What Would an Alternate Approach to Software Look Like?

DIMITRIS LYRAS
Director Ulysses Systems

What makes shipping different for the software industry

Ship Management is a co-ordination business and risk management business.
Remember the Hebei Spirit, remember how many decision only last week were made last week involving hundreds of thousands of dollars.
Its not the transportation that differentiates us in how we use software to manage our businesses
Its the co-ordination intensity and risk levels we face

No matter how elegant the software design

Many examples show us that improving the software is hard:

ERP systems are inflexible to changes making upgrades expensive.

UI's often make the software difficult to use because it so hard to adapt the UI's to client needs. Imagine of there are hundreds of UIs to change for each data column needing to be added

No matter how elegant the software design

ERP providers that sell software in more than one vertical have even greater problems with flexibility making them Expensive to adapt to client needs, Expensive in customizations and Expensive in upgrades

ERP providers often suggest you discard all your customizations and start again, when they come up with a major upgrade

Domain Situational Awareness

Usability deteriorates with an inflexible ERP not only because of the UI updating problem, but also because of DOMAIN SITUATIONAL AWARENESS

By domain situational awareness we mean the awareness the system needs so as to stop annoying people with notifications and requirements that have little relation to current situation and priorities

Domain Situational Awareness

There are many examples:

1. Maintenance intervals that are not sensitive to ambient working conditions.
2. Inventory systems that are not tuned to component lifecycle
3. Quality management procedures that are unrelated to current operation of the vessel
4. Safety management checklists that have questions that have no relationship to the current situation, likewise for risk management templates.

Domain Situational Awareness

5. Crew work rest period systems that are unaware of current ships operation.
6. Maintenance or asset management systems unconnected to crewing systems
7. Purchasing systems that can't alert buyers to ships inspection deadlines or other urgent needs or adjust to ships itinerary changes.
8. Change management and auditing systems that adapt to daily changes of itineraries, costs, priorities.

No wonder senior managers and busy users find software to be bureaucratic:
Without DOMAIN SITUATIONAL AWARNESS software is always going to be bureaucratic

Situational Awareness

Who will make software SITUATIONALLY AWARE?

Why would an ERP vendor addressing 10 or 20 verticals integrate software modules that interact differently in each vertical. Asset Management, Procurement, HR, and Operations interact very differently in each vertical?

Only a vendor who can truly integrate different modules to achieve DOMAIN SITUATIONAL AWARENESS

Situational Awareness

You may ask how this all ties together? Is a different way to build software also a way to achieve domain situational awareness?

Two things help achieve domain situational awareness;

1) Software which makes it easier to model the real world
2) Software dedicated to one vertical. Just one

Positives of software that is hard to change

Indeed there are, conventionally built software is hard to plagiarize

Software companies seeking an easy way to take ideas from other software cannot easily make the changes work because it’s too much trouble.

It’s hard to figure out how the other software works because there is not enough information in the data schema. The schema tells you what’s there but not why tis there.

Positives of software that is hard to change

It’s la bit like copying someone else’s movements in a sport like football or boxing

In addition it’s hard to adapt software features from another system because of the many UI and logical adaptations needed. This may be why software architecture has remained inflexible

So, for systems needing little improvement this is OK!

But which systems do you know that don’t need improvement?

Plagiarisation

Competition Announcement Summer 2018

You achieve the flexibility to revise jobs at fleet level and can structure components based on makers and model numbers. This means you can manage tasks and procedures for a given model in the entire fleet by categorizing maintenance jobs into procedures and demands from the maker, class and company. With Jobtext Management, it becomes easy to prevent reoccurrence of defects on different ships with the intelligence gained through a central database.

Jobtext Management assists in governing a uniform job structure by inviting to align jobs. Updates and job procedures are centrally managed, and automatically distributed to relevant vessels. This includes the distribution and management of service letters from manufacturers.

Key Features

- Technical data available on board the vessels
- Flexibility to change procedures and parameters on several ships at once
- Advanced document management with a controlled document flow

Benefits

- Optimized Asset Management
- Share knowledge on fleet level
- Improved task performance
- Prevent reoccurrence of defects

Ulysses SPD 2007

COMPONENTS LIBRARY MANAGEMENT

- 'Ulysses' have developed a set of tools that allow the PMS administrator to manage a central Library with all the components that can then be assigned to ships.
- Excel Spreadsheet export and input of components.
- Quality control facilities to help understand the quality of population of components before they are assigned to other ships.
- Ability to share components between companies and the central components repository held by Ulysses.

Ulysses Client Testimonial 2013

Is the data easy to manage and improve?

Although we have **data on 328.000 individual items of machinery** such as pumps prime movers etc., the data we actually manage in the system is only **20.000 unique machines**. The remaining **308.000 machinery items share common data**, thus adding hugely to the ease of management **cost of inventory and reusable value** of the know-how collected.

