Distributed Enterprise Modelling Guided by Business Rules

Nahuel Di Paolo and Alejandra Cechich

Departamento de Informática y Estadística – Universidad Nacional del Comahue
Buenos Aires 1400, 8300 Neuquén, Argentina
E-mail: nahueldipaolo@hotmail.com, acechich@uncoma.edu.ar

Abstract

Enterprise architectures create a technical platform to meet current and future business needs. In this sense, architectural components ensure IT capabilities are modelled in every business process. Current approaches to enterprise architecting tend to focus on reducing the effort required to design business processes by using the resources provided by the Web to support new inter-organisational processes. This often leads to the development of services where some generic features or rules are particularly similar across different processes. We describe and illustrate the use of business rules to share components across many different views. Having a business rule front end lets a component offer different access rights and visibility to different clients. We present an approach that might be used in the process of enterprise architecting in conjunction with a service-oriented perspective.

Key words: Object-Oriented Modelling – Web-based Systems – Enterprise Architecting – Business Rules

1 Introduction

The way you define architecture, execute architectural strategy, and record the results makes a critical difference in your ability to deal with Information. With the migration to distributed-computing environments, an enterprise architecture should define and demonstrate the interoperability, scalability, and portability of applications and their subcomponents across the architecture. An IT architecture is a series of principles, guidelines, drawings, standards, and rules that guide an organisation through acquiring, building, and modifying resources. When defining enterprise architectures, the need for a technical perspective is clear, but a business perspective is equally valuable. In the architecture dimension, a system that is technically flawless but useless from a business perspective fails just as much as a system that cannot operate without errors [1]. Many articles and reports cover the architecting process but primarily from a technical view. The technical perspective includes steps and heuristics for creating a good architecture. These are, of course, critical to the architecture's success. But so is the business part, which includes steps to ensure that the architecture is successfully implemented.

On the other hand, the Internet provides an underlying framework on which today's organisations build their computing systems. But past computing paradigms – like the centralised mainframe and client-server architecture – don't fit well with the needs of today's organisations. As a possible solution, a Server-Based Computing (SBC) infrastructure facilitates centralised corporate management by letting system administrators impose standard software configurations reducing

administration to various business locations and giving users a standard, updated version of applications [2,3].

Before deploying SBC technology, a company must establish future system architecture and resources according to projected system growth. During design, a service-oriented perspective facilitates the definition of appropriate components and data-sharing strategies. A service encapsulates a subset of the application domain into a domain component and provides clients contractual access to it. In a sense, services do for components what interfaces do for objects [4]. Domain components are shared by different clients, each having its own contractual view of the component. A domain component offers services that represent business behaviour as well as semantic validation of the enterprise environment [5,6].

Semantic validation is based on current practices that are dynamic in the sense that they depend on changing regulations. Business rules are the dynamic for the sentences that are found in the language of business. The best way to identify business rules is to look at the organisation from its customer's point of view. Thus, a good enterprise model reflects a concise understanding of the structural qualities of the business plus dynamic regulations. Developers must understand the purpose of the business as well as the current practices of the business. Therefore, the quality of an enterprise model is measured in part on its ability to model both the structural business entities and the current (and dynamic) business practices. A related work, the Accessible Business Rule (ABR) framework [7], describes a framework that enables enterprises to develop distributed business applications that systematically externalise the time-and-situation-variable parts of their business logic as externally applied entities called business rules. Decoupling business rules from the application can provide a number of advantages. Because business rules are external to the applications that depend upon them, the variable business logic contained in them can easily be changed. Because the management of the externalised business rules is done explicitly through a rule management facility, it is easy to understand what rules exist and to locate those that need to be changed.

In this paper, we present an approach that might be used in the process of IT architecting in conjunction with a service-oriented perspective in which business rules offer an opportunity to share components across many different views. In Section 2 of the paper we introduce the main characteristics of our modelling approach. Section 3 then presents how business rules are used to analyse a particular enterprise domain by describing a case study. We conclude with an overview of future research directions.

2 A Modelling Approach Guided by Business Rules

Developing an enterprise's business model includes overall planning of information and infrastructure requirements, along with detailed planning of the model elements and their properties, functions, and interactions. An enterprise's major information service frequently involves satisfying the following general information architectures [8]:

- Query/Response. A query/response architecture can be used for querying and for producing reports.
- Conversational. A conversational architecture is used by managers and strategic planners. Interactions between the client and the server take place in dialogue mode and are context dependent.

