MAMS: Multi-Agent Microservices

Rem Collier, Eoin O'Neill, David Lillis, Gregory O'Hare

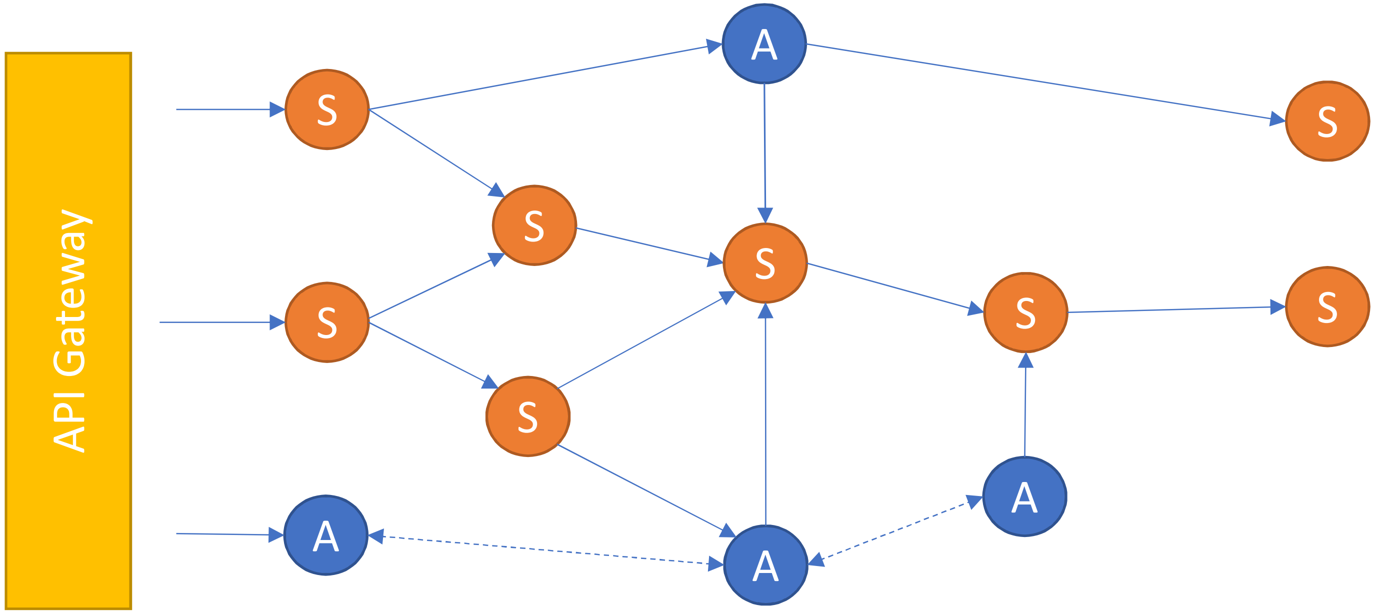
School of Computer Science

CONSUS

CONSUS is funded under the SFI Strategic Partnerships Programme (16/SPP/3296) and is co-funded by Origin Enterprises Plc

The Bigger Picture

- We want to build applications that seamlessly combine RESTful microservices and Multi-Agent Systems.
  - There should be no delineation between agent and service frameworks.
  - We require that services be able to interact directly with agents and agents to interact directly with services.



CONSUS

MAMS = Multi-Agent Systems + Microservices

- Multi-Agent Systems:
  - Established (since 1980s) research area that views systems as consisting of one or more loosely-coupled entities that have private (isolated) state and which work together to solve problems that are beyond their individual capabilities.
  - Key concepts include: autonomy, reactivity, proactivity, social ability

- Microservices:
  - Established (since 2011) as a key architectural style for modern software systems. Adhere to the IDEAL principles: Isolated state, Distribution, Elasticity, Automated management, and Loose coupling.

- Both approaches are concerned with the creation of loosely-coupled distributed systems comprised of small independent (autonomous) components with internal state.

Microservices in 1 slide

- Microservices often adopt a resource-oriented view of systems:
    - A system consists of a set of resources and (composite) resource types.
    - Representations of resource state in tandem with CRUD-style operations can be used drive system behaviour.
    - In the spirit of the Web, relations between resources are modelled as links.
    - URIs are used to identify (parts of) resources.
- Microservices co-locate resources for practicality:
    - Instances of the same resource types (e.g. records held within a database)
    - Highly coupled / composite resources (e.g. blog entries and user comments).
- Microservices can be passive or active:
    - Changes of state are not only driven by interaction, but also by internal/hidden (to the resource) processes.

