Industrial Architectural Assessment Using TARA

WICSA 2011
Boulder, Colorado, USA
June 2011

Eoin Woods
www.eoinwoods.info

Scenario Based Architectural Assessment

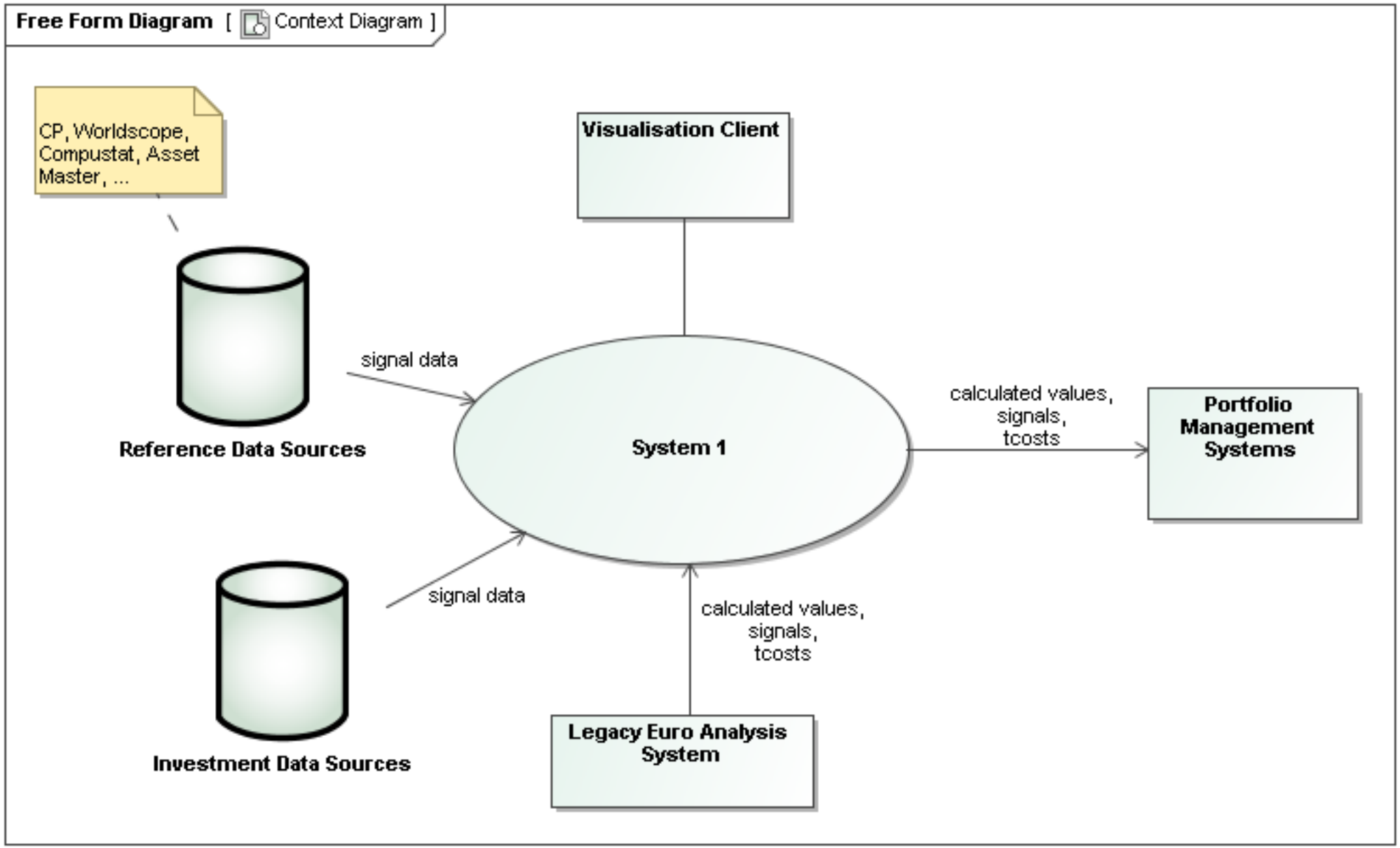
- Scenario based assessment has existed "forever"
  - SAAM was defined in 1994
- Lots of research defining and comparing methods
  - SAAM, ATAM, SAAMMCS, HoPLAA, ALMA, CPASA, ALPSM
- Relatively little use in industry
  - beyond large defence projects and some public sector ones
- Industrial perceptions
  - expensive, looks complicated, outputs & benefits unclear
  - needs a lot of input from busy (non-technical) stakeholders
- However industry still needs to review architectures
  - the norm is "*assessment by uninformed debate*"