- Queued-message. A queued- message architecture is usually used for routine transaction processing and for producing reports.

To satisfy these architectural requirements and develop a web-based distributed model, our approach models an organisation business process as an OO structure of hierarchical classes. It assumes an initial number of business processes where the identification of entities depends on particular functions and rules. Business rules are used to capture and implement precise business logic in processes, procedures, and systems (manual or automated). They can also provide the basis for expert systems. Enterprises that take a model-based approach to software component development can use business rules to refine the models and create better designs. An enterprise that properly documents its business rules can also manage change better than one that ignores its rules. Business rules may be any of the following:

- Definitions of business terms
- Data integrity constraints
- Mathematical and functional derivations
- Logical inferences
- Processing sequences
- Relationships among facts about the business

The basis for good business rules is a single, coherent model for all the rules, no matter their source or where they are implemented. Business rules take several forms when documented in enterprise business models. There is one best representation that is appropriate for each particular type of business rule. For example, strategic business rules are best modelled as simple business statements, while operational business rules are best modelled as static data instances and derivations. Our business-rule guided model consists of a set of UML interaction diagrams [9] that may in turn consist of domain as well as application classes. To model business rules, we followed a typical cycle [10]:

1. Describe the enterprise mission in a brief statement of purpose: what the enterprise does, how, and for whom.
2. Make assumptions and gather data about external factors: government policies, rates of inflation, markets, and demographic changes.
3. Assess enterprise strengths and weaknesses.
4. Establish goals, objectives, and measures linked to the enterprise mission.
5. Develop strategic and operational plans to meet the goals and objectives, utilise strengths, overcome weaknesses, counter threats, and address opportunities.
6. Design/re-design and integrate cross-functional processes to meet goals and objectives.
7. Implement information systems that support enterprise processes and assist in decision-making.
8. Evaluate performance to ensure that goals and objectives are being met.
9. Re-evaluate and change goals, objectives, processes, and measures as necessary.

During each of these steps, business rules are discovered, documented, and modelled. First, business requirements and business rules are defined in real-world terms: goals, strategies, objectives, tasks, critical success factors, etc. As business requirements are being defined, an Enterprise Architecture model is developed using logical models that are both unambiguous to developers and understandable to business experts. The Enterprise Architecture is made up of

logical data, process and activity models that represent the information and business rules necessary to support the defined business requirements.

Following, we set a number of activities that we undertook for rule analysis and design at this point. Additionally, there are a number of concepts that need to be taken into consideration during the analysis and design processes.

2.1 Extracting business rules from use cases and interaction diagrams

Business rules are found during analysis by inspecting the use cases or user interaction scenarios [9,11] that are typically developed as statements of requirements, which are input to the analysis process. The analyst will study this description to output the essential information, identify concepts, and distinguish static requirements (describing, classifying, and organising the problem domain concepts) from dynamic requirements (representing system operation and business rules). We have developed some guidelines that help identify business rules from use cases and interaction scenarios:

- Sentences that express different types of classifications between the subject and the predicate might indicate different processing rules. Investigating processing source an analyst could identify organisational constraints on the process.
- When the set of concepts into which the subject can be divided is expressed in the subject (for example, invoice and receipt are types of documents), it might indicate different and dynamic treatments.
- Differentiate sentences that express a permanent relationship between concepts from those that express dynamic behaviour. For example, “a person is located at his home address” from “people are paid monthly”.
- Identify conditional sentences that might express changing procedures, for example “when a sale is made, the vendor reports the order to the sales manager”.
- Identify conditional procedures in which authorisations are required, for example “once the orders are reported to the sales manager, he send confirmation to the vendor and forwards the orders to the suppliers weekly”.

In general, analysts build the business rules representation based on the dynamic requirements that specify interactions and events that affect the information described in the static requirements. Therefore, they specify how the system under development should behave constrained by changing organisational aspects. The first step should be the preparation of use cases and their scenario. It is a concrete, focused, informal description of a single feature of the system from the viewpoint of a single actor. A scenario is an instance of a use case describing a concrete set of actions. Scenario descriptions usually have three components: (1) the name of the scenario, (2) the participating actor instances involved in the scenario, and (3) the flow of events that describe the sequence of interactions step by step.

After applying the guidelines mentioned above to a use case, its description is transformed to differentiate business rules. Notice that it is important to check that all the operations belonging to the same use case are interrelated.