CONSUS

Agents as Microservices

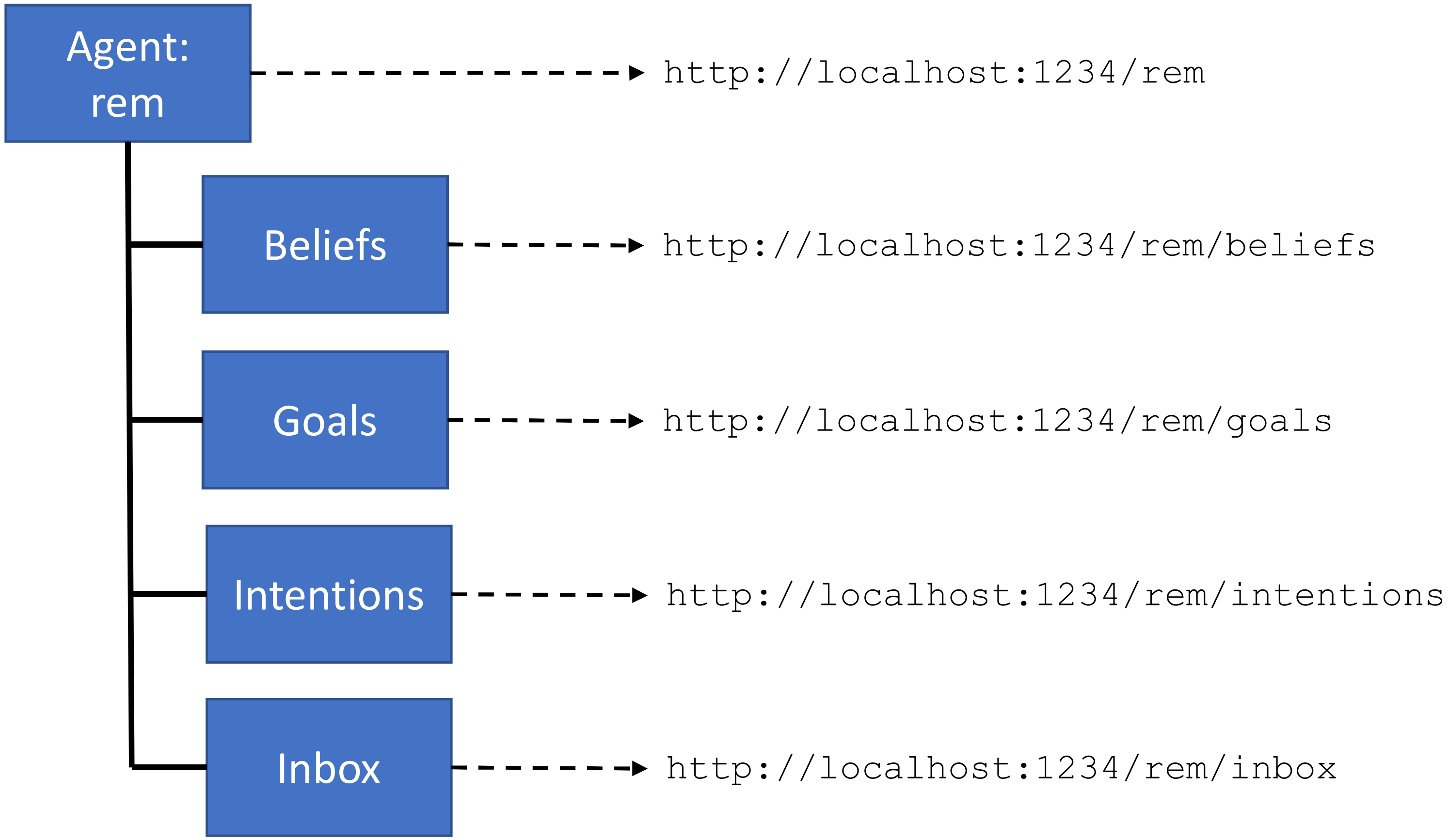
- Agents are complex resources:
  - Agents have complex (composite) state – beliefs, desires, intentions, rules, messages, …
  - Not all states are/should be externally mutable.
  - Agents normally interact by sending messages to one another (via their inbox resource).

- Agents should have unique identifiers:
  - e.g. FIPA Agent Identifiers

- Agents must deal with Chattiness/Bounded Context:
  - Coordination / collaboration often results in increased interaction between agents.
  - Organisationally speaking, we expect more interaction within an organisation than between organisations.

