

An MDA based environment for modelling executable business Negotiations with a DSL

Pierfranco Ferronato Dr¹

¹ Chief Architect, Soluta.Net, via Edificio 1, Caselle d'Altivole,
31030 Treviso, Italy
pferronato@soluta.net

Abstract. ONE is a 6th FP EU funded project, it consists of an environment that supports the modelling, creation and execution of negotiations using an MDAtm approach. A Domain Specific Language has been defined in ECORE and used to generate the editor with the GMF Eclipse framework, the XMI models are executed by JBoss and AJAX web interfaces are created on the fly to coordinate the participants. In addition the project provides a Recommender and Leader component which are metamodel driven. This paper describes the strategies, the architecture, the process used in the development of the project, with focus on the MDA key decisions they drove the architecture. It describes why MDA was the founding principle that drives the entire architecture, this allows the solution to functionally scale, to have software components that are not bind to any specific negotiation and are adaptable to custom need. The consortium has successfully and widely adopted the MDA Eclipse framework as a set of specifications and tools, this has provided also benefit in the daily organization of the work easing the interoperability issues among partners and software components. The highly scalable architecture allows the infrastructure and the model to grow at the pace of the community as it needs to. The underlying DSL, the open source infrastructure and the MDA will help creating an open library of computable negotiation models that can be modified, shared and executed across a community in different domains.

Keywords: Negotiations, domain specific language, MDA, metamodeling, open source

1 Introduction

Supporting negotiations in a web based environment is a task that many tools and vendors already offers. These have also some level of customization in the tender/auction specification; for example the price and the duration is customizable so do 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 of 5 minutes after the receive of

an offer, or adding new models like “reverse auction” or generic tenders, without the need to undergo a rather long set of changes in the code itself requires to re-think the entire architecture of such environment.

The project consist of two main environments: the Factory where model are defined, and the Runtime environment where the models are instantiated and executed; Figure 1 below depicts this design and it also indicates that the execution is further defined in two phases: the Setup where negotiation are configured with data and the Execution where they are started and users actually participant to arrive at an agreement.

The Open Negotiation Environment project is a research project funded by the Information Society and Media DG ICT for Enterprise Networking Unit in the 6th Framework Program.

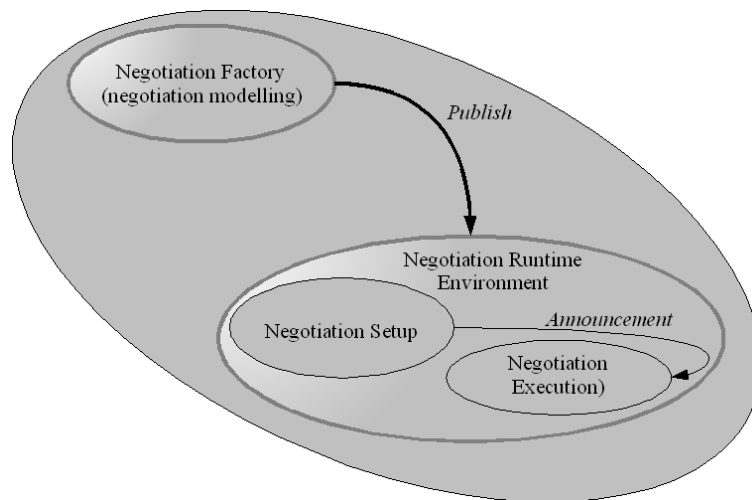


Figure 1. Environments in ONE

2 The MDA Approach

In order to support new negotiation models without undergoing the classical develop, compile test and software deployment life cycle, ONE consortium has decided to leverage the Model Driven Architecture(MDA [10]). ONE provides a tool where negotiation models can be created in a visual environment specifying the negotiation protocol, and the underlying information model. The protocol is modelled using a UML [19] state diagram notation where messages, flow and conditions are defined: with this model it's possible for example to model First Price Sealed-bid auction or any other custom negotiations.

