: **number = IC2356**
- Updated code: code for booking and taking high speed train
- Variables: **source**, **destination**, **number**
- State update: **number = FR82**
- Non functional properties: **time = 6h23m, cost = 115€**

Triggering adaptation

- Adaptation can be either application-triggered or manager-triggered
- Different approaches can also be combined
- Tradeoff between resources consumed for adaptation and timeliness of adaptation
  - On time adaptation is more easy if checks are frequent
  - Checks consume resources
    - » both on the application and on the manager side

Application-triggered approaches

- On initialization: when the application is started
  - Good for configurable applications
  - Does not provide further evolution
  - Low resource consumption
  - Increased bootstrap time
- On wait: when the application is idle (or has low workload)
  - Optimizes performance and resource allocation
  - No guarentee of timeliness
  - May never be possible for stressed applications
- On activity enter: before starting an adaptable activity
  - The last moment where adaptation is possible
  - Ensures an updated activity is used
  - May stop the application progress

Manager-triggered approaches

- On registration: when the application registers to an adaptation server
  - Good for location-based adaptation
  - No further update
- At time intervals: after a fixed amount of time after last check
  - Ensures updates are applied within a time bound
  - Mobile applications may escape before
- On rule update: whenever a rule is added or modified, it is checked for applicability
  - Ensures new rules are applied as soon as possible
  - Good if rules are frequently changed
  - Old rules are “forgot”

Order of rule updates

- Many rules can apply to the same activity
- Applying them in a smart order can avoid useless updates
- Nondeterministic update: rules are checked for update and applied in a random order
  - The simplest strategy
  - Also natural for rules from distributed adaptation servers
  - May cause useless checks and updates
- Priority update: high-priority rules are applied first, and low-priority rules cannot undo their effects
  - Rules must be sorted according to the priority
  - Difficult for distributed adaptation servers
  - Minimum number of rule checks and rule applications

Checking for adaptation

- The adaptation manager checks the matching between a rule and an activity
  1. The two descriptions should match
  2. All the variables needed should be available
  3. The applicability condition should be satisfied
  4. The non functional properties provided by the new implementation should be better than the old ones according to the user defined preferences
- Requires interaction with the adaptable application
  - Should expose the information on the activity
  - Should expose the part of the state needed for evaluating the applicability condition
  - Should provide the comparator for non functional properties

Non functional properties and more

- Normally the check is used for ensuring improvement of non functional properties…
- …but may also work for other aims
  - Code version: to ensure progressive updates
  - Compulsory flag: applies the update regardless of the old implementation
  - …
- The comparator function has to be programmed accordingly

Enacting adaptation

- If all the conditions are satisfied then adaptation can be performed
- The new code for the activity is sent from the adaptation manager to the application replacing the old one
  - Requires code mobility
  - Requires dynamic binding
    - » The new activity is invoked by the old code
    - » The new activity can exploit the public state
- The non functional properties are updated accordingly
- The state update is applied to the state of the application
  - Only allows to change values

JoRBA

- Jolie Rule-Based Adaptation framework
- Written in the service oriented language Jolie
- A proof-of-concept implementation of our approach including
  - A skeleton for adaptable applications
  - An adaptation manager
  - A skeleton for adaptation servers

Roadmap

- Adaptation
- Dynamic adaptation
- A rule-based approach
- An introduction to Jolie
- The JoRBA prototype
- Conclusions

The Jolie language

- A language for programming service oriented applications
- Developed by ItalianaSoftware s.r.l., a spin-off of Bologna University
- Implemented as an interpreter written in Java
- Open source project, under the LGPL license
- Based on the process calculus SOCK
  - Fully formal semantics
  - Good framework for proving properties

Service oriented computing

- A programming paradigm for distributed applicati
- Created as compositions of simpler services
- A service is a software entity providing some functionality with a well-defined, publicly available interface
- Services can be looked for, invoked and composed dynamically
- The internal implementation of a service is hidden
- Need of standards for interoperability
  - Web services: XML, SOAP, WSDL, UDDI…

Jolie main features

- Takes inspiration from
  - sequential languages (Java, C) (to be not too surprising – cfr. Klint)
    - » assignment, if-then-else, while, …
  - concurrent calculi (CCS, pi-calculus)
    - » parallel composition, …
  - workflow languages (BPEL, WSDL)
    - » one-way and request-response communication patterns
- Has native mechanisms for implementing loosely-coupled interactions
  - E.g., the ones between adaptation servers and adaptable applications