Perhaps a Better way to build Software

Normally when we design software

we deal with things that the computer does well

and we don’t do well as humans

Such as remembering thousands of schedules

of machinery maintenance

Perhaps a Better way to build Software

But it’s easy to get over involved in the details

and unaware of the more important decisions made

in an enterprise that constitute

the real knowledge and the real risks in the enterprise

Perhaps a Better way to build Software

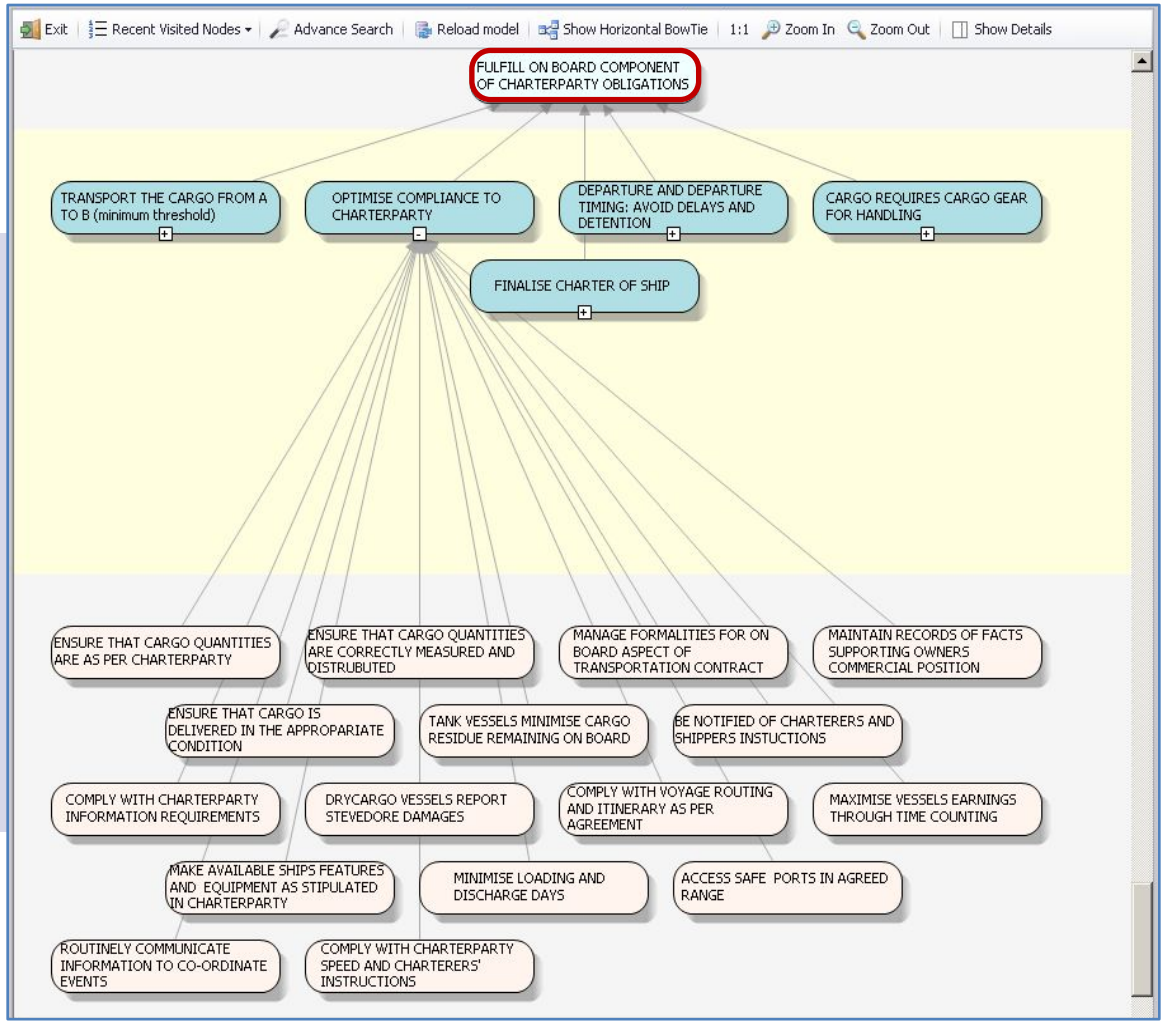
Goal models make it possible

to join the processes in an enterprise

into a converging goal based graphical structure

Transportation according to charter party agreement

A model of how technical functions,
commercial processes,
and ships operating processes
converge to fulfil the goal
of satisfying charter party obligations

