BPM and SOA – an integrated modelling approach

Introduction

Services and Processes are new architectural layers raising the level of abstraction required to develop enterprise applications in a SOA environment.

Several vendors offer tools and technologies that support the implementation of these new concepts. However efforts to integrate these concepts in the application development process and related model-driven approaches, are lagging behind the technology, as evidenced by the fact that the most recent version (2.1) of the Unified Modelling Language [1] lacks these concepts.

This paper shows how Services and Processes can be integrated in the RDF Development Process and associated model architecture, with little disruption to their fundamental principles and structure. This is made possible by the specification-oriented modelling approach used at RDF, in which system and component functionality are specified early in the development process, using the 'design-by-contract' approach.

The RDF process

The RDF software development process has been used for several years in projects of medium to large size (i.e. approximately from one thousand to ten thousand person days). It is heavily influenced by the Unified Process [2], but it adopts a more rigorous, specification-based, modelling approach influenced by robust object-oriented methodologies such as Catalysis [3]. The fundamental principles of the process are:

- It is use-case driven. Use cases capture requirements from the user's standpoint and drive all activities: analysis, design, development, testing, project planning and progress tracking. Progress is measured by the delivery of visible functionality of value to the business rather than by technical metrics such as lines of code completed.

- It is iterative and incremental. Functionality is built and delivered in increments using time-boxed iterations. Advantages of this approach are: risk reduction, early visibility and higher quality achieved through continuous testing.

- It is architecture centric. Analysis is object-oriented, while architecture and high-level design are component-based. Object-oriented design does not scale up well to application level, so it is used only inside components ('design in the small'). Component interfaces play a central role in the architecture. This means the level of reuse and encapsulation is raised from objects to components, creating stable and coarse-grained abstractions, similar in nature to services, as explained in more detail later.

- It is model driven. UML models drive progress and are used to specify and design business processes and software artefacts. RDF models have a standard structure and a strong set of best practice guidelines. The picture below illustrates the modelling layers and key models:

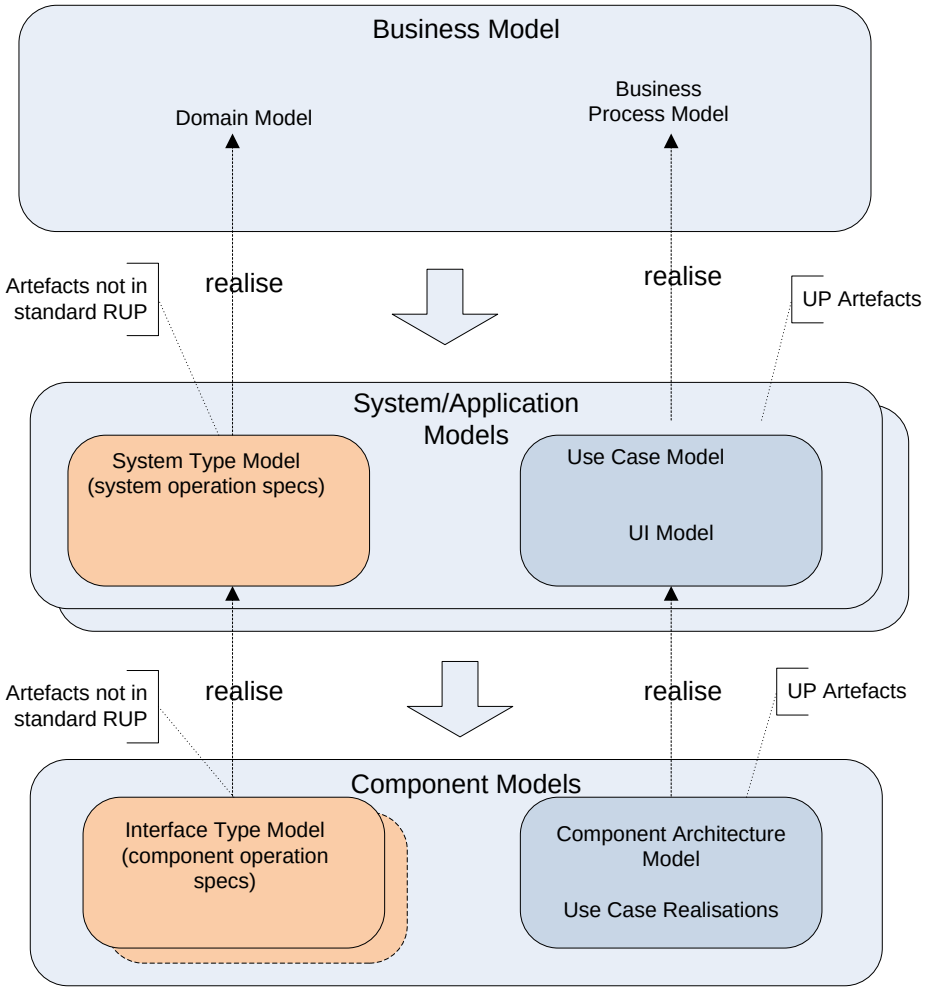

Figure 1: RDF's platform independent model structure

System and component interface specifications

In the RDF process, system functionality is specified rigorously by modelling the system under construction as a typed object providing one or more interfaces, with associated operations. External systems are modelled as required interfaces.

Operations are specified in a declarative style, using pre- and post-conditions, based on an underlying System Type Model, which provides the functional semantics of the operations.

The same approach is used to specify functionality offered by the main application layer components in the system design. A component Interface Type Model provides the semantics of component operations [4], just as the System Type Model provides the semantics of system operations.

As noted in the diagram, the System Type Model and Interface Type Models are not standard UP artefacts. Their use makes the RDF modelling approach easily portable to a SOA environment, as explained in the following section.

Modelling services

"Services are published capabilities that can be dynamically discovered and composed" [Kroll and MacIsaac 2006]. In order to utilise a service, a client needs to obtain its interface and invoke one of the operations published in that interface.

Services are similar to components. In UML2 a component is "a modular part of a system design that hides its implementation behind a set of external interfaces"[1].

The main difference between a service and a component is that component interfaces cannot be discovered dynamically. In all other respects, components and services are similar in nature. They both offer functionality that can be invoked through interfaces, and both need to have specifications associated with those interfaces. It follows that the modelling techniques used to model components can be reused, bar the discovery side, to model services.

The diagram below shows how the RDF models can be adapted to the modelling needs of SOA architectures with little disruption:

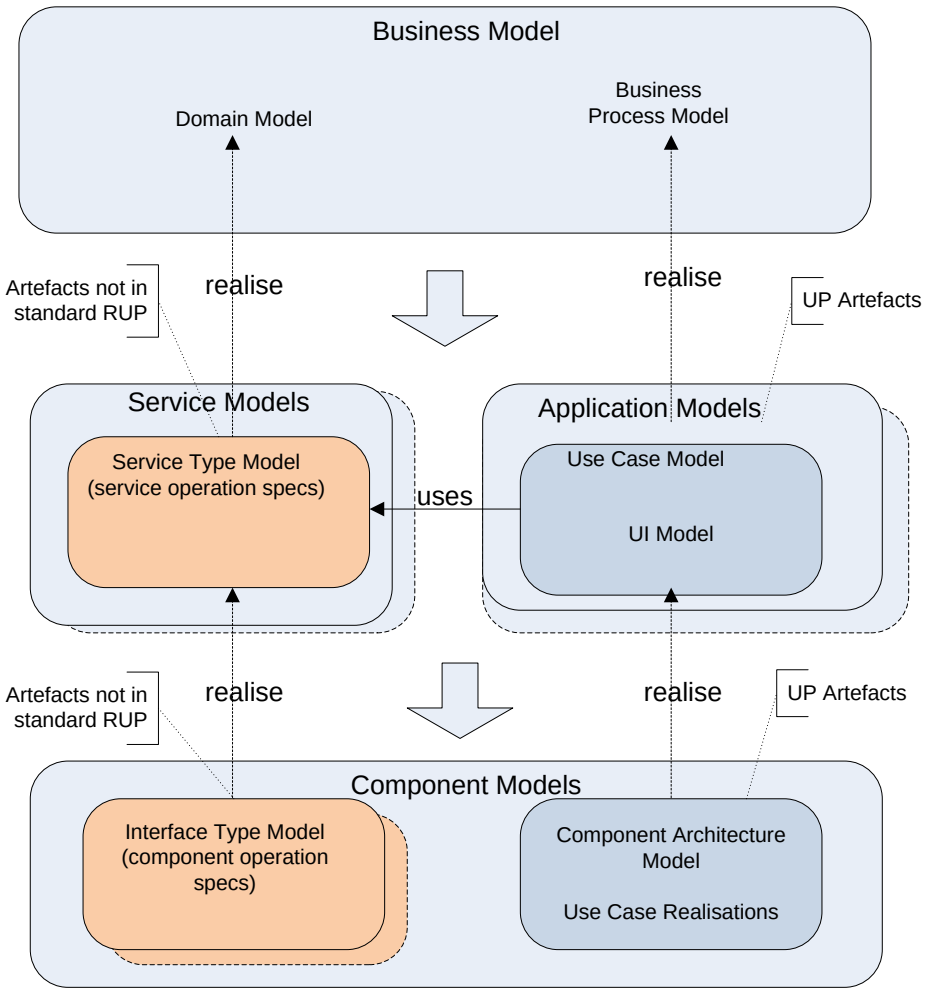

Figure 2: Replacing system with service models

Service definition

Services are specified using the same technique currently used to model systems and components. Each service is modelled as an interface, with a number of provided operations, and the semantics of the operations are defined using pre- and post-conditions, based on an underlying Service Type Model.

Applications continue to exist in a SOA environment, so use cases and user interfaces will continue to be applicable modelling techiques. However, system functionality is captured as Services rather than being provided by a 'system' abstraction.

Service identification

Currently there is no explicit step or method in existing methodologies to identify and define services, as acknowledged in [6]. In RDF's experience, services need to become visible artefacts as early as possible in the development process, in the analysis stage, when system functionality is discovered, analysed and specified, rather being just design artefacts. In other words, we see services playing a role, in a SOA project, similar to the role played by the System type in our pre-SOA projects. This has the advantage of making services visible to the user and testing community, rather than being seen as an internal implementation mechanism, increasing the opportunity for reuse in different business processes/applications.

As we believe that visible functionality, modelled as use cases, should continue to drive the development process even in a SOA enviromment, we have added a Service Identification step in our use case modelling technique. Services are identified and their purpose and scope captured at the same time as use case flows are described.

Modelling processes

We model business processes, both in their 'as-is' and 'to-be' forms, using UML 2.0 Activity Diagrams [1].

Traditionally applications automate part(s) of business processes, but the processes themselves are not directly automated, i.e. they do not appear as an implementation artefact in the enterprise architecture.

In SOA, business processes can be automated in the Service Orchestration layer of the Enterprise Service Bus (ESB) [7], and become both consumers of services as well as suppliers of services. For example, to initiate or resume a process, a client of the ESB will invoke a service which will be routed to and consumed by the correct process. The process can reply to the client (request-response invocation) or not (one way invocation). Requests originating from UI clients will normally be of the request-response type.

Processes can invoke services during their execution. They can also generate events and can wait for events to occur. For example, generating an external request and waiting for a reply can be modelled with generating an event and receiving an event. A process can generate a workflow task for a human agent - this can be achieved by invoking services provided by the underlying Workflow platform. The process may or may not have to wait for the agent to take action before proceeding.

This means that business process models have to be formalised and interfaced with the Service Model, and used to generate executable process descriptions (e.g. BPEL descriptions) rather than used as a mere documentation item.

Putting it all together – business processes, use cases and services

A use case represents a slice of a business process seen from the perspective of an application user. The use case models the user-system dialogue. Whereas use cases are normally of short duration, business processes are long-lived and can interface to a number of use cases during their execution. For example, in the business process 'Process Mortgage Application', there will be a number of use cases for different users, i.e. the customer, the underwriter, the bank processing agent, etc. Each of these actors will perform a number of use cases, possibly using different applications, to achieve their individual goals, which represent parts of the overall business process.

The diagram on the next page shows a typical interaction scenario between use cases (only partially shown via their primary actors for simplicity), processes, and services. Use cases invoke the Mortgage Application Service, which provides entry points into the Mortgage Application Process. The process invokes other services that encapsulate business logic. Note how the Workflow Service allows the process to create tasks for human agents and suspend itself waiting for the appropriate event.

