Reliable Agent Systems

This lecture looks more closely at reliability of agent systems in an open environment:

- Why this is difficult but (maybe) important.
- Two short examples showing how a small change in our formal view alters practice.
- Different ways of assessing reliability.

Many SE/KE technologies/methods *could* apply. The issue is how they can be applied effectively.

Reminder: Demand v Supply

"What is particularly impressive is the way that scientists are now undaunted by important complex phenomena...The emerging field of e-science should transform this kind of work...One of the pilot e-science projects is to develop a digital mammographic archive, together with an intelligent medical decision support system for breast cancer diagnosis and treatment....So the surgeon in the operating room will be able to pull up a high-resolution mammogram to identify exactly where the tumour can be found."

Tony Blair, Speech to Royal Society, 23rd May 2002

“Design and Development: Software Architecture Design… Artificial Intelligence…NR [Not Recommended]”

IEC 61508 standard for safety-related software

Reliability for Traditional Systems

A traditional form of reliability estimation:

- Define the boundaries of the system.
- Identify the components within these boundaries.
- Assess the reliability of each component under the range of operating conditions expected.
- Estimate the overall reliability as some function of component reliabilities and their interactions.
- Use this stable reliability estimate for the system whenever it is run.

Reliability for Agent Systems

The Semantic Web (and related Grid initiatives) assumes:

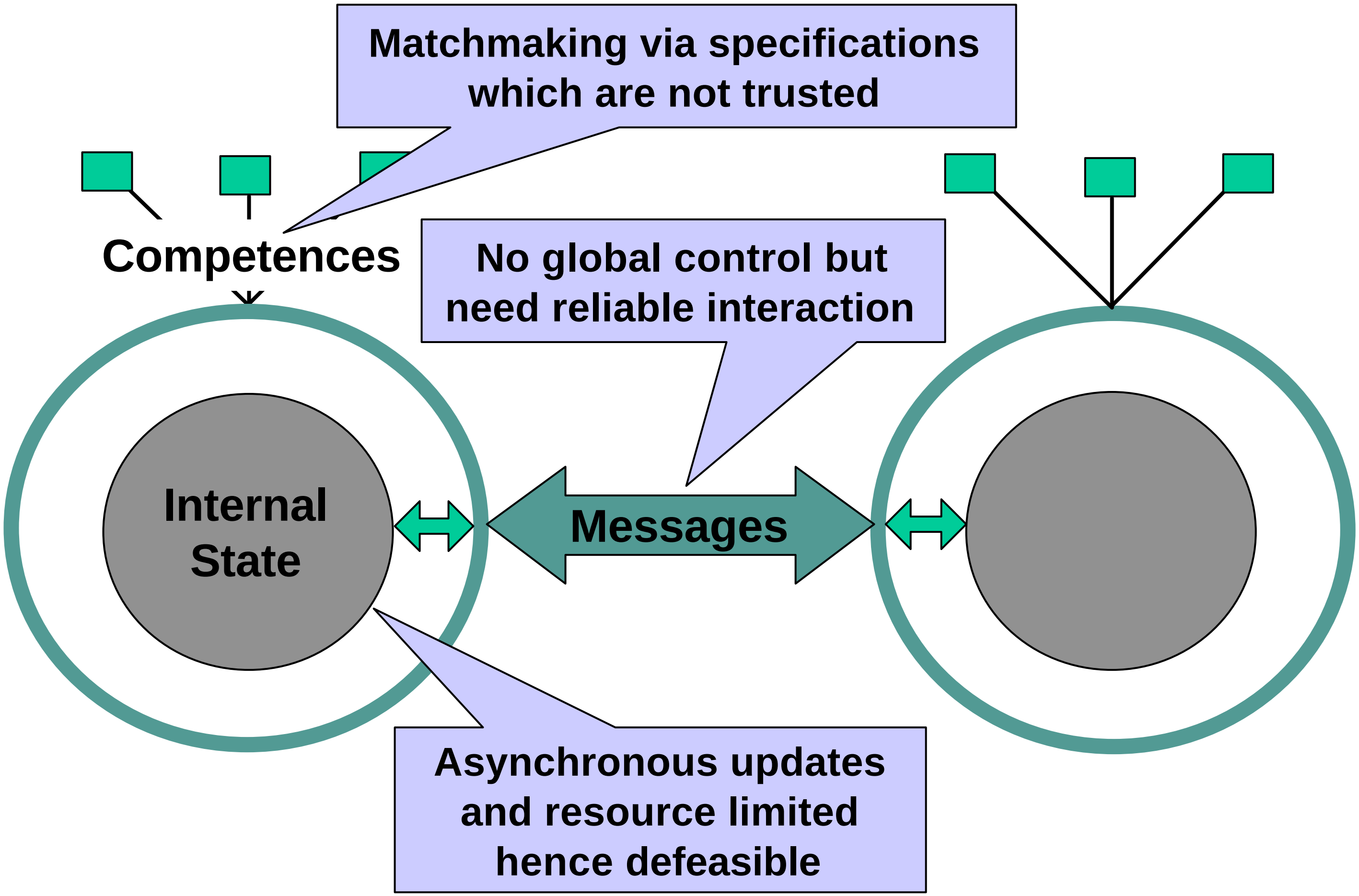
- Characterisation of components via a “semantic layer” (DAML-S, *etc*).
- Incremental and evolving characterisation.
- Standardisation only at interfaces between components (protocols, ontology mappings, *etc*).
- Context for evaluating “truth” of information is not always static.