How the Approach was Born

- An existing system was perceived as deficient
  - or at least its "quality" was unknown
- A request was made to assess its "quality"
  - really its suitability for strategic global deployment
- The whole process needed to be fairly quick
  - no expectation of wide stakeholder involvement
  - one architectural assessor working with the team
- ATAM clearly wasn't going to be accepted
  - no organisational history of architectural assessment
  - desire to keep it simple and "lightweight"
- So a simpler approach was required
  - that took implementation artefacts into account

The Tiny Architectural Review Approach

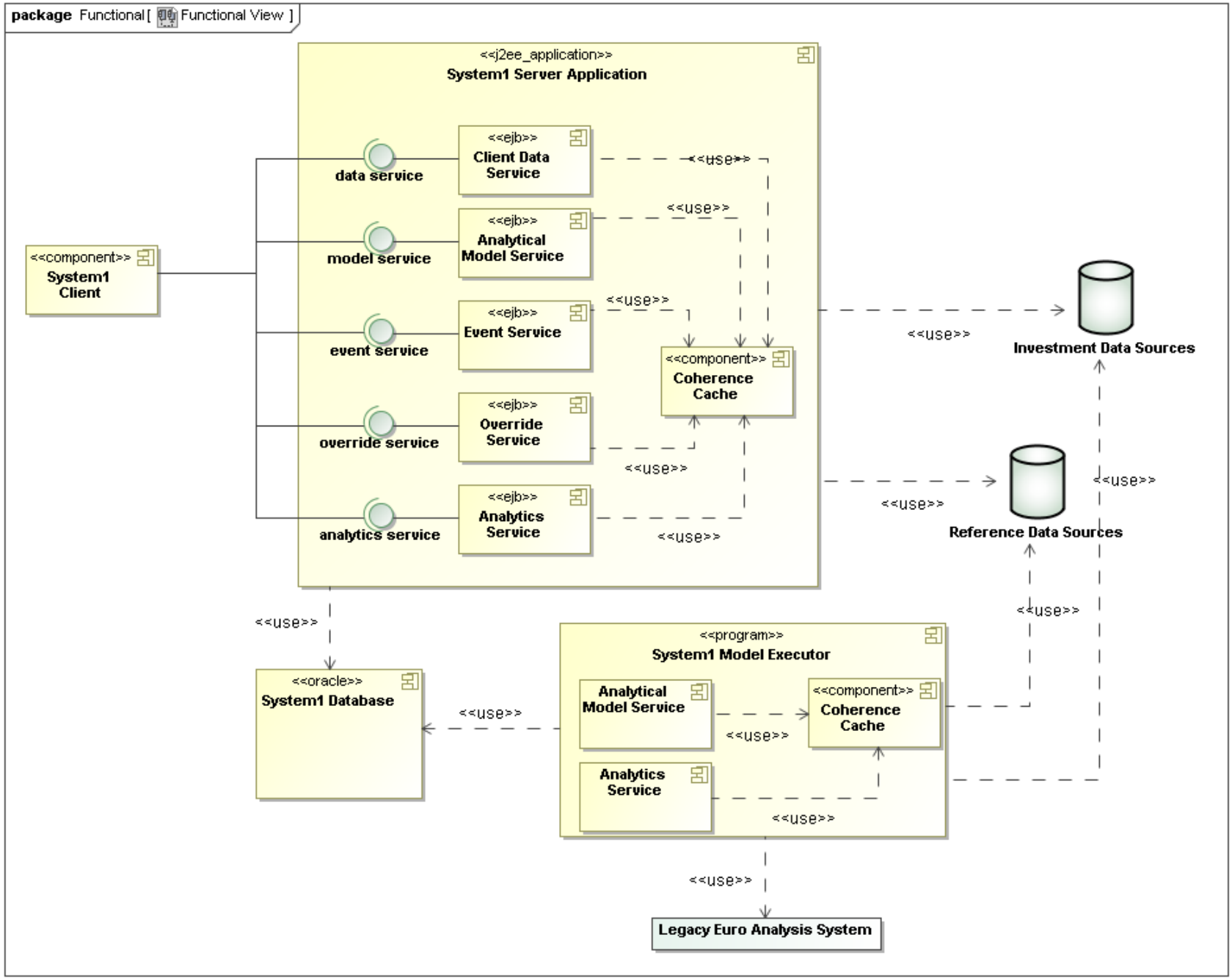
Context


Free Form Diagram [Context Diagram]
CP, Worldscope, Compustat, Asset Master, ...
Visualisation Client
Reference Data Sources
signal data
System 1
calculated values, signals, tcosts
Portfolio Management Systems
signal data
calculated values, signals, tcosts
Investment Data Sources
Legacy Euro Analysis System

Example Requirements

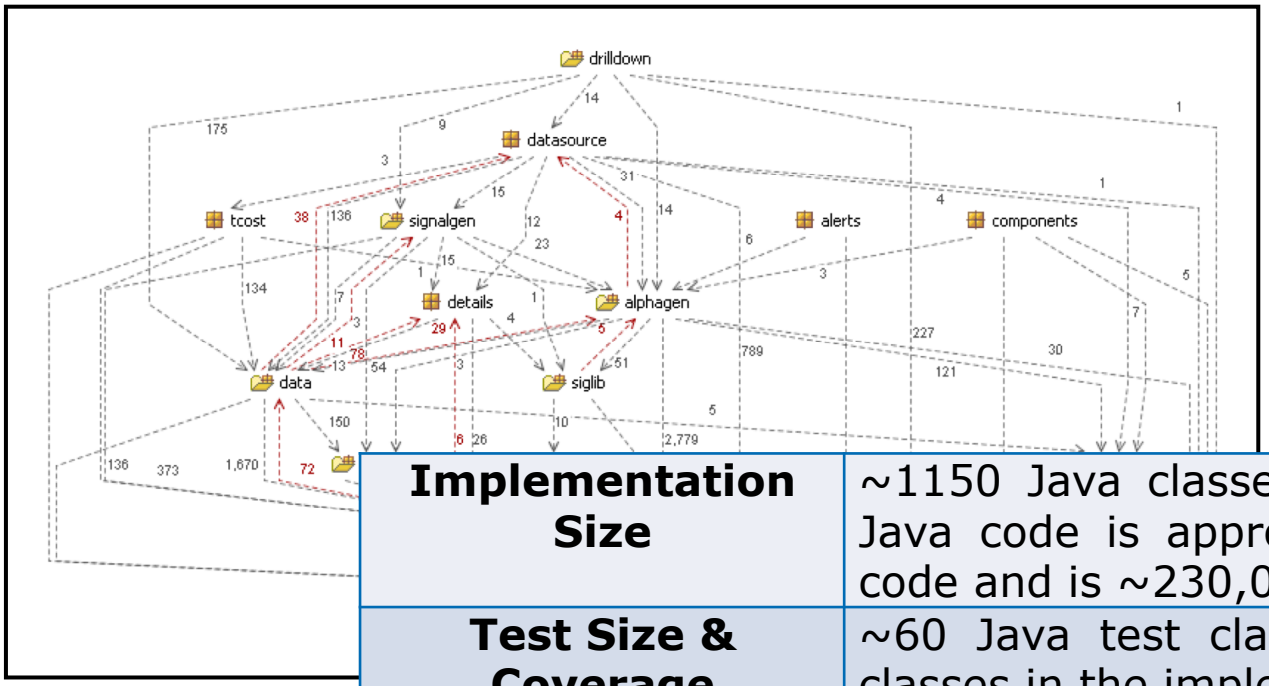
FR1	Quantitative Model Management and Execution – the core responsibility of the system is to allow quantitative model to be defined and executed when required. The model defines the input data, calculation status and output data that result in the generation of the quantity and cost values which are the system's main output.
FR2	Override Management – in many cases, users of the system will want to be able to override individual values or groups of values in the source data being used by the system. The system must provide the ability to create, remove and report on overrides and how they have affected the quantity value calculations.
NFR1	Performance – the key performance metric is the time taken to perform a model calculation run and generate results. Currently this is assessed to take in the order of 30 minutes in the system, but the target time for this is about 10 minutes. The other important performance requirement is the implicit requirement for the user interface to be usably fast (defined by the organisation to mean never freezing, responding instantaneously to local UI events and new data being available within 10 seconds of a request).
NFR2	Scalability – the key scalability requirement is likely to be maintaining the bound on the quantative model execution time as the size and sophistication of the model and the input data grow. This is likely to be a key challenge in the future. A related scalability requirement is the implicit requirement for the user interface to remain usable as the amount of data in the system and in each model run grows. Finally, the system's user base will never be very large but it will probably need to support 30 or 40 users per region in the long term.

