A Multi-Agent Reflective Architecture for User Assistance

Antonella Di Stefano, Giuseppe Pappalardo,

Corrado Santoro, Emiliano Tramontana

University of Catania - Italy

Motivations

- Supporting the evolution of applications by enriching them with assistant agents:
  - Extending existing applications *without embedding* code implementing assistance tasks into their source code
  - Clearly *separating* applications and assistants, making applications unaware of assistants and assistants easily reusable for various applications
- Providing an architecture that *interfaces* several special purpose assistants to an application independently of specific access points

Computational Reflection

- A *reflective* system embeds some structures that represent its own aspects, which allow it to act on itself
  - Actions are performed by means of two mechanisms: *introspection* and *interception,* together they provide *reification* to a system
- A reflective system is generally structured as a two-level system
  - *baselevel* (application)
  - *metalevel* (assistance activity)

Computational Reflection

- Characteristics of reflective systems
  - *Transparency*: objects at the baselevel are not aware of metalevel objects
  - *Separation of concerns*: each level deals with a different aspect
- Connection between baselevel and metalevel
  - Some objects at the metalevel (said *metaobjects*) observe the behaviour of objects
  - *Metaobjects* capture some operations of objects, execute some computation and then hand control to objects letting them perform their operations

Interfacing Assistants

- The interface between an application and assistants is realised by means of a reflective system:
  - Baselevel holds objects that implement an application
  - Metalevel holds two types of agents:
    - ***Coordinator*** that captures control from application objects, notifies proper assistants, pours results of assistants to the application and allows exchanging data between assistants
    - ***Assistants*** that implement specific tasks by using Inference Engines and Knowledge Bases