2.2 Architecting the system

There are numerous architectural styles and patterns. Some are widespread, and others are specific to particular domains [12]. The architectural issues for a distributed system include both the network architecture connecting databases and application components, and the database system architecture. The great popularity of the Web has resulted in a huge growth in use of Internet, which has also led to the growing popularity of building systems on the Web. A Web application is defined as any software application that depends on the Web for its correct execution [13,14]. Software explicitly designed for delivery over the Web and software that uses the Web infrastructure for its execution fall under this definition. Web technologies provide the basis for constructing actual Web objects supporting the interfacing and messaging aspects of distributed object systems.

Our business-rule architecture is based on a mediated architecture [15], which is partitioned into three layers as shown in Figure 1: end-user applications, mediating information servers, and information repositories. Users send requests through a Web browser interface; and the System Manager mediator fills the request by gathering and processing data from the distributed databases. To process data, the System Manager uses business rule information that dynamically constraints processing. Results are incorporated into an HTML/XML page and sent to the requesting Web browser, as Figure 2 shows. The System Manager runs continuously. When it receives requests from the Query processor, the IT System mediator processes the query by using the appropriate business rule and sends the results to the requesting process.

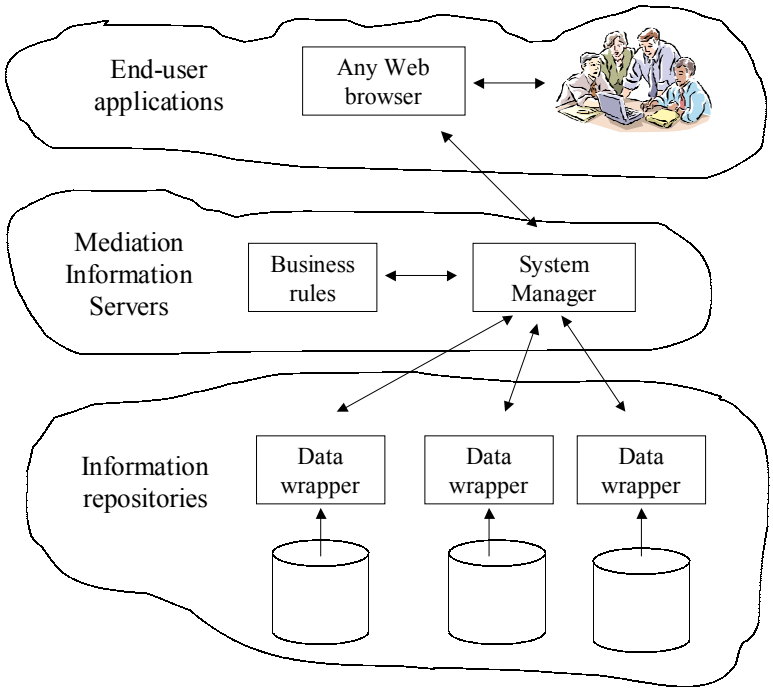

Figure 1. Three-layered business-rules architecture

The architecture offers the flexibility to make organisational changes. Using a business rule constraint procedure lets managers control the information they provide and require. Also, using wrappers at the information repositories lets data owners limit access to the data by controlling repository access. This offers the basic security measures typical of current large-scale development projects.

Finally, a Web browser at the end-user level provides data access on a variety of platforms.

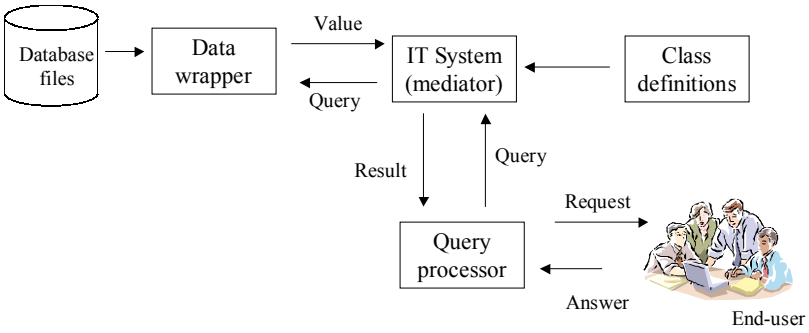

Figure 2. User requests passed to a mediator

2.3 Deploying on distributed environments

Enterprise systems frequently consist of many components distributed in highly complex configurations: heterogeneous, geographically distributed servers linked through many networks interconnections, databases of enormous size, and so on. Understanding the interactions within such a system is the first step in building an effective model [16].