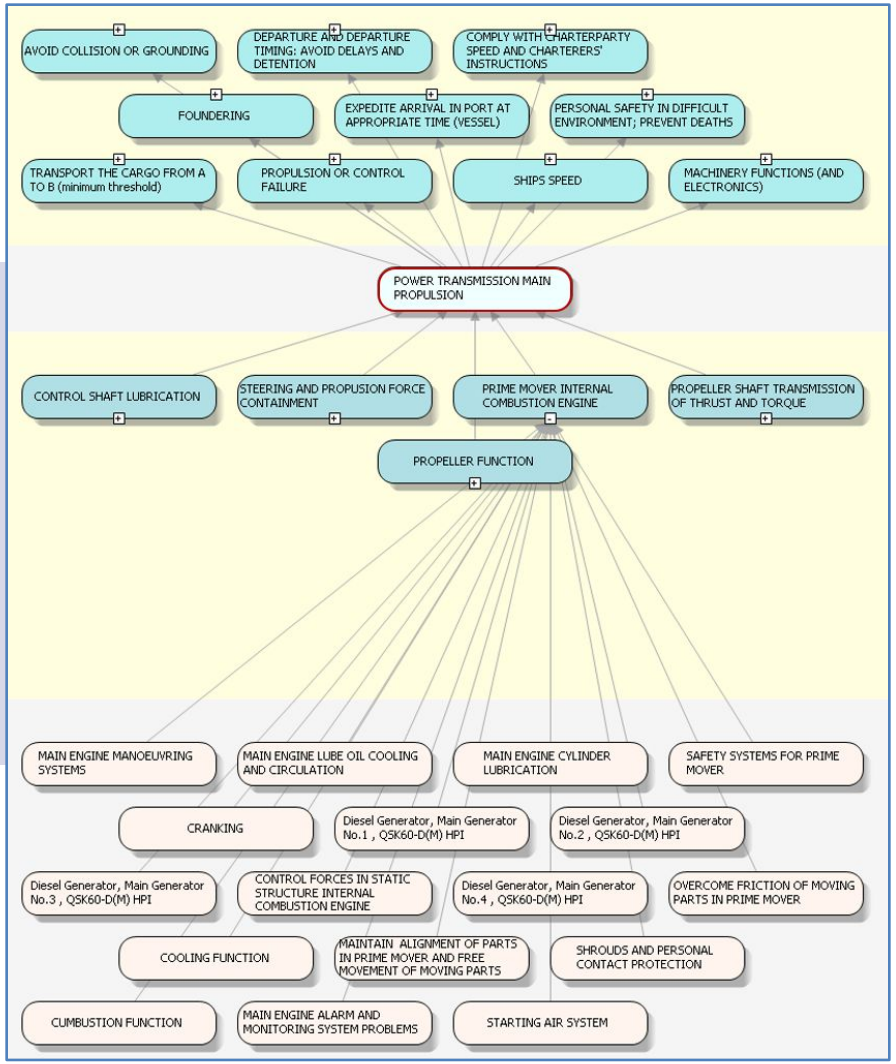


PMS isn’t only about remembering that some asset needs maintenance

The **propulsion function**

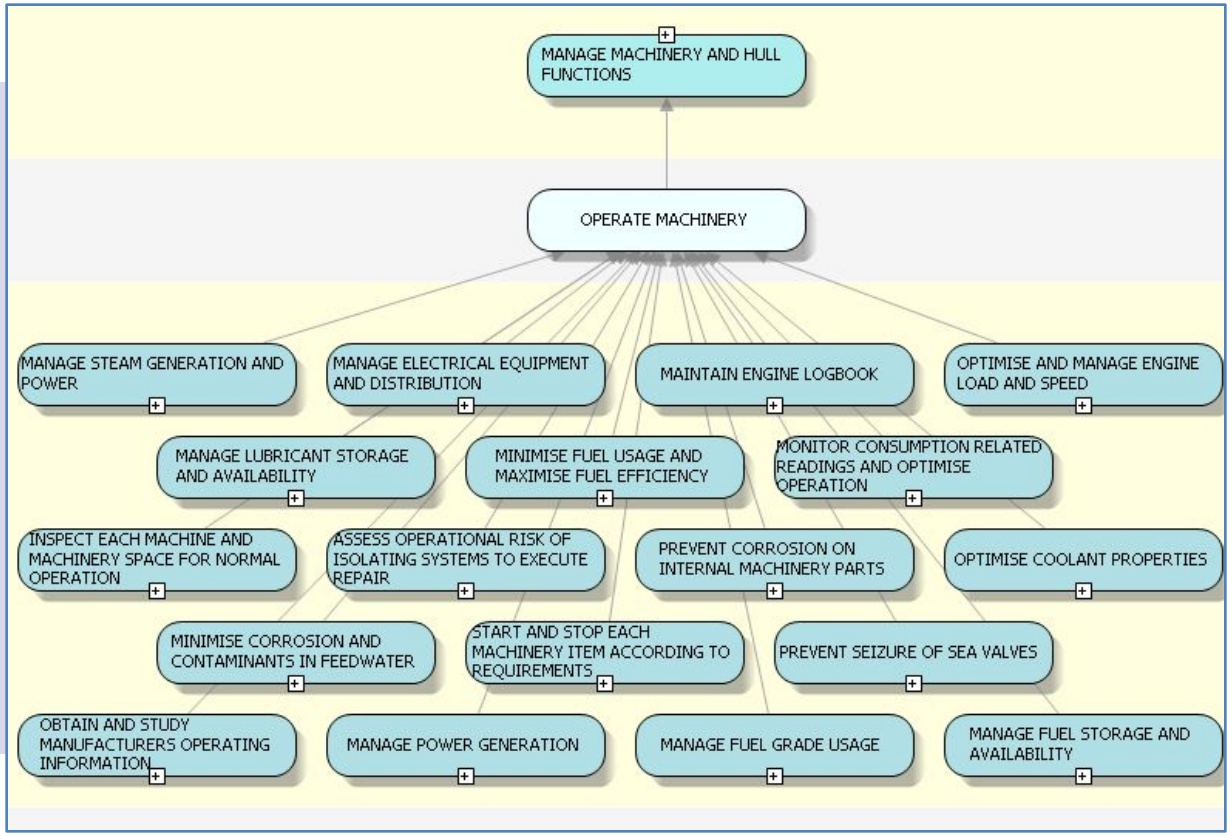
of a ship affects ship processes

and depends on other machine functions



Asset management is a small part of operating machinery

In a classic
Planned Maintenance System,
on-board operating and handling
of machinery is not modelled
although it is just as important
as the scheduling processes

