

## A CASE For Change

This White Paper draws on a number of themes to demonstrate the need for CASE tools that are flexible yet built on standards – ‘flexible’ in terms of the notations they accommodate, their openness to integration and configurability to evolving practices; ‘standard’ in terms of the use of UML and promotion of the Model Driven Architecture philosophy.

### Maturing Development Processes

A key driver for many organisations today is the recognition that software development is an industrial process that can be improved to offer strategic advantage. A standard framework for assessing the maturity of an organisation’s software development processes is the Capability Maturity Model (CMM). This model provides a series of attainable levels (1-5) that allow an organisation to measure their own processes against industrial benchmarks. Most organisations setting out on the track of CMM initially ‘procure’ an off-the-shelf process – something like the Rational Unified Process (RUP). This will often suffice in educating the organisation on the potential benefits of structured managed processes. But over the course of time an organisation’s processes will mature and diverge as strategic benefit is found in different working practices. This should be encouraged.

Deviations are needed to accommodate such issues as project size (large versus small), out-sourcing, procurement policies, legacy system integration, componentisation, safety-criticality, deployment environment etc. These deviations place point stresses on the development methodology used, for example stresses on requirements and/or design specifications, state modelling, business domain modelling, transformations. While these method variations are required, it is interesting to note that the same variability is not offered in the tools that support them. Rational Rose and similar products cling to the idea that UML – the Unified Modelling Language – by offering a rich vocabulary, will in some way address these method variations. The UML is a set of standardized graphical notations; it is not intended to be a definition of a development methodology – that is left to each organisation to develop.

So what is missing? Well the obvious answer is ‘method awareness’. Rose and other conventional CASE tools are not method-aware – they skirt over issues such as semantic integrity, traceability (beyond simple navigation links), consistency, correctness and completeness leaving these problems to developers and managers to solve. By taking this approach, Rose and its compatriots, degrade the role of CASE tools to that of graphical editors. It is for this reason that many software engineers have become disillusioned with CASE over the last few years and have resorted to tools like Visio on cost-benefit grounds and bypass formal design all together, vis-à-vis Extreme Programming (XP) / Agile Development.



At the other extreme are CASE tools that are method specific – examples include iUML, Software Through Pictures. These tools are hardwired for a particular software development method – e.g. Schlaer-Mellor, OMT. While offering increased method awareness these tools require you to adopt their method, warts and all. In some circumstances this may be fine. However there is generally little flexibility offered in terms of extending, enhancing or adapting such tools. The direction you must follow controlled by the vendor.

### Flexibility

For an organisation to be successful in today's highly competitive markets its IT function must be adaptable. Productivity, quality-of-service, flexibility are some of the hallmarks of today's successful companies.

Software engineering is a moving target – we daily see new initiatives that offer strategic benefits but seem to replace yesterday's new initiative. You only have to look at the succession of software platforms offered by Microsoft over the last 5 years to see the dynamics at play – VB, ADO, COM, DCOM, .NET.

So how is an organisation expected to track such rapid evolution with an eye on achieving competitive advantage? Well part of the answer may lie with the vision provided by the Object Management Group's Model Driven Architecture (MDA) initiative. MDA recognises that different problems need different approaches, specifically in relation to varied vertical-markets and delivery environments. MDA creates a clear separation between the solution domain (the platform independent model PIM) and the implementation environment (the platform specific model PSM). This model separation provides 'longevity' to the solution domain while recognising the dynamics characterising the implementation domain. This separation facilitates the possibility of increased use of tools by creating a standard framework for defining and transforming software models.

Utilising UML profiles and stereotypes, it is possible to define the notations and transformations needed to support the various MDA models - PIM and alternative target PSMs.

MDA therefore provides a practical platform for leveraging 'competitive advantage' through methods and tools. But where are the tools that support this initiative? The answer to this is – they are coming – there are number of early offerings on the market that address specific platforms such as J2EE and .NET. However there are few tools on the market that handle the whole lifecycle or can be tailored to an organisation's own methods.

But why would an organisation want to develop its own methods and tools? Well the answer to that is simple - again it is 'competitive advantage'. If an organisation can or has established effective software development practices, resulting in products being brought to market earlier and meeting their intended purpose, then the organisation will be



successful. Tools form an enabling technology – by automating the complex task of maintaining consistency, integrity, traceability etc. and enriching the decision-making process, tools increase ‘competitive advantage’.

