Complex Systems and Agent-Oriented Software Engineering

Juan Pavón[1], Francisco Garijo[2], Jorge Gómez Sanz[1]
[1]Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense Madrid
[2]Telefónica I+D

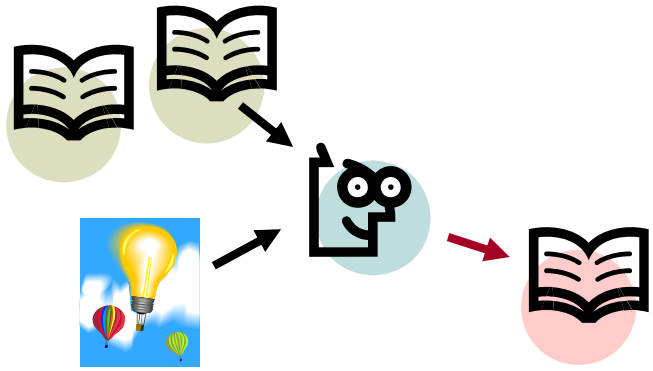
http://grasia.fdi.ucm.es

Points of discussion

- How Agent Oriented Software Engineering (AOSE) contribute to the development of complex software systems?
  - Better than other paradigms?
  - Do we have examples?
  - How to measure this?
- The gap between academia and industry
  - How can AOSE get more acceptance?

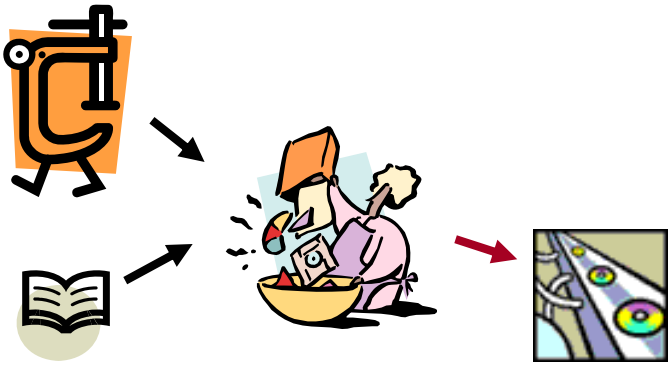
Doing software engineering – the Academic way

- The scientist
  - Collects and contrast information
    - Journal and conference papers, books, …
  - Synthesis of papers and experimentation
  - Results: modeling languages, methodology, examples, some tools, PAPERS

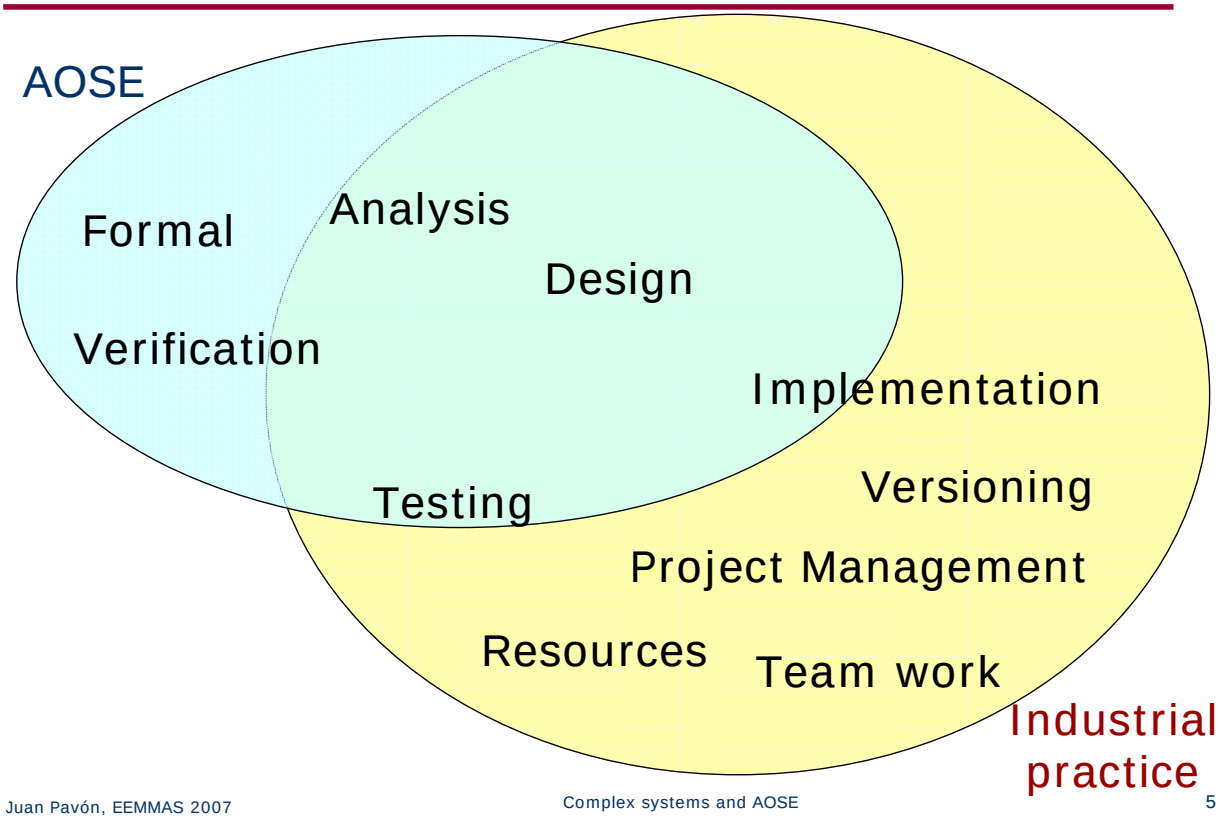


Doing software engineering – the Industrial way

- The engineer
  - Analyses, designs, implements, tests
    - Systems, services, applications
  - Synthesis of successful practices
  - Results: products, recommendations, practices, tools, reports (almost no papers, scarce dissemination)



Focus areas



What about other approaches

- Component based frameworks (J2EE, .NET, etc.)
- Model Driven Software Development
- Aspects
- Software Product Lines
- Service centric systems
- ...