Defining clear objectives is the key to a successful modelling project. However, objectives may differ depending on the stage of the design process. When used during early design, an enterprise system model cal help build performance into the system at a time when the costs of doing so are minimised. The performance model may also be used to study different implementation choices. For example, should data be replicated at different sites to ensure reasonable response time? Are local application servers a good idea? Modelling is all about tradeoffs, furthermore modelling objectives should drive the tradeoffs between model abstraction and model accuracy [17]. One of the greatest difficulties of this analysis is understanding the customer problem and turning it into a clear problem statement. Business rules help guide the analysis and build a model based on user roles, which can then be defined in terms of the business processes and decomposed into the computational transactions required to be operated within an architectural framework.

Reusing business rule components allows a better categorisation of component capabilities according to a component's contribution to an overall component-based system's functions and helps organise non-functional constraints. A key feature of this categorisation of business rule component characteristics is the idea that some components provide certain rule-related services for other components (or end users) to use, while other components require certain rule-related services from other components. These details aim to increase developers and end users knowledge about business behaviour by providing a more effective categorisation and codification mechanism for the business rule services.

When reasoning about operations of business rule components we analyse these in terms of particular tasks. Hence, task architecture is developed in which the system is structured into concurrent tasks where task interfaces and interconnections are defined. To help determine the tasks, business rule structuring criteria are provided to assist in mapping the model of the system to a concurrent tasking architecture. These criteria are a set of heuristics, which capture designer knowledge about the application domain, the design of concurrent systems, and the business rule behaviour.

3 A Case study – People registration

In this section, we introduce a case study developed in conjunction with the Neuquén Province Government to motivate the need of business-rule-based developments. The complete example is omitted here for brevity.

The example consists of part of a distributed application for people's registration where one or more instances of a user interface component are used to access a remote database containing people-related information. Registering people's data consist of recording facts and acts that safely conform, alter, or modify the marital status and capacity of people, and preserving documentation by transferring information to a National Repository. The kinds of facts able to be recorded are: deceases, births, marriages, and people's identifications. Different areas inside the organisation, which requires fulfilment of different forms and recording books to consider a procedure done, carry them out. Our example is focused on developing software subsystems that allow dealing with all the mentioned procedures; however, we only show here a business-rule-oriented modelling of the birth procedure for brevity reasons.

3.1 Extracting business rules for birth registrations

As we mentioned in Section 2.1, use cases are the main diagrammatic tool to be used when detecting business rules, providing easier analysis on functionalities and procedures. A class diagram is added to include main interacting domain entities on interaction diagrams describing how business rules apply.

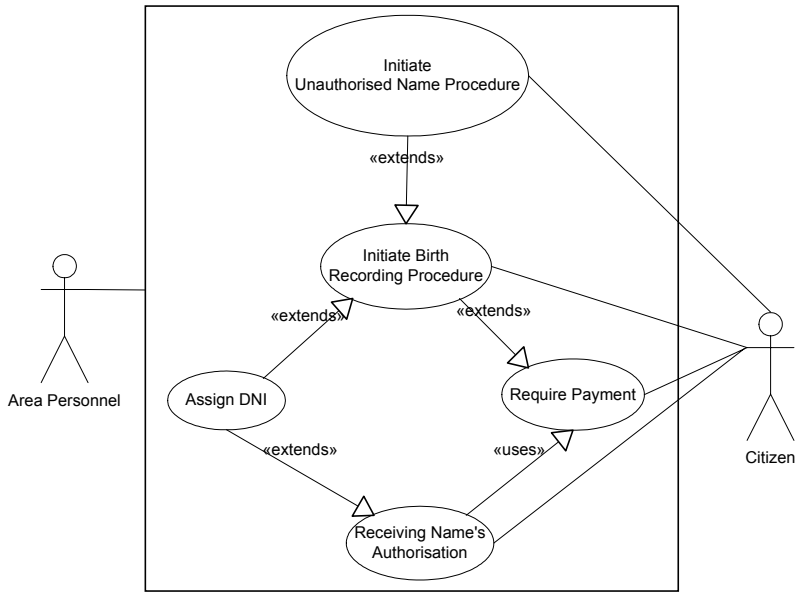

Figure 3: Use case diagram for births

The use case diagram in Figure 3 shows how a birth procedure should start. Two actors are identified on the diagram: the *Area Personnel*, who will operate the system, and the *Citizen*, who will supply the required information. The *Area Personnel* also will perform the tasks of requiring data to the *Citizen* and recording the birth information. They will operate the system by using standard input/output devices. In this context, *Area Personnel* is considered as a role, which might represent actually more than one actor playing the role. On the other hand, the *Citizen* will be a

person willing to announce a new birth. He/She will actively participate in the procedure by supplying all required information. *Citizen* is also considered as a role in this context.

