Using OCL for expressing temporal validity constraints

Juliana Küster Filipe and Stuart Anderson
LFCS, School of Informatics
The University of Edinburgh
United Kingdom

SVERTS@UML 2003, 20 October, San Francisco, California

Background

In DIRC, a UK EPSRC funded project, we are looking at *dependable socio-technical systems*.

In particular, we are interested in:

- the specification and design of large-scale distributed dependable systems,
- how formal approaches can be used to analyse dependability requirements and help designers
  $\Rightarrow$ *developing verification tools*

Context

- Dependability of a system reflects a property of that system such that "reliance can justifiably be placed on the service it delivers" (Laprie).
- Attributes of *dependability* are reliability, availability, safety, integrity, security, and so on.
- One aspect we are particularly concerned with here is *data integrity* in a distributed real-time application with replicated data.

Problem

- Distributed application where tasks on several nodes/components require access to the same data.

- There are many alternatives to deal with this...

  – *Data replication*
    data is duplicated in several locations; local copies of replicated data have to be updated (for consistency).

$\Rightarrow$ Clients have different temporal validity constraints on the data (accuracy).

Temporal Validity in Design

- At the design level we are not concerned with the procedures that are implemented to make sure that the data replications are kept consistent.

- We are concerned only with the *constraints* that we want to impose (at the component or architectural level) and that have to be satisfied by these procedures.

Publish/Subscribe

- A client "subscribes" to the server to be notified about the changes on the value of some data according to some *policy*.

- A policy can be "when the value changes", "at least every so often", "at most every so often", and so on.

- These policies reflect an aspect of a component *contract*, which we need to express at the design level. *They may reflect temporal validity constraints.*

Our Approach

- We consider UML as a modelling language for the (system and detailed) design of distributed real-time applications.

- In particular, we use OCL to capture the required temporal validity constraints (we need an extension of what is the standard $\leadsto$ time-enriched liveness template).

- The OCL constraints are mapped onto a logic, in this case a *real-time temporal logic of knowledge*.

ParcelCall

- Explored the development of a low cost information infrastructure that improves business processes in transport and logistics by enabling the continuous information of the exact geographic position of parcels at any time (*Parcel localisation system*)

- Open distributed system which integrates with the legacy systems of the transport and logistic companies (carriers).

- Carriers can offer more services to customers.