These should be integrated with AOSE

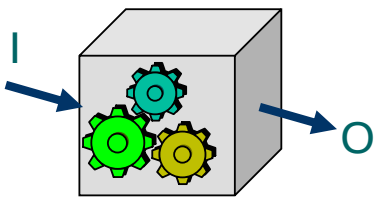
Some issues towards AOSE integration

1. The role of MAS in the development of complex systems
2. Agent-based Software Architectures
3. Agents in the Software Process
4. Metrics of Agent-based Software

- There are many other issues

MAS for the development of complex systems

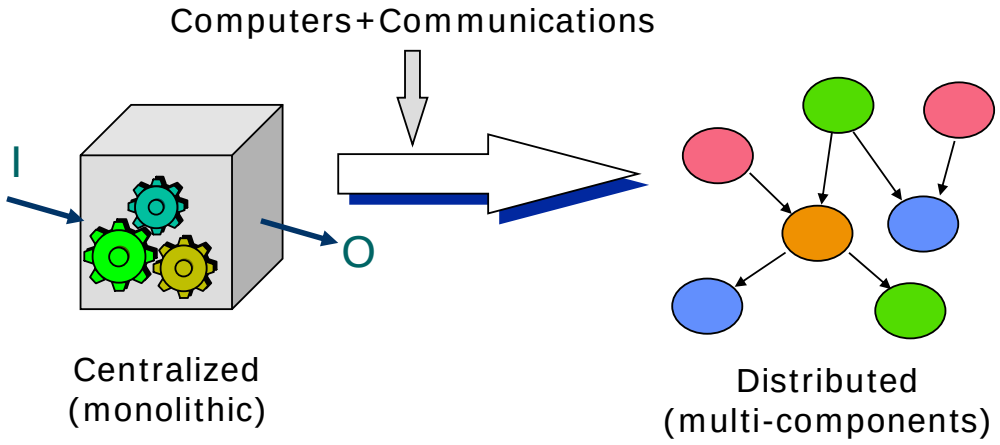
- Software engineering as a reaction to the software crisis (late '60)
  - *Machines have become several orders of magnitude more powerful* [Dijkstra 1972]
  - Waterfall process model, formal methods, structured programming, software metrics, …



Centralized
(monolithic)

MAS for the development of complex systems

- Evolution to object oriented computing
  - Networking
  - Graphical user interfaces
- Objects, client-server, interfaces, …



MAS for the development of complex systems

- Evolution of complexity in the environment
  - From one-to-one interactions to many-to-many interactions
    - Many parallel sessions
    - Some interactions may depend on others (e.g. negotiation for best offer)
  - Partial knowledge of context
    - Continously changing, partially observed, partially understood
    - Example: Which services are available, how to access them, what quality of service, …
  - Bounded rationality
    - *boundedly rational agents experience limits in formulating and solving complex problems and in processing (receiving, storing, retrieving, transmitting) information* [Herbert Simon]
  - Uncertainty
    - Resources availability, failures, security risks, …
  - New opportunities
    - More service providers, more clients

MAS for the development of complex systems

- Complex environment requires ability to adapt
  - Centralized control: Dependent => Objects
    - Predefined behaviour, ad-hoc mechanisms
    - Subject to communication failures
    - Subordination
  - Distributed control: Autonomy => Agents
    - Ability to act locally
    - No global knowledge
    - Cooperation

MAS for the development of complex systems

- Other important environmental aspect: the development process perspective
  - Multi-disciplinary teams
  - Continous evolution of software products
  - Customization of products
  - Organizations

Agent-based Software Architectures

- Patterns
  - *From idioms that shape the use of a particular programming language to mechanisms that define the collaboration among societies of objects, components, and other parts*
    [Booch, Handbook of Software Architecture]
  - Cumulative experience of software practicioners
  - Reuse of well-proven solutions

- A system architecture enforces the use of a set of patterns
  - Behaviour principles
  - System structure
  - Separation of concerns
  - *Guideline for identification of relevant system features and application of patterns*

Agent-based Software Architectures

- What agent-based architectures?
  - Micro level: Classical agent architectures
    - Reactive, cognitive, hybrid
  - Macro level: Organization architectures
    - Emergent vs. Predefined organizations
    - Electronic institutions, Gaia, INGENIAS, TAEMS, AGR, AMAS, ...

- They have to be *usable, replicable, flexible, robust*

Agent-based Software Architectures

- Classical examples

  [taken from M. Wooldridge, *Intelligent Agents*, in G. Weiss (ed.) *Multiagent Systems-A Modern Approach to Distributed Artificial Intelligence*, MIT Press 1999]

Generic BDI architecture

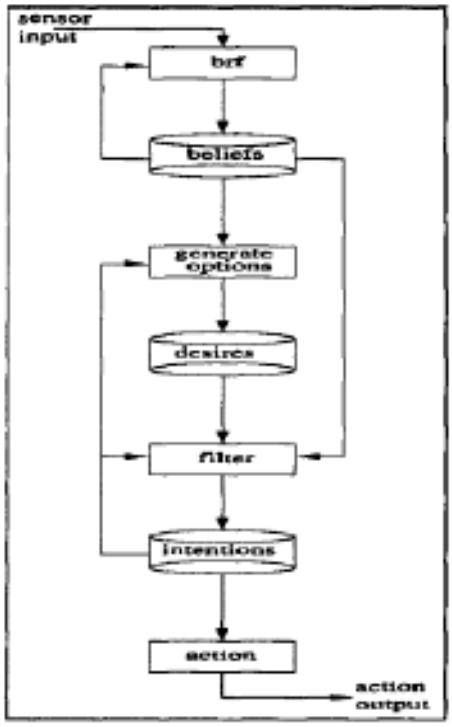


Figure 1.5
Schematic diagram of a generic belief-desire-intention architecture.