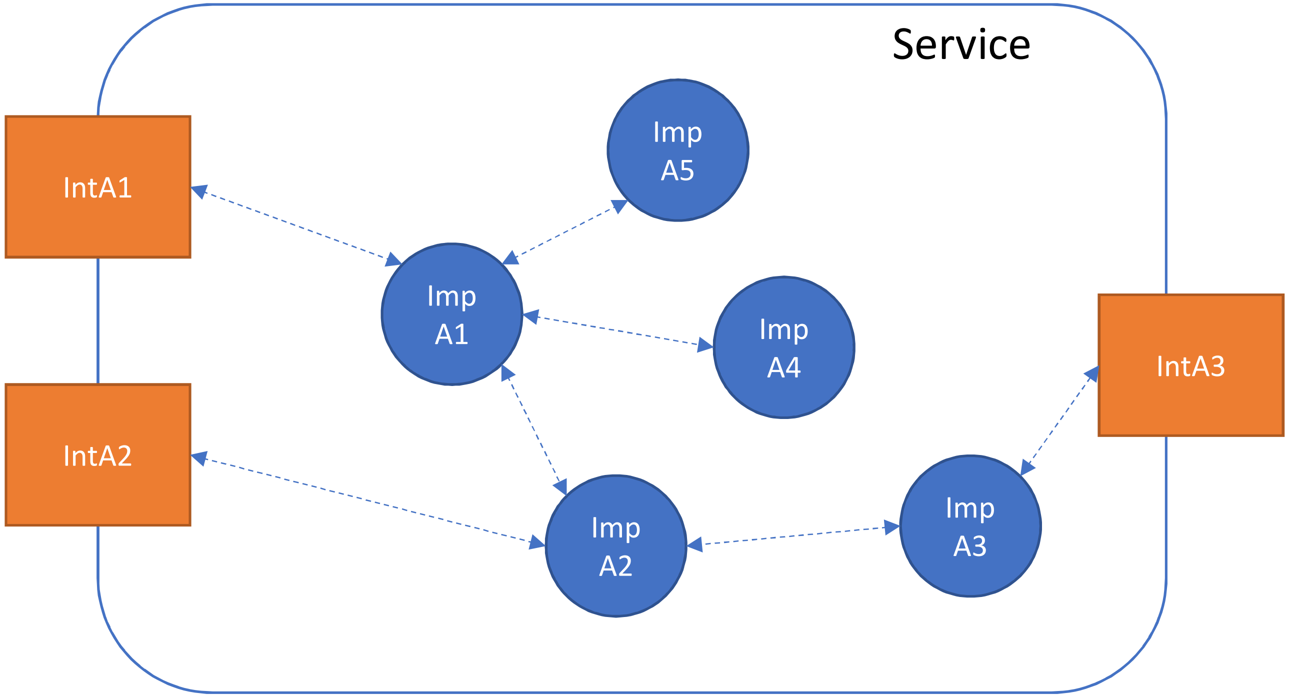
Agents as Microservices

- Adopting a view of agents as resources offers a simple model for exposing agent state.


Agent: rem
http://localhost:1234/rem
Beliefs
http://localhost:1234/rem/beliefs
Goals
http://localhost:1234/rem/goals
Intentions
http://localhost:1234/rem/intentions
Inbox
http://localhost:1234/rem/inbox

Public/Private Agents

- Internal agents can augment interface agent functionality.