No boundary. No definitive measure of dependability. No stability over time.

An Example

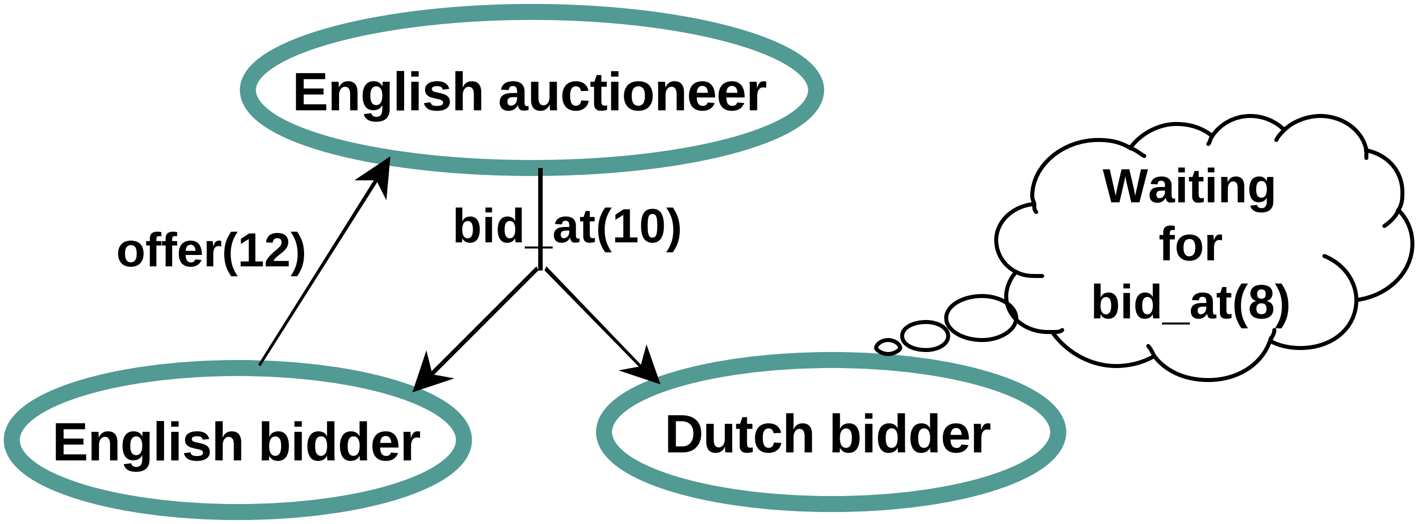
- It is believed to be impossible now for medical practitioners to know all they should know.
- In some sub-disciplines (e.g. oncology) automated support is being provided via decision support systems.
- These are now being made available as components (open source) via Web portals.
- The legal position in doing this is unclear. There is a duty of care in medicine. Normal medical safety arguments (“the patient would have died anyway”) won’t work. Normal defence in depth arguments won’t work.
- So how do we make a safety case for such systems?

