

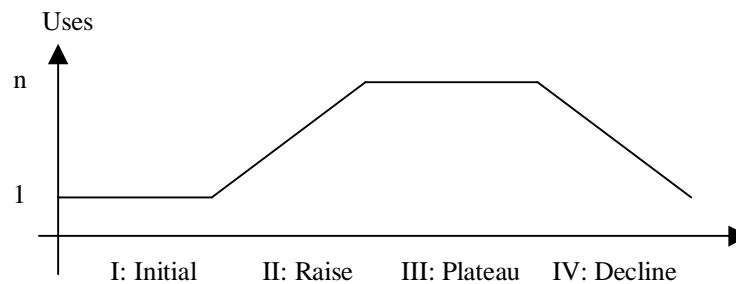
Quicksilver 0.6 Whitepaper

© Jan Burse, 18. April 2000

The Component Based Approach

Agent-based computer models and simulations form a special area of the applications of computer systems. Quicksilver is an attempt of an investigation into this area of applications in the light of modern component based software development. Software development has many years suffered from the lack of a component based engineering approach. This approach is available in many other engineering disciplines. Only recently has the component based engineering approach become well accepted in software development through object oriented technology.

The basic idea behind the component based approach to application development is to view an application in an organization as a collection of software components and as well to view software components as used by many applications in an organization. If software components are used by many applications then something like a lifecycle of a software component can be observed. In the beginning a software component is used by an initial application. The number of applications might then rise, arrive at a plateau and later decline:



Figur 1: Lifecycle of a Software Component

The cost of developing an application consists of the cost of developing the software components and of the cost of wiring the software components. Assume that the overall cost of the applications in an organization is x , that a software component costs c and that this software component has been independently developed by $n \geq 0$ applications in the organization. If this software component were first developed and then shared by the applications, then the overall cost changes to $x' = x + c - n \cdot c$. We can distinguish the following cases:

$n=0$) The software component is not used: The cost would raise $x' > x$

$n=1$) The software component is used: The cost would remain $x' = x$

$n > 1$) The software component is used more than once: The cost would lower $x' < x$

Figur 2: Benefit of first developing and then sharing a software component

The first case is not an unusual case for an organization. For example a software component could have been developed for an application that turned out not to deliver the expect business benefit. Or a software component itself can turn out not to fulfil the functional or the non-functional requirements of an application. Thus a new software component can be a risky investment. Only

after a software component has been successfully applied and is applied again, an additional payback results when the software component is shared. Business requirements can change and thus software component can get less used. Investment in new software components is then necessary to possibly accomplish a turn-around.

To get the additional payback similar software components across applications have to be identified. This effort can result in additional costs that might in turn annihilate the additional payback. These costs can be reduced to some degree by repository techniques. Also the sharing itself might include additional costs. For example if software components evolve updates have to distributed and new requirements have to be sorted out. Finally if I can reduce the cost of wiring components, I can also reduce the overall costs.

Contribution to Agent-Based Simulation

Quicksilver provides shareable software components and custom wiring in the area of agent-based computer models and simulations. It does so in complementing the Java platform. The modeler thus has to access the Java platform and Quicksilver for his modeling business. The decision in Quicksilver to cooperate with the Java platform is motivated by the divide of software components into classes and instances. Classes and instances are assumed to have different characteristics in two respects. First of all it is assumed that classes are not great in number in an application whereas instances can be great in number. Second it is assumed that classes can contain program code and data values whereas instances can only contain data values.

Instances and classes are instrumented in the following way for agent-based computer models and simulation. Instances can be identified with empirical data whereas classes can be identified with expert knowledge. The explanation is that we view expert knowledge as derived from empirical data by generalization. In agent-based simulations these generalization are used to derive the behaviour of the agents. Expert knowledge has thus to be available as program code. This is a characteristics of the simple application area of computer models and simulation. Other application areas, as for example medical expert systems, could require the fusion and generation of expert knowledge that is not program code.

	Characteristics	Instrumented	Java Platform	Quicksilver
Classes	Small number Program code	Expert Knowledge	Basic Library Class Wiring	Weak Agents Strong Agents
Instances	Great number No Program code	Empirical Data		Data Analysis Instance Wiring

Figure 3: The division of labor between the Java platform and Quicksilver

Figure 3 positions the Java Platform and Quicksilver with respect to the divide between classes and instances. The divide should be taken with a grain of salt. For example there is expert knowledge that can also be covered by instances. The Java Platform delivers with the "Basic Library" classes with basic mathematical functions, file system access, network access and graphic device access. The Java Platform further delivers with "Class Wiring" a compiler where one can generate Java classes and a virtual machine where one can execute Java classes.