Agent-based Software Architectures

- Classical examples

  [taken from M. Wooldridge, *Intelligent Agents*, in G. Weiss (ed.) *Multiagent Systems-A Modern Approach to Distributed Artificial Intelligence*, MIT Press 1999]

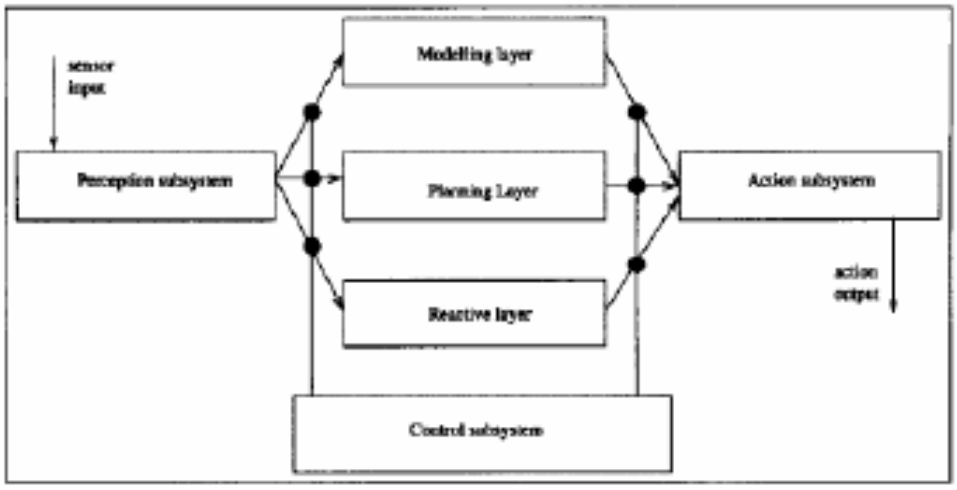


Figure 1.7
TOURINGMACHINES: a horizontally layered agent architecture

Agent-based Software Architectures

- Examples [taken from this conference]

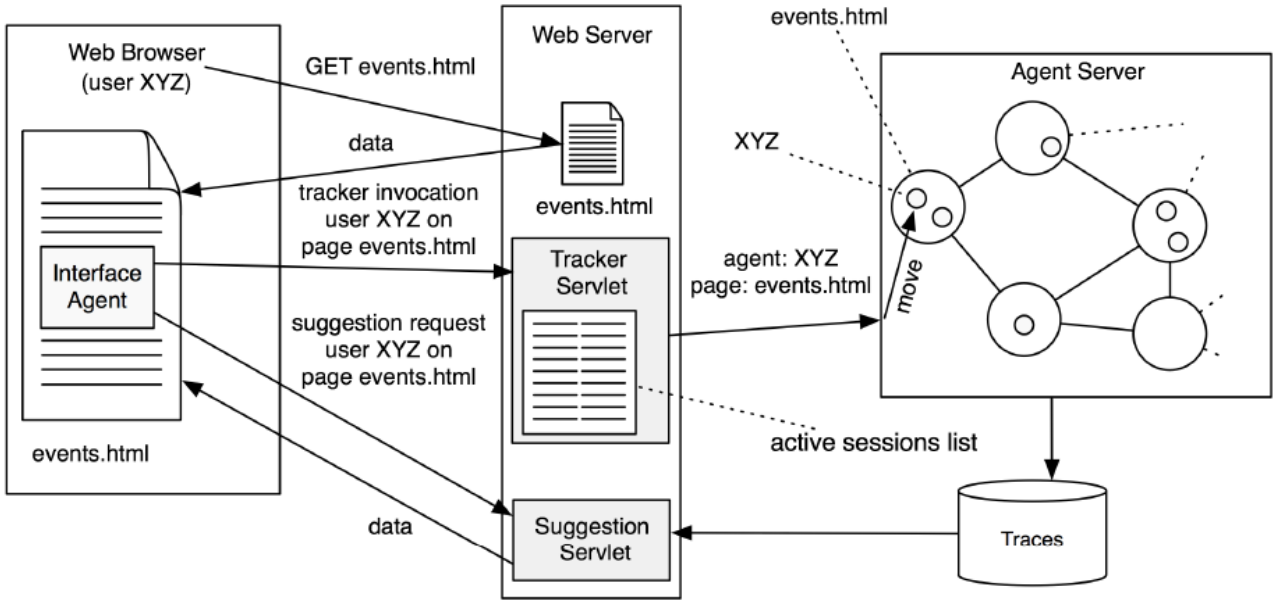


Fig. 1. A diagram showing the different components of system architecture and their interactions.

Agent-based Software Architectures

- Examples [taken from this conference]

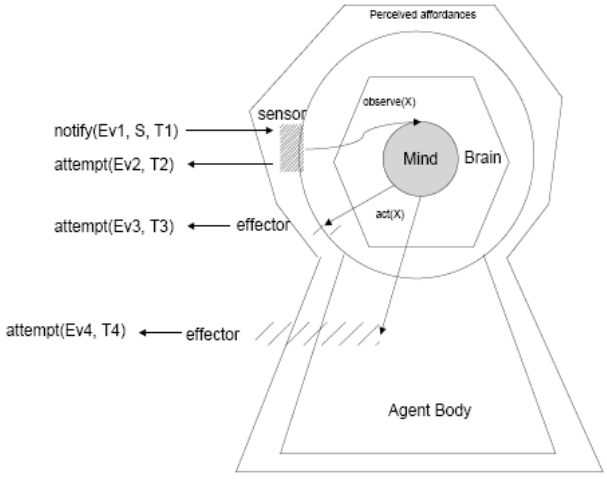


Fig. 3. The Agent Architecture in GOLEM (adapted from [11]).