Some Reliability Issues



Interaction Control

An auction without social conventions…



Current Standard Practice

Build into each agent:

- Its interaction strategy.
- Its algorithm for deciding how interaction strategy translates into message passing.
- Its decision procedures.

This doesn't scale because:

- We must predict all the interactions for each agent.
- The resulting code is complex.
- We must change it whenever the implementation platform or communication environment changes.
- This change may have an impact on other agents.

Reliable Interaction Control

Agents must operate independently but their interactions must be reliable. Conversation shouldn’t require preparation by all involved.

- Solution 1: Performative languages
- Solution 2: Global controllers.
- Solution 3: Proxies.
- <u>Solution 4: The protocol is in the message.</u>

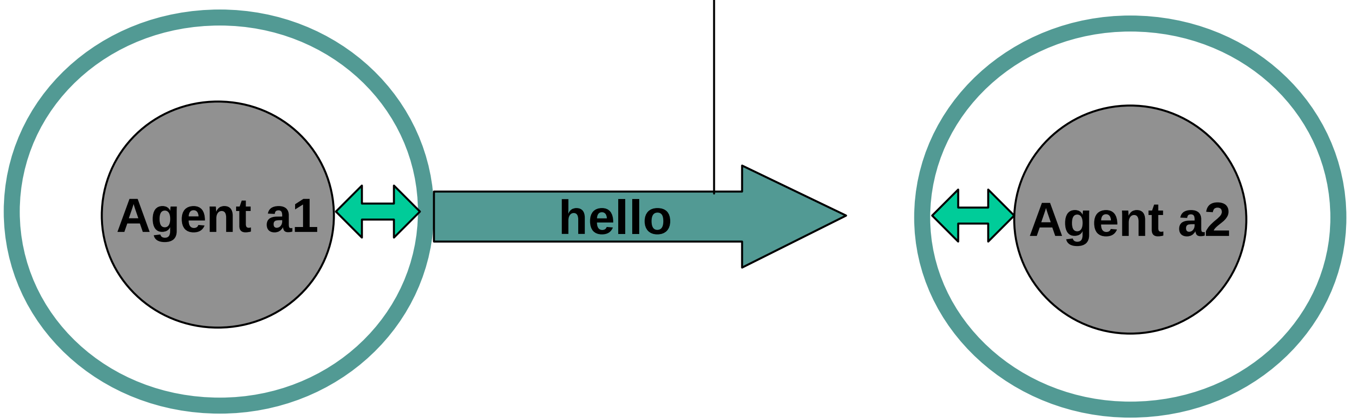
Basic Example

General

```
agent(A) ::=
    hello => agent(B) then
    recognised <= agent(B) then
    agent(A).

agent(B) ::=
    hello <= agent(A) then
    recognised => agent(B).
```

Specific

```
agent(a1) ::=
    hello => agent(a2) then
    recognised <= agent(a2) then
    agent(a1).
```