Functional View



Nearly always add a deployment view too; sometimes others (e.g. data flow)

Static Analysis Measurement



Implementation Size	~1150 Java classes and 20 database tables. The Java code is approximately 111,300 (raw) lines of code and is ~230,000 Java byte code instructions.
Test Size & Coverage	~60 Java test classes which reference ~100 Java classes in the implementation.
Structure	Code organised into 10 modules and 8 layers, with about 15% of the leaf level packages considered to be "tangled" together.
Tangled Code	Engine - package com.abc.system.engine (46% of the code tangled); Server - package com.abc.system (42% of the code tangled) and package com.abc.system.service (31% of the code tangled); Base – package com.abc.system (32% of the code is tangled) and package com.abc. system.cuboid.dimension (30% of the code tangled).

Example Findings and Recommendations

Finding 1	Model Implementation - The quantitative model implementation is very nicely done and a significant innovation when compared to previous such systems. The fact that the model definition is now effectively data, rather than code, means that it can be evolved much more quickly than previous systems allowed and also (in principle) understood and modified by people outside the development team. It also opens up the possibility of implementing multiple execution engines for different scales and type of workload.
Finding 2	Internal Dependencies - The inter-module, inter-package and inter-class dependencies in the system could do with some review. In particular, the number of inter-module dependencies suggests that many sorts of change could be difficult in the future. Some of dependencies within the modules also appear to be very complicated and would benefit from a review by the development team to ensure that this level of inter-package and inter-class coupling is really required.
Recommendation 1	Operational Documentation - When installing and running the system, people in other regions will need simple, task oriented, installation and operational documentation to guide them. This could be as simple as a Wiki page of common procedures.
Recommendation 2	Simplicity Supporting Variation - There is going to be a need to support variation within the codeline (for example providing different override logic in one region compared to another). In order to minimise the complexity of achieving this, refactoring parts of the code to make the internal dependencies as simple as possible is likely to pay dividends later. Simplifying the dependencies will also help people to understand the code.

Assessing TARA Itself

- Strengths
  - Simplicity
  - Implementation structures are a key input
  - Brings Structure to an otherwise ad hoc situation
  - Speed and Efficiency
  - Simple, Concise, Usable Outputs
- Weaknesses
  - Subjectivity of the reviewer
    - still some ad hoc elements to the process
  - Lack of (mandatory) trade-off analysis
  - Primary input is implementation structures
    - rather than the design artefacts or process
  - Relatively shallow assessment

Comments and Questions?

Eoin Woods
contact@eoinwoods.info
www.eoinwoods.info