Jolie communication

- Communication is a main aspect in Jolie
- Primitives for one-way and request-response communication
  - Compatible with WSDL interfaces
- May exploit different protocols
  - SOAP
  - SODEP
  - …
- Data automatically converted into/from the required format

Hello World!


```
include "console.iol"
main
{
    println@Console("Hello, world!")()
}
```

Invoking a service


```
outputPort MathService {
Location: "socket://localhost:8000"
Protocol: sodep
RequestResponse: twice
}
main
{
twice@MathService(5)(result)
}
```

Creating a service


```
inputPort MathService {
Location: "socket://localhost:8000"
Protocol: sodep
RequestResponse: twice
}
main
{
twice(number)(result)
    { result = number * 2 }
}
```

Handling data

- Jolie provides basic integers, doubles and strings together with standard operations
- Variable types are determined at runtime and can change dynamically
- Variables can be undefined
- Every Jolie variable is a vector
  - Automatically resized and packed as needed
  - Handy to deal with large amount of data
  - a refers to a[0]

Beyond arrays

- Every variable is also the root of a tree
- a.b[5].c
- Jolie supports dynamic lookup
  - Name="Bo";
    Age.Bob=25;
    println@Console(Age.(Name+"b"))()
- Whole structures can be copied with <<
- Aliases can be defined with ->
  - currStud -> Persons.Students[i]
  - currStud points to different elements when i changes

Control flow statements

- Standard (more or less…) control flow statements
  - Sequential composition: ;
  - Conditional: if (cond) {…} else {…}
  - Loop: while (cond) {…} for(init,cond,incr) {…}
- For each: foreach (var:root) {… root.(var) …}
- Built-in parallel composition: |
- Nondeterministic input choice:
  - [in1] {…}
    [in2] {…}
    [in3] {…}

Sessions

- A Jolie service is actually a session manager
- 3 possible modalities:
  - Single (default)
  - Concurrent: many concurrently active sessions
  - Sequential: interleaved sessions
- Each session has its own state
  - Variable global is shared
- Synchronization can be obtained by synchronized (lock) {code}
- Init{} procedure is executed before starting the session mechanism

Embedding

- Jolie service can embed other Jolie services
  - Executed in the same JVM
  - Their accessibility may be restricted
- Embedding can also be done at runtime
- loadEmbeddedService@Runtime(Info)(Loc)
- The code is dynamically loaded and executed

…and much more

- Error handling
  - Scopes, faults
  - Error notification
  - Termination, compensation
- Typing annotations
- Redirection
- http://www.guideonepage.com/gop/GoP.html

Why Jolie for JoRBA?

- Suitable for rapid prototyping
- Suitable for programming dynamic applications
  - Flexible data management
- Jolie communication primitives allow to implement all the interactions needed for dynamic adaptation
  - Interactions between application and adaptation manager
  - Interactions between the different activities
- Dynamic embedding good for loading new activities at runtime
  - Directly from code

Roadmap

- Adaptation
- Dynamic adaptation
- A rule-based approach
- An introduction to Jolie
- The JoRBA prototype
- Conclusions

JoRBA architecture

- An AdaptationManager coordinating the adaptation process and including the environment State
- Distributed AdaptationServers managing adaptation rules
- A set of adaptation rules
- A skeleton for applications (AbstractClient), embedding an ActivityManager and an application State
- A skeleton for activities (AbstractActivity)
- A set of activities

Writing a JoRBA application

- Start with a standard Jolie application
- Individuate the activities that can be updated
- Externalize them as separate services including AbstractActivity.iol
  - Define its non functional properties and the Comparator function
- Individuate the variables that have to be public
  - Needed by many activities
  - Needed for evaluating conditions
  - Updated by state updates
- Externalize them by inserting them in the State
- Include AbstractClient.iol in the main application

Adapting a JoRBA application

- Write an adaptation rule
  - Including the new code of the activity
- Launch an AdaptationServer loading it
  - This can be done at any time

AdaptationManager

- Keeps track of all active adaptationServers
- Embeds the environment State
- Provides two operations
  - registerAdaptationServer, for registering a new adaptationServer
  - checkForUpdate, for asking all registered adaptationServers to check whether an adaptation rule is applicable to a given activity

AbstractAdaptationServer

- When initialized, it
  - Registers to the adaptationManager
  - Loads all the adaptation rules in its own subdirectory rules
- Provides an operation checkForUpdate, that checks all its rules for applicability
  - Exploiting the checking algorithm described before
  - Exploiting embedded service Matcher for matching activity descriptions
    - » In the current implementation, only syntactic match of activity names
  - Interacting with the activityManager and adaptationManager to get the values of variables
  - Invoking the activityManager for comparing the non functional properties
  - If the rule applies, it sends the updated code as a response

