Goals, Scenarios, Models and Architectures:

a tasty requirements recipe

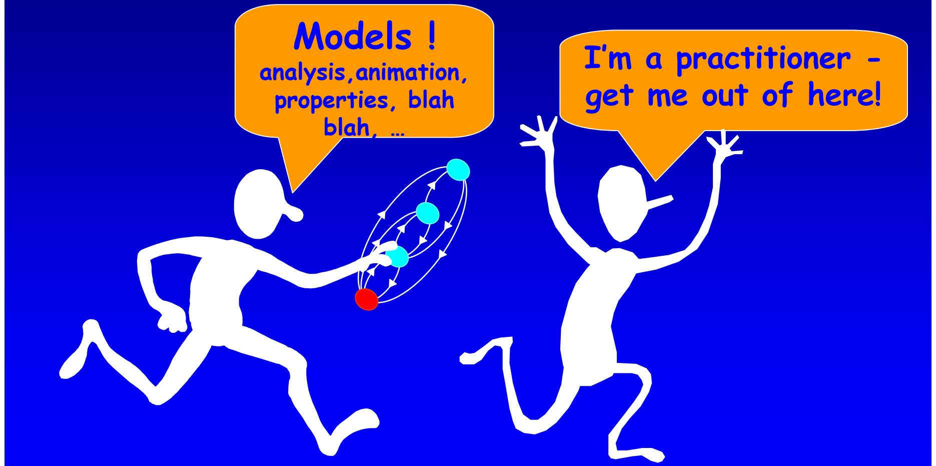
Jeff Kramer
Jeff Magee
Sebastian Uchitel

We believe in ….

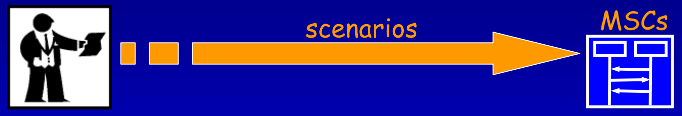
… model construction as part of the requirements process.

- Early use of a behaviour model can form part of a requirements specification.
- Model checking and animation of model behaviour and misbehaviour (property violations) help in performing requirements analysis .

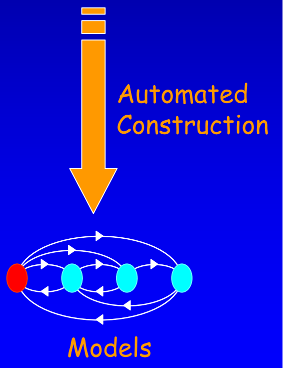
Motivation

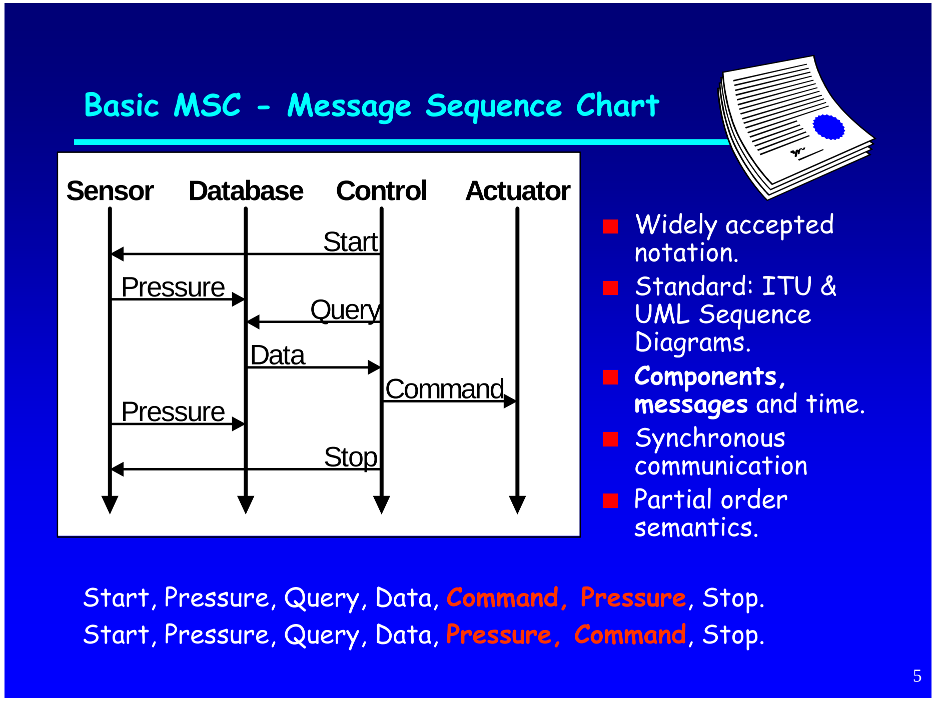

Models ! analysis,animation, properties, blah blah, …
I'm a practitioner - get me out of here!