The models defined in the editor at abstraction level M1[FRANKEL] are instances of the negotiation modelling language, called the One Negotiation Meta model,

defined in the consortium[ONM] that represent the core expression capabilities of the models. The language resides at M2 abstraction level.

The ONE consortium has defined a domain specific language (DSL) for describing negotiations (a DSL is hence 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 (parts of) applications by instantiating these models and generating the code, documentation, and other artefacts.

As stated by the OMG: “The MDA defines an approach to IT system specifications that separates the specification of system functionality from the specification of the implementation of that functionality, on a specific technology platform. To this end, the MDA defines an architecture for models that provide a set of guidelines for structuring specifications expressed as models.” [11].

Models are processed by the run-time environment which allows users to tailor them to their needs by specifying the products or the services, the start and the end date, and in general to given values to 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. The Setup phase essentially creates an instance (M0 respect to the MDA abstraction level) that can be finally executed in a web collaborative environment under the control of the execution engine.

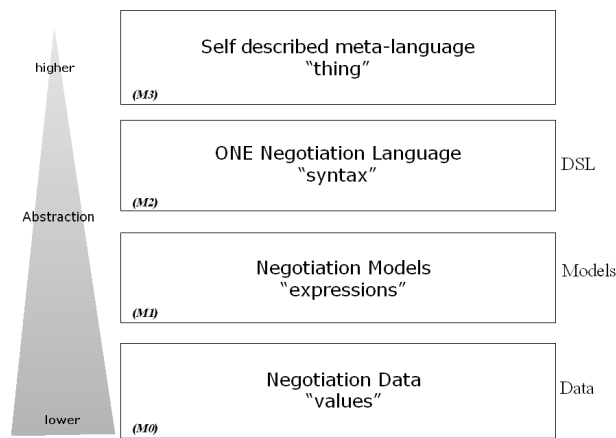


Figure 2: ONE simplified standard stack

Figure 2 above depicts the abstraction layers: M3 is the language used to create the negotiation language, in our case it is ECORE. Level M2 is the One Modelling Language itself, inspired by FIPA protocol and by the OMG Negotiation Facility [14]. M1 are the models created with the Editor, and M0 are the negotiations themselves enriched with data and executed.

3 The Implementation Strategy

This project has been conceived for an open community of users and as a such it has adopted an architectural style that spouses the EOA approach (Ecosystem Oriented Architecture) [4]. The main features in this sense are:

- decentralization of the model repository, p2p based;
- evolutionary repository: the life-cycle of the models is regulated by the community;
- use of folksonomy [5] to tag models and information model entities;
- community driven trust and reputation model;
- a negotiation topology that reflects the star based model of real life negotiations;
- scalable: the infrastructure is able to scale at runtime as the community needs to;
- use of open source frameworks.

The strategy when defining the Architecture has been to reflect the topological model found in real life business negotiations and in ecosystems.

As defined in the DBE project [1] business ecosystems are based on a peer to peer model, where each participant plays both the role of a server and of a client in a point to point communication protocol, with the absence of a central control point. This topological model well reflects the nature of social networks: they self sustain and are regulated by a mechanism where each participant offers its resource to the community: this approach enforces the sense of being part of a living organism. In particular it's key the absence of a central regulation node, the absence, technically speaking, of a single point of failure. From the business perspective this avoids the "big brother" syndrome¹.

On the other hand when in the community a negotiation is initiated, the topology is star based; the owner represents the central of the star, while the participants are the edges of the star itself: it's a plain client-server approach.

The ONE architecture has reflected this approach in the architecture. The run-time environment is based on a p2p topology: there is no single point of failure, participants offer their resources to help persisting the knowledge, to execute negotiations or to host a Web Portal.

