An MDA Based Environment for Modelling and Executing Business Negotiations

Pierfranco Ferronato[1] and Jason Finnegan[2]

[1] Chief Architect, Soluta.Net, Italy, EU
pferronato@soluta.net
[2] Waterford Institute of Technology, Ireland, EU
jfinnegan@tssg.org

Abstract. This paper presents the results of a EU FP6 funded project named "Open Negotiation Environment" (ONE), including it's features, the strategies, the use of Model Driven Architecture (MDA), the tools, the process used in the development and the key decisions that drove the entire architecture's specifications.

Keywords: Modelling, negotiations, e-procurements, MDA, p2p, peer to peer, metamodelling, agreement.

1 Introduction

Supporting negotiations in a web based environment is a task that many tools and vendors already offers. Some of these also offer the ability to customize to some extent the tender/auction; for example the price and the duration is customizable, as is the fact that it can be a public or a private negotiation. In general these solutions support a rather small number of negotiation types (auction or direct buy usually) and for each some level of customization is allowed. Supporting new behaviors like for example extending the termination of the negotiation by 5 minutes after the receipt of an offer, or adding complete new models like "reverse auction" or generic tenders, is hard without the need to undergo a rather long set of changes in the application code. The ONE projects has been entirely conceived for being able to dynamically support new negotiation models.

The project is constituted of two main environments: the Factory, where models are defined, and the Runtime, where models are instantiated and executed. The latter is further defined in two phases: the Setup where negotiations are configured with data and the Execution where they are started and users actually participate to arrive at an agreement.

2 The MDA Approach

In order to support new negotiation models without undergoing the classical develop, compile, test and software deployment life cycle, the ONE consortium has

L. Telesca et al. (Eds.): DigiBiz 2009, LNICST 21, pp. 27–34, 2009.

decided to leverage the Model Driven Architecture(MDA [1]). ONE provides a tool where negotiation models can be created in a visual environment specifying the negotiation protocol, and the underlying information model: with this tool it is possible to model virtually any kind of eBusiness negotiation.

The models defined in the editor are instances of the negotiation modelling language, called the One Negotiation Metamodel (defined by the consortium [2]) that represents the core expression capabilities of the models.

The ONE consortium has defined a Domain Specific Language (DSL) for describing negotiations (a DSL is a specifically tailored language for describing the structure and the behavior of vertical domains). Adopting this approach, primarily aims at raising the level of abstraction at which the software is produced. The main idea is to capture the domain knowledge by means of models and to develop the supporting applications by instantiating these models and generating code, documentation, and other artifacts.

Models are processed by the run-time environment which allows users to tailor them to their needs such as the products information, the services, the start, the end date, and in general the values of the various variables defined in the model. During this phase, called “Setup”, users can also invite users and specify if the negotiation will be public or private.

3 The Implementation Strategy

The project has been conceived for a community of users and as a such it has adopted the EOA approach (Ecosystem Oriented Architecture) [3].

As defined in the DBE project[4] business ecosystem are based on a peer to peer model, where each participant plays both the role of a server and of a client, with the absence of a central control point as depicted in Figure 1 below. This topological model well reflects the nature of social networks: they are self sustained and are regulated by a mechanism where each participant offers part of its infrastructure as a resource to the community: from the business perspective this avoids the “big brother” syndrome.

The execution environment on the other hand is star based topology: the owner of a negotiation is hosting the negotiations itself, while the participants are using their nodes to access the negotiation. Figure 1 below. There are two dotted circles that represent p2p networks of resources. The first, in the upper part of the figure, are the nodes that can run the negotiation as either an owner or a participant in a negotiation. The second network, in the lower part of the figure, represents the p2p network of the Distributed Knowledge Base. The two other clouds represent two running negotiations: 'A' and 'B'. Each cloud is called a “Negotiation Arena” and their life cycle is regulated by each negotiation: they appear and disappear in sync with the negotiations.

The environment, due to the redundant peer-to-peer nature of the topology, is highly reliable and able to remain effective in case of a random failure of nodes.