### Textual Notations

Prior to the advent of UML, methods had drawn on both textual and graphical notations. Rational’s original view was that visualisation was the key to successfully modelling software systems – this philosophy, through the widespread adoption of UML has implicitly become the cornerstone of most tools that support UML. The same however cannot be said for methods – methods continue to make use of both textual and graphical notations. While visualisation is a valuable tool for modelling certain aspects of software engineering such as class diagrams and messaging, it is an impractical vehicle for modelling aspects such as Use Cases, business rules, non-functional requirements and formal specifications.

Furthermore some UML artefacts are in fact placeholders for textual artefacts e.g. Use Cases. But where is the CASE tool support for these textual notations? One approach adopted by a number of vendors is to provide hyper-links to external documents. While providing some benefit, in general documents are at the wrong level of granularity. If a designer references a class by name in a design document, they want to know that it exists in the project dictionary (embedded in their CASE tool). If the class is deleted or renamed in the project dictionary, then they want this to be reflected in their document in real-time. Likewise they want to know that they are working with the appropriate version of the artefact.

Perhaps this is asking a lot of tools integration, but the ability to write structured text that is kept semantically consistent with the project dictionary and diagrams is feasible.

### Life-cycle Coverage & Integration

Which parts of the life-cycle can be automated? The historical answer to this question has been governed by the choice of CASE tools and the ability to integrate them. CASE-tool vendors have progressively tackled only specific parts of the life-cycle e.g. analysis, design or code generation with little concern for how their tool integrates with other tools covering some other part of the lifecycle.

Recognising these issues, the OMG has defined XMI, a data-exchange protocol for tools. This offers a vehicle for information inter-change among tools – its goal being that XMI should form the basis for ‘live’ tools integration. However most vendors implementing this standard do so only to provide a one-off import capability – a means of getting data out of a competitor’s product into their own. This approach does not lend itself to the notion of a single ‘live’ project-wide repository.



Consequently it is virtually impossible to ask 'what-if' type questions or seamlessly follow tracability links across tool boundaries.

In a similar vein, what options do you have if you need to integrate with tools that do not support XMI? Well you could approach the vendor for a bespoke enhancement or utilise some intermediary product to provide bridging but these solutions are not normally practical or cost-effective.

### **MaxUML – the CASE tool for Change**

MaxUML heralds a new generation of CASE tools aimed at maximising the benefits afforded by automation and standards. Its back-bone is UML – currently providing full UML 1.4 coverage, and when ratified, the UML 2.0 standard. Going beyond UML, MaxUML also supports Snapshot diagrams and Film Strips – used to model the progress and ultimate effect of methods and other behavioural elements on a collection of objects. These techniques are used in a number of leading methods including Catalysis.

But MaxUML's true potential is realised when developing Profiles and Stereotypes to support bespoke methods. Taking a standard UML construct as a base element, it is possible to define a new Stereotype - controlling its constituents, its appearance, its usage and the semantic rules governing its use. Using tagged values one is able to extend the stereotype to include new fields that may be either textual, literals or links to other elements in the repository. These features combine to allow you to configure rich profiles comprising well-behaved stereotypes that can be tailored to a vertical-market, different target platform or life-cycle phase. Furthermore stereotyping is not limited to UML meta-types – it is also possible to define new diagram types e.g. Process Flows, Domain Models.

MaxUML also incorporates a structured text editor. Using this feature you can develop structured text documents that can embed references to other model elements such as classes, attributes, Use Cases etc. and elements that are 'in-scope' e.g. objects, object valued attributes and tagged values. These references are maintained consistent across renames and deletes. These features allow you to define structured documents such as formal design specifications, Use Case descriptions and specifications that have live content.

Another key feature is the ability to associate a 'language' against a text field. One standard supported language is the OMG's Object Constraints Language (OCL). MaxUML utilises OCL to support design-time constraints and the definition of the semantic rules governing stereotypes. But OCL is only one possible language – you may wish to incorporate support for other languages such as lifecycle expressions.

MaxUML is built on a meta-CASE technology that has a strong pedigree – used to build tools like HOOD, SSADM and IDEF0, and as a result MaxUML benefits from this highly flexible development technology. This allows SoftCASE to quickly track emerging standards like UML and MDA



while also offering specific enhancements required by its clients such as tool integration, new modelling notations, specific code generation.

MaxUML is not available as an off-the-shelf product – SoftCASE's strategy is to develop relationships with its clients aimed at maximising the benefits afforded by MaxUML's flexibility. Providing tailoring services and method consultancy, SoftCASE delivers a 'whole product' solution to organisations wishing to pursue initiatives such as MDA or leverage home-grown development methods.

For information please contact us at:

SoftCASE Consulting  
Rowley House, The Quay, Poole, BH15 1HL  
Tel. 01202-749643