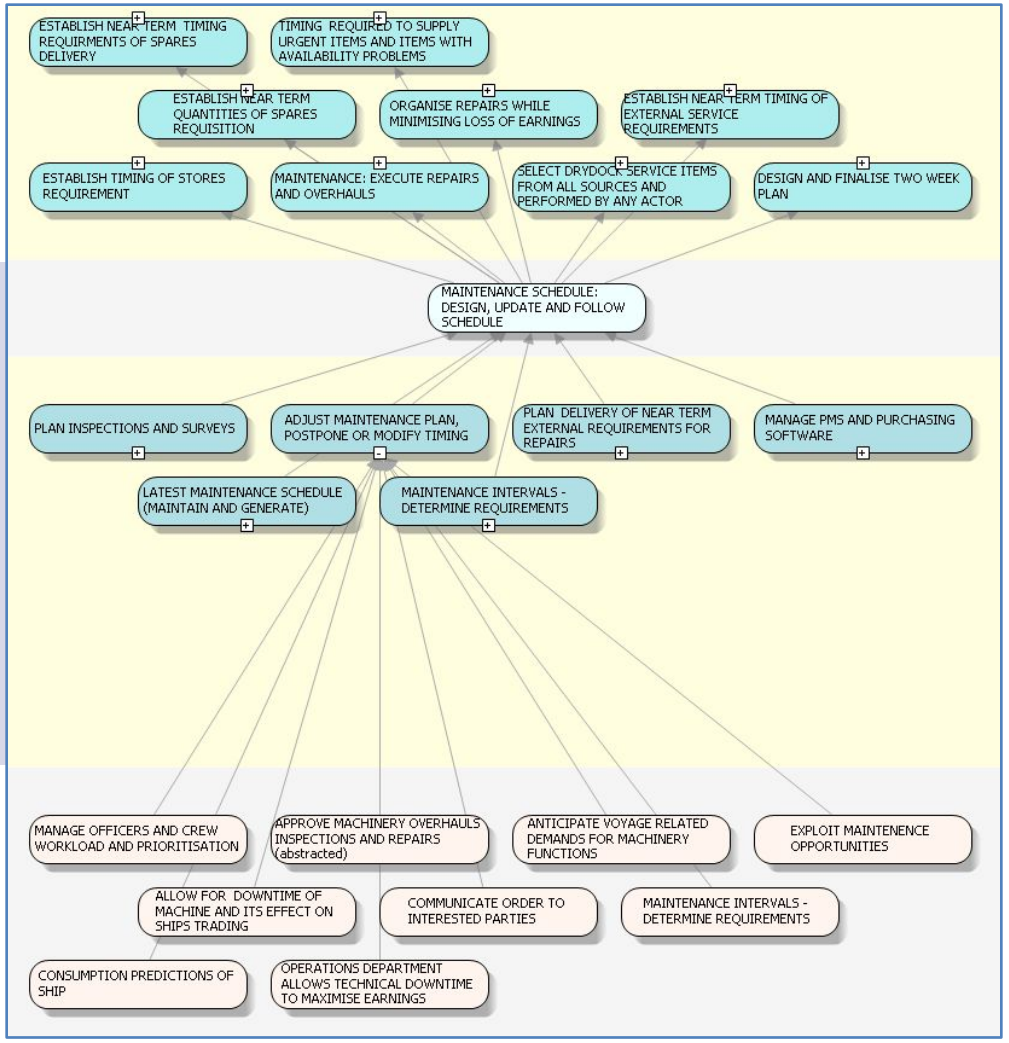


The domain expert's approach is to model both Data and Logic

starting from higher level processes,

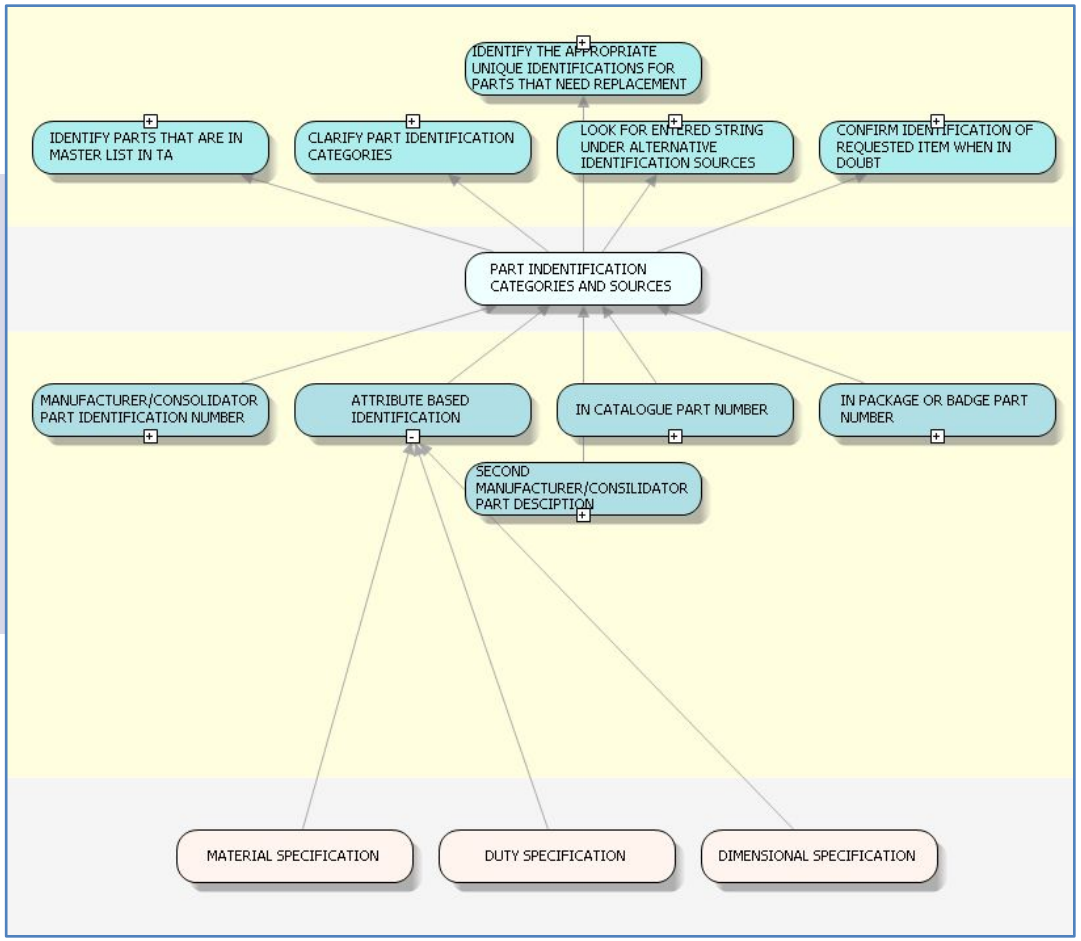
and going lower to more

detailed and less encompassing processes



The domain expert's approach is to model both Data and Logic

all the way
to very granular detail
we often see in a
Planned Maintenance System



Evolving the software

Adjusting to new requirements

in conventional software development:

Above all it is likely that with conventional development

the detail aspects are not easy to evolve

after the original design has matured

Why is conventional software inflexible?

This is because the building blocks

of conventional software are aimed at making the software function.

They are not aimed at understanding

the world around the software that is often

many times larger than what we build into software…

Current software practices make it hard to make improvements

Data is incorporated in related objects which make it hard to make changes.

In addition the code is not in a map or Mimic diagram like we use when we emulate complex systems for example main engine control systems

Current software practices make it hard to make improvements

We have event logs but event logs nether tell us how the event came about nor how the logic connects the events that take place.

The result is a system that is hard to change because we cannot be sure how the code interacts with the data because we can’t see how the code works as we would if we had a mimic diagram as we have in engine room automation..