Agent-based Software Architectures

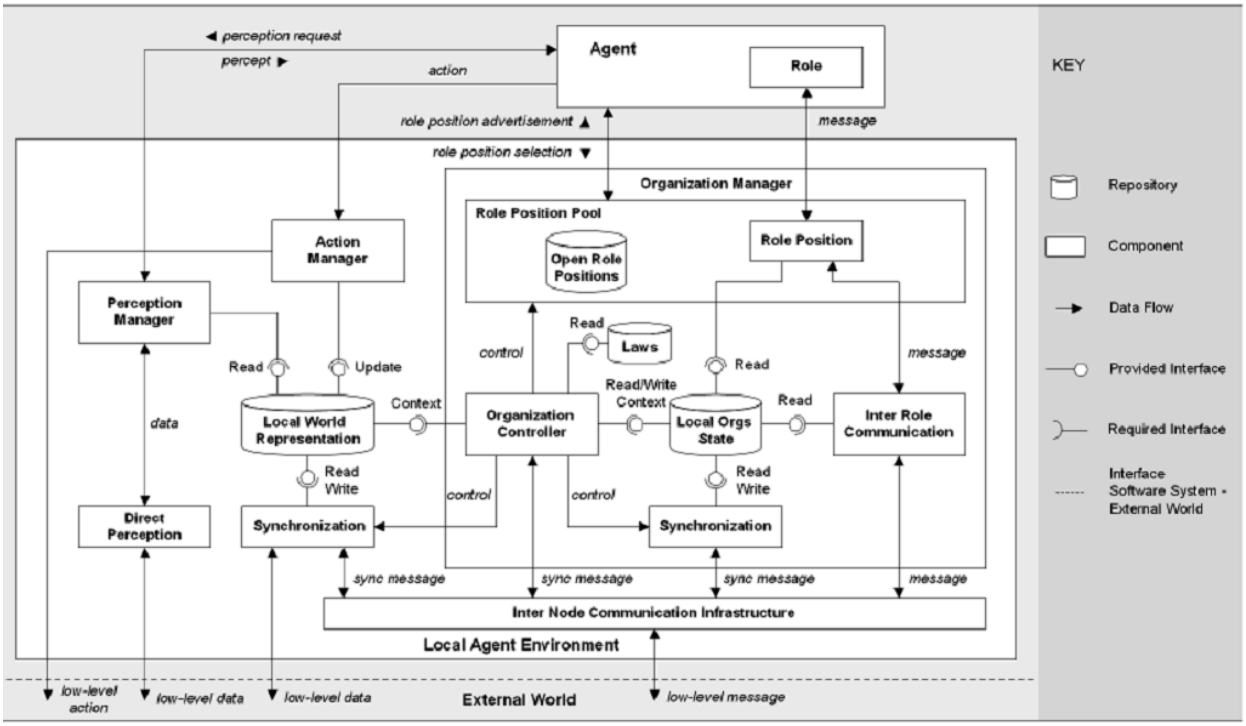


Fig. 7. Collaborating Components view of the software architecture for context-driven dynamic organizations.

Agent-based Software Architectures

- Examples [taken from this conference]

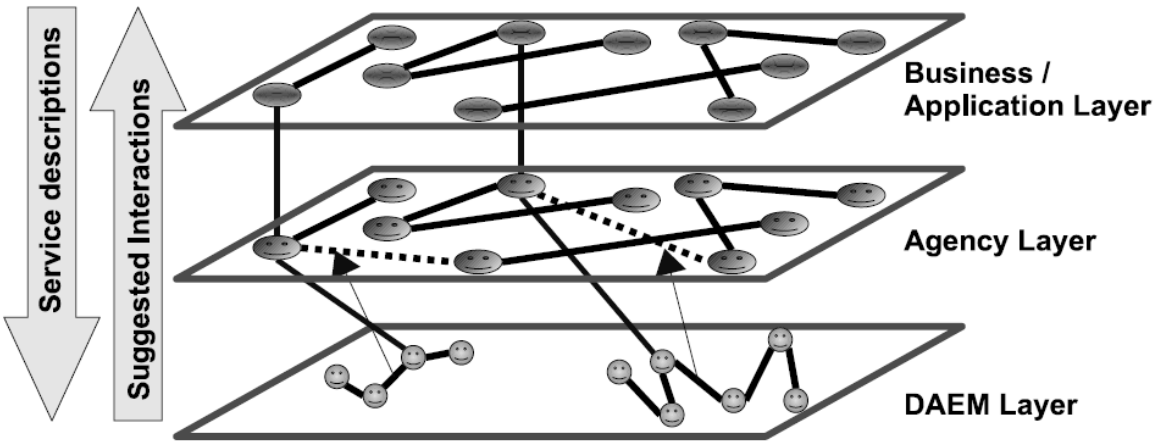


Figure 1. Business ecosystem conceptual architecture.

Agent-based Software Architectures

- An example of pattern application: TID Agent Framework
  - This is the result of cumulative experience, implementing several agent-based applications:
    - Cooperative working system for network bandwidth negotiation
    - Project network management for IN services development
    - Flight notification system (MESSAGE project)
      - Reactive and cognitive agent architectures
      - Communication session mechanisms
      - Planning
    - Web service personalization
    - On-line discussion and decission making
      - Scalability of the cognitive agent model
      - Agent management
    - Voice recognition services
      - Refinement
  - Currently being used in two European projects: SECSE and MOMOCS

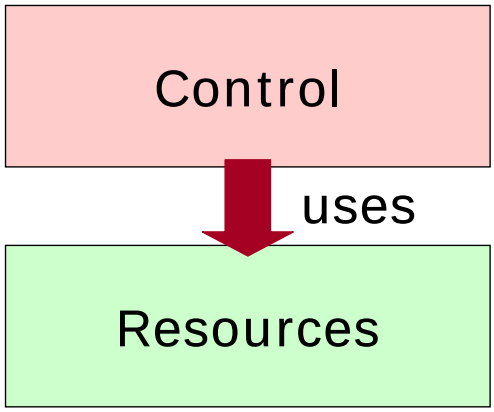
TID Agent Framework

- Basic principle
  - An **organization** made up of **agents** and **resources**