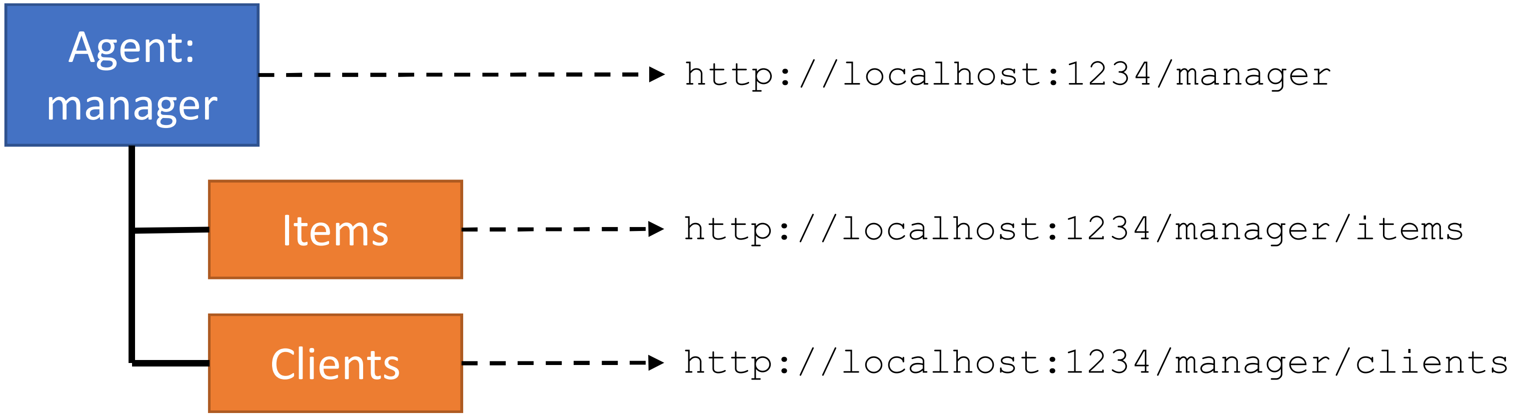
Agent-Service Interaction

- What about Plain Old MicroServices (POMS)?
  - POMS interact through REST – simple and effective

- Agent-POMS interaction needs to be as simple!
  - Beliefs are internal representations of concepts that the agent uses to reason about how best to act.
    - E.g. how to bid for a given type of item
  - Interacting with agents through messaging or state update requires in depth technical knowledge.
  - Another approach is required…

Agents & Virtual Resources

- **Idea**: Agents are able to manage internal resources that are externally accessible through REST.
  - Agents expose concepts (e.g. bidding strategies) as resources
  - Internal representations of the resources are implementation specific.
  - For example, bidding strategies may be modelled as a set of beliefs
    - best-price(Item, Amt), required(Item, Qty), increment(Item, Inc)
    - strategy(Item, Amt, Qty, Inc)
  - Agents should be aware of incoming requests and be able to decide on how to respond (based on the request and the current context).


Agent: manager
http://localhost:1234/manager
Items
http://localhost:1234/manager/items
Clients
http://localhost:1234/manager/clients

CONSUS

ASTRA: AgentSpeak(TR)

- Variant of AgentSpeak(L) that includes support for Teleo-Reactive Programming.
  - Event : Context -> Plan rules
  - State -> Action rules
- Strongly Typed
  - closely aligned to Java type system
  - Includes object references
- Extension/Reuse mechanisms
  - Modules: Sensors, Actions, Terms, Formulae, Events
  - Multiple Inheritance: Agent Classes
- Minimal Run-time
  - Configurable directly by agents.
  - System started by running an agent.