When a *Birth Recording Procedure* is initiated, the possibility of detecting an unauthorised name as requirement exists. Hence, the standard use case is extended by *Initiate Unauthorised Name Procedure* to attend this possibility. *Receiving Name's Authorisation* describes how to proceed when information that allows continuing the procedure *Initiate Unauthorised Name Procedure* is obtained. After receiving a birth registration requirement, a new person's identification (DNI) is obtained (*Assign DNI*) and the corresponding payment is requested. Figure 4 shows a partial description of the use case.

Initiate Birth Recording Procedure
GOAL
Gathering relevant data to start a birth procedure

ACTORS
Area Personnel
Citizen

MAIN FLOW
1. Area Personnel shall gather all relevant data to start the procedure. At least the following data should be obtained: Birth Certificate; Father's ID; Mother's ID; Baby's name
2. If there exists a birth certificate then the Area Personnel shall verify its validity
   Otherwise two witnesses are required to certify the birth's data
3. All relevant data must be checked and loaded. The birth certificate shall be kept in the "Folder of birth certificates"
4. Baby's name shall be authorised.
5. If the name is unauthorised, then an authorisation shall be requested and the use case "Initiate Unauthorised Name Procedure" is initiated.
6. A new identification shall be assigned and the use case "Assign DNI" is initiated.
7. All data must be recorded on the main database system.
8. The corresponding payment shall be required and the use case "Receiving Payment" is initiated.

RESULTS
The birth certificate is kept in the "Folder of birth certificates" and all data is loaded into the system.

Figure 4: Use case description for Initiate Birth Recording procedure

Statements in Figure 4 were analysed to produce a preliminary set of business rules, according to the guidelines mentioned in Section 2.1. Therefore, we differentiate sentences that express a permanent relationship between concepts from those that express dynamic behaviour, and we identify conditional sentences that might express changing procedures, for example "if there exists a birth certificate, then the Area Personnel shall verify its validity". We also identify conditional procedures in which authorisations are required, for example "if the name is unauthorised then an authorisation shall be requested and the use case *Initiate Unauthorised Name Procedure* is initiated".

Of course, a conceptual model of domain classes is part of our modelling approach. A traditional hierarchical representation of domain entities builds a structure in which aggregation, and inheritance relationships are identified. But, business rules are the main issue of our approach. They are modelled as part of this static structure as well as part of the interaction diagrams – as we suggested in Section 2.1. Scenarios as instances of a use case allow us to follow the flow of events that describe the sequence of interactions step by step. So, we identify different business rules for the birth procedure example and represent them as part of the class and interaction diagrams as Figure 5 shows. For the sake of the example, we chose the *Receiving Name's Authorisation* business rule to proceed architecting the enterprise system. In Figure 6, the use case description and

the interaction diagram is shown. The actor *Area Personnel* in the use case description indirectly produces the information recorded by the entity of the same name in the interaction diagram.

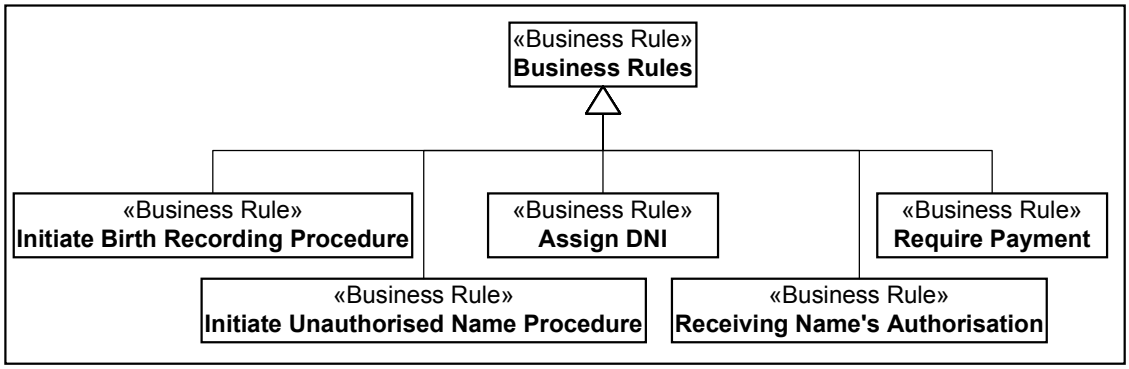

Figure 5: Class diagram for business rules

Receiving Name Authorisation Procedure
GOAL
Receiving the name authorisation to proceed with the Birth Recording Procedure.
ACTORS
Area Personnel
Citizen