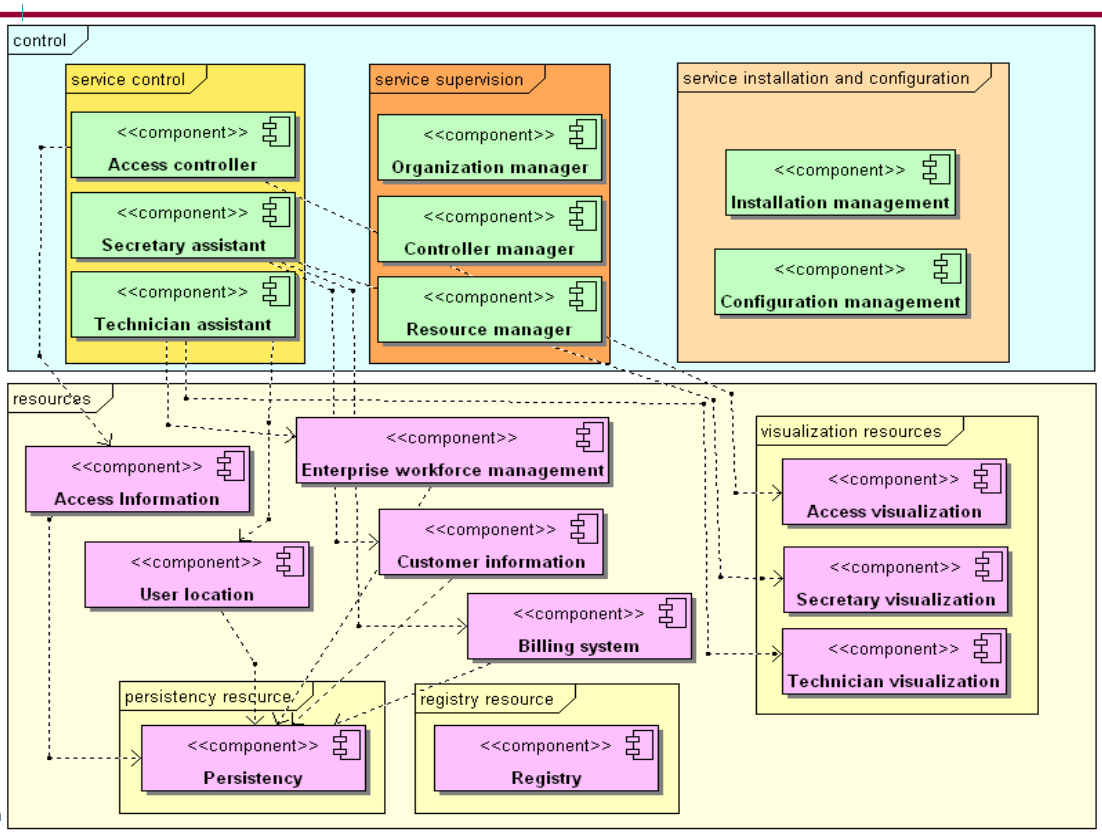
TID Agent Framework

- Two layers architecture
  - Control layer: Agents (controller components)
    - Managers: service mngt
    - Specialists: service functionality

    Similar interfaces and structure
    But different roles
  - Resource layer: Resources
    - Persistency
    - Registry
    - Visualization
    - Support functionality



TID Agent Framework



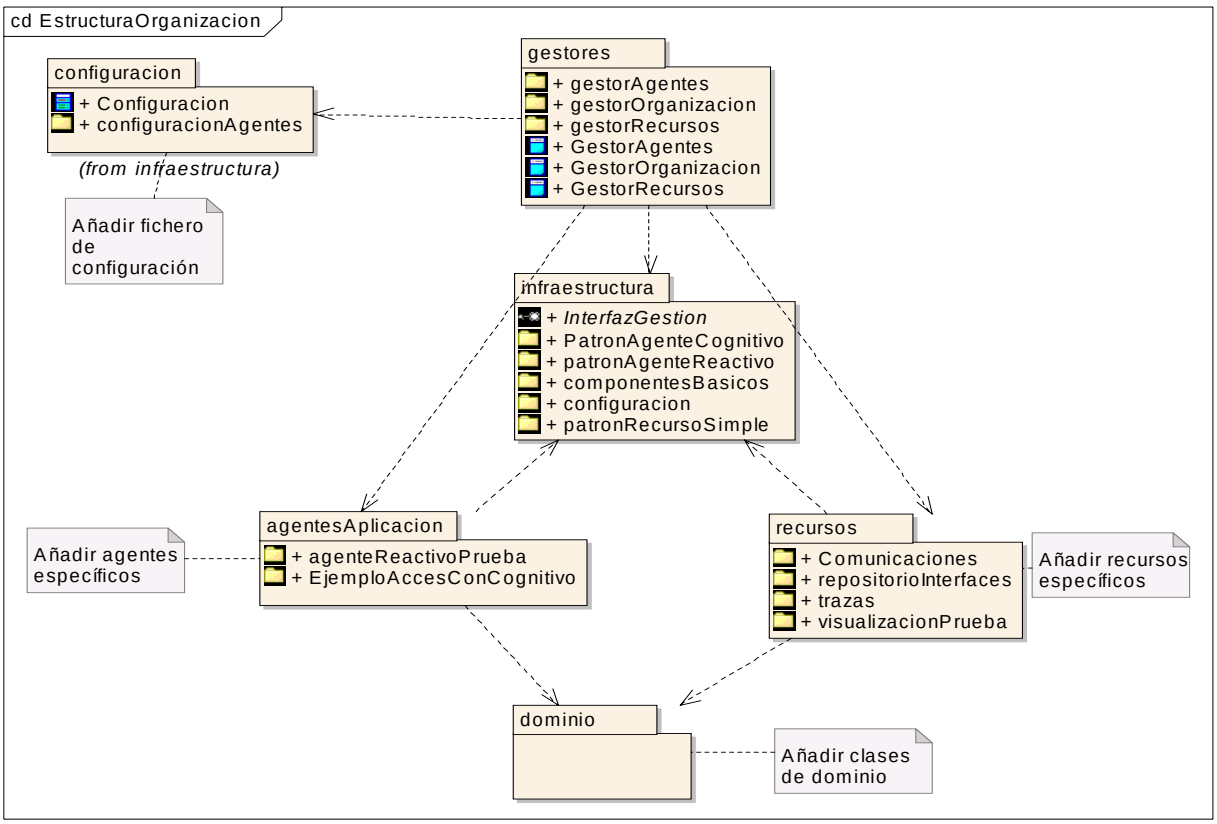
TID Agent Framework

- Management Patterns (life-cycle, management organization)
  - Every component has a management interface
    - Developers are forced to make manageable components
  - Management components (agents) provide a reusable way to cope with common problems
    - Installation
    - Configuration
    - Monitoring