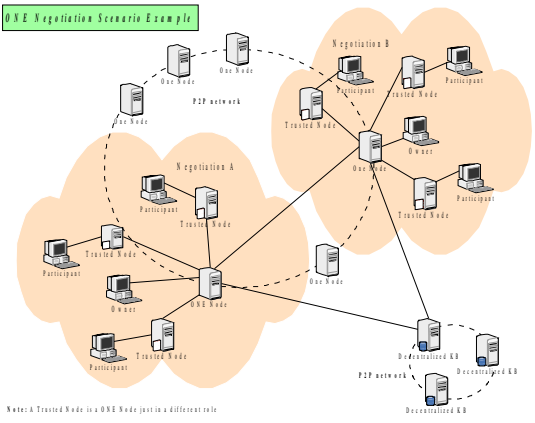

Fig. 1. ONE topological model

4 The Factory Environment

The Factory environment is where models are created, modified and published in the Distributed Knowledge Base. consists ssentially of a “boxes & lines” [5] tool that has been generated from the negotiation metamodel itself.

The GEF [6] and the GMF [7] Eclipse framework has been considered as the primary source for MDA related frameworks for the construction of the Negotiation Model Editor. More detailed information can be found in chapter 6, The ONE Negotiation Metamodel..

The generated code has been further customized in order to fit special requirements which can not be expressed in the meta model, for example creating wizards for adding Variables, Protocol Rules, Criterias, Action Language support or how to access to the repository.

In ONE it was decided to draw two state diagrams, one represents the Owner (left side), the other the Participant (right side). The two State Machines communicate via Messages that are sent by Actions in one side and that are captured by Events on the other side.

In order to describe precise behaviors, like Actions, preconditions and in general behavioral logic, it is necessary to support an action language, this means to augment the ONE DSL with an imperative language. Since scripts are to be executed at run time by the Negotiation Engine and since the entire project is based on Java, is was decided to adopt a subset of the Java Language [8] (JSR-274, implemented by Bean Shell [9]) in order to avoid further transformations in the Engine at runtime.

The editor is a rich client application: the models can be stored locally in the file system or in the Distributed Knowledge Base (DKB) as an XMI [10] encoded file. In line with the MDA approach, these models are not specific to an execution engine, they are a Platform Independent Models (PIMs [11]), there is no assumption about the Execution Engine, Jboss, Ilog or others, not even it assumes if the participants of the

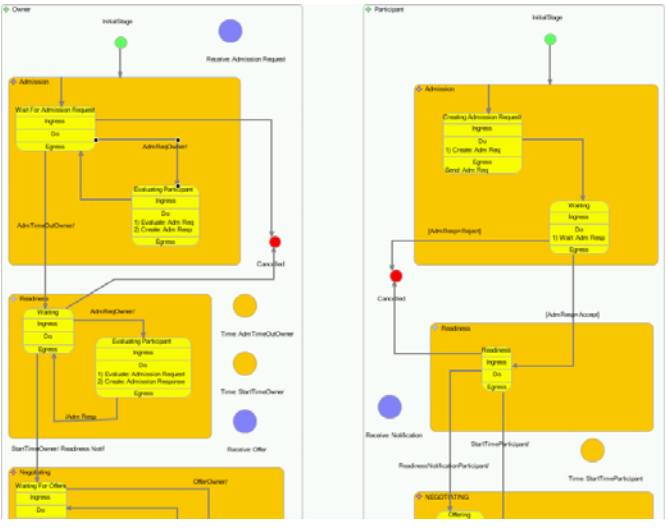

Fig. 2. Negotiation Editor, two state machines, fragment

negotiations are humans using Web based interface or software components using Web Services.

There is no use of an ontology, the projects decided to adopt folksonomy [12] to tag models and the information model.

5 The Runtime Environment

The Runtime is the Web based environment where negotiations' instances are created from models and executed; in addition this environment provides a set of side functionalities[13] like searching, inspecting for negotiations, registration, login and the management of a personal workspace where to keep track of past negotiations, called the My One [14]. The connection point between the Factory and the Runtime is the Distributed Knowledge Base (Distributed Knowledge Base) that receives the models created in the first environment and they are then retrieved, configured and executed from the second.

6 The ONE Negotiation Metamodel

The ONE DSL was created in order to allow users to design any type of negotiation process as simply as possible, while still being precise enough to allow the process to be machine executable: the language has been inspired by FIPA protocol [15] and by the OMG Negotiation Facility [16]. The metamodel uses essentially a subset of UML [17], as no existing UML profile met our requirements.

