Engineering Multi-Agent Systems

Dagstuhl, August 20-24, 2012

Prof. Tom Holvoet
DistriNet labs
Dept. of Computer Science
KULeuven, Belgium

Research @ DistriNet

“Software engineering for distributed software / systems”

Scope:
• Distributed software
• Secure software
Distributed SW architectures (dyn/ls/…)

Architectural tactics
- MAS & coordination
  - delegate MAS
  - bio-inspired mech
  - other
- Arch. based self-ad

Modeling and specification
- Specification
  - Arch. doc.
  - Patterns/PL
- Modeling
  - MACODO
  - Forms+

Applications
- • traffic
  - logistics
  - smart grid
  - mc (cold end)
  - …

Execution/experimentation platforms

Simulation environments
- Logistics – RinSim
  - Traffic – GridLock
  - Smart grid sim

Middleware
- MW for service composition
- MW for collab. applications

distributed system

One case: warehouse logistics…
One case: warehouse logistics…

- **Problems**…
  - flexibility / adaptivity (autonomy)
  - scalability

- Company was interested in **solutions** for these non-functional requirements
  - Hów we do it was less a concern
  - But should be ‘easily integrated’ with their environment
    - legacy
    - developers team / skills
      - not interested in new methodologies/languages/etc.
    - clients
Conclusions

• In our experience…

→ Engineering MAS IS (at least) 90% software engineering, 10% MAS

→ MAS is all about software architecture (aligns with mainstream SE)

→ Thorough evaluation / guarantees are crucial

Conclusions (1)

• Engineering MAS IS (at least) 90% software engineering, 10% MAS

→ a real system is not only concerned with the features (nfr/prop) that MAS offer!!!

→ but also real world stuff, like
  • safety
  • security
  • taking into account team skills
  • programming languages
  • middleware
  • existing code base / proprietary systems
  • …
Conclusions (2)

Architecture-centric perspective on MAS

- Multi-agent systems as a particular family of software systems
  → Specific way to *structure* software system

Architecture-Centric Software Engineering
Development Life Cycle

1. Domain Modeling
2. Requirements Engineering
3. Architectural Design
4. Develop Core System
5. Develop Version
6. Deliver Final Version
7. Incorporate Feedback
• Consequences?

→ extremely easy to sell !!
  ▪ stick to standard practice in software engineering
    ▪ there ARE established best practices, e.g. (R)UP
    ▪ proven their effectiveness (and weaknesses)
    ▪ all engineering activities/tools/techniques/… could be reused
  ▪ define a clear delta
    ▪ models that define the MAS / agent behaviour & coordination
  ▪ IT DOES NOT ENFORCE A DOMINANT STYLE throughout the engineering process just to encompass with ‘some’ non-functional requirements

→ security – could be integrated
→ advanced (e.g. reflective) middleware – could be integrated
→ …

→ But indeed ‘punctual’ additional support may sometimes be interesting
  ▪ Viewpoint for modeling organisations
  ▪ Middleware for coordination / organisations
  ▪ …
Conclusions (3)

• Evaluation / guarantees…

→ simulation results are insufficient…
→ ‘collective behavior’ scares clients…

(Eval4SASO – workshop at SASO-2012)

Conclusions

• In our experience…

→ Engineering MAS IS (at least) 90% software engineering, 10% MAS

→ MAS is all about software architecture (aligns with mainstream SE)

→ Thorough evaluation / guarantees are crucial

• Startup in AGV warehouse management…
Disclaimer

- In *our* experience in *our* cases
  → logistics
  → traffic
  → smart power grid management
  → manufacturing control
  a software-architectural perspective is more than enough

But there may be other cases, of course!

But the danger is that by adopting an AOSE, you throw away everything that comes with mainstream SE, such as

- Such as
  → requirements engineering – a whole body of work there!
  → architectural design – a whole body of work there!
  → detailed design – …
  → security – …
  → middleware – …

  → ATAM

  → If you would need to integrate all that in a NEW methodology…??
Further thoughts / statements / …

1. (Stop whining, and) GO TO MAINSTREAM SOFTWARE ENGINEERING EVENTS!!

• If we want to bring engineer real systems using MAS, we need to know(!) and speak their language!!
  → e.g. software architecture
  → e.g. design patterns
  → …

Further thoughts / statements / …

1. (Stop whining, and) GO TO MAINSTREAM SOFTWARE ENGINEERING EVENTS!!

• …

• Our MAS research IS appreciated, if realistic (!) and presented right
  - OOPSLA/SPLASH
  - ECSA/WICSA
  - ESEM
  - SEAMS
  - (ICSE, …)

• Grasp low-threshold opportunities!
  - AGERE @ SPLASH
  - (SELMAS @ ICSE)
Further thoughts / statements / …

2. We must actively engage in real-world applications

→ tremendous experience
→ not only applications that we help define in proposals!
→ AOSE-2012 conclusion…

Further thoughts / statements / …

2. We must actively engage in real-world applications

→ …
→ Keep our eyes open for interesting application domains

- Multi-agent systems have been identified by the IEEE Power Engineering Society’s Working Group as a promising distributed control approach in power engineering


3. A Pattern language for MAS

'Proven, reusable solutions for recurring design problems’
Technology independent
Well-understood / well-accepted

Useful for communication with industry / software engineers / students

Our experience
- Tom Holvoet, Danny Weyns, Paul Valckenaers, Patterns of Delegate MAS, Self-Adaptive and Self-Organizing Systems, pages 1-9, San Francisco, California, USA, 14-18 September 2009
- Tom Holvoet, Danny Weyns, Paul Valckenaers, Delegate MAS patterns for large-scale distributed coordination and control applications, Proceedings of the EuroPLoP ’10 European Conference on Pattern Languages of Programs, Germany, July 2010
Conclusions

→ Engineering MAS is (at least) 90% SE, 10% MAS
→ MAS is (almost) all about software architecture (aligns with mainstream SE)
→ Thorough evaluation / guarantees are crucial

→ Go to mainstream software engineering events
→ Real-world applications
→ Towards a pattern language for MAS