    of agents and resources

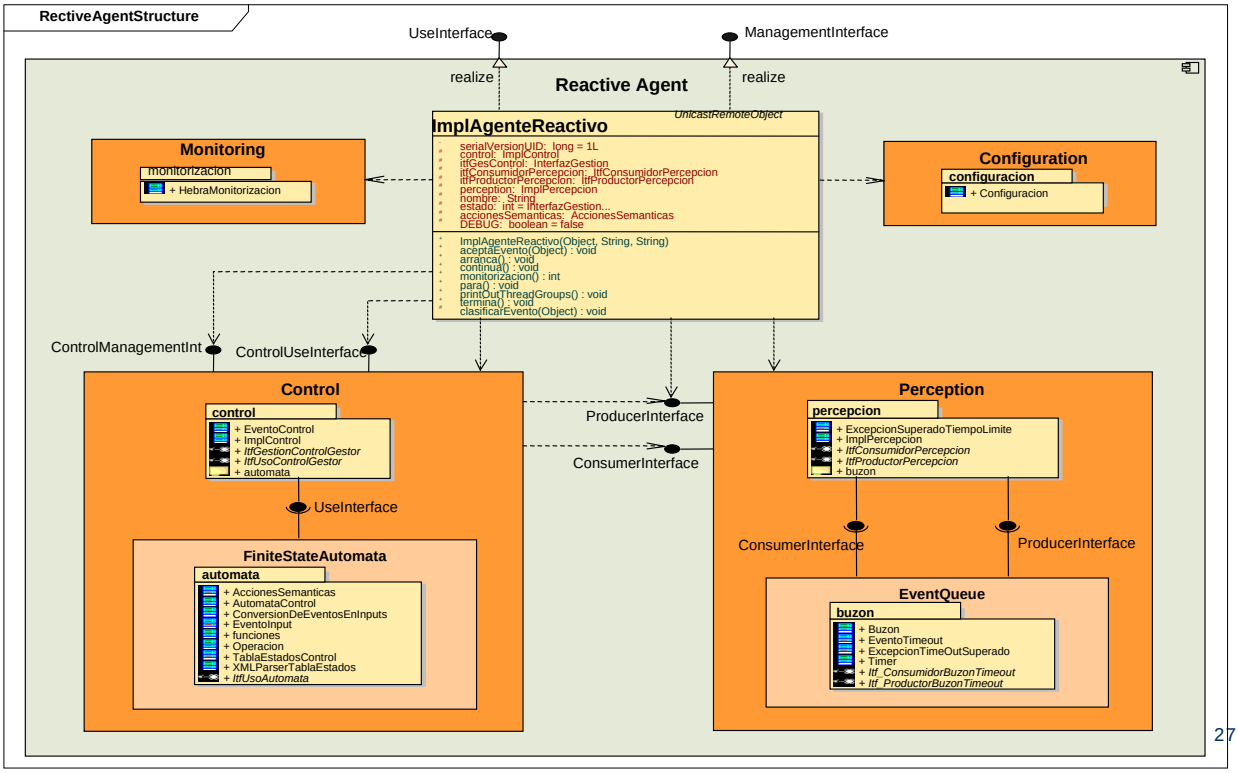
    The framework takes care of them

TID Agent Framework



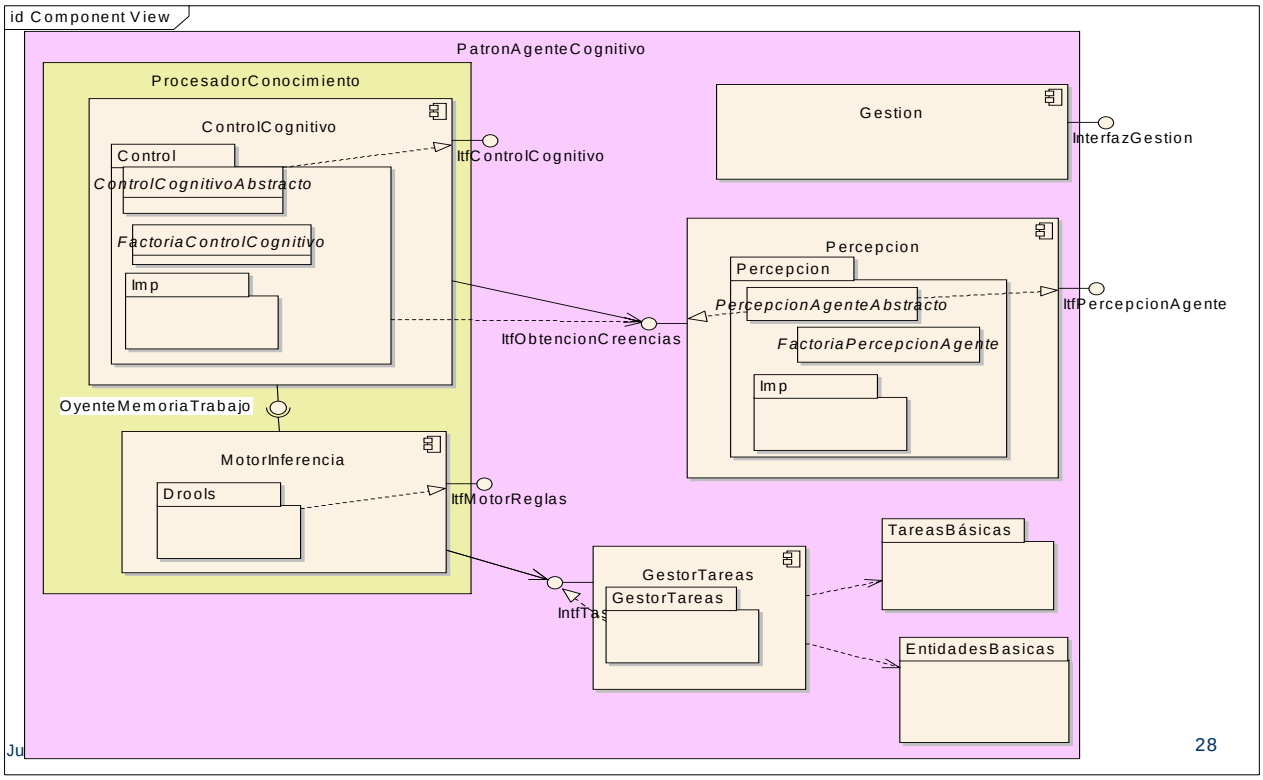
TID Agent Framework

- Agent Patterns: Reactive agent pattern



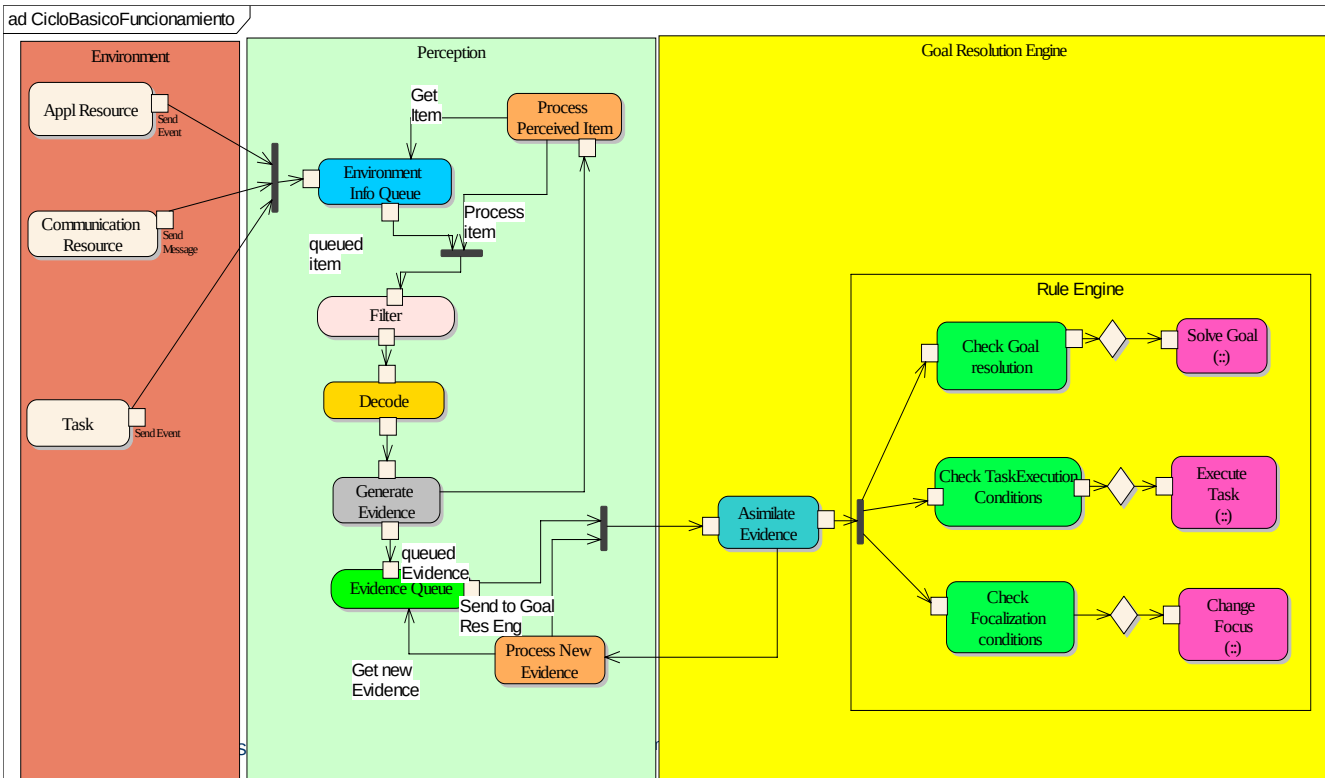
TID Agent Framework

- Agent Patterns: Cognitive agent pattern



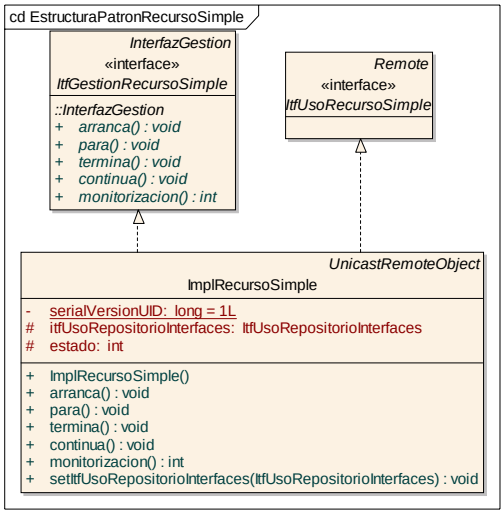
TID Agent Framework

- Agent Patterns: Cognitive agent pattern



TID Agent Framework

- Resource Patterns
  - Encapsulation of computing entities providing services to agents
    - Message oriented middleware
    - Transaction monitors
    - Security and authentication services
    - Information services
    - Databases
    - Visualization
    - Speech recognition and generation
  - Two interfaces
    - Usage
    - Management



TID Agent Framework

- A ***component-based*** MAS architecture
  - Software entities are categorized either as agents or resources
    - This implies a clear design choice for the developer
  - Environment is modelled as a set of resources
    - Standard patterns and mechanisms to facilitate their access
  - Relevance of management patterns for agents and resources
    - Relieves the developer of a considerable amount of work
    - Guarantees that components will be under control
    - Enforces a pattern for system initialization
  - Agents work as autonomous entities and encapsulate their behaviour (reactive, cognitive, hybrid) behind their interfaces
  - Interactions can be defined at application level
    - Independently of underlying middleware

Agents and Software Project Management

- The basic elements of a software project
  - Persons
    - Difference in skills of team members