Easily adapting software design to the real world

…a world that is bound to stretch the limits

of the original software design as more and more concerns

need to be added to the part originally computerised

Make the software easy to extend

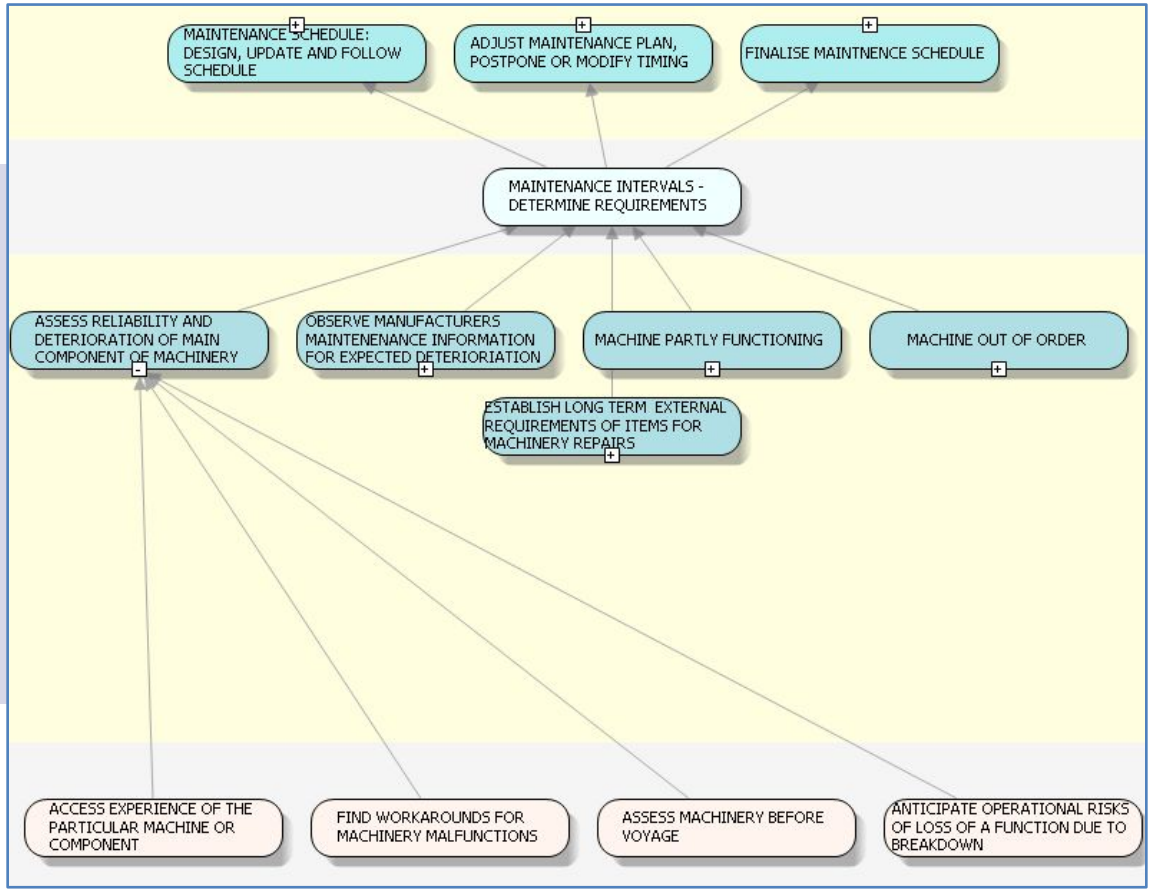
For example,

in Asset Management we provide intervals between maintenance activities

but we do not persist decisions that refine the intervals

Maintenance Intervals: Persisting decisions & concerns

However, the domain stakeholder
will benefit
from software that persists
decisions and concerns