1. model synthesis



- Statechart models [Khriss et al, Krüger et al, Whittle and Schumann]
- Live Sequence Charts [Harel]
- OO models [Koskimies, Systä et al]
- ROOM models [Leue]
- Timed Automata models [Somé]
- LTS models in FSP [Uchitel et al]

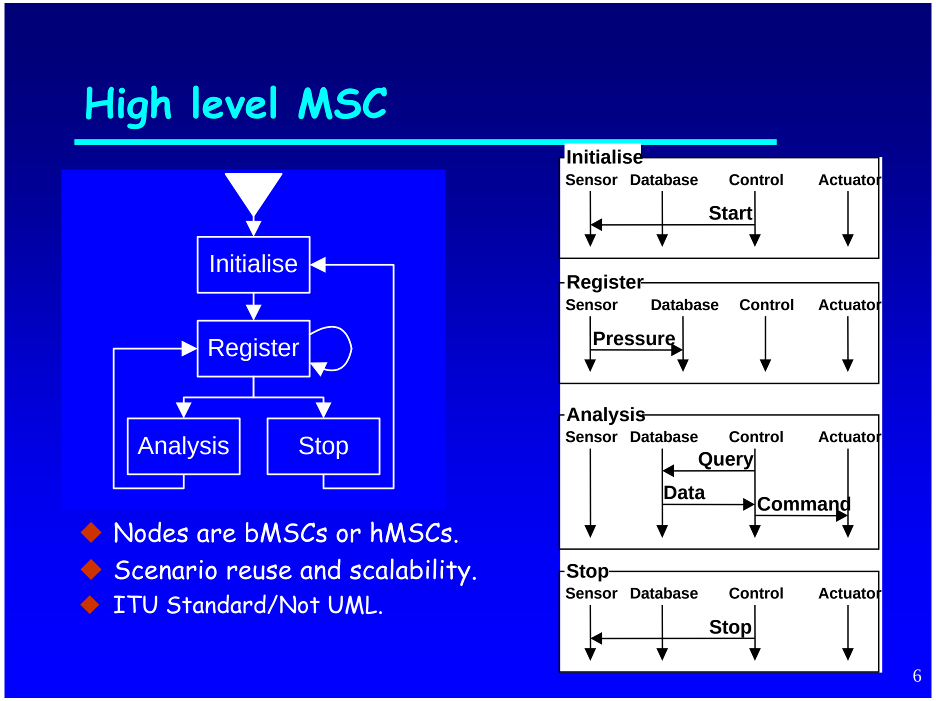




Basic MSC - Message Sequence Chart

- Widely accepted notation.
- Standard: ITU & UML Sequence Diagrams.
- **Components, messages** and time.
- Synchronous communication
- Partial order semantics.

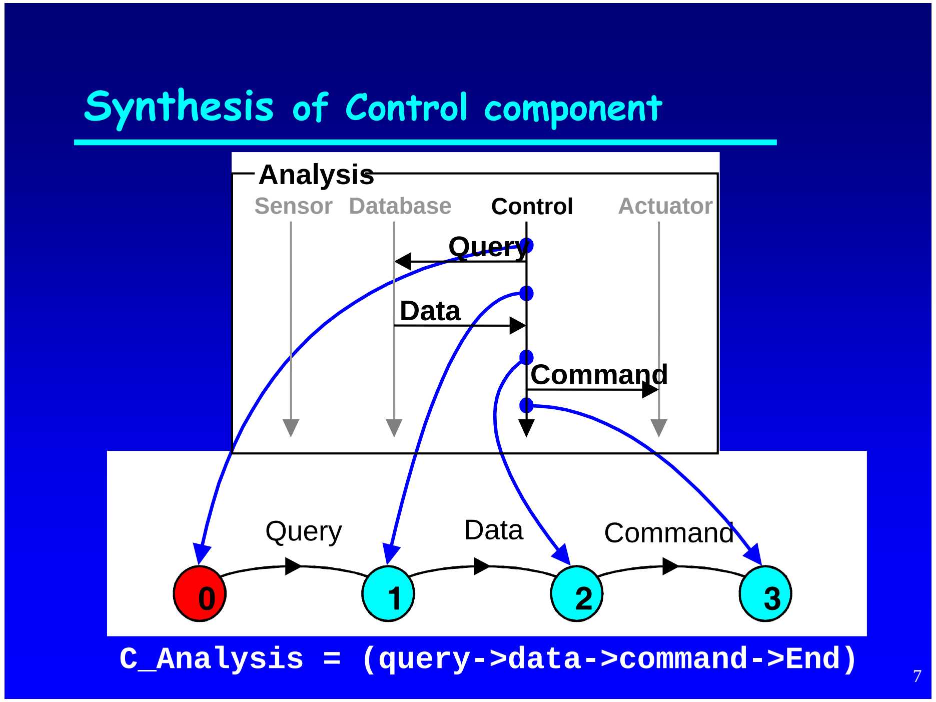
Start, Pressure, Query, Data, Command, Pressure, Stop.
Start, Pressure, Query, Data, Pressure, Command, Stop.