    - Changes in team composition
    - Organization structure
    - Corporate culture
    - Development strategies and tactics
  - Process
    - Sequential, evolutionary, agile, etc.
    - Risk management
    - Software quality assurance
  - Product: the MAS

Agents and Software Project Management

- There is not too much about process management in AOSE methodologies
- But there are implications of the agent paradigm in project management
  - Task distribution is easier
    - Agents and resources can be developed indenpendently as interfaces are well defined
  - Promotes evolutive/incremental/explorative iterations
    - It is relatively easy to change the internal processing of the agent as well as interactions
  - Conflicts management
    - It is easier to locate and delimite responsibilities
  - Less integration problems
    - Great modularization and encapsulation

Metrics for MAS

- How much does it cost the development of a MAS?

Metrics for MAS

Estimating Costs for Agent Oriented Software, AOSE 2005

- Using experience in previous developments
  - Decomposing the product to be built into smaller pieces and estimate the cost
  - Decomposing the development process to be applied into fine grain activities and estimate the cost for each one
- Empirical models: COCOMO II, Putnam
- Use several methods and evaluate the difference in the estimations

Metrics for MAS

- COCOMO II
  - An empirical estimation model developed by Boehm

$$PM = A \cdot (Size)^{1.01+\sum_{j=1}^{5} SF_j} \cdot \prod_{i=1}^{17} EM_i$$

PM: Personnel Month
SF: Scale Factor (VL,L,N,H,VH,XH)
EM: Effort Multiplier (VL,L,N,H,VH,XH)
A: a constant to adjust to local domain
Precision: 30 % of actuals 75% of the time