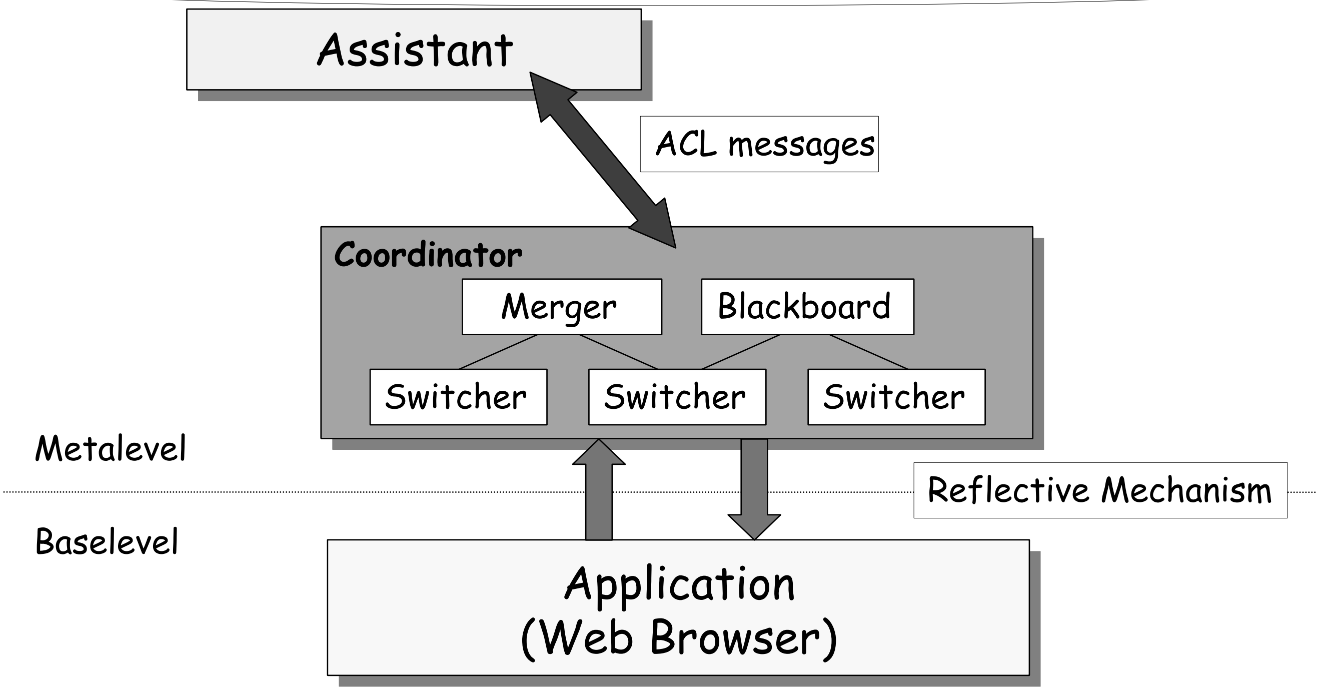
Coordinator

- Coordinator allows assistants to be plugged into the application according to user needs
  - It knows only that some assistants are interested on application events and that some data are exchanged
  - It does not need to know the number of assistants nor their tasks
    - assistants can be created even after the Coordinator
  - It is able to intervene to modify the behaviour of the application
  - It enables independent assistants to work cooperatively and share results

Coordinator

- Coordinator consists of:
  - **Switcher**s *detect* application events and pour results of assistants by interacting with application objects
    - use the capability of metaobjects to trap control from associated objects and to detect the context of events
  - **Merger** receives all the application events and *notifies* interested assistants
    - works, as in the *Observer* design pattern, by handling a list of event observers (i.e. assistant agents)
  - **Blackboard** is a repository that allows assistants to exchange their outcomes
    - It exploits the *Blackboard* architectural style

The Architecture



Assistant

ACL messages

Coordinator

Merger

Blackboard

Switcher

Switcher

Switcher

Metalevel

Reflective Mechanism

Baselevel

Application
(Web Browser)

Model of Assistant Agents

- Assistant agents are composed of:
  - Inference Engine
    - provides assistant agents with reasoning ability
    - works by processing rules that depend on the application and the assistance task
  - Knowledge Base
    - stores facts that the Inference Engine generates
  - User/Agent Interface
    - interacts with users to provide results and accepts inputs

Constructing the Architecture

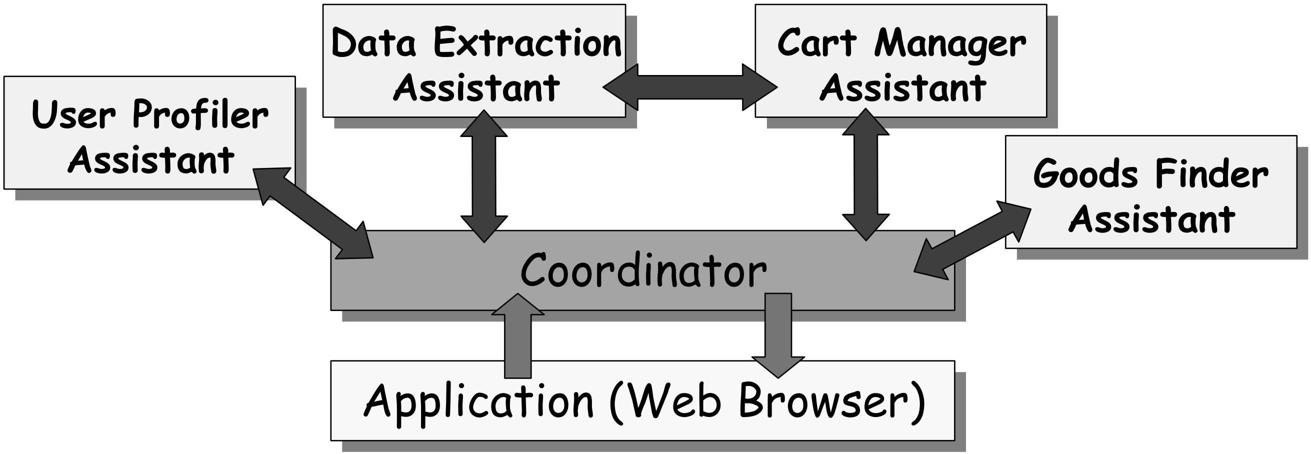
- Three hypotheses to employ the architecture:
  - The application is implemented in an object oriented language
    - this enables the metaobject model to be used
  - The source or the Java bytecode of the application is available
    - this allows necessary hooks to be inserted to capture events of the application
  - Some knowledge of the application is available
    - this makes it possible to understand which objects and methods implement the events of interest

