An Approach to the Dynamic Evolution of Software Systems

Manuel Oriol
June 28th, 2006
RoadMap

• Motivation
• Toward Disconnection
• General Infrastructure
• Service Descriptions
• Experiment Reports
• Conclusion
Motivation

• Motivation
• Toward Disconnection
• General Infrastructure
• Service Descriptions
• Experiment Reports
• Conclusion
Evolution of Applications

- Applications are often modified over their lifetime.
- It has been said that an application becomes mature after several years of maintenance.
- Though there are different types of evolution:
  - Marshaled / unmarshaled evolution.
  - Dynamic / static evolution.
  - Anticipated / unanticipated evolution.
Unmarshaled Evolution

• A marshaled evolution is an evolution constrained by invariants.
  - Real Life (RL) Example: a chair will always be used to seat.
  - Object-Oriented Programming (OOP) Example: subtyping constraints for polymorphism.

• Unmarshaled evolution has no such constraint.
  - RL Example: a chair may be modified to be transformed into a table.
  - OOP Example: redesigning an application.
Dynamic Evolution

• A static evolution is an evolution that is made without the application running.
  – RL Example: the original plans of the chair are changed.
  – OOP Example: stopping an application, modifying its code and relaunching it.

• A dynamic evolution is an evolution that is made while the application runs.
  – RL Example: repairing, modifying a chair.
  – OOP Example: only few, dynamic loading
Unanticipated Evolution

• An anticipated evolution is made when some parts of an application are evolvable
  – RL Example: a chair that may have diverse different colored legs.
  – OOP Example: interfaces.

• An unanticipated evolution is an evolution that may concern the whole application and have not been foreseen at design time.
  – RL Example: a chair that looses its back and becomes a stool (or the reparation).
  – OOP Example: a bug of security that modifies the general behavior of the application (or its correction).
Focus

• We decided to work on unmarshaled unanticipated and dynamic evolution of object-oriented applications.

• In few words, we want to offer to programmers an infrastructure for handling such evolutions in order to allow to build highly available applications.
Applications

• At the moment most applications are programmed using an object-oriented language.

• More and more systems controlled by software need to run continuously:
  – Example: PDA, mobile phones, car systems, satellite control systems, nuclear power plant systems...

• These systems need to be dynamically upgraded!
Toward Disconnection

- Motivation
- Toward Disconnection
- General Infrastructure
- Service Descriptions
- Experiment Reports
- Conclusion
Traditional Object-Oriented Applications
Problem?
Connections
What to do?
Remove Connections!

An Approach to the Dynamic Evolution of Software Systems
General Infrastructure

• Motivation
• Toward Disconnection
• General Infrastructure
• Service Descriptions
• Experiment Reports
• Conclusion
Entities and Service Manager

Service Manager

Entities
Announcing Services
Invoking Services

Service Manager

T

?
Returning an Answer
Evolution: A Target Disappears

Service Manager

? T

T

?
Evolution: A Target Evolves
Evolution: A Caller Evolves
Model Properties: Anonymity and Associative Naming
Model Properties: Asynchrony
Service Descriptions

• Motivation
• Toward Disconnection
• General Infrastructure
• Service Descriptions
• Experiment Reports
• Conclusion
How to Describe Services?

• Functionality:  
  What does it do? (comparable to a method name)

• Behaviour:  
  How does it do it? (comparable to method signature)

• QoS:  
  How well does it do it? (comparable to the description of a method in an API)
An Approach to the Dynamic Evolution of Software Systems

Trees

- Root
- Node1
  - Node21
  - Node22

Precision AND Relation

OR Relation
Comparing Trees?

The same Number?

number of successive common nodes?

An Approach to the Dynamic Evolution of Software Systems
Matching Depth (1)
Matching Depth (2)
Matching Depth (3)
Service Descriptions

Diagram showing a tree structure with nodes labeled as follows:
- Functionality
- Behavior
- QoS

Each node has multiple children, labeled with identifiers such as F1, F2, ..., Fn, B1, B2, ..., Bq, Q1, Q2, ..., Qr, and so on, with further levels of sub-nodes like F11, F12, ..., Fnm, B21, B22, Q11, Q1t, etc. The diagram illustrates the dynamic evolution of software systems.
Matching Service Descriptions

- We match each tree separately and the best adapted service is the one whose service description matches the best with the required service descriptions. (F, then B, then Q)
- We impose to have minimal values while matching, to guarantee a minimal adequacy: if no service description is precise enough, nothing occurs and the caller is notified.
Matching Service Descriptions

(F « Sort »)

(B « SlowSort »)

(QoS « Slow » « Fast » (2, 5, 2))

(argument int [] char [] return int [] char [])
Experiment Reports

• Motivation
• Toward Disconnection
• General Infrastructure
• Service Descriptions
• Experiment Reports
• Conclusion
Implementation: LuckyJ

- Allows arbitrary changes to be made in classes and objects used in an application.
- Considers components as the units of evolution of the application.
- Uses a custom communication model based on services and their descriptions.
LuckyJ Characteristics

- Built on top of a standard JVM (Java 2).
- Each entity has its own ClassLoader.
- Decoupling between Description Passer and Service Manager (possibility to modify the matching module at runtime).
- A lightweight core platform (5000 lines of code).
Other Implementation

- 2 distributed implementations:
  - A centralized distributed one
    - Allows to distribute a LuckyJ implementation at runtime
    - Organised in a tree-like structure.
  - A semi-centralized one (Stadler 2003)
    - Client peers
    - Server peers
Applications

• Web server WeeselJ:
  - http://www.weeselj.org
  - Implementation online for the last 18 months.
  - Up to 160 versions of some parts of the code.
  - 99.99% availability (4 restarts of the application).

• Tic-Tac-Toe
  - Open Days implementation.
  - Dynamically recoded versions for participants.
  - Student work on top of LuckyJ.
Conclusion

• Motivation
• Toward Disconnection
• General Infrastructure
• Service Descriptions
• Experiment Reports
• Conclusion
Conclusion

• We presented our architecture that addresses the problem of services in very dynamic environments through associative naming, late binding and asynchrony.

• We give a way of quantifying how well (possibly complex) descriptions match and prefer some services to others.

• And... It works quite well. ;)}
Future work

- Correctness???
- Self-Organizing Applications ???
- Semantic Service Descriptions ???
- Object-level Evolution ???