The metamodel has two sections; the *Information model* which describes *what* is being negotiated, and the *Negotiation Protocol* , which describes *how* an agreement is

negotiated. The Information Model needs to be very flexible to support almost any business domain, so it has a simple structure with attributes and issues that can be negotiated (changed by negotiators). The ONE DSL does not contain any standard agreement or contract types; it is up to users to define these kind of models.

From examining many existing negotiation methods and systems, a number of features common to virtually all negotiation processes were determined. An example of one of these concepts is that to reach an agreement a user must make an offer and that offer must be accepted by another user. These concepts were formalized into the negotiation protocol section of the ONE Negotiation metamodel.

Negotiations in ONE do not require a neutral third party to host or coordinate a negotiation, the parties communicate directly with one another. This brought us to the approach of using independent state diagrams for each user, with the transitions between states being triggered by the sending or receipt of messages, or other stimulus. Each Negotiation State machine have a number of SuperStages that a process would go through on the way to reaching an Agreement. The SuperStages and the basic negotiation protocol were created to help ensure that every negotiation achieved certain legal norms inherent in contract creation [2].

User defined Stages represent specific steps in the negotiation process, such as when a user is creating an offer or other message, waiting to respond to an offer, or waiting after a sealed bid has been made.

As part of the metamodel we defined a number of message types such as: Offers, Agreements, Admission Requests, Information and Notification messages. These Message types have precise meanings but can be extended to support different kind of negotiation models.

In the metamodel there area also Actions that are triggered at certain steps in the negotiation protocol; such as when a user enters or leaves a specific stage. In designing the metamodel a difficult decision to make was the level of granularity for predefined actions. The more functionality that was included in the predefined actions, the less flexible the system becomes. With less functionality the metamodel provides no benefit over existing modelling languages. The decision was to define a few simple common actions, and to define the more complex behavior in action Scripts: small sections of Bean shell code[9], embedded in the models. This approach allows a user to write custom rules like sorting or filtering messages.

6.1 Transformation and Execution

MDA dictates that negotiation models are to be executable. This could have been achieved by developing a state machine engine to step through the ONE stages. However an existing open source state machine engine (JBPM) was found to have all of the core features required for ONE. It was demonstrated that by transforming models from the ONE DSL into JBPM compatible model (JPDL), the development effort could have been greatly reduced.

The process of transforming negotiation models from the ONE metamodel into JPDL was not a straightforward task, since not every feature in ONE metamodel had an equivalent in JPDL. A multiple layer transformation process was developed that supported all the features of the ONE DSL [2]. This process uses a number of ATL rules (Atlas Transformation Language [18]) with XSLT also used for formatting, see Figure 3 below.

The greatest benefit of transforming the models into an existing process model language was that it allowed us to rapidly prototype negotiation processes in a stable state machine environment and use a number of features that would not have been possible to implement with the available resources.

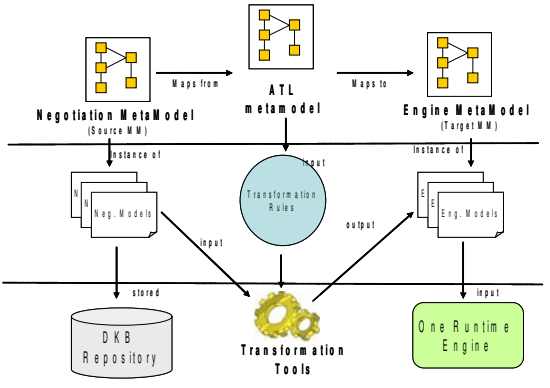

Fig. 3. Models and metamodels

The transformation layer would allow different implementations to use the same negotiation model, with Web Services providing the standard for messaging. As such, it might be possible to replace components of the current ONE platform with new components and remain compatible with the Architecture.

7 The ONE Web Portal and the ONE Node

In the ONE Web Portal participants can create, join and negotiate under the rules specified by the model. There is also a user workspace, where contacts, and past negotiations are stored managed.

There can be several Web Portals in the Internet each with its own different IP address and any of these can be used to log in and to enter the community. The ONE Web Portal is a stateless application server; all the storage of information are managed via the Distributed Knowledge Base only. The One node is the technical infrastructure (see Figure 4), is built on top of the Spring [19] service container. All the functional components are then plugged on top and some of them make also use of the Portal that make use of an Open Source implementation [20] of the Portlet specification [21].