MAIN FLOW
1. Area Personnel shall receive the authorisation and declare the name as "authorised".
2. If the registration procedure corresponds to a baby then initiate the procedure "Assign DNI"
3. If there exists a birth certificate then the Area Personnel shall verify its validity
   Otherwise two witnesses are required to certify the birth's data
4. All relevant data must be checked and loaded. The birth certificate shall be kept in the "Folder of birth certificates"
5. All data must be recorded on the main database system.
6. The corresponding payment shall be required and the use case "Receiving Payment" is initiated.

RESULTS
The birth certificate is kept in the "Folder of birth certificates" and all data is loaded into the system.

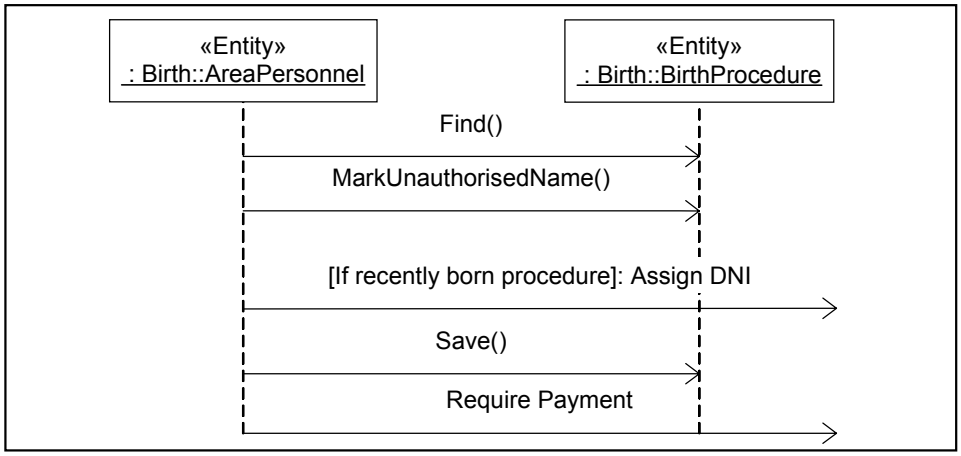

Figure 6: Use case and interaction diagram – Business rule "Receiving Name's authorisation"

3.2 Architecting unauthorised name registrations

This section mainly describes the role of business rules as mediators to process data. The System Manager uses business rule information that dynamically constraints processing in such a way that interaction among business rules, user interface and data repositories satisfy the system's goals. Requirements of different business rules are analysed from two viewpoints, which in turn constitute two different layers: (1) the end-user layer and (2) the data storage layer (End-user applications and Information Repositories respectively in Figure 1).

Business-rules guided architecting implies analysing and identifying goals of every class in the system, and their relations to the central business rule of each collaboration diagram. Typically, we have one collaboration diagram for each business-rule's behaviour. In such a diagram, we identify some classes as data entities, which are the basis to identify abstraction classes or database wrapper classes. Analysing different views helps define a more consistent and complete specification of interface and storage needs. Identified interfaces are part of the abstraction level specified by using collaboration diagrams. In Figure 7, a collaboration diagram – following notation suggested by [16] – representing a business rule's behaviour is shown. The *Receiving Name's Authorisation* business rule manages a business process whose main goal consists of initiating a birth registration procedure in which an unauthorised name is received as input. User and data administrator participation are required by the business rule in order to achieve its goal. These requirements allow us to identify new domain and system classes that eventually complete the system. For example, when analysing user's participation we notice that users should supply the identification of the procedure as a first step. Hence, user interface classes are updated to get the corresponding identification.

Once user's participation is analysed, a similar analysis for other kind of requirements is carried out, i.e. analysis on requirements from data storage management or other business rules. Thus, in Figure 7 invocations to *Birth Procedure* and *Citizen Born* entities indicate a repository access; and invocations to *Assign DNI* and *Require Payment* procedures indicate business rule invocations.