Basic Example

General

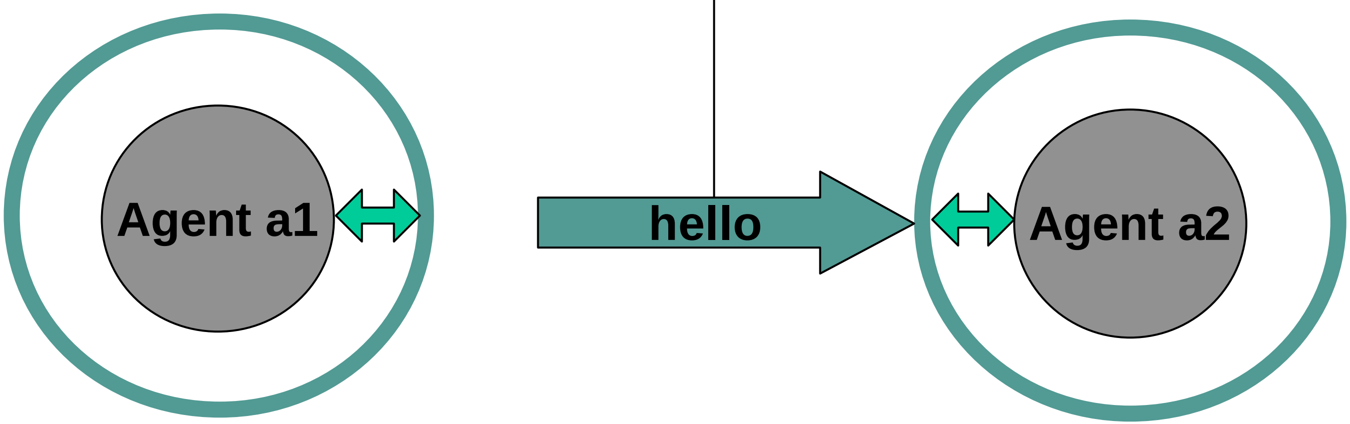
agent(A) ::=
hello => agent(B) then
recognised <= agent(B) then
agent(A).

agent(B) ::=
hello <= agent(A) then
recognised => agent(B).

Specific

agent(a1) ::=
hello => agent(a2) then
recognised <= agent(a2) then
agent(a1).

agent(a2) ::=
hello <= agent(a1) then
recognised => agent(a1).



Basic Example

General

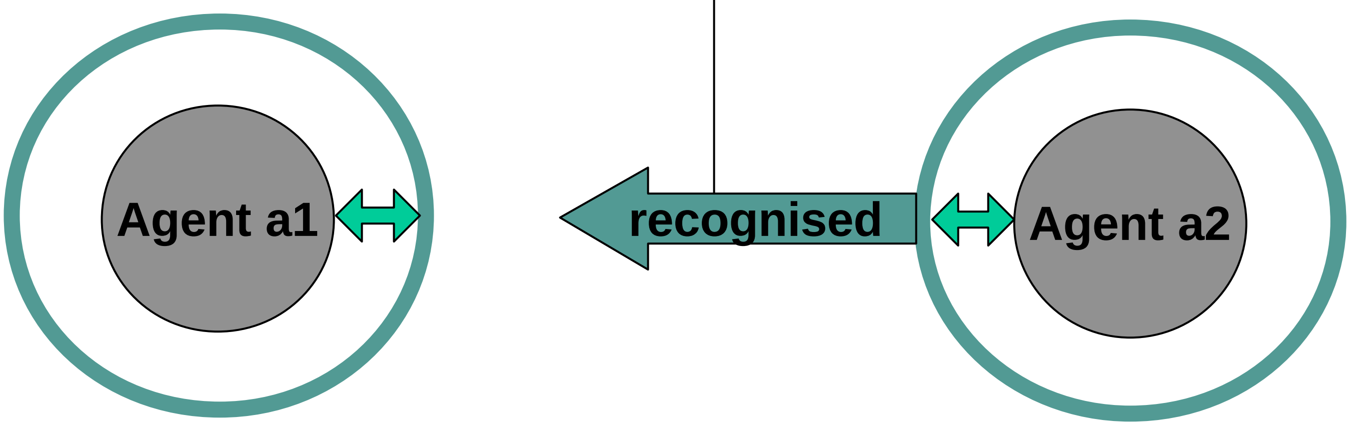
agent(A) ::=
hello => agent(B) then
recognised <= agent(B) then
agent(A).

agent(B) ::=
hello <= agent(A) then
recognised => agent(B).

