IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITP4244

A Method Tailored For System Integration

Jon Oldevik and Arnor Solberg
SINTEF Telecom and Informatics, Norway, {jon.oldevik, arnor.solberg}@sintef.no

Audun Jensvoll
EDB4Tel, jensvoll@edb4tel.com

ABSTRACT

Large companies that have been in business for some years necessarily aggregate lots of history concerning their business. The business lives and breaths in pursue of new markets, better price/earning rates, etc. Their business systems that were designated to support their business does not always follow the dynamic needs of the business and the surrounding technology; the architecture and technology demands changes; requirements for interchange and interoperability with other, maybe external system, are put forward. Today, there are increasing demands to support open interfaces that can be easily integrated with Internet applications and integrating legacy systems into state-of-the-art component infrastructures.

In order to achieve this, we need to identify the structure of the business in terms of systems and components. We need to identify the dependencies between components and the structure and semantics of their interfaces. In this paper, we describe a method based on Unified Modelling Language (UML) and Unified Process (UP) that is tailored to describe component architectures in terms of their collaborations, their structure and semantics of interfaces, to facilitate integration capabilities between components in heterogeneous environments. We also use forthcoming concepts in UML 2.0 to model protocols between components. We aim to provide a method that supports a model driven architecture (MDA™[1]) tuned towards integration.

BACKGROUND AND INTRODUCTION

Enterprises today exist in an environment of rapid technology changes. The flavour of the month of implementation languages, database systems, and hardware platform is quickly changing. The choices made in the 1970ies and 1980ies are still alive in terms of legacy systems. Although these still may support the business fine, we need methods and techniques for integration with new systems built on newer technology to support new business and market requirements.

The industry put forward demanding requirements for flexible and customisable development processes that support development within diverse communities, towards heterogeneous hardware and software platforms. In addition, there is an increasing need for standardising the specification of the system, not in terms of process, but in terms of artefacts. A standard set of deliverables from a specification and design process leverages the value of this activity. We encourage the model-driven (or artefact-driven) development that focuses on standardised artefact deliverables. This standard should be equally valid for specifications of new and existing systems. The goal is to enable integration of existing components and new components by using the same framework for specification and design.

The method presented here focuses on delivering models on different abstraction levels; the models are the operative force of achieving deliverable artefacts and provides a structural organisation of deliverables for the developers. It is not a full-fledged method backbone, but aimed at supporting component specification within heterogeneous environments.

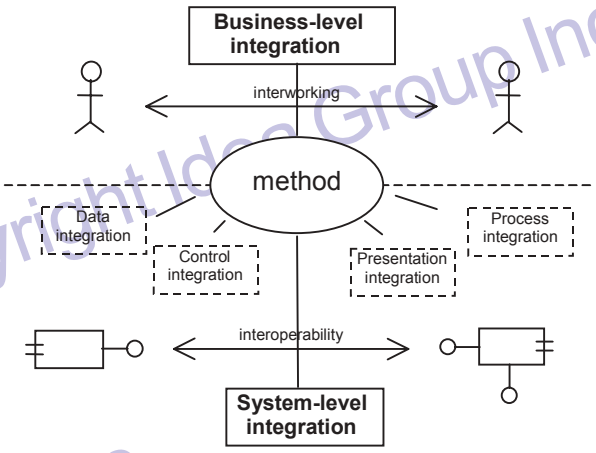
The most recent background for this work was conducted for the Norwegian Telecom Company Telenor and has been further refined since then. Here we applied results and ongoing work from several EC projects (OBOE[4], DISGIS[5], COMBINE[6]) and Norwegian research projects (MAGMA[7] and DAIM[8]). UML and RUP/UP are major input factors in all the methodology work done here, but also concepts from Ooram[9], Catalysis[10], and others.

The following sections describe the details of our method and how it facilitates system integration.

FACILITATING SYSTEM INTEGRATION

System integration spans a range of complex problem areas that hardly is solved by a single solution. We can organise complementary integration aspects within an integration architecture, representing different aspects of integration; control integration, data integration, process integration, presentation integration, and business-level integration. (These integration areas are defined in the ECMA/NIST toaster model[11] and refined in Berre93[12].) Within such an architecture, a key catalyst for successful integration (and migration) is a method that enables a unified way of describing existing systems, new systems, and their integration. Figure 1 depicts that interoperability between systems and inter-working between people is facilitated/leveraged by the method. Interoperability is achieved by being able to integrate systems at different levels. Inter-working between people is achieved by means of shared tools and methods, which leverages human communication and common understanding.