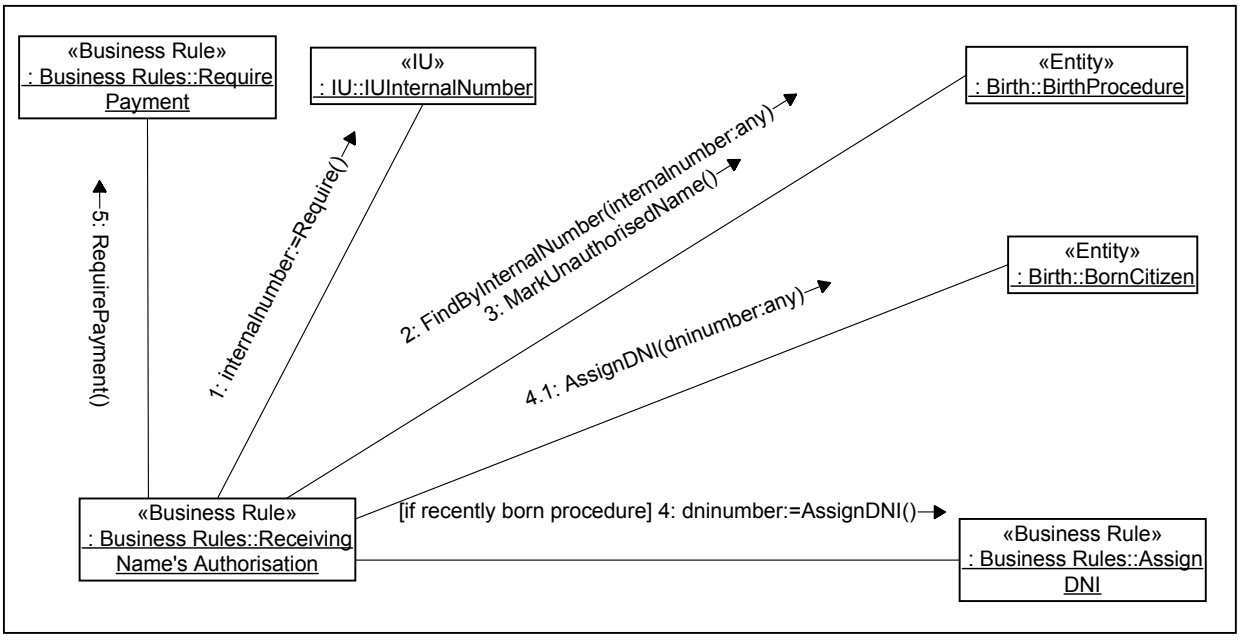

Figure 7: Collaboration diagram – Business rule "Receiving Name's authorisation"

Invocations in Figure 7 are representing broad sense collaborations by which a business rule executes its goals. Particularly, an entity invocation might be translated into a database access – perhaps requiring the definition of other collaborations – when going deep into further states of design.

3.3 Deploying on distributed environments

An important objective in task and class structuring is separation of concerns. A task uses information hiding to address concurrency concerns, in particular details of timing, control, sequencing and communication. A class uses information hiding to address structural concerns, of which are different kinds, such as hiding device information or data abstraction. The business-rule approach allows identifying components whose well-defined interfaces easily create distributed architectures. Business rule perspective let define a communication schema based on messages passed among classes. Message-based architectures are well known in modelling distributed environments [12]. In our modelling approach, collaboration diagrams clearly defining component interfaces explicitly represent messages.

The activity of defining components is based on identifying components that might be distributed among different nodes. For the sake of the example, the business rule component of the *Birth Procedure Subsystem* is located in the *Business Rule Server*. The other components encapsulate the user interface specifying the control logic for accessing data. Physical location of the user interface components might correspond to any client node; meanwhile the *Data Repository* component will be located in the *Data Repository Server*. Figure 8 shows a possible deployment diagram. We should notice that the resulting configuration is flexible enough to allow that business rule components and data access components be allocated in the same physical node as well as in different nodes. We have chosen a Server-Based Computing (SBC) infrastructure to facilitate centralised management by imposing standard software configurations. However, the same components might be reallocated and reused in a traditional client-server infrastructure. In addition, Internet technology allows a client to be located on any physical location and on different implementation platforms.