Specific

agent(a1) ::=
hello => agent(a2) then
recognised <= agent(a2) then
agent(a1).

agent(a2) ::=
hello <= agent(a1) then
recognised => agent(a1).



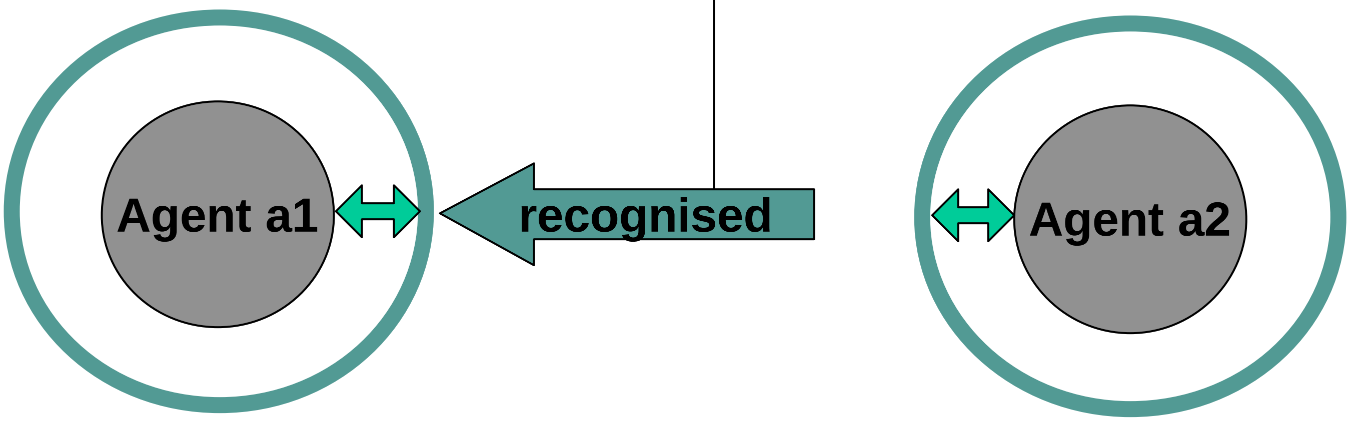
Basic Example

General

```
agent(A) ::=
    hello => agent(B) then
    recognised <= agent(B) then
    agent(A).

agent(B) ::=
    hello <= agent(A) then
    recognised => agent(B).
```

Specific

```
agent(a1) ::=
    hello => agent(a2) then
    recognised <= agent(a2) then
        hello => agent(B) then
        recognised <= agent(B) then
        agent(A).
agent(a2) ::=
    hello <= agent(a1) then
    recognised => agent(a1).
```