Figure 1: Integration architecture



Integration between components and systems can only be achieved if we know the interfaces and the behaviour of the engaged parties. We need a common way of describing components and interfaces and common means of locating, inspecting, and using them.

Another aspect of integration is how interrelated systems and components reflect towards the business level. Ultimately one must be able to identify the business benefits of integration, i.e. we should be able to map between business goals and the technical interconnection of systems, be it existing or new interconnections.

Unified process (UP) and UML provides a baseline for describing systems in terms of process framework and standard notation. These

This paper appears in *Issues and Trends of Information Technology Management in Contemporary Organizations,* the proceedings of the Information Resources Management Association International Conference.

are, however, too flexible and too widely scoped to support the specific needs within an enterprise. Most importantly, they do not target the specific need of describing system and component integration. We provide a more specialised set of models and techniques that standardises how to describe components, their interfaces, their integration, and their link with the business level.

MODEL-DRIVEN PROCESS

Our overall process is phase-driven like the Unified Process. It contains a set of phases that ends in a review activity. Each phase is divided in a set of iterations, where each iteration delivers a review artefact. The process diverges from UP in its dedicated focus on components and their interfaces. This is particularly visible in the delivered artefacts, which are focused around the interfaces of components. Even though the goal of the business modelling is to understand and describe the business and its requirement, the reason for doing it is to identify components supporting the business and describe the details of their interfaces.

An important factor is the combined support for existing (legacy) and new systems in the process. The specification of an existing component developed in e.g. Cobol should be according to the standardised framework and thus compatible on a model-level with other specifications e.g. for EJB components. The main difference from UP is the dedication towards platform independent models, focusing on components, their protocols and interfaces. In addition, we provide special techniques to support bottom-up analysis of existing components.

ARCHITECTURE-DRIVEN METHOD

The method recommends establishing of reference architectures that supports the domain within which we are currently working. The goal is that a reference architecture will help organising concepts according to well-known rules, which in turn will facilitate interoperability between specifications.

The reference architecture defines a logical architecture, in which general modelling concepts are identified and described in a UML profile. This profile defines which concepts to use during modelling and their semantics. An example of this could be a profile that defines modelling concepts for web-service, business service, business entity, web-presentation-component, and business event. This profile reflects a reference architecture that should be enforced when modelling and supported by modelling tools.

Adopting reference architectures that discipline the end-artefacts of the development method leverages the odds of reusability and increases system integration ability, since artefacts follow the same structural and behaviour rules to some extent.

MODELS

The models are the central focal point of our method. These are replacement of UP's core workflows. The model abstraction provides a simplified view of the developers workspace, focusing on the roles and their responsibilities during development through the models they are responsible for.

We distinguish between four modeller roles: The business modeller, the architecture modeller, and the interface modeller, with the following responsibilities:

- The business modeller handles domain modelling, use case modelling, and inter-domain use case modelling.
- The architecture modeller handles component collaboration, i.e. the structuring behaviour of components.)
- The interface modeller handles interface design detailing, i.e. the details of component interfaces and the protocols they are involved in.
- The technology modeller handles technology-specific issues, i.e. how a specification is realised in a particular technology.

The most important result of our process is a specification complying to the reference architecture and that specifies all necessary details of a set of components, their interfaces and protocols. We have an iterative and incremental method and with many paths towards a specification, depending on the experience and habits of the actors fulfilling the modelling roles, and the starting point of the modelling. When specifying existing systems, a bottom-up-oriented approach might be appropriate, whilst a top-down-oriented approach might be more suitable when specifying new systems.

Either way, and regardless of target technologies, the end products should follow the standard defined.

The Business Modeller

The business modeller describes the domain within consideration and identifies the requirements to the system. The business modeller is responsible for the following artefacts: (1) The domain model, comprising rich pictures, business processes, and business information model. UML class models, use case diagrams, and activity diagrams are produced in this model. (2) The use case model, comprising high-level and detailed use cases. UML use case diagrams are produced in this model. (3) Inter-domain use cases, comprising a use-case oriented view on a set of inter-related systems, focusing the area of concern on the integral parts of several systems. UML use case diagrams are produced in this model. The business models provide a common way of describing the business and thus a means of enabling business-level interworking and integration.