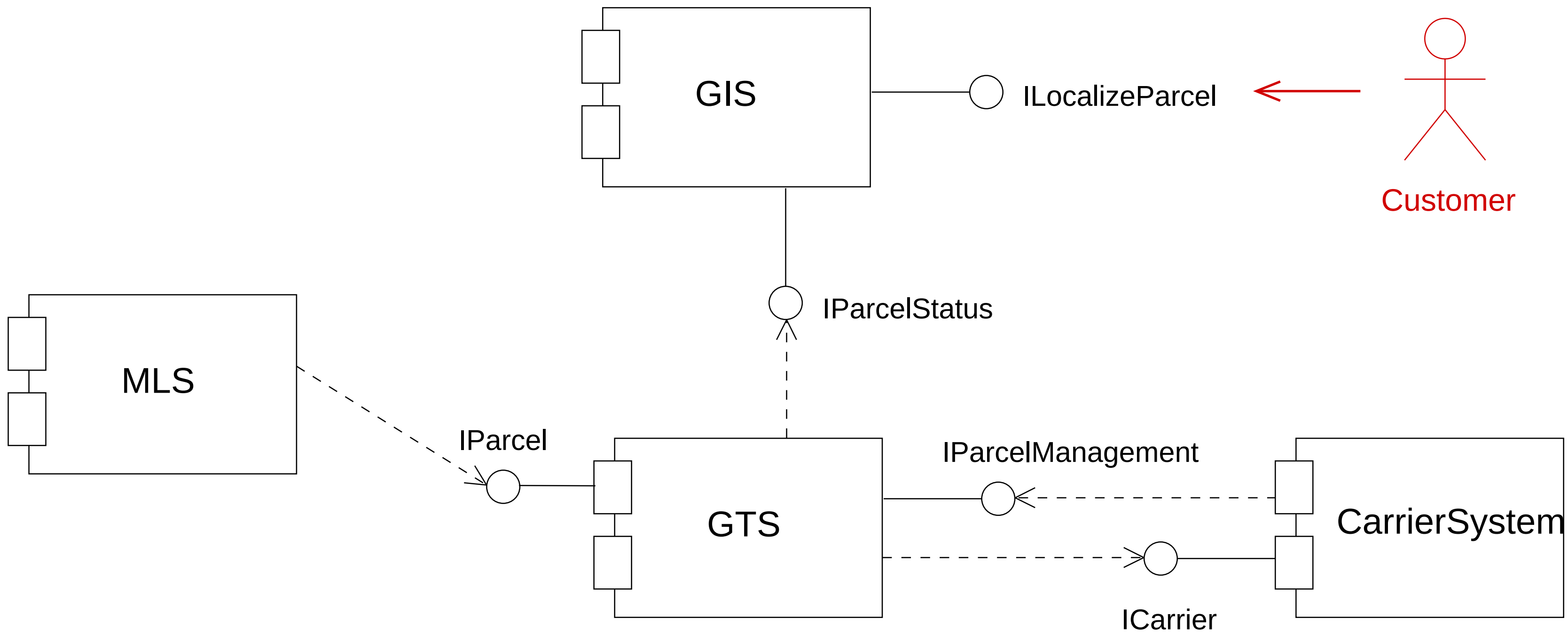
ParcelCall components

- *Mobile Logistic Server* (MLS): exchange points, transport units (container, trailer, freight wagon, etc). Units carry the parcels. MLS's build hierarchies.
- *Goods Tracing Server* (GTS): databases containing MLS hierarchies. Knows about registered parcels. It is integrated with the legacy system of carriers.
- *Goods Information Server* (GIS): interacts with the customers, and provides the authorised customer the current location of her parcels, keeps her informed in case of delivery delays, etc.

Where is my parcel?

- A customer can query the location or status of her parcel at any time.
- How accurate provided information can be depends on the established delivery agreements at send time.