High level MSC

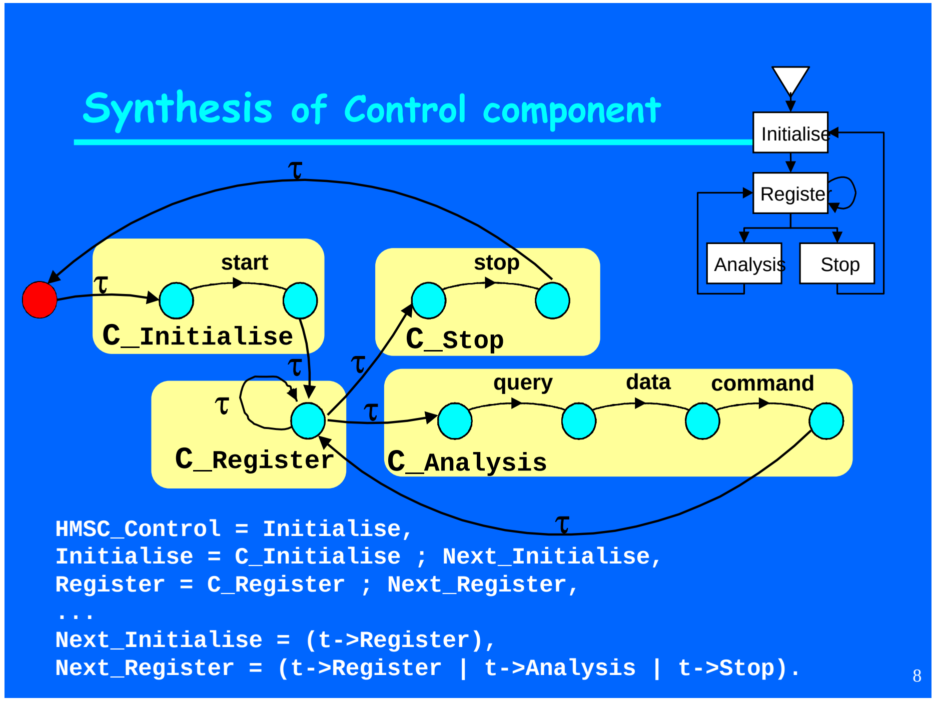
Initialise
Register
Analysis
Stop
Initialise
Sensor Database Control Actuator
Start
Register
Sensor Database Control Actuator
Pressure
Analysis
Sensor Database Control Actuator
Query
Data
Command
Stop
Sensor Database Control Actuator
Stop

- Nodes are bMSCs or hMSCs.
- Scenario reuse and scalability.
- ITU Standard/Not UML.



Synthesis of Control component

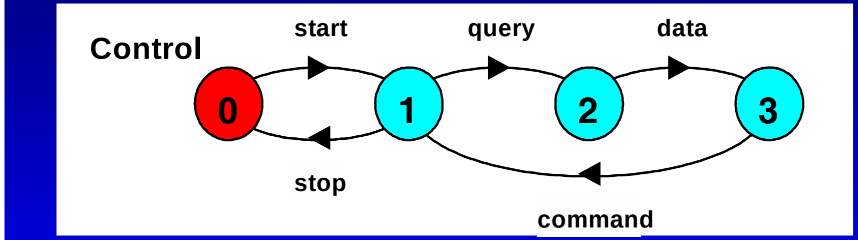
```
C_Analysis = (query->data->command->End)
```



Synthesis of Control component

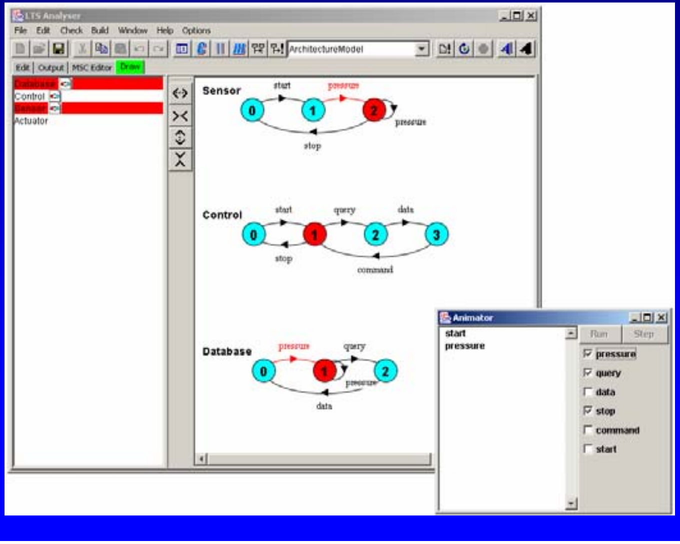
```
HMSC_Control = Initialise,
Initialise = C_Initialise ; Next_Initialise,
Register = C_Register ; Next_Register,
...
Next_Initialise = (t->Register),
Next_Register = (t->Register | t->Analysis | t->Stop).
```

Synthesis of Control component



```
deterministic ||Control = HMSC_Control\{t}.
```