The Architecture Modeller

The architecture modeller identifies the components within the area of concern, identifies their interfaces, and describes the details of their interaction. He/she is responsible for two main artefacts: (1) The component structure model, comprising the components, their dependencies, and their interfaces. Components are modelled using UML subsystems or stereotyped classes. (2) The component interaction model, comprising the collaboration behaviour of components. Interactions are modelled using UML sequence diagrams. The structure of components provides an overall picture of how components are interrelated. The interactions show how components interact to achieve their responsibility. Together with the interface details, these describe the protocols needed for component interaction and integration.

The Interface Modeller

The interface modeller describes the details of component interfaces and is responsible for one artefact: The detailed interface specification, comprising the semantics of each component interface. This artefact details the necessary logical structural and behavioural aspects of the interfaces of components. The interfaces are modelled as UML interfaces, detailed with properties (operations, attributes, and abstract relationships). The goal is a technology independent specification that provides a general abstraction of many possible technology realisations. The detailing of the interfaces is in close relation with the component interaction artefact, which together aims to describe the protocols between components. The resulting artefacts are part of a platform-independent model of the problem domain.

Technology Modeller

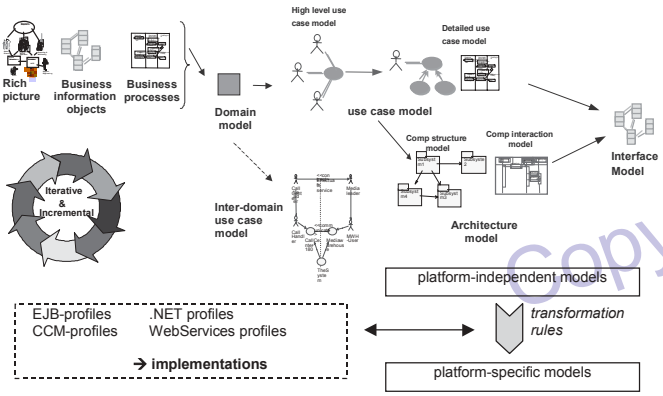
The technology modeller defines the appropriate transformation from a platform-independent model to a platform-specific model (PSM). Within a specific domain, the transformation rules can be tuned towards a specific reference architecture, which defines generally valid rules for technology transformations towards a specific technology platform. These rules can also define interoperability properties, i.e. how a component will access another, possibly legacy, component.

The process of going from platform-independent to platform-specific model is part of the model-driven strategy supported by this method.

Figure 2 depicts the artefacts and indicates one possible flow of a modelling iteration. The result of this iteration, is a platform-inde-

pendent component and interface specification. Then, from a set of transformation rules, possibly described in terms of technology profiles, we produce platform-specific models.

Figure 2: The method in a nutshell



By basing the systems on platform-independent models and rule-based transformation to platforms, we reduce the problems of changing technology and mapping towards heterogeneous platforms.

EXAMINING EXISTING SYSTEMS

Within an enterprise's IT-systems portfolio of existing components and systems, there will arise need of integrating parts of existing systems with new, or other existing systems, in order to support new business ideas in terms of IT-services.

In other words, we want to obtain and create new business opportunities by combining existing IT-services to achieve new services to the customer. A simple example is to provide a new web-based end-user application that integrates billing and ordering systems that are run and managed separately, perhaps on different software and hardware platforms.

Eventually, it requires an integration of several diverse components or systems to achieve this goal. Our area of concern with regard to this new problem is not one component, the other component, or both components. It is an intersection between the involved components that together solves the problem. This is the focus of the inter-domain use case model, depicted as a part of Figure 2.

If we look at an imaginary order/billing situation, we can analyse this from a use case perspective. The customer orders some product of the on-line web ordering system. The order is processed and the order engine checks the economic viability of the customer. Coming through ok, the order is sent to the billing system, which creates an electronic invoice sent to the customer.