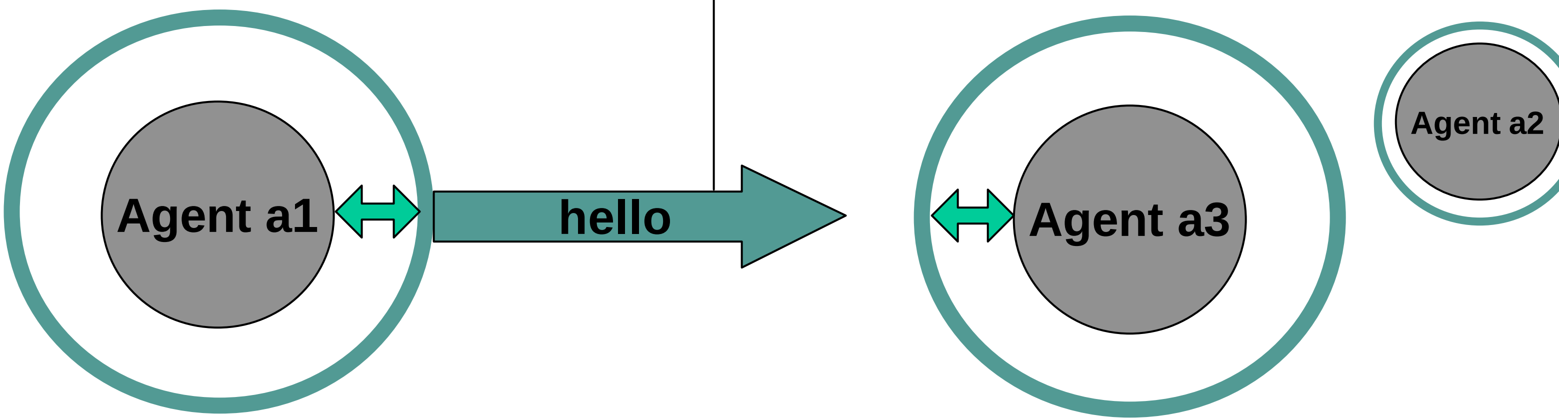
Basic Example

General

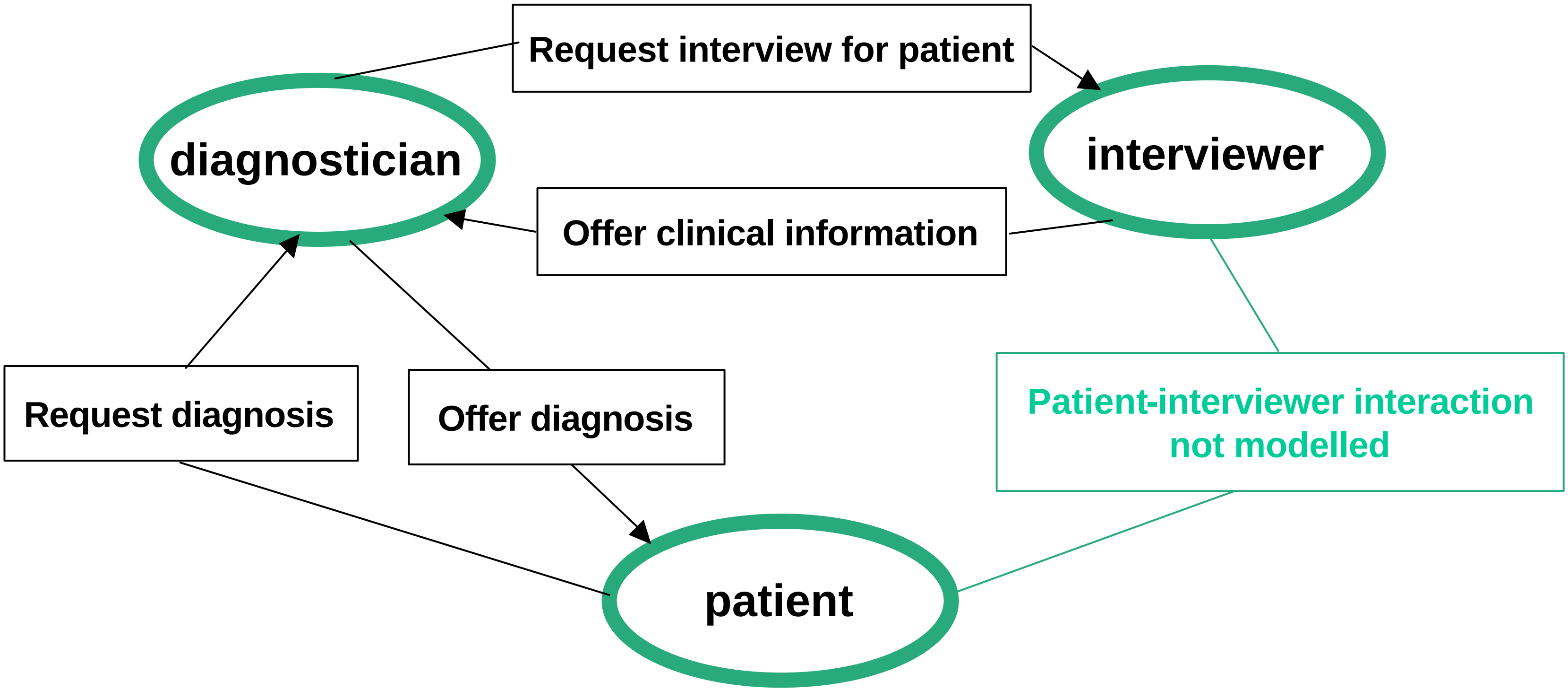
```
agent(A) ::=
    hello => agent(B) then
    recognised <= agent(B) then
    agent(A).

agent(B) ::=
    hello <= agent(A) then
    recognised => agent(B).
```

