

On Agentware

Ruminations on Why We Should Use Agents

Federico Bergenti

AOT Lab

Parco Area delle Scienze, 181/A

43100 Parma, Italy

bergenti@ce.unipr.it

<http://aot.ce.unipr.it>

FRAMeTech S.R.L.

Via San Leonardo, 1

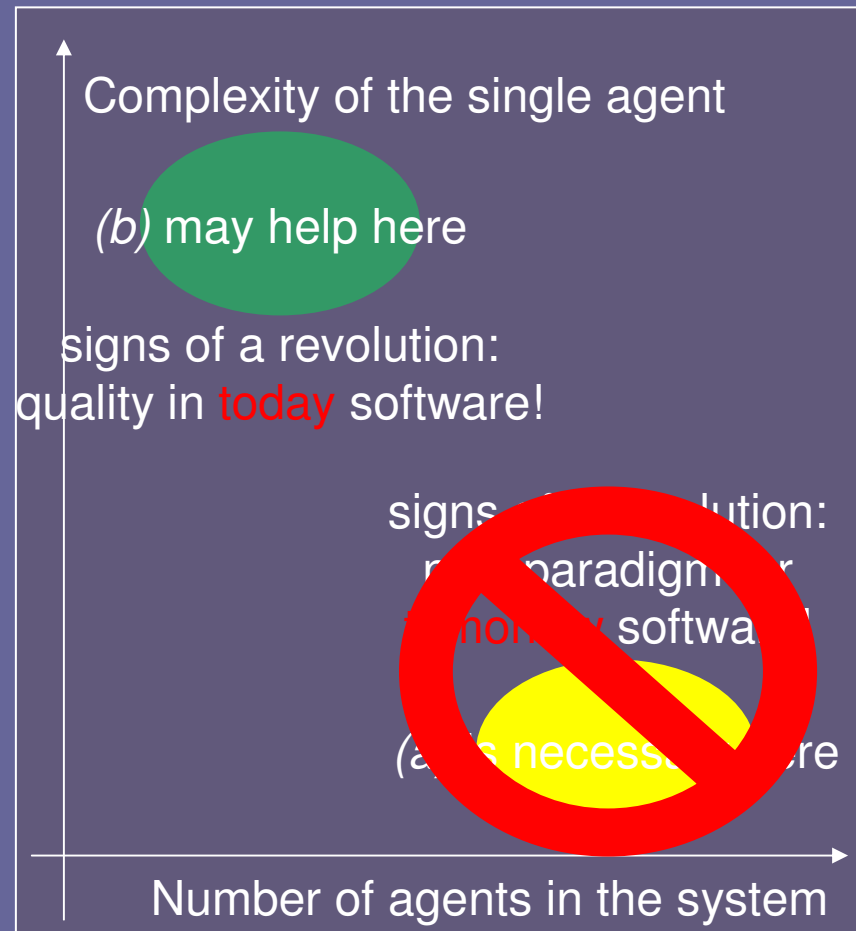
43100 Parma, Italy

bergenti@frame-tech.it

<http://www.frame-tech.it>

Agent-Oriented Software Engineering

- Two points of view:
 - a. Engineering of multiagent systems where agents are mainly software;
 - b. Agent-based engineering of software systems.
- These approaches are complementary:
 - a. Necessary when dealing with a large number of simple agents, e.g., CA, ants, GRID;
 - b. May help to improve the quality of today software.