When analysing the use cases, we see that at least three different components are involved. Now, if we assume that these three components already existed and that they solved the above indicated user case, we know that these components are collaborating, interacting through their interfaces. We need a way of analysing the collaboration, relating it to the business case, and describe the protocols and interfaces of these components. The inter-domain use case model has proven a useful technique in this regard. Here, in a bottom-up approach, we identify the area of concern involving the parts of the components involved, and unfold the actors external to the involved components. This can be done recursively until the necessary level of business actors and interest has been identified. This technique helps bridging system integration with business requirements and goals.

The next step is to describe the details of the component collaboration, including the details of interfaces to provide the technical foundation for the integration.

Component Collaboration, Interfaces and Protocols

Components work together in order to solve some problem, in what we call collaborations. A collaboration of components is defined by the way the components interact to solve the problem. Interactions are defined by the protocols enforced by the components, i.e. the exact semantics of the behaviour.

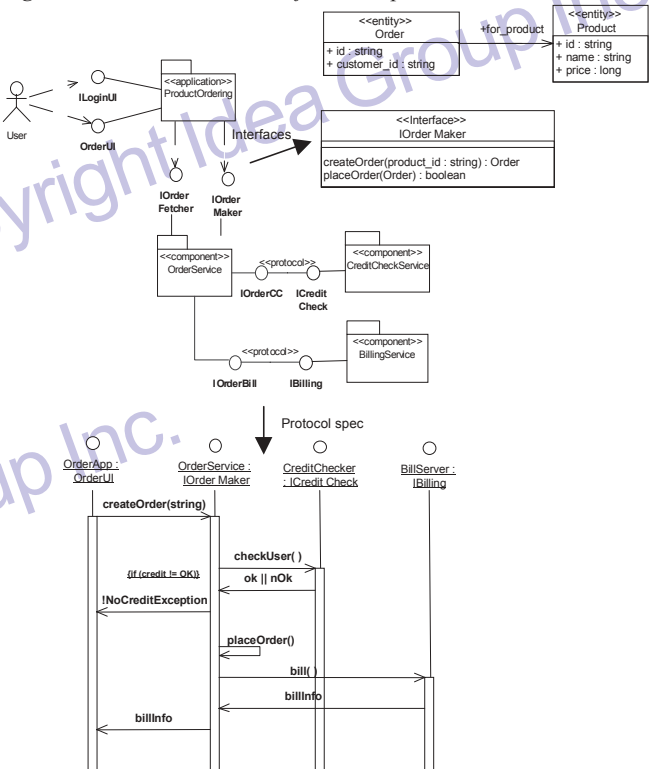
The interfaces of a component define how other components can plug into this component. This is also known as incoming interfaces. A components outgoing interfaces defines which external components that are used. The way two components interact (i.e. the messages sent between them) defines their protocol.

In order to define the protocol, we describe the interactions between components for different use case scenarios in terms of UML sequence diagrams. Each interaction scenario identifies parts of collaboration protocols in terms of the messages sent and the messages responded from/to a component. Each message targeted for a component must correspond to an operation specification on an interface supported by that component.

The focus in our method is describing the protocols in the component interaction model and detailing the interfaces in the interface model. We have three views on a component collaboration. The component structural model, the component interaction model, and the interface model. Figure 3 depicts parts of how this can be modelled in UML. In order to model to-way protocols between components, we relate interfaces owned by components and associate protocol specifications to that relationship. This is not allowed in UML 1.x, but is likely to be part of UML 2.0. The concept is similar to that of identifying ports for components though which components can interact.

In order to support a standard artefact production, we standardise on notation (i.e. we use UML) and how to use UML in the context of component modelling. For describing interfaces, a standard template must be followed. It covers completeness of signatures (operation names, parameters, types, exceptions, pre- and post-conditions), complying to an agreed-to type-system when modelling (e.g. using ISO IDL types), using name conventions for names of components, interfaces, operations, attributes, parameters, etc.

Figure 3: Collaborations, interfaces and protocols



Even though the platform-independent specification appears as a model of standard, synchronous components interacting, this is not necessarily the case. The target technology may as well be a message-oriented or other asynchronous platform.

The complete details of component collaboration require that we for each invocation from one component to another, specify the expected behaviour in the sending and receiving end. Specifications in terms of semi-formal OCL (Object Constraint Language) expressions can define this behaviour. An OCL pre-condition can specify the required state of the receiving or sending end. A post-condition can specify the required end-state of the receiving or sending end.