It is assumed that participants have a preference for a trusted node through which they access a ONE node that will be responsible for the traffic and for the activities of their users, as such there is a cross trusted relation that will be properly implemented by the Trust and Reputation component [22]. Users who intend to join a negotiation are to be redirected, together with their trusted/owning node, to the central node were the negotiation will be run. It is important to enforce the concept that a node becomes "central" only for the duration of the negotiation: it is not an fixed property of a node.

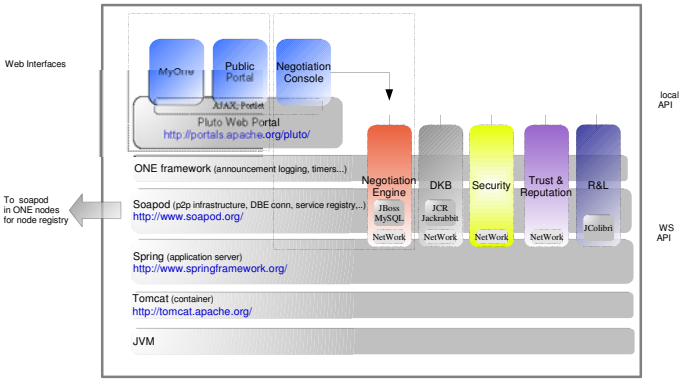

Fig. 4. One node, layers of technologies

Since negotiations are supposed to last for several days or weeks it is hence required to provide a fail-over mechanism: during the negotiation; the Engine will take care of duplicating the negotiation state in a backup node.

8 Recommender and Learner

A node, in addition, could host the connections of many users who could participate to different negotiations and hence the Negotiation Areas do overlap and intersect: a node can support many users in the role of owner and many users in the role of participants in the same or in different nodes (Figure 1 represents an ideal scenario where two separate Arenas do not share any Node).

ONE also provides a recommendation framework and an intelligent learner system that given the negotiation models and the history of past anonymous executions is able to provide feedbacks and infer suggestions to the participants.

The Recommender System [23] that plays a key role in supporting users during the negotiation processes, it recommends how to reply or respond in a negotiation. It is aimed at implementing a learning agent, able to suggest to the user a negotiation strategy. This has been achieved by observing, in the right context, the user behavior and the actions of other partners in past negotiations. The strategy learning algorithms is based on the observation of real interactions that happens in the Execution Environment.

9 Summary and Conclusions

The consortium has successfully adopted the MDA that allowed the solution to functionally scale, to have software components which are not bound to any specific negotiation and are adaptable to custom needs. This provided benefits in the daily organization of the work, easing the interoperability among partners and most of software components that have been generated rather then coded by hand. The project (released of open source [24]) can already be modelled, published and executed.

References

1. Many, Model Driven Architecture
2. Boudjemil, Z., Finnegan, J.: D3.1 – Languages, Models and Agreements (2008)
3. Ferronato Pierfranco, Architecture for Digital Ecosystems: Beyond ServiceOriented Architecture (2007)
4. Many, Digital Business Ecosystem Project (2004)
5. Cavalcance, P., Ferronato, P.: Del 3.3 Set of Software Design Tools (2008)
6. Many, Graphical Editing Framework
7. Many, Eclipse Graphical Modeling Framework
8. Many, Provides classes that are fundamental to the design of the Java programming language
9. Pat Niemeyer, Bean Shell
10. Many, MOF 2.0/XMI Mapping, Version 2.1.1 (2007)
11. Many, OMG MDA Guide (2001)
12. Many, Folksomy
13. Pierfranco Ferronato, D2.1 – Functional Architecture (2008)
14. Stanoevska-Slabeva, K., Faust, R., Hoyer, V.: D1.2 – Software Requirements and Use Cases (2007)
15. Manyù, FIPA Interaction Protocols
16. Many, OMG Negotiation Facility (2002)
17. Many, UML
18. Many, Atlas transformation language
19. Many, The Spring Framework
20. Many, Reference Implementation of the Java Portlet Specfication
21. Java Community Process, JSR 168: Portlet Specification
22. Ion, M., Danzi, A.: D5.2 Design of a peertopeer reputation model (2007)
23. Avesani, P., Malossini, A., Penserini, L., Serra, A.: D4.1 Interaction Design (2007)
24. Many, ONE code repository