Figure 3 also shows the added value that Quicksilver gives to the Java platform with respect to agent-based computer models and simulation. The added value is mainly provided in the form of shareable software components. Namely "Weak Agents" and "Strong Agents" are in fact software components that a modeler can use to derive its own model agents from. The software components

of "Weak Agents" are for the notion of a weak agent [Woolridge], i.e. an active object, whereas with the software components of "Strong Agents" we are heading towards more cognitive agents. Further "Data Analysis" stands for software components that can be added to a model to accommodate for the visualization and analysis of a simulation. Only "Instance Wiring" is rather a tool than a shareable software component. "Instance Wiring" allows for the interactive creation, linkage and stepping of model and analysis agents.

Weak Agents: This framework provides the classes "Agent" and "Swarm". The modeler can derive an active object in his model from the class "Agent". The current scheduling is such that an active object is called every simulation step. An active object is free to change the linkage of objects and/or to create objects, the simulation can thus use variable structures. The linkages between active objects may form any network and must not form a tree. Additionally by means of the class "Swarm" active objects can be collected in hierarchical activity clusters.

Strong Agents: This framework provides the classes "Base" and "BeliefNetworks". Standard Java program code is good for numeric and collection processing. In symbolic processing the Java programming language lacks some functionality that is found in artificial intelligence programming languages. For this purpose the class "Base" provides Prolog knowledge bases such that one can handily use pattern matching and backtracking. Further there is the class "BeliefNetworks" in development that will extend the Prolog knowledge bases and provide Bayes inferencing, decision optimization and learning.

Data Analysis: This framework provides the classes "Viewer" and "MonteCarlo". The modeler can derive custom views for his model agent from the class "Viewer". There are various visual components in stock such as graphs, nets, etc.. that can be used to build these viewers. Viewers can be used to only browse model agents or even to edit model agents. The "MonteCarlo" class is used to insert an analysis agent into a simulation that will perform a monte carlo analysis of the model. The analysis will search the specified parameters and determine correlations.

Instance Wiring: This tool supports the editing and browsing of simulations. For this purpose the modeler can interactively insert a new agent in his model and set the linkage to other agents. He can save the resulting model in so called model files and later load the model again. He can also use cut & paste to move around sets of agents, even between different model files. Finally the tool provides the interactive stepping of a model, and utilization of the default and custom viewers of the model agents.

The resulting agent-based computer models and simulations allow for further sharing of software components. Model agents can be shared over several model files as long as in the environments the same agent classes and thus expert knowledge is shared. Model files can also be delivered as applets, the expert knowledge is then automatically loaded from the applet server. Besides that no other distribution mechanisms for agent classes and thus expert knowledge are provided. The model agents can be cataloged by means of the javadoc facility of the Java Platform. Besides that no other repository techniques are provided.

Further Developments

We have seen how Quicksilver is based on the component approach. Quicksilver delivers software components for weak agents, strong agents and data analysis. Quicksilver also delivers a tool for the building and exploration of models and simulations from these software components. It does so by a division of labor with the Java platform. Further the component approach also holds for the resulting

models and simulations, in that the same model agent can be used in different model files. The weak agents and part of the data analysis has been applied by [Tillman]. The weak agents and the data analysis are currently applied by [Hare] and [Kottonau]. The strong agents are still in development [Burse].

Quicksilver does not support the identification of similar model agents or the engineering of requirements. There are UML based tools which could fill this gap and which would cooperate with the Java platform and with Quicksilver. Some explorations in the application of UML were already pursued by [Tillman]. We plan to start new explorations soon, namely when [Kottonau] needs documentation. The idea is to install the Together/J product and to practice UML and to eventually work on a Quicksilver profile for UML.

Although the applet mechanism works for Quicksilver model files, Quicksilver models and simulations have not yet been used in participatory settings. We have a total lack of any experience with agent based models in participatory settings. What model agents are used in participatory settings? Are belief networks or monte carlo methods useful? What viewers could be used in participatory settings? What changes to the tool are needed? Hopefully there will be some projects concerning participatory settings.

Finally Quicksilver should keep up with the newest technological developments. One important area is XML. The XML approach could be used for the model files. This could open up a greater interoperability what concerns both the empirical data and the expert knowledge. There are already XML definitions around for belief networks. XML model file readers and writes have to be provided. Further with the release of JDK 1.3 some software innovations have become available. These concern the possibility to wire classes by a tool and thus to acquire and present expert knowledge. This could be also useful also in a participatory setting, to make models more self explanatory in that the models become automatically white boxes.

References

Woolridge
Tillman
Kottonau
Hare
Burse