Make the software easy to extend

We do identify parts needing replacement

but do we anticipate the variety of identifications

of the parts we might be using and the problems

of not being able to reconcile

this variety of descriptions**?**

Let the software show you what it knows

By using models we can go

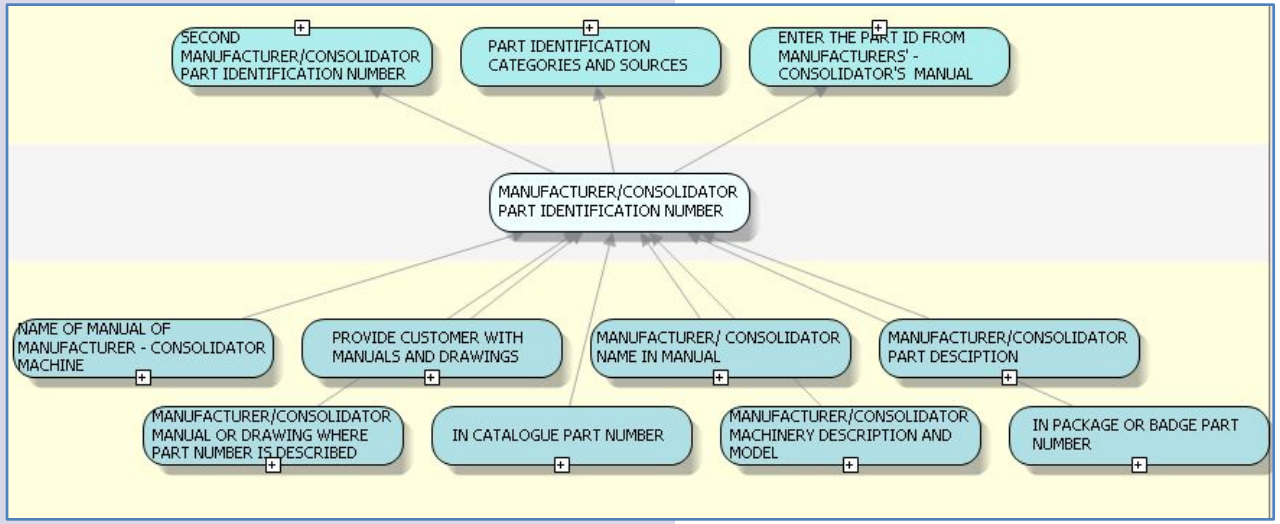
from high level goals

to granular transactions,

which remain readable

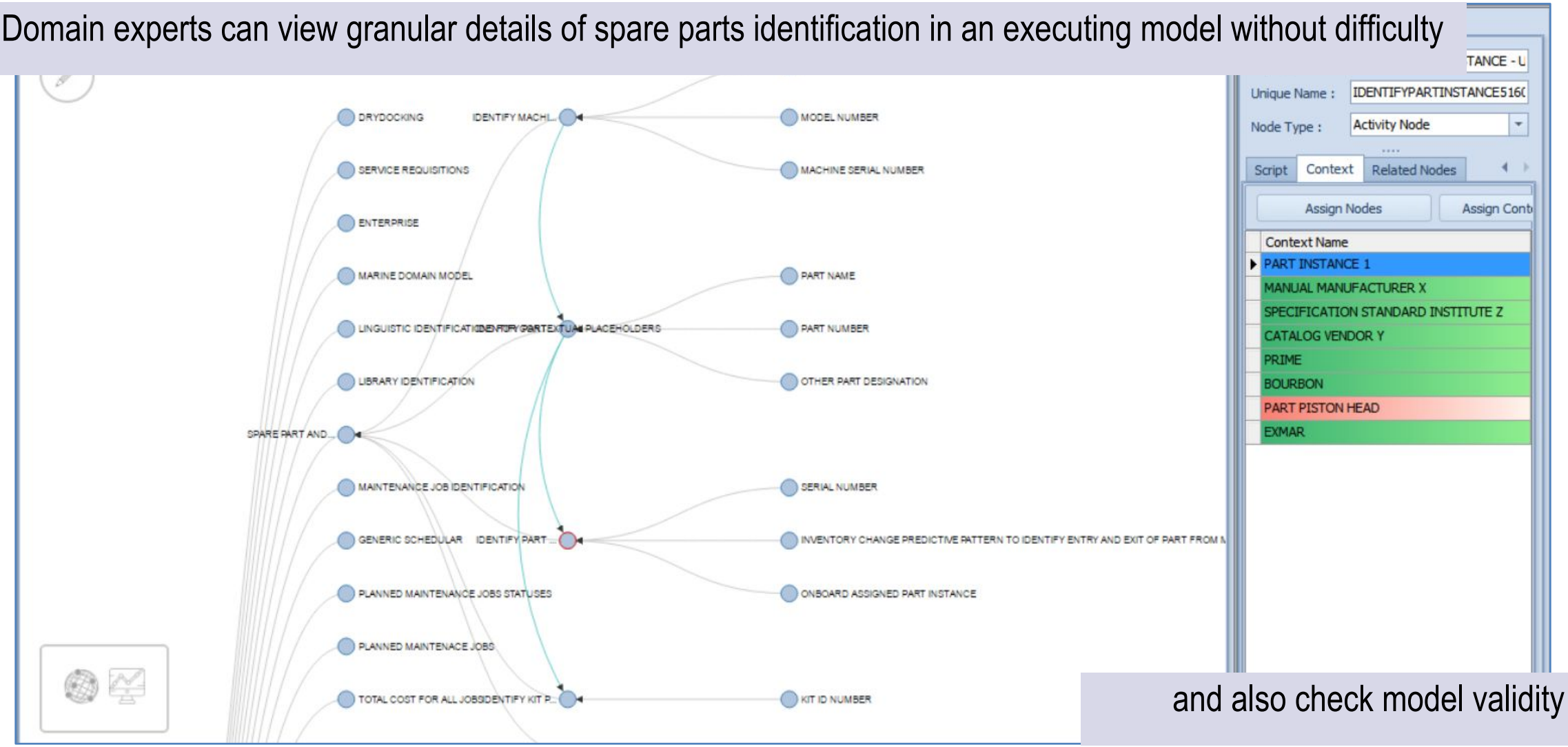
and comprehensible

by all stakeholders



Granular Detail

Domain experts can view granular details of spare parts identification in an executing model without difficulty