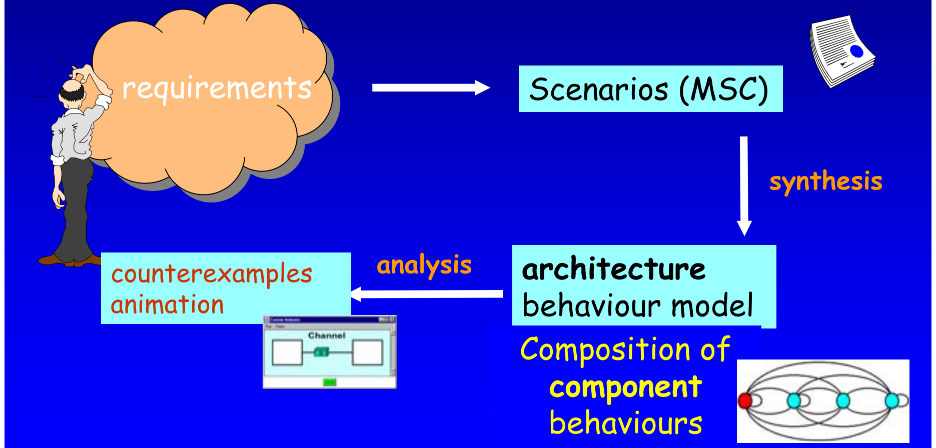
component behaviour model - animation



start
pressure

☑ pressure
☑ query
□ data
☑ stop
□ command
□ start

Model synthesis from Scenarios



requirements

Scenarios (MSC)

synthesis

architecture behaviour model

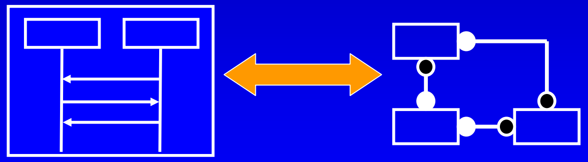
Composition of **component** behaviours

analysis

counterexamples
animation

2. What about Complex Systems?

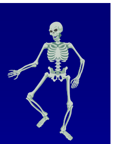
Architecture-based Synthesis:
scenarios and architecture fragments



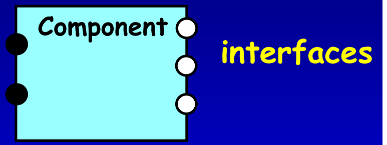
MSCs

ADLs
(Architecture Description Languages)

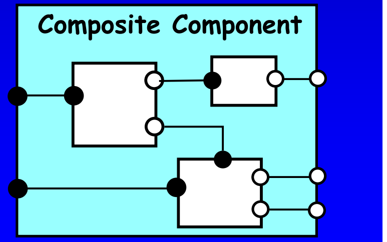
Darwin ADL - structural view



- *Component types* have one or more interfaces. An interface is simply a set of names referring to actions in a specification or services in an implementation, **provided** or **required** by the component.

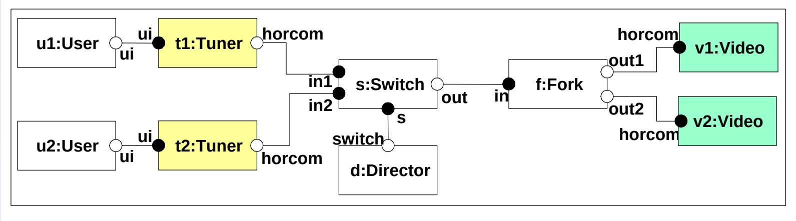


- *Systems / composite component types* are **composed** hierarchically by component **instantiation** and interface **binding**.



The motivation: A real system

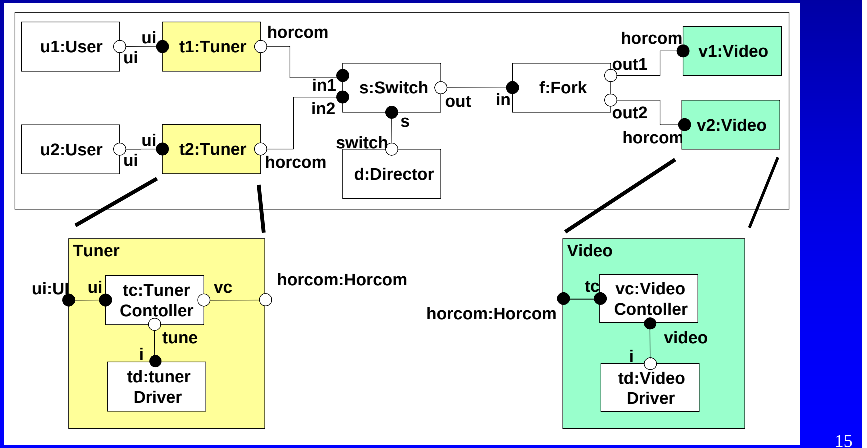
- A family of TVs that support multiple tuners and video output devices …