Constructing the Architecture

- The programmer may take the following steps:
  - ***Identifying*** the set of events that should trigger the work of assistant agents
  - ***Understanding*** how the application handles the selected events, by establishing which methods are involved with them
  - ***Connecting*** the application objects handling the identified events with the Coordinator, thus associating them with some metaobjects (**`Switcher`**)
  - ***Mapping*** the output of Assistants onto actions on the application (**`Switcher`**)

Assistants for E-Commerce

- Assistants are designed to help the users of a web browser performing e-commerce, by reacting to application events and working autonomously



Assistants for E-Commerce

- ***User Profiler Assistant*** understands user preferences from visited web pages
  - it is informed by *Coordinator* when a new web page has been visited
    - a **`Switcher`** traps this application event and informs **`Merger`**
    - **`Merger`** notifies the assistant
  - it uses a set of page categories and a set of weighted keywords for each category to classify pages [Mase98]
  - as outcome, it provides to Coordinator a *web profile* of the user that consists of a ranked list of keywords
    - the **`Blackboard`** stores the web profile
    - a **`Switcher`** uses the web profile to modify the colour of keywords on visited web pages

Assistants for E-Commerce

- ***Data Extraction Assistant*** stores data on goods by extracting them from visited web page
  - it is informed by *Coordinator* that a new page has been loaded
  - it uses the *web profile* to select interesting goods
  - it builds a structured version of the data of a web page (by using an ontology)
  - its outcome is a ranked list of goods where the most accessed data come first
- ***Cart Manager Assistant*** handles a virtual cart that compares potential user's purchases
  - stores sensitive and personal data on the client side
  - provides a common repository of data from different web sites
  - enables the user interact through a graphical representation

Assistants for E-Commerce

- ***Goods Finder Assistant*** searches on the web offers for user selected goods
  - accesses web pages where goods can be found
  - analyses web pages looking for interesting goods
  - asks Data Extraction Assistant to gather new data from selected web pages
- ***Goods Monitor Assistant*** watches the trend of prices of user selected goods
  - periodically accesses known web pages
  - asks Data Extraction Assistant to gather data for a good
  - updates the price of the good

Implementation Issues

- An object oriented application whose source or Java bytecode is available and whose design and implementation are (partially) known can be extended with assistant agents
  - For Java applications, the agent framework JADE can be used to implement assistants
  - For C++ applications, CLIPS (C-Language Integrated Production System) can be used to implement assistants

Performance Issues

- The performance penalty introduced can be tuned:
  - The overhead due to the computation of Assistants
    - is reduced by caching results into metaobjects, and giving assistants the ability to work *asynchronously* from the application
  - The cost of jumping to the metalevel
    - is reduced by carefully choosing the intercepted operations
  - The overhead due to the transformation of bytecode of application classes
    - can be paid just once, since the reflective abilities can be added to bytecode permanently

Conclusions

- The proposed architecture integrates several assistants into applications by means of reflection:
  - Is independent of hooks provided by applications or OS
  - Lowers complexity
    - Reduces difficulties when developing both assistants and applications
    - Makes applications not aware of assistance issues
    - Allows both applications and assistants to be developed and evolved independently, without affecting each other
  - Enables assistants to be plugged in just when needed
  - Allows assistants to be reused for several families of applications