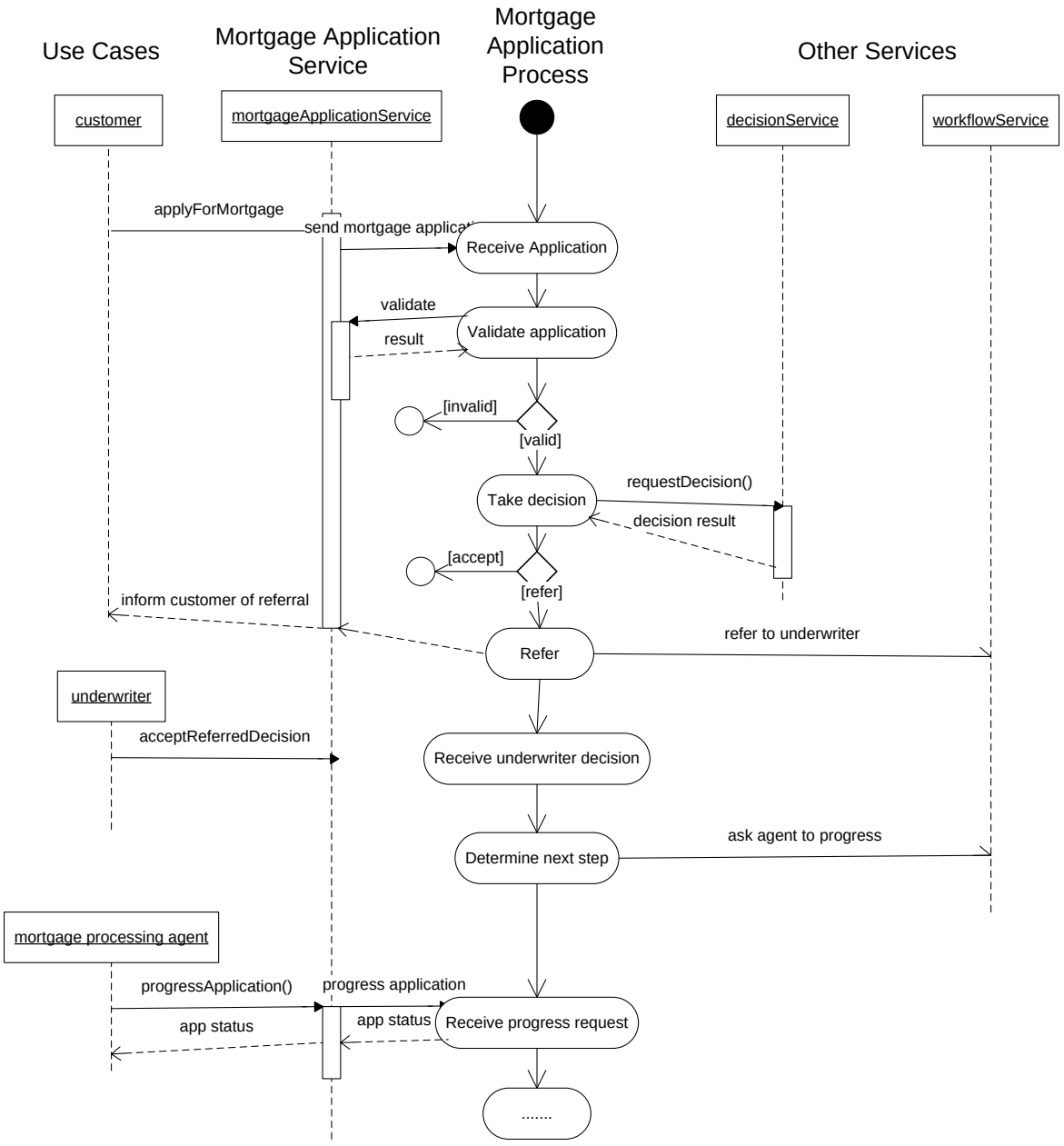

Figure 3: Typical Interaction pattern between use cases (actors), services and processes

Conclusions

As the level of abstraction in enterprise application development is raised to include services and processes, it is important that these concepts are explicitly modelled in a coherent and integrated manner. RDF's strong model-driven development process can readily incorporate these new concepts with relatively small changes to the structure of the models and the steps involved in system analysis and design.

References

[1] Grady Booch, Ivar Jacobson and James Rumbaugh, *The Unified Modeling Language Reference Manual*, 2nd Edition, Addison-Wesley 2005

[2] Ivar Jacobson, Grady Booch and James Rumbaugh, *The Unified Software Development Process*, Addison-Wesley 1999

[3] Desmond D'Souza and Alan Wills, *Objects, Components and Frameworks with UML: the Catalysis Approach*, Addison-Wesley, 1999

[4] John Cheesman and John Daniels, *UML Components: A Simple Process for Specifying Component-based Software (Component-based Development),* Addison-Wesley, 2000

[5] Per Kroll and Bruce MacIsaac, Agility and Discipline Made Easy – Practices from OpenUP and RUP, Addison-Wesley 2006

[6] Olaf Zimmermann, Pal Kroqdahl and Clive Gee, *Elements of Service-Oriented Analysis and Design*, Article on IMB DeveloperWorks web site, http://www.ibm.com/developerworks/library/ws-soad1/

[7] Thomas Erl, *Service-Oriented Architecture – Concepts, Technology and Design*, Prentice-Hall, 2005