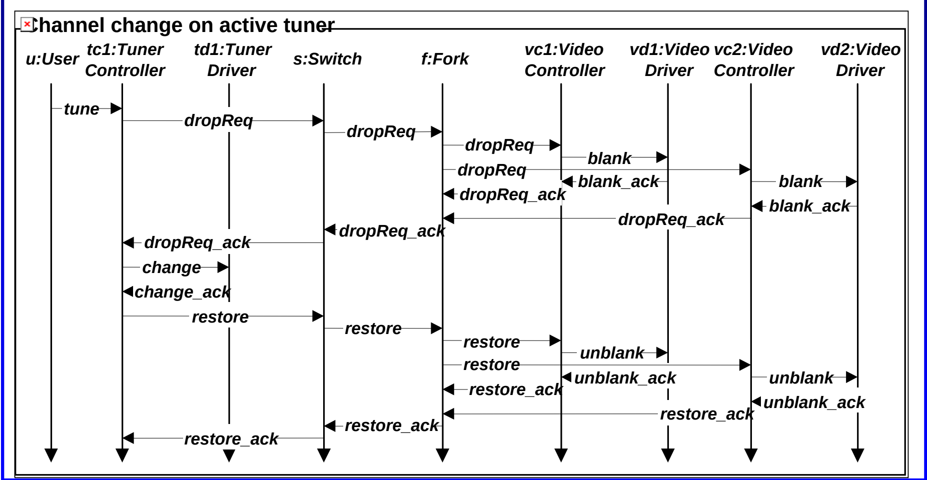
TV with 2 tuners – 2 Videos specified in **Darwin**

The motivation: A real system

Component types in a hierarchy



A scenario for a real system …



More complex?

How should we go about describing systems with

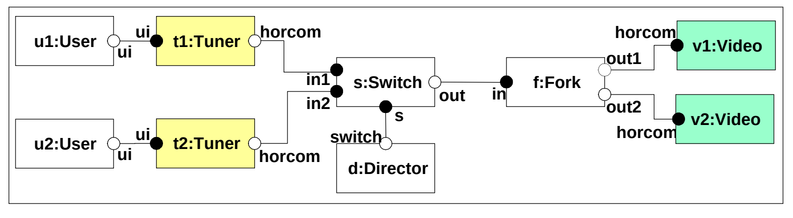
… additional tuners/video devices?

… introduce other complex devices?

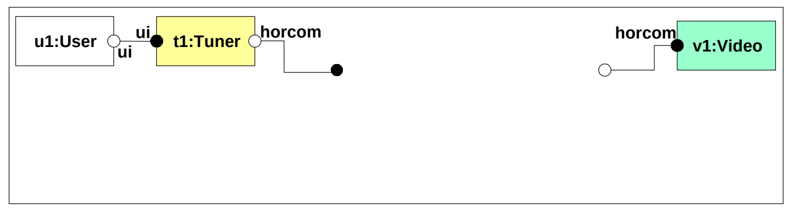
Do we have to use more complex scenarios?
Or is there an alternative approach …

> *Can we build complex systems by **composition**, using combinations of **simpler architectural fragments** …?*