ASTRA & MAMS

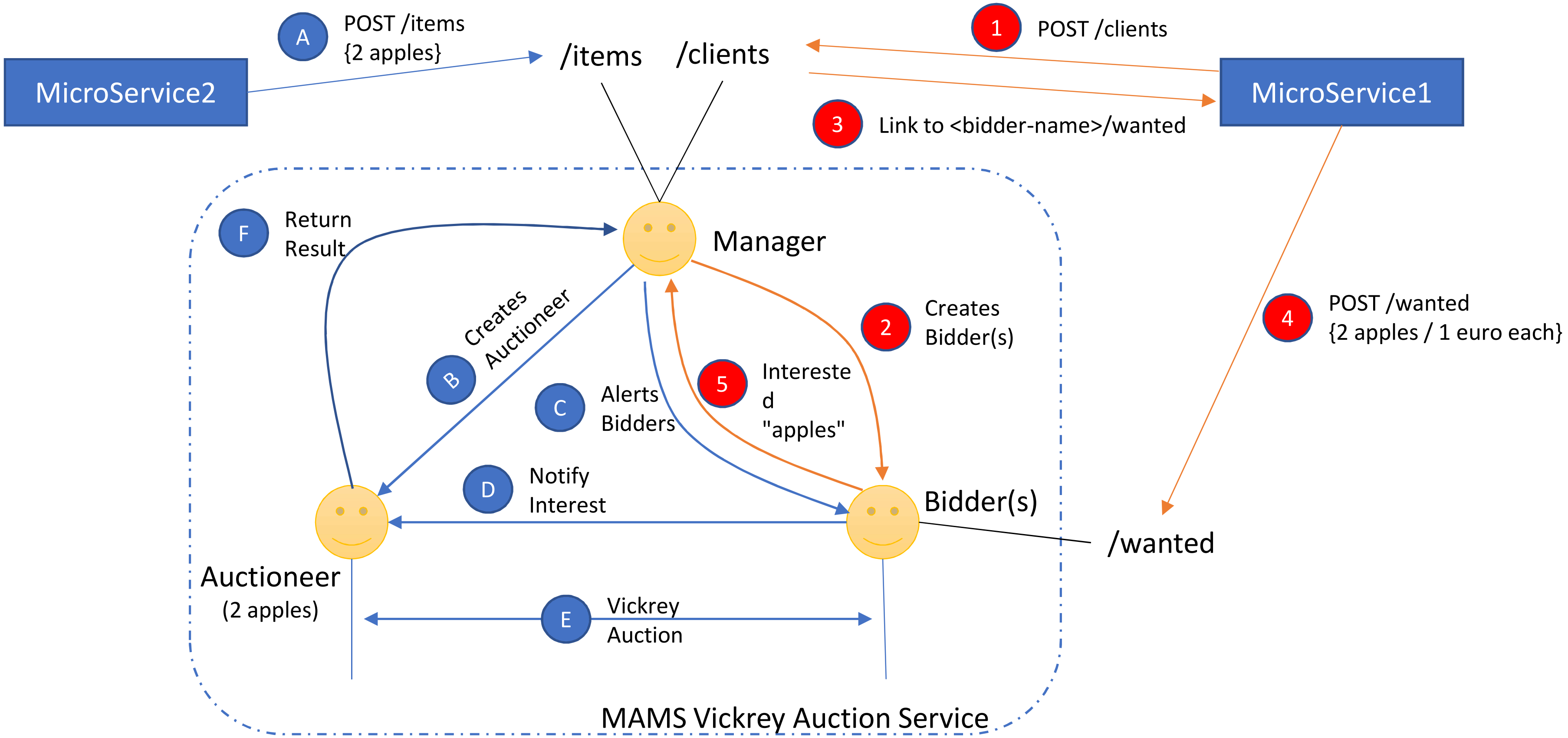
- Integration of a web interface based on Netty.io
  - A Http module that links agent to the web interface (creates URI) and provides custom actions, events, and terms.

```
agent Hello {
    module Http http;

    rule +!main(list args) { http.register(); }

    rule $http.get(ChannelHandlerContext ctx,FullHttpRequest req,["hello"]) {
        ResponseObject obj = http.createResponse();
        http.setStatus(obj, 200);
        http.setType(obj,"text/html");
        http.setContent(obj, "<html><body>Hello World!</body></html>);
        http.sendResponse(ctx, req, obj);
    }
}
```

Example: Vickrey Auction



CONSUS

Example: Vickrey Auction

```
agent Manager {
    ...
    rule $http.post(ChannelHandlerContext ctx, FullHttpRequest req, ["items"], string bdy) {
        Item item = il.itemFromJson(bdy); il.storeItem(item, string id);
        !!auctionItem(id, il.getItemName(item));
        ResponseObject obj = http.createResponse();
        http.setStatus(obj, 200);
        http.setLocation(obj, http.myAddress()+"/items/"+id);
        http.sendResponse(ctx, req, obj);
    }

    synchronized rule +!auctionItem(string id, string item) {
        !auctioneer("auctioneer"+id, item);
        foreach (interest(string name, item)) {
            send(inform, name, available(item, "auctioneer"+id));
        }
    }
    ...
}
```

CONSUS

Conclusions

- MAMS offers a simple model for defining open decentralised multi-agent systems.
  - URIs provide a global naming system for agents and a way of exposing the state of an agent.
  - Feels like something akin to defining a “body” (modelled as observable state) for agents…
  - Enables further concepts: Joint Intentions, Conversation Modelling, Conversation Histories, Acquaintance Networks, …
- MAMS promotes the creation and use of pre-built components that can be tested in isolation and used in confidence
  - Seamless interaction between agents and services facilitated through the concept of virtual resources.
  - Public/Private agents allows the creation of robust services with clearly defined interfaces.
  - Leads to concepts such as Organisation as a Service (OaaS)

Conclusions

- From a Linked-Data / Semantic Web perspective:
  - Agent-Agent interaction can enhanced through semantic models.
  - Agent-Service interaction can also benefit.
  - Need to move from a model of implementing internal models of the environment to embracing shared models.

- Need to design a class of agent programming languages that fully embrace linked data / semantic web / REST concepts.