One-Million-Euros Question

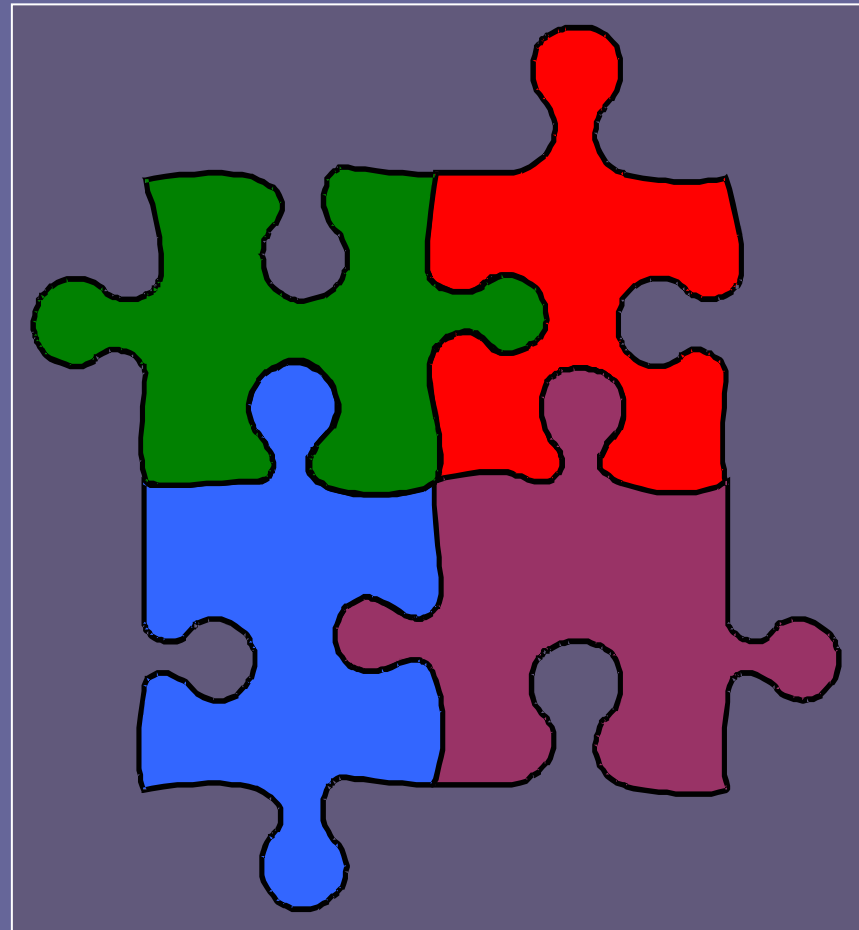
- Approach (b) needs justification:
 - Current technologies are used to implement complex systems with pretty good results;
 - Systems run properly, without any single agent been implemented.
- The important issue we have to tackle is:

“Why shall we use agent technologies instead of choosing any other available technology?”
- The rest of this talk discusses this issue.



Targeting the Question

- **Componentware** is the most promising technology for realizing quality software today.
- Similarities between agents and components are somehow understood:
 - Pro: agents are on the path that seems right one today;
 - Pro: agents may be complementary to components;
 - ...
 - Con: why yet-another looks-like-components technology?



Agentware

- **Agentware** is software (*as we know it today*) made through the composition of interacting agents:
 - The whole system is implemented in terms of agents and only occasionally we need to go underneath;
 - Each single agent is coarse-grained, just like a CORBA object or a .NET component.
 - Ideally, agents are taken from a repository of **commercial off-the-shelf (COTS)** agents and the system is composed and reconfigured on the fly exploiting *semantic composability*.

A Model of Agentware

- One possible model of agentware, implemented in **ParADE**:
 - Agents are known to the multiagent system by means of a unique identifier;
 - Their state is characterized through beliefs and intentions;
 - Agents have capabilities:
 - What the agent can do, i.e., the possible outcomes of its actions;
 - How the agent can interact with other agents.
 - Agents communicate through message passing.
 - Communication is meant to exchange **representations** of theirs and others beliefs, intentions and capabilities.
 - To support fruitful communication, we equip agents with **roles** and **interaction laws**:
 - Rules an agent adopt when it *decides* to play a role in the system;
 - They govern its interactions with other agents;
 - The interaction laws that an agent decides to follow are part of its capabilities and they are published.

Comparison with Components

Abstraction	Component-oriented	Agents-oriented
Communication	Task delegation	Task and goal delegation
Message	Requests for actions	ACL messages
Interaction with the environment	Events	Updates of beliefs
State	Properties and relations	Mental attitudes
Interactions between parties	Interfaces, Interface repository	Capabilities, Semantic matchmaker
Runtime	Application server	FIPA platform

Comparison: Communication Models

- Main difference between agents and components:
 - Agents use ACLs, i.e., a message is sent to transfer part of the sender's mental state to the receiver;
 - Components use a metaobject protocol.
- One important special case is goal delegation:
 - Basic mechanism that agents use to delegate responsibilities (**declarative message passing**);
 - The components' metamodel does not comprise an abstraction of goal and components cannot exploit goal delegation; rather they use task delegation (**imperative message passing**).

Comparison: Interactions with the Environment

- The environment is a *structural* part of the agents' metamodel.
- Agents can:
 - Measure it;
 - Receive events from it.
- Agents react to any change in the environment through changes in their mental state.



Comparison: Interactions with the Environment

- In the component-oriented approach, the environment communicates with components only through *reified events*:
 - Components react to reified events after constructing a relation with them.
- Both approaches respect encapsulation:
 - The state of the component is changed only when the component itself decides to change it;
 - Agents' reasoning capabilities are the only responsible for any change in the mental state.

Comparison: Representation of the State

- Both agents and components are abstractions that comprise a state:
 - The state of a component is represented though a set of *properties* and *relations* that other components can manipulate;
 - The state of an agent is represented though a set of beliefs, intentions and capabilities.
- Main differences are:
 - Agents have an explicit representation of their goals;
 - Agents have explicit knowledge of their environment;
 - Agents do not have properties, they only have implicit relations with other agents and with entities in the environment;
 - Agents can exploit general approaches to manage knowledge. Such a process is always application-specific in components.

