Coordination Architecture for Evolvable Event-Based Systems

L. Andrade\textsuperscript{1} and J.L. Fiadeiro\textsuperscript{1,2}
landrade@atxsoftware.com, jose@fiadeiro.org
\textsuperscript{1}ATX Software SA, Alameda António Sérgio 7, 1C, 2795-023 Linda-a-Velha, PORTUGAL
\textsuperscript{2}Dep. Informatics, Fac. Sciences, Univ. Lisbon, Campo Grande, 1749-016 Lisboa, PORTUGAL

It is clear that event notification services, event-based and publish/subscribe systems provide levels of flexibility and agility that bring us much closer than, say, object-oriented techniques, to what is required for operating in business environments that are “time critical”, namely those that make use of Web Services, B2B, P2P or operate in what is known as “internet-time”. Indeed, because customership, the basic mechanism of object-oriented computation, relies on the application of specific features of specific classes, determined at design time, to specific instances of these classes, interactions in object-oriented systems are too tightly coupled and rigid to be able to respond to changes occurring in the business domain while ensuring critical business requirements such as time-to-market and responsiveness to immediate user-driven reconfigurations of service (B2C) or the procurement of services from which global properties of the system can emerge (B2B). Event-based interconnection circumvents the rigidity of explicit invocation, allowing for interaction to be separated from computation and managed independently.

Although event services are now available in major middleware platforms, directly or indirectly via design patterns, their use at the more abstract levels of abstraction in which system configurations can be established directly in terms of business concepts is not yet supported in a satisfactory way. Semantic primitives are required that allow for business architectures to be modelled explicitly so that the levels of agility and flexibility provided by event services can be effectively used. Indeed, for achieving the higher-level business properties that we mentioned in the previous paragraph (e.g. time-to-market and responsiveness in B2C or B2B), it is essential that the evolution of systems be driven in terms of the business concepts that underlie the requirements for change or determine the dynamics of service procurement, and not the lower level design mechanisms available in the middleware and in terms of which the system is designed.

Our purpose in this abstract is to present an overview of a set of modelling techniques that address these issues. Our proposal is based on a layered abstract architecture that enforces a strict separation between two levels of modelling, the rationale for which is the realization that, when addressing the dynamics of system evolution, it is possible to distinguish two different rates of change and address them separately.

On the one hand, we can identify core business entities that are relatively stable in the sense that they provide very basic services whose functionalities tend to remain invariant even if their implementations may change, e.g. as a result of new technologies being introduced. For instance, in a typical financial system, a bank account can be seen to offer three basic services — the current balance and operations for debits and withdrawals. The way these operations affect the current balance are business invariants whose implementation can vary, say depending on the way the current balance is represented and the language that is being used for programming the operations. In terms of the architectural approach that we are proposing, core business entities like bank accounts are represented as components that reside in what we call the computational layer.

The contexts and conditions under which these services can be used at any given moment are, however, much more volatile. They are determined by the business rules and policies that the bank has in place as part of its current business strategy, the legislation that currently applies, and other factors that are bound to change during the life-time of the system. For instance, a regular customer will only be allowed to withdraw amounts that are smaller than the current balance but, after a while, “loyal” clients may be “upgraded” and given a credit limit that they can use for overdrawing their accounts. New ways of optimising interest-earning are constantly being offered as a way of attracting new customers or keeping current ones loyal. New legislation may be introduced that requires services to be adapted, e.g. accounts may need to be monitored for big-deposits as part of a new scheme aimed at detecting money-laundering.

In order to allow for such changes to be reflected in the system without intruding in the way the basic components are implemented, we provide a second architectural layer in which such volatile business rules are represented as connectors that can be superposed, dynamically, over the components in the computational layer so that the behaviour that is required of the system at any given moment can emerge from the computations performed lo-
ally by the components and the interconnections that are established between them through the connectors. A strict layering is enforced in the sense that interactions between components in the computational layer cannot be established directly between them but have to be coordinated through connectors sitting in what we call the “coordination layer”.

This is where event-based systems represent a real advantage over object-based ones: they allow for interactions to be externalised whereas interactions between objects become coded in the way features are (directly) called, making any change to be intrusive on the implementation of the objects involved. This does not mean that coordination mechanisms cannot be put in place over object-based middleware. In fact, we have proposed in [5] a micro-architecture that makes this externalisation possible, but at the cost of introducing an additional layer of adaptation that, basically, enforces an event-based approach. In our approach, we provide a set of semantic primitives that allow for such connectors to be modelled and superposed directly over components assuming an event-based component middleware.

Our overall approach capitalises on research in the areas of Coordination Languages and Models [4], Parallel Program Design [6], Software Architectures [1] and Reconfigurable Distributed Systems [7]. It was initially proposed as an extension to the UML [2] and is also supported by a full mathematical semantics [3].


Visit www.atxsoftware.com for more information and a demonstration of these technologies.