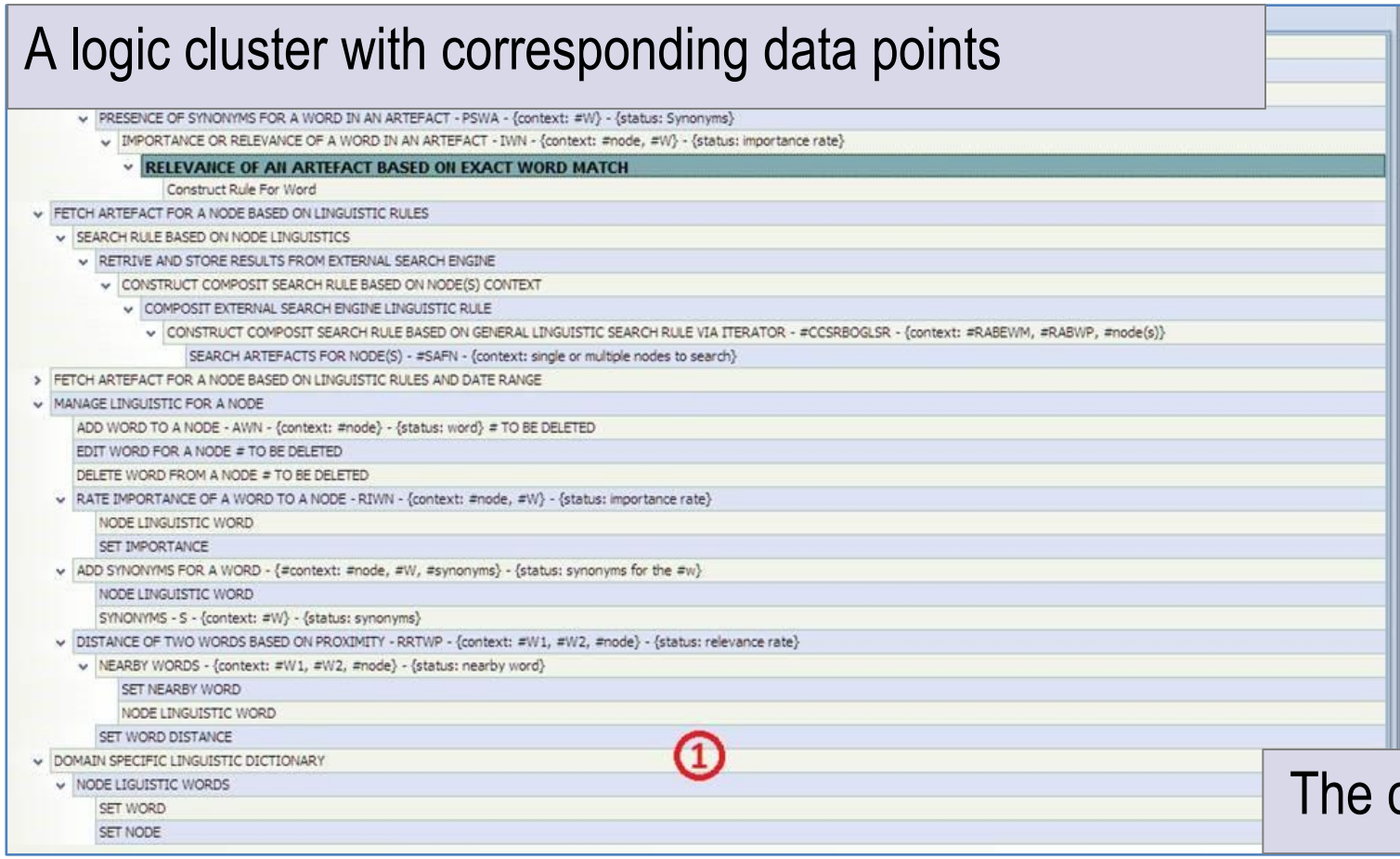
and also check model validity

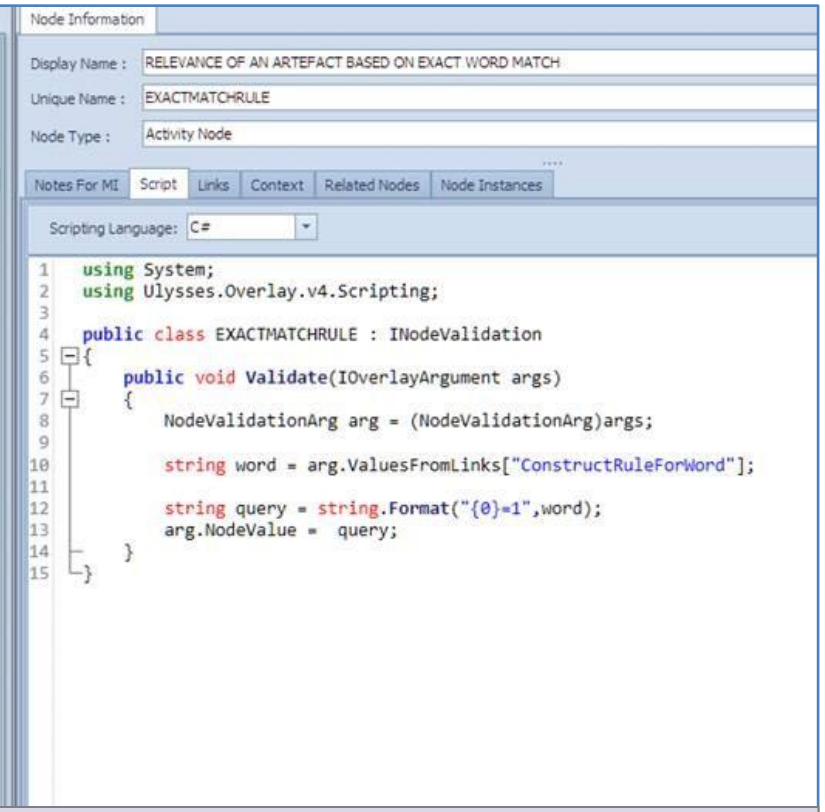
Let’s see how modular modelling can be:

We have seen that
models are graphical depictions of how processes
and the environment they take place in
are related to each other,
from a high level view point down to the finest transactional detail

Let’s see how modular modelling can be:

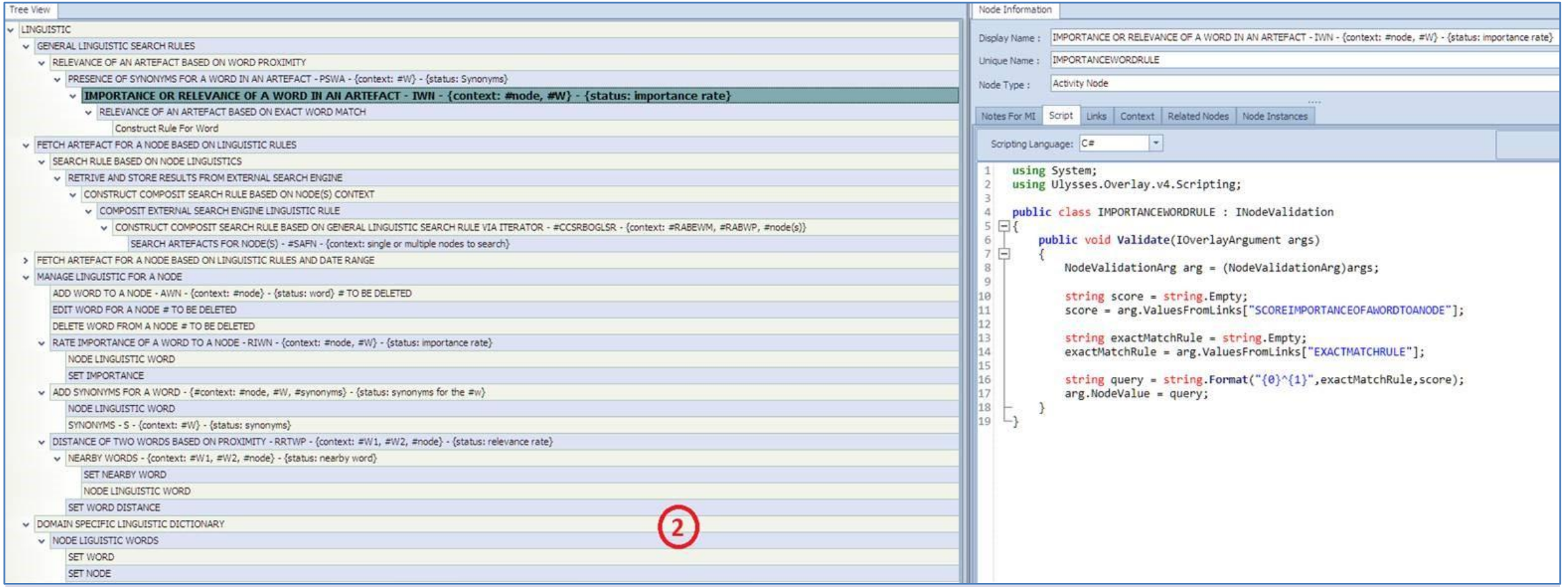
A logic cluster with corresponding data points