Comparison: Interaction between Parties

- The different communication mechanisms influence how agents and components open themselves to the outer world:
 - Components use interfaces to *enumerate* the services they provide and to tell clients how to get in contact with them. Sophisticated component models support design by contract.
 - The agent-oriented approach eliminates interfaces and give agents capabilities to describe what the agent can do and how it can interact with other agents.
- The use of capabilities instead of interfaces has the advantage of easily describing the semantics of the services that an agent offers, using:
 - High-level, mentalistic abstractions, i.e., beliefs and intentions;
 - A model of the environment, i.e., entities and their relations.

Benefits of Using Agents

- The chosen ranking criteria for the comparison is based on two important features of a development technology:
 - Reusability;
 - Composability.
- Agents **improve** both reusability and composability with respect of components.
- First, agents can use goal delegation:
 - Decouples the agent that delegates a goal from the delegate agent. The two are no longer implicitly coupled through a sequence of actions, but only through an explicit goal;
 - Solves the *interface mismatching* problem, i.e., the problem of substituting a component with another component that offers the same services through a different interface.

Benefits of Using Agents

- Second, agents use ACLs:
 - They turn many common tasks into application specific only in the sense that they still depend on the ontology;
 - Components implement common task, e.g., informing and querying, through properties and relations and so they couple communication with the information model.
- Third, the logic-based model of the mental state allows using deduction and means-end reasoning:
 - Many messages and events can be turned to fewer common situations;
 - The concrete behavior of the agent is described only for a limited set of situations while allowing it to react to a wider range of events and messages;
 - The behavior of the agent is partially decoupled from the concrete messages and events that it can handle.

A New Level of Composability

- Agents support **semantic composability**:
 - They rely on a common model of the executor that moves most of the semantics of the composition to the semantics of the ACL;
 - Components do the opposite, they shift most of the semantics of the composition to the semantics of the action that a component executes in reaction to a message or an event (**syntactic composability**).
- When adopting a shared and accepted ACL, agents become **semantically interoperable** while components are only **syntactically interoperable**.

A New Level of Abstraction

“Agents introduce a new level of abstraction.”

- We must give this sentence a scientific meaning by means of a **system level**.
- The literature proposes two system levels:
 - Newell’s **knowledge level**;
 - Jennings’ **social level**.
- None of them is satisfactory from an engineering standpoint. They do not provide any hint on realizing tools for modeling systems at that levels.

System Levels

- A system level is a set of concepts that provides a means for modeling implementable systems.
- System levels are layered in a stack and each level is mapped on one lower level.
- System levels abstract away for implementation details and higher levels provide concepts that are closer to human intuition and far away from implementation.
- System levels are structured in terms of the following elements:
 - *Components*: atomic building blocks for building systems at that level;
 - *Laws of compositions*: laws that rule how components can be assembled into a system;
 - *A medium*: a set of atomic concepts that the system level processes;
 - *Laws of behavior*: laws that determine how the behavior of the system depends on the behavior of each component and on the structure of the system.

The Agent Level Defined

Element	Element Element of the Agent Level
System	Multiagent system
Components	Beliefs, intentions, roles and interaction laws
Law of composition	Any composition is admissible.
Medium	Representation of beliefs, intentions and capabilities
Behaviour	Principle of rationality

- The agent level takes benefits from both Newell's and Jennings' proposals:
 - Single agents are modeled through mentalistic attitudes;
 - The system is a society where roles and rules are considered.
- At the agent level, single agents and the society are modeled together.

Discussion

- Two results:
 - Agents provide high-level abstractions to developers;
 - Agents have some advantages over components in terms of reusability and composability.
- Both points have well-known advantages but they also have a typical drawback, i.e., **speed**:
 - In order to fully exploit the possibilities of agents, we need to implement reasoning agents;
 - Goal delegation instead of task delegation requires means-end reasoning and we may end up implementing slow agents.

Discussion

- In both cases, performances degrade gracefully.
- We can choose how much reasoning we want for each and every agent:
 - We may use reasoning for agents that are particularly complex and that could benefit from high level abstractions;
 - On the contrary, we can fallback on reactive agents, or components, when we have an urge for speed.

This decision criterion seems sound because the more an agent is complex and value-added, the more we want to reuse and compose it with other agents.

- Anyway, performance issues seem not to be blocking because today speed is not always the topmost priority.

Thanks...

Questions?

AOT Lab
Parco Area delle Scienze 181/A
43100 Parma, Italy
bergenti@ce.unipr.it
<http://aot.ce.unipr.it>

FRAMeTech S.R.L.
Via San Leonardo 1
43100 Parma, Italy
bergenti@frame-tech.it
<http://www.frame-tech.it>