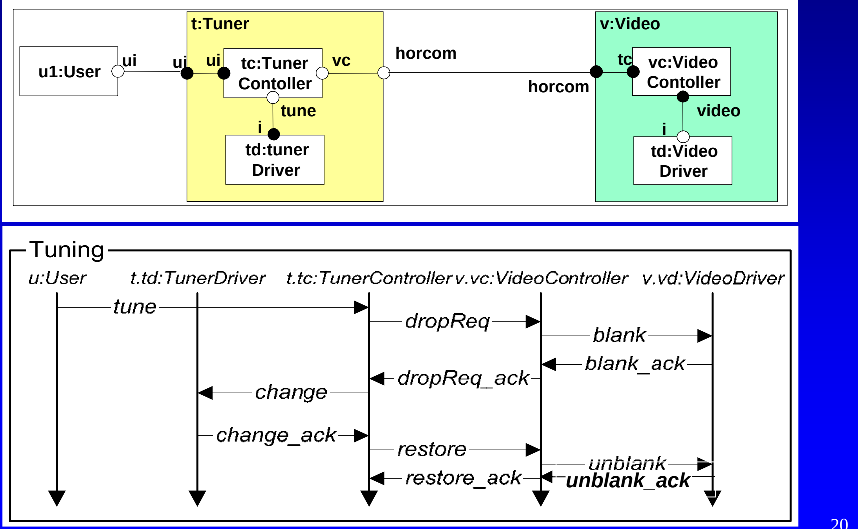
Simpler architectural fragments …

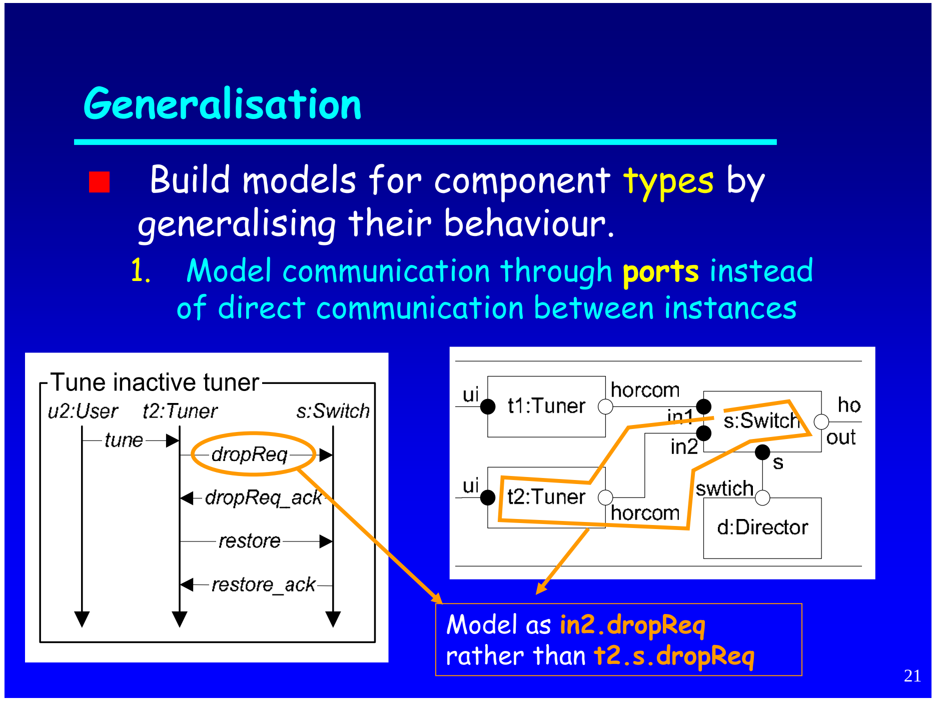


Simpler architectural fragments …



Simpler architectural fragments …

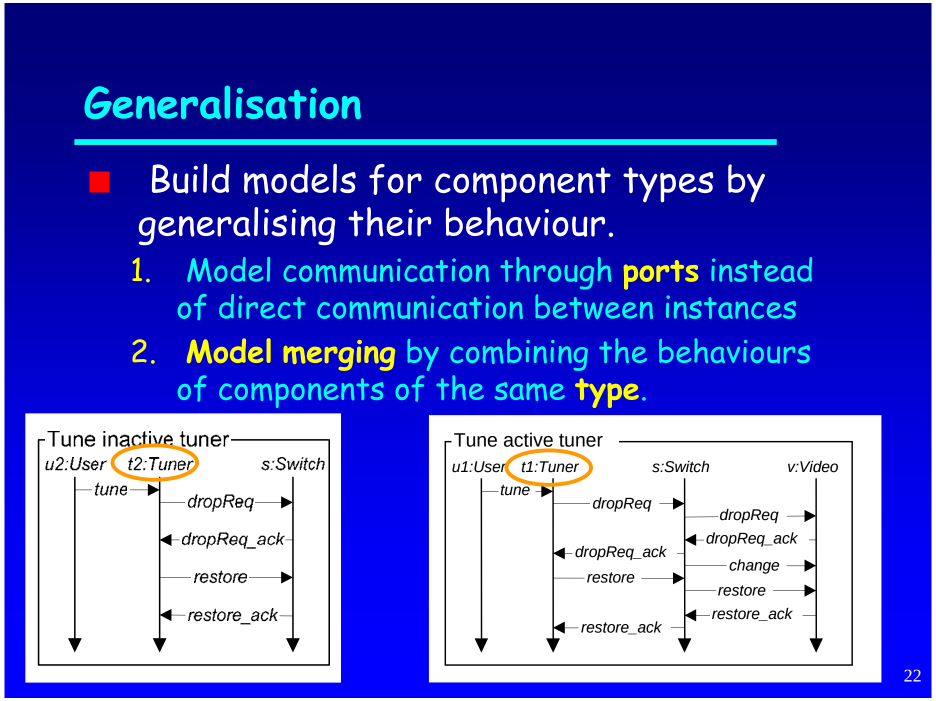




Generalisation

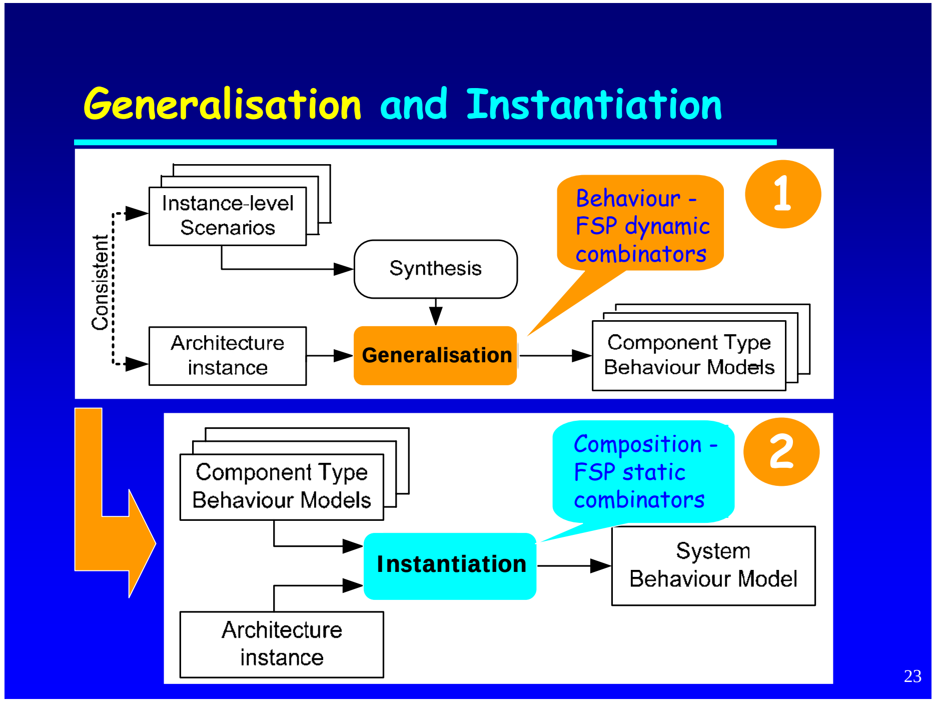
- Build models for component types by generalising their behaviour.
  1. Model communication through **ports** instead of direct communication between instances