ActivityManager

- Provides all the services necessary to manage activities
- During initialization, it loads all the starting activities
- Provides two operations, Run and checkUpdateNFP
- Operation Run, allowing to run a specific activity
  - Before running, the activity is checked for updates by invoking the AdaptationManager
    - » According to the “On activity enter” policy
  - If an update is found, then it is applied
    - » The old service is removed
    - » The new one is dynamically embedded
- Operation checkUpdateNFP, allowing to compare two sets of non functional properties according to user preferences

State

- Manages a public state
- Provides three operations
  - Get, to read the value of a variable
  - Set, to set the value of a variable
  - getVariableNames, to get the list of variables

Activities

- Activities are composed by their main body and their comparator
- Activities include abstractActivity.iol, that defines the basic operations an activity have to support
  - Run, to execute the code of the activity
  - getVariableNames, to get the list of variables used by the activity
  - getNFP, to get its non functional properties

Adaptation rules

- A rule named Name is composed by two files
  - NameRule.ol, defining the rules
  - NameCode.ol, defining the new code for the activity
- NameRule.ol defines
  - The elements of the rule,
  - The function for evaluating the applicability constraint
  - The function for applying the state update
- NameCode.ol is a normal activity file, including the definition of the non functional properties

The travelling application

- A simple instance of the travel application described before, just for one travel
- A main activity GoToADAPT
  - Two subactivities, TakeTrain and TakeBus
- Additional services modelling the train station and the bus station

Adaptation servers

- Two adaptation servers
- Server_1 with 1 rule
  - For booking the high-speed train
- Server_2 with 2 rules
  - For taking a taxi instead of a bus
  - For booking (another) high speed train

JoRBA at work

Roadmap

- Adaptation
- Dynamic adaptation
- A rule-based approach
- An introduction to Jolie
- The JoRBA prototype
- Conclusions

Summary

- A novel approach for dynamic adaptation
- Based on the interaction between adaptable applications and external adaptation rules
- Very flexible framework
- A prototype implementation in Jolie

Enhancing the JoRBA framework

- Descriptions of the activity
  - Allowing more general descriptions
  - Should include the purpose and the expected behavior of the activity
- Non functional properties
  - Now given by the programmer
  - Non functional properties of activities may depend on the ones of subactivities
- Updating running activities
  - Requires to build the state of the new activity from the one of the old one
  - Should the activity be restarted?
- Someone interested in working on the implementation?
  - A Klint's scientific programmer?
  - Unfortunately this is not linear algebra

A JoRBA compiler

- In the current implementation JoRBA applications are written extending the Jolie skeletons
- Another possibility is to develop a compiler
  - From a Jolie program extended with an Activity construct
  - Including informations on the non functional properties and the comparator
  - To the real Jolie code
- A similar compiler may translate an abstract rule description into a JoRBA rule

Trustworthiness

- Are the updates safe?
- Typing of rules and activities to avoid some errors
- Verification techniques for the evolving application
  - Compositional verification, proof carrying code
- Can an application trust external updates?
  - Credentials, authentication, …

Towards object orientation

- Is the approach applicable to an object oriented language?
  - Java, ABS, …
- How can objects be changed dynamically?
- How to ensure consistent updates of different objects?
- How updates influence the class hierarchy?
- How to change data types?

The end

- Main references
  - JoRBA: http://www.jolie-lang.org/examples/tgc10/JoRBAv0.1.zip
  - Jolie: http://www.jolie-lang.org/
  - HATS: http://www.cse.chalmers.se/research/hats/
  - JoRBA paper: http://www.cs.unibo.it/~lanese/publications/fulltext/tgc2010.pdf.gz
- For later questions: lanese@cs.unibo.it

The end

- Interesting papers
  - Jolie:
    Fabrizio Montesi, Claudio Guidi, Gianluigi Zavattaro:
    “Composing Services with JOLIE”. ECOWS 2007: 13-22
  - SOCK:
    Claudio Guidi, Roberto Lucchi, Roberto Gorrieri, Nadia Busi,
    Gianluigi Zavattaro: “A Calculus for Service Oriented
    Computing”. ICSOC 2006: 327-338
  - Dynamic update of object oriented systems:
    Einar Broch Johnsen, Marcel Kyas, Ingrid Chieh Yu:
    “Dynamic Classes: Modular Asynchronous Evolution of
    Distributed Concurrent Objects”. FM 2009: 596-611