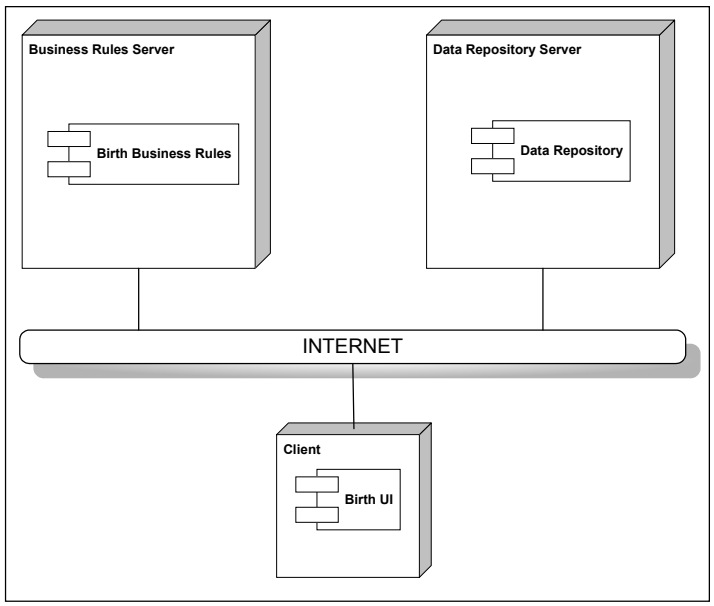

Figure 8: Deployment diagram for the Birth Recording Procedure

Conclusion

We have presented an enterprise modelling approach that might be used in the process of IT architecting in conjunction with a service-oriented perspective in which business rules offer an opportunity to share components across many different views. The architecture is defined in general way, so to extend our business-rule approach it is necessary to include two new capabilities: consistent combination of data updates from multiple sources, and model building for these combined data. Our future work includes the identification and definition of different task architectures for the most common techniques of combined data storage on web-based applications.

Acknowledgments

Authors wish to thank personnel of the *Registro Civil* of Neuquén Province for their invaluable collaboration in developing the case study.

References

1. M. Boster, S. Liu, and R. Thomas: Getting the Most from Your Enterprise Architecture, *IT Professional*, *Vol 2 (4),* July/August 2000, 43-50.
2. M. Stang and S. Whinston: Enterprise Computing with Jini Technology, *IT Professional, Vol.3 (1),* January/February 2001, 33-38.
3. A. Volchkov: Server-Based Computing Opportunities*, IT Professional, Vol. 4 (2)*, March/April 2002, 18-23.
4. C. Roe and S. Gonik: Server-Side Design Principles for Scalable Internet Systems*, IEEE Software, Vol. 19 (2)*, March/April 2002, 34-41.
5. M. Fayad et al: Enterprise Frameworks Characteristics, Criteria, and Challenges, *Communications of the ACM, Vol. 43 (10)*, October 2000, 39-46.
6. L. Willcocks and R. Sykes: The Role of the CIO and IT Function in ERP*, Communications of the ACM, Vol. 43 (4),* April 2000, 32-38.
7. Rouvellou I. et al: Extending Business Objects with Business Rules, *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 33)*, IEEE Press, 2000, available at http://dlib.computer.org/conferen/tools/0731/pdf/07310238.pdf
8. A. Umar: *Distributed Computing and Client/Server Systems*, Prentice-Hall, 1993.
9. J. Rumbaugh, I. Jacobson, G. Booch: *El Lenguaje Unificado de Modelado – Manual de Referencia*, Addison-Wesley, España, 1999.
10. A. Perkins et al: Business Rules = Meta-Data, *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34)*, IEEE Press, 2000, http://dlib.computer.org/conferen/tools/0774/pdf/07740285.pdf
11. I. Jacobson et al: *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1995.
12. M Shaw and D. Garlan: *Software Architecture – Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.
13. Gellersen H., and M. Gaedke: Object-Oriented Web Application Development, *IEEE Internet Computing, Vol. 3 (1),* January/February 1999, 60-68.
14. F. Maurier and G. Kaiser: Software Engineering in the Internet Age*, IEEE Internet Computing Special Issue, Vol. 2 (5)*, September/October 1998.
15. G. Booch: *The Visual Modeling of Software Architecture for the Enterprise*, MicroSoft Development Network, MSDN Library Visual Studio 6.0.
16. H. Gomaa: *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Addison-Wesley, 2000.
17. R. Zahavi*: Enterprise Application Integration with CORBA Components and Web-based Solutions*, John Wiley & Sons, 2000.