Specific

```
agent(a1) ::=
    hello => agent(a2) then
    recognised <= agent(a2) then
        hello => agent(a3) then
        recognised <= agent(a3) then
        agent(a1).
agent(a2) ::=
    hello <= agent(a1) then
    recognised => agent(a1).
```



More Complex Example



Interaction Protocol

```
agent(Scene, Type, Name) ::=
    InputMessage1 <= Agent1 then
    (OutputMessage1 => Agent2 par OutputMessage2 => Agent3) then
    agent(NewScene, NewType, Name).
```

```
agent(referral, diagnostician, D) ::=
    request(diagnosis,_) <= agent(external, patient, P) then
    request(clinical_interview,_) => agent(diagnosis, interviewer, I) then
    offer(clinical_information,_) <= agent(diagnosis, interviewer, I) then
    offer(diagnosis, _) => agent(external, patient, P) then
    agent(referral, diagnostician, D).
```

Reactions

agent(Scene, Type, Name) :: InputMessage <= Agent1 → Actions.

agent(referral, diagnostician, _) **::**
    request(diagnosis,D) **<=** agent(external, patient, P) →
        believe(diagnosis_request(P, D)).

Proactions

agent(Scene, Type, Name) :: OutputMessage => Agent1 ← Conditions.

```
agent(referral, diagnostician, _) ::
    offer(diagnosis, D) => agent(external, patient, P) ←
        decision(referral_decision(P), D).
```

Separation of Concerns

For each form of dialogue define a protocol:

- Define interaction protocol.
- Define proaction and reaction constraints.

For different interaction environments:

- Build the software for transporting protocols.