The code that converts the data values

Let’s see how modular modelling can be:



Most importantly executing models persist data and process together, this enables each increment of functionality to stand alone

Conclusions

Even after 40 years, enterprise software typically still covers

only a very small part of processes

which involve people

in the working enterprise,

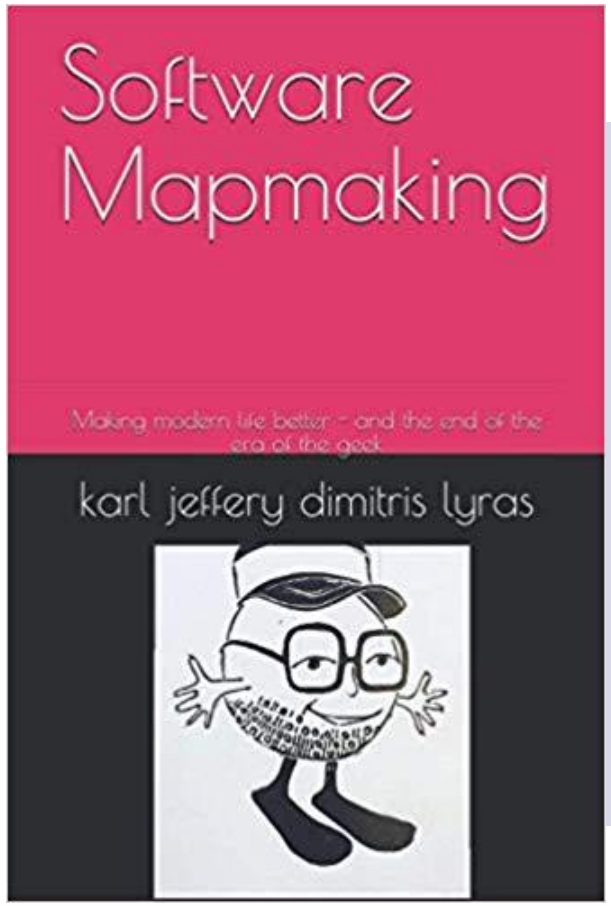
Conclusions

Software needs to develop fast

with minimal risk of instability.

To achieve this, software has to combine logic and data

Software Mapmaking



We think there is scope for the software
which supports and enriches modern life to be better

Our recipe,
which we call map making,
involves spending more time looking at software in an abstracted way;
as a plan, map, or model with less time spent looking at the code

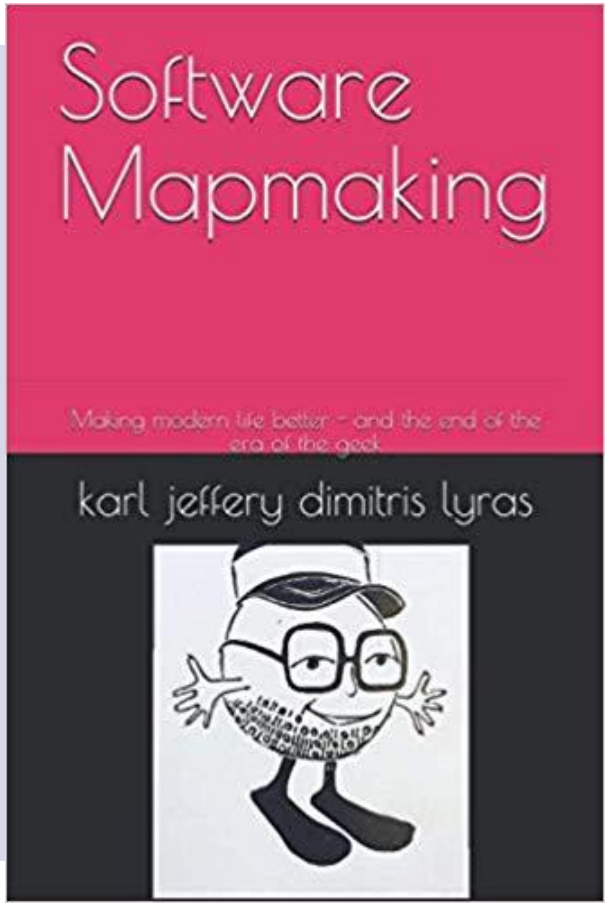
Software Mapmaking

In the same way that a builder follows a plan,
a musician follows sheet music
and an actor reads out the lines provided by somebody else
we suggest keeping this map in the forefront
of software development work
as a plan to be followed rather than a code

PDF: www.d-e-j.com/softwaremapmaking.pdf

Amazon Store: https://www.amazon.co.uk/s?k=software+mapmaking&ref=nb_sb_noss

Bit,ly www.Bit.ly/softwaremap



THANK YOU!