In the COMBINE[6] project, we are working on problems related to guaranteed pluggeability of a component within an enterprise, based on concepts defined in the EDOC[2] (Enterprise Distributed Object Computing) submission to OMG (Object Management Group). EDOC defines how components can interact through protocols and ports, allowing a complete specification of a component. A protocol is defined by the communication along two ports attached to components. By combining several specifications, we can describe complete collaborations of components. The same problems addressed in UML 2.0, where the need for ports and internal parts for components has been identified.

FACILITATING REUSE OF ARTEFACTS

Standardising description of notation and semantics is for the purpose of reuse. If reuse of modelling artefacts can be implemented in an organisation, the benefit of a model-driven approach will really be surfaced. In UP, the set of artefacts and techniques can appear too versatile for establishing common understandings. A packaging into a stricter framework of highly specific and standardised way of providing deliverable artefacts, is an important step towards benefiting from existing assets:

- Business model reuse: The artefacts from business modelling can be reused in terms of business process models, domain models, requirement models. We attain business specifications that provide a common platform for accessibility, availability, and usability.
- Component model reuse: The artefacts from the component and interface models can be reused as platform-independent specifications as starting point for platform specific model generation/transformation and to supply system integration wrappers based on component and technology transformation specifications.
- Implementation reuse: Implementation of components that comply to technology profiles specifications can be reused (in terms of their specifications as well as their implementations and deployed installations) to provide integration with new or existing components.

CONCLUSIONS AND FUTURE WORK

In this paper we have described a method that facilitates system integration by standardising a simple way of describing components, their interfaces, and how they collaborate with other components.

Our approach tunes the apparatus available in UP/UML towards specifying interfaces of components and how components interact through interfaces. Our process is lightweight and aimed at producing platform-independent artefacts that describe the essence of the boundaries of components.

In the future, we will look at implementing tools to support the mapping from platform-independent models to platform specific models and how to provide tool- support for automation of system-level integration frameworks. We will also harvest experiences on the deployment and usage of out method and use these for improvements and extensions. We will work closely in line with the ongoing OMG standardisation work on model-driven architecture and pursue a more complete support for it through technology transformation techniques. In the CAFÉ[13] project, we will build on these experiences in further refinement and application of the MDA aspects. The standardisation work on UML 2.0 is an important discussion arena for the evolution of component modelling and will be followed closely within the scope of our projects.

REFERENCES

[1] OMG's Model Driven Architecture: http://www.omg.org/mda (MDA is a trademark of OMG)
[2] OMG EDOC profile – UML profile for enterprise distributed object computing, http://www.omg.org/techprocess/meetings/schedule/UML_Profile_for_EDOC_RFP.html
[3] UML 2.0 Superstructure and Infrastructure RFPs, http://www.omg.org/uml
[4] OBOE - Open Business Object Environment (OBOE). Esprit IV project 23233, http://www.opengroup.org/public/oboe/Home.html
[5] DISGIS – Distributed Geographical Information Systems, Esprit IV project http://www.cordis.lu/esprit/src/22084.htm
[6] COMBINE – COMponent-Based INteroperable Enterprise system development, ESPRIT V project IST-1999-20893,
[7] MAGMA – Modular architecture, reuse, object-oriented methodology, and workprocesses in software product development, a research project sponsored by the Norwegian Research Council.
[8] DAIM - Distributed Architecture, Internet, and Multimedia, a research project sponsored by the Norwegian Research Council.
[9] Ooram, Reenskaug, Wold, Lehne: Working With Objects. The OOram Software Engineering Method, Manning/Prentice Hall 1996. ISBN 0-13-452930-8
[10] Catalysis, Desmond D'Souza, Alan C. Wills, 'The Catalysis Approach', ISBN 0-201-31012-0, www.catalysis.org
[11] ECMA/NIST Reference Model for Frameworks of Software Engineering Environments, 3rd edition, http://www.ecma.ch/ecma1/TECHREP/TECHREP.HTM
[12] Arne-Jørgen Berre, An object-oriented framework for systems integration and interoperability, Phd thesis, University of Trondheim,1993
[13] CAFÉ – from Concept to Application in system-Family Engineering http://www.extra.research.philips.com/euprojects/cafe/