Model as in2.dropReq rather than t2.s.dropReq

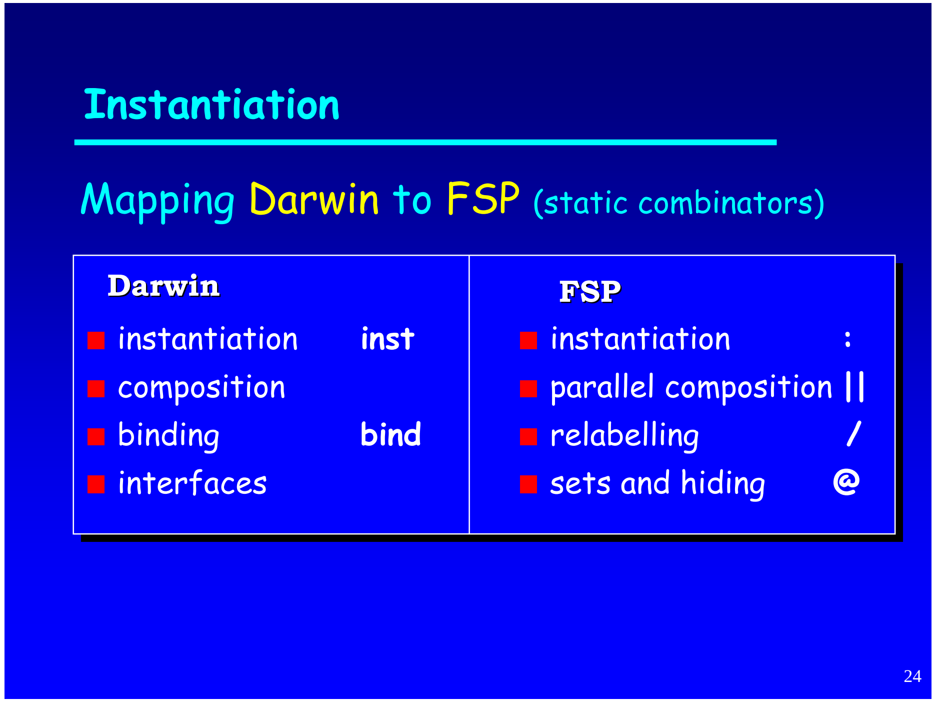


Generalisation

- Build models for component types by generalising their behaviour.
  1. Model communication through **ports** instead of direct communication between instances
  2. **Model merging** by combining the behaviours of components of the same **type**.



Generalisation and Instantiation



Instantiation

Mapping Darwin to FSP (static combinators)

Darwin		**FSP**	
instantiation	inst	instantiation	:
composition		parallel composition	||
binding	bind	relabelling	/
interfaces		sets and hiding	@

3. Scenarios, architectures, models …

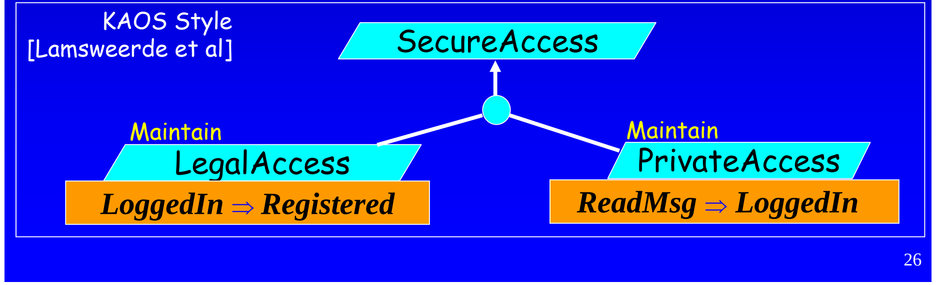
What about …

GOALS ?

Goals

- Declarative statement of intent about system behaviour
- Goal graphs structure and model refinement relations

The web mail system shall provide secure access to email in that a user must be registered before he/she can logon and must be logged in before he/she can read email via Web browser.