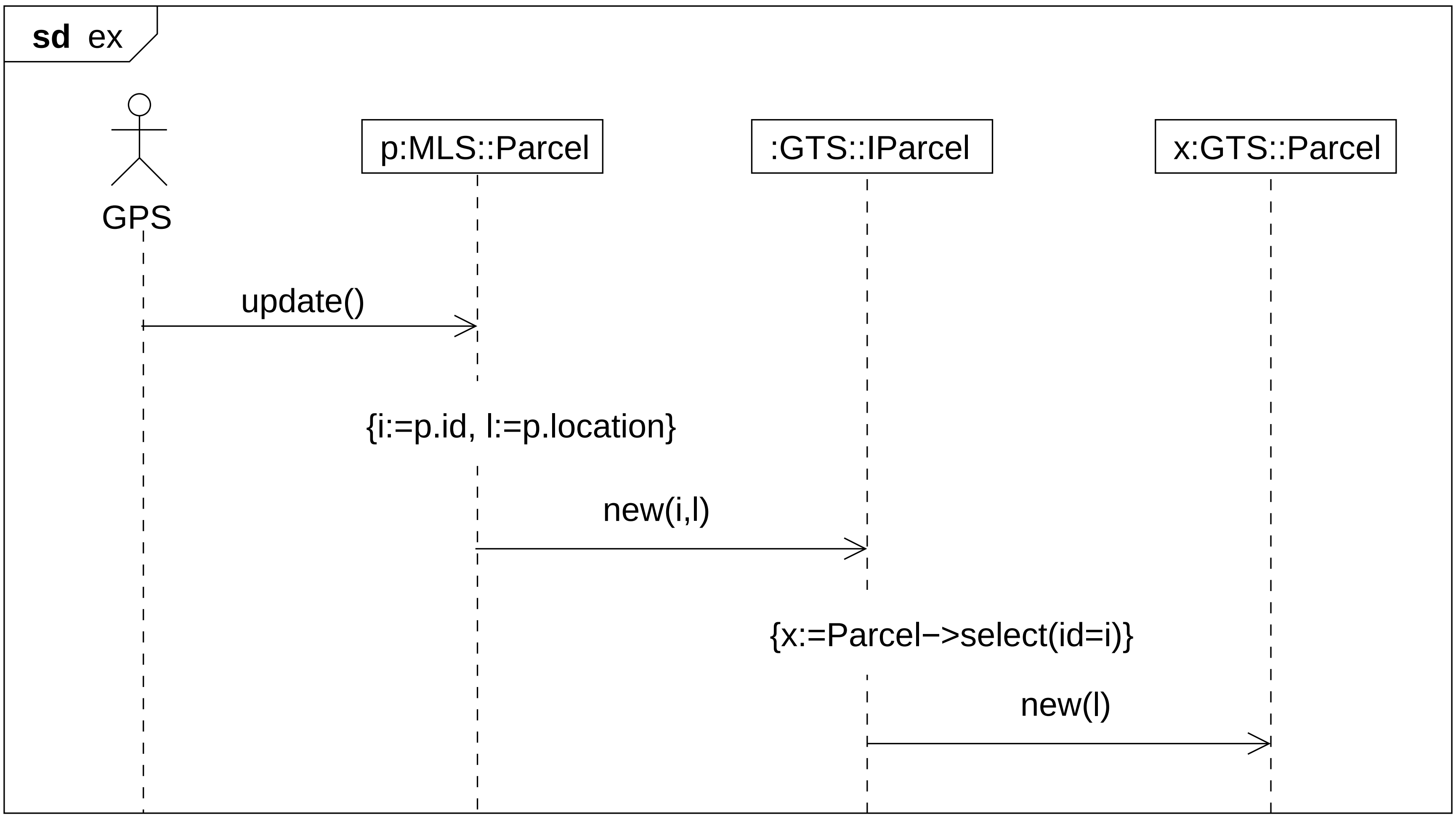
ParcelCall Architecture



Assumptions

- MLS: there is a class `Parcel` with attributes `id` and `location`, and an operation `update()` which updates the value of `location`.
- GTS: `Parcel` is replicated with attributes `id` and `location`, and an operation `new(l)` which updates the value of `location` to `l`.
- GIS: `Parcel` is replicated with attributes `id`, `location` and `deliverymode`, and an operation `new(l)` which updates the value of `location` to `l`.

An illustration



Contracts in OCL

context `MLS::Parcel`
  after: `self.update()`
  eventually: `GTS::IParcel::`
                `new(self.id,self.location)`

here MLS eventually publishes the changes on the parcel location. *This is not standard OCL.*

Temporal Validity in OCL

context `GTS::Parcel`
  after: `new(a)`
  eventually: `new(b)`
  within: `t`

context `GIS::Parcel`
  after: `new(a)`
  eventually: `new(b)`
  within: `deliverymode` $\times$ `10`

in these cases a time constraint has been added.

Logics of Knowledge

- Epistemic modal logics, or modal logics of knowledge, originated in work by J. Hintikka in the 1960s to formally capture some intuitions about the nature of knowledge.

- Knowledge can change throughout time (through local observations, communication, etc).
  $\Rightarrow$ Temporal logics of Knowledge.

- Numerous applications in AI and distributed computing.

Knowledge and Real Time

- What about real time? No known work here.
- Certain observations have a limited temporal validity.

  Take an observation $\Rightarrow$ gain some knowledge

  $\Rightarrow$ but it will elapse at some point.

Real-time Temporal Logic of Knowledge

$$\varphi := \text{false} \mid p \mid \varphi \Rightarrow \varphi \mid K_j\varphi \mid \langle a \rangle\varphi \mid \varphi\, \mathcal{U}_{\theta c}\, \varphi$$

- p is an atomic proposition, a is an action, j is a system component, c is a rational number, and $\theta \in \{<, \leq, =, \geq, >\}$

- the K operator gives us a notion of *locality*.

Example Formulae

```
context MLS::Parcel
  after:  self.update()
  eventually:  IParcel::
                  new(self.id,self.location)
```

$$K_{MLS::Parcel}(\langle self.update() \rangle$$

$$\mathcal{F}_{>0}\ (\langle IParcel :: new(self.id, self.location) \rangle\ true))$$

Example Formulae(2)

context `GTS::Parcel`

after: `new(a)`

eventually: `new(b)`

within: `t`

$$K_{GTS::Parcel}(\langle p.new(a)\rangle \mathcal{F}_{<t}\ \langle p.new(b)\rangle\ true)$$

Conclusions

- Timing constraints in general, and *temporal validity* constraints in particular, should be captured earlier as precise component contracts or local timing constraints.

- The constraints may reflect choices already: push versus pull or a combination of these.

Conclusions (2)

- We do not need (or *want*) a very expressive temporal OCL: a timed liveness template is enough!
  $\Rightarrow$ Let other diagrams do the rest:

  tlt + Seq.Diag. UML2.0 $\leadsto$ Time-enriched LSCs

- Mapping extended OCL into our logic is straightforward (both are *locality*-based).

- Verification is possible: *data integrity constraints* can be verified. Failures can be detected at an early stage.