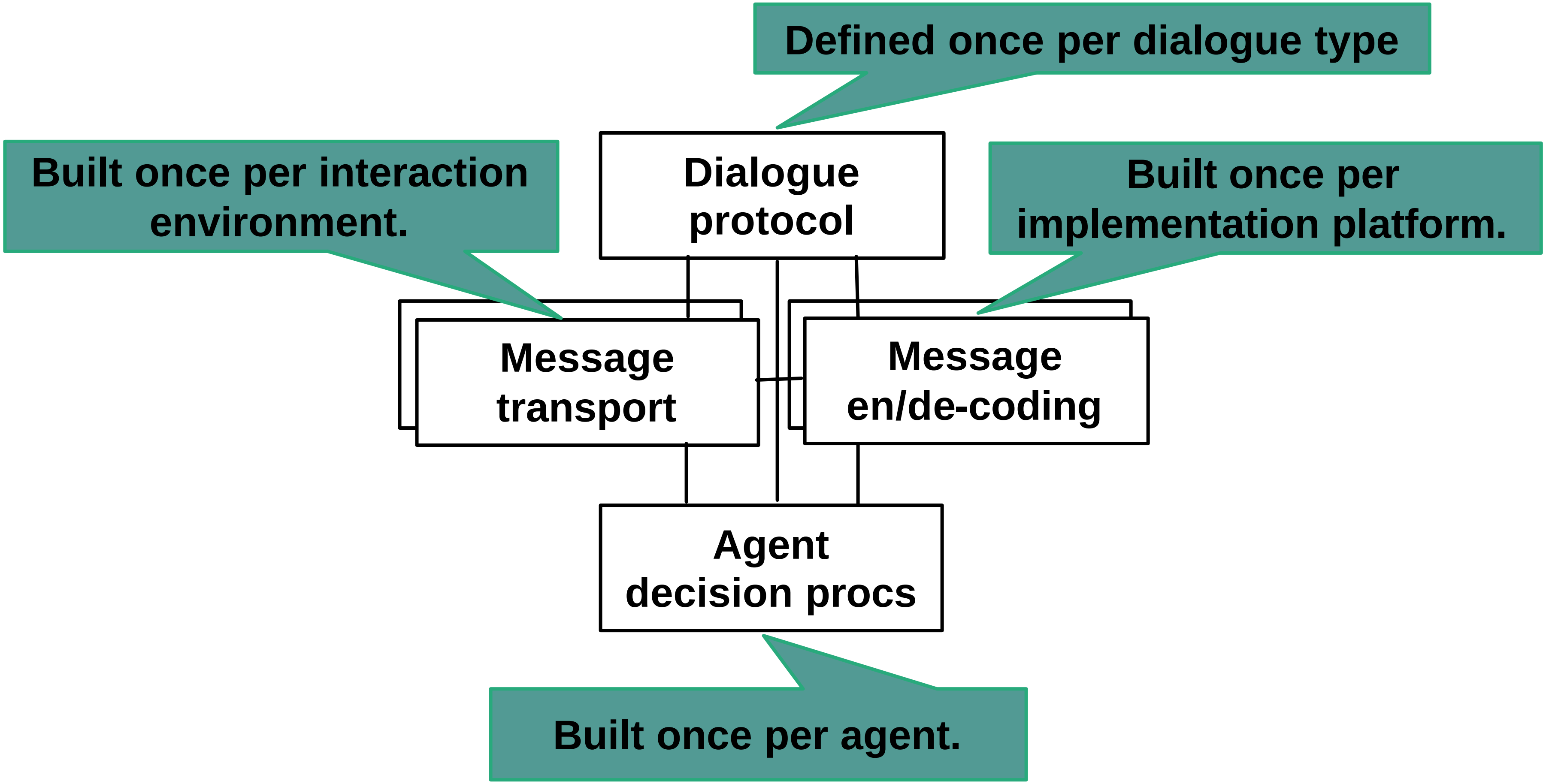
For different agent implementation platforms:

- Build the software for en/de-coding protocols.

For each agent:

- Build decision procedures for pro/re-actions.

Reliability: Process



Reliability: Proof

Proofs of properties seem attractive.
e.g. Every time there’s a request for a referral there will be an offer of diagnosis to the patient.

But we can prove this only with assumptions about the agents involved (e.g. that decision(referral_decision(P), D) always succeeds).

Component reliability can only be determined at deployment time so vendor has to provide a reliability function, not static evaluation.

Reliability:Testing

How do we test specific behaviours?
One way is to simulate.

```
agent(referral, diagnostician, D) ::=
    request(diagnosis,_) <= agent(external, patient, P) then
    request(clinical_interview,_) => agent(diagnosis, interviewer, I) then
    offer(clinical_information,_) <= agent(diagnosis, interviewer, I) then
    offer(diagnosis, _) => agent(external, patient, P) then
    agent(referral, diagnostician, D).
```

e.g. Every time there’s a request for a referral there will be an offer of diagnosis to the patient *in reasonable time*.

Reliability: Experimentation

How do we predict population-level behaviours?
One way is multi-agent simulation.

e.g. Can we invent a dialogue protocol which significantly reduced mean waiting time between presentation of symptoms and diagnosis.

Requires a “laboratory” for rapid modelling of large agent systems.

A “Laboratory”