Metrics for MAS

- Three projects
  - Eurescom P815. Communications Management Process Integration Using Software Agents (1999-2000)
  - Eurescom P907. MESSAGE: Methodology for Engineering Systems of Software AGEnts (2000-2002)
  - PSI3. Personalized Service Integration Using Software Agents (2001-2003)

Project complexity	P907	P815	PSI3
Number of classes	172	482	130
Number of packages	31	45	23
Average methods per class	4.09	5.17	5.3
SLOC Logical lines	5393	15843	9862
SLOC Physical lines	7007	20009	13102

Metrics for MAS

SF	P815	P907	PSI3
Precedentness	Nominal	High	High
Development Flexibility	High	High	Nominal
Architecture/ Risk resolution	Extra High	Extra High	Nominal
Team Cohesion	Hi	Very High	Very High
Process Maturity	Low	Nominal	Nominal

EM	P815	P907	PSI3
Product Reliability and Complexity (RCPX)	High	Nominal	Very High
Reusability (RUSE)	Very High	Very High	Nominal
Platform Difficulty (PDIF)	High	Very High	High
Personnel Capability (PERS)	Nominal	Nominal	Nominal
Personnel Experience (PREX)	Low	Nominal	High
Facilities (FCIL)	Nominal	High	High
Required Development Schedule (SCED)	Nominal	Nominal	Nominal

Metrics for MAS

- Measuring P815 cost according to pure COCOMO II
  - SF and EM were those previously presented.
  - Early design model

Project Name: p815 | Scale Factor | Schedule | Development Model: Early Design

X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK
	AI part(rules)	S:2130	0.00	2.21	AI Shell	7.3	16.1	132.5	0.00	0.0	1.0	0.0
	OO part	S:15843	0.00	2.21	Object-Orient	54.1	119.5	132.5	0.00	0.0	7.2	0.0

Total Lines of Code: 17973

Estimated	Effort	Sched	PROD	COST	INST	Staff	RISK
Optimistic	90.9	14.7	197.8	0.00	0.0	6.2	
Most Likely	135.6	16.7	132.5	0.00	0.0	8.1	0.0
Pessimistic	203.4	18.9	88.4	0.00	0.0	10.8	

Metrics for MAS

- Adapting to agents
  - The way we adapted COCOMO II to the agent domain was by means of the ***size*** variable
  - Following the same approach as with Function Points estimation, we provided empirical equivalence between agent concepts and SLOC

- Two approaches
  - Method 1.
    Measuring only parts relevant to agent implementation.
    (It was facilitated due to the structure of the application: organized into packages)
  - Method 2.
    Obtaining the average SLOC per Item, just as it is done for Function Points estimation

Metrics for MAS

Measurement of agent concepts	P815	P907	PSI3
Total number of Interactions with other agents	3	5	4
Total number of messages interchanged	15	19	11
Total number of events considered	61	10	10
Total number of rules	198	48	39
Total number of tasks	71	9	39
Total number of state machines applied	5	10	8
Total number of states in every state machine	13	46	37

Metrics for MAS

Measurement of BDI concepts	P815	P907
Total number of types of mental entities (F=Facts, G=Goals, E=Events)	8F+135G+61E= 204	3F+29G+10E= 42
Total number of Goals	135	29
Rules dedicated to management of mental entities	190	46
Events	61	10

Metrics for MAS

- Recomputing costs for P815. Method 1
  - Early design model

Project Name: p815 | Scale Factor | Schedule | Development Model: Early Design

X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK
	tasks	S:793	0.00	1.00	Object-Orient	2.5	2.5	312.6	0.00	0.0	0.3	0.0
	goals	S:1581	0.00	1.00	AI Shell	5.1	5.1	312.6	0.00	0.0	0.6	0.0
	state machines	S:142	0.00	1.00	Object-Orient	0.5	0.5	312.6	0.00	0.0	0.1	0.0
	rules	S:2130	0.00	1.00	AI Shell	6.8	6.8	312.6	0.00	0.0	0.8	0.0
	events	S:443	0.00	1.00	Object-Orient	1.4	1.4	312.6	0.00	0.0	0.2	0.0

Total Lines of Code: 5089

Estimated	Effort	Sched	PROD	COST	INST	Staff	RISK
Optimistic	10.9	7.7	466.6	0.00	0.0	1.4	
Most Likely	16.3	8.7	312.6	0.00	0.0	1.9	0.0
Pessimistic	24.4	9.8	208.4	0.00	0.0	2.5	

Metrics for MAS

- Method 2: Obtaining the equivalent in SLOC
  - We counted how many lines of code were needed for each concept

Element	SLOC P815	SLOC P907	SLOC PSI3	Average SLOC	Average SLOC per item
Event	443	86	172	233.66	11.02
Rule	2130	923	1047	1366.66	18.94
Goal	1581	110		845.5	7.75
Task	793	520	303	627	76.17
State machines	142	691	1048	233.66	11.02

Conclusions: Final results

Project	Real cost PM/months	(A) PM/months	(B) PM/months	(C) PM/months
P907	6 / 5	34.4 / 10.6	7.2 / 6	6.8 / 6.6
P815	18 / 9	135.6 / 16.7	16.3 / 8.7	25 / 9
PSI3	69 / 18	63.7 / 13.4	7.6 / 6.8	7.3 / 6.7

(A) Measuring as with conventional programs
(B) Measuring only code for tasks, goals, state machines, rules, events
(C) Measuring average SLOC per task and goal

Statistical data and COCOMO II files are available at http://grasia.fdi.ucm.es/gschool

- More projects are needed. COCOMO II uses 160.
  - Before this work, there were none available
  - We do not expect accurate results, since the number of projects is low
  - We are applying to more recent projects
  - We would like to have data from other projects as well
- Further work is needed
  - Determine if there are specialised EM and SF for agent projects
  - Calibrate COCOMO II EM and SF according to agent based results
  - Gather more data about agent based projects

Conclusions

- The myth of agents being so different…
  - They are a contribution to distributed computing engineering
- Reuse current practices if applicable
- What needs to be reviewed when using agents?

Summary

- Agents cope with new types of complexity
  - There are application areas for agents
- Reuse experience in agent-based developments
  - Architecture
  - Patterns
  - Component based frameworks
- Most AOSE methodologies consider the production process but not the management process
  - What are the implications of using agents in project management?
- Need for metrics
  - Collect data from agent-based developments
  - Adapt metrics to agent paradigm