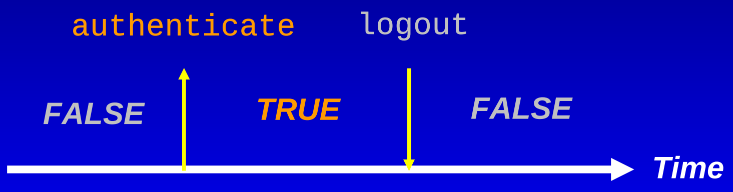


Early Validation of Requirements

- Goals and Scenarios are complementary
  - State vs. Events
  - Declarative vs. Operational
  - General vs. Example
  - What & Why vs. How
- Crucially, a formal relation between goals and scenarios needs to be defined…

Fluent Propositions

Defined in terms of sets actions



```
fluent LOGGEDIN =
     <{authenticate},{disable,logout}>
                            initially False
```

[Magee & Giannakopoulou, ESEC/FSE'03]

Defining Fluents

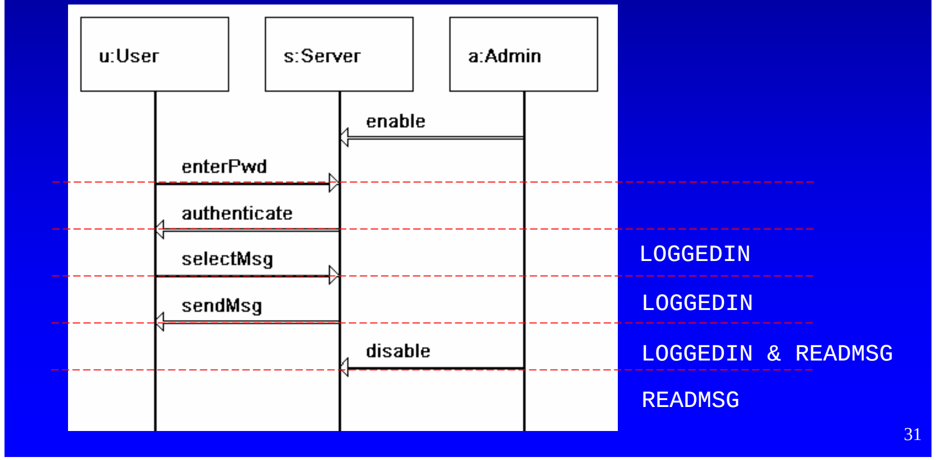
```
fluent LOGGEDIN
   = <authenticate,{disable, logout}>
fluent REGISTERED
   = <enable, disable>
fluent READMSG
   = <sendMsg, {closeMsg, logout}>
//-----------------------------------------
assert SecureAccess
   = [] (LegalAccess && PrivateAccess)
assert LegalAccess
   = [] (LOGGEDIN -> REGISTERED)
```

Goals as properties

- Fluents provide basis for model-checking operational descriptions against goals.
- We can build animations that execute scenarios but present them in terms of the goals

Violation of PrivateAccess

```
assert PrivateAccess =
     [](READMSG → LOGGEDIN)
```

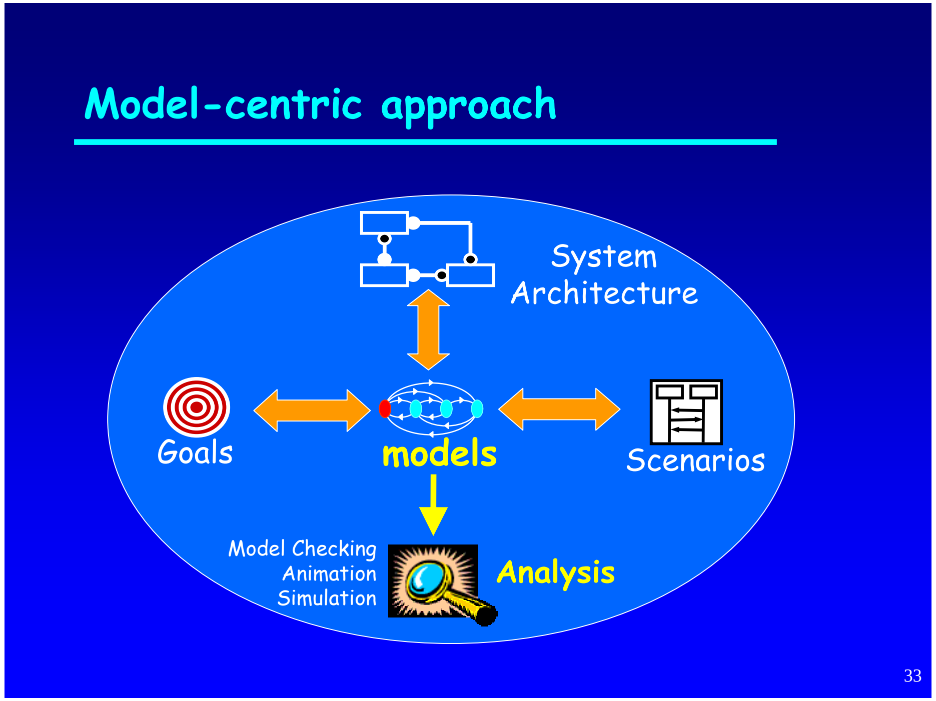


The tool support: LTSA

Extended LTSA to deal with Fluents and FLTL.

Plugins developed for:

- ***Model synthesis from Message Sequence Charts***
- ***Model generation from Darwin architecture description.***
- *Graphic Animation of Models*



Model-centric approach

System Architecture

Goals

models

Scenarios

Model Checking
Animation
Simulation

Analysis