The execution environment on the other hand is star based: the owner of a negotiation is hosting the negotiations itself, while the participants are using their nodes to access the negotiation. Figure 3 below depicts this approach: the dotted circles represent a p2p network, the upper one of all the nodes that will be sometime either a client or a server in a negotiation, the smaller in the lower part represents the p2p network of the Distributed Knowledge Base. The left and right clouds represent two running negotiations: 'A' and 'B'. Each cloud is called in ONE terminology a Negotiation Arena and their life cycle is regulated by each negotiation: they appear and disappear in sync with the negotiations. The runtime environment is a Web based application, the Distributed Knowledge Base is a Web Service based application.

¹ The fears of many people concerning computers and privacy: "Big Brother", the government, or someone else that can in theory assemble a great deal of information people and use it for its, or third parties, benefits

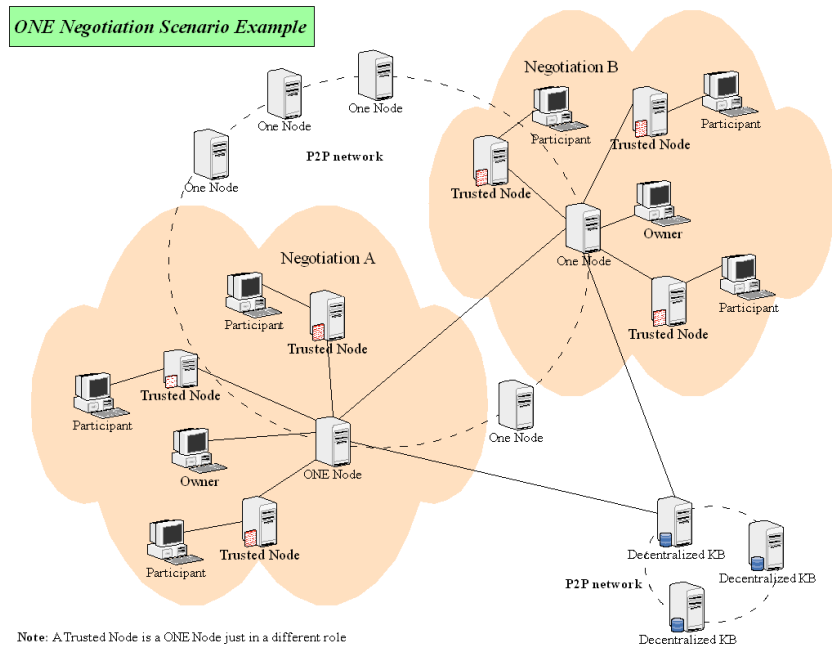


Figure 3: ONE topological model

The consortium has made use of sophisticated topology models and technologies in order to avoid the architecture to rely on a single point of failure. This means that the Runtime environment will not allow any single entity to gain “control” over the negotiation process: there is not a single element in the architecture where data and messages are stored or where they transit, as such there is not way to tamper a server in order to alter the behavior of an entire negotiation for the benefit of an organization or a single user.

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.

There are essentially two reasons for introducing a peer to peer topology in ONE infrastructure: one is technical and another is to provide an immediate perceivable benefit to the user.

From the technical perspective the p2p topology is a way to replicate information and access points. This gives a real economical benefit because no one in the community is asked to provide a important investment in terms of resources like database and application servers; no one is in charge of supporting a possible increase of the workload by retrofitting new hardware or bandwidth. With a proper p2p topology the resources will grow as the community needs to since the each participant will sustain the grown with it's own resources: the set of nodes will behave like a single unique organism. In particular data are safely replicated across nodes to guarantee fail safe mechanism without special hardware, in addition, by a proper p2p strategy, it's possible to avoid the typical bottleneck in portals, since connections are

routed to different nodes: scalability is key since it does not requires huge investment to start a community.

From the user perspective the adoption of a p2p architecture is essentially invisible from the usability perspective but relevant when explained that all the models and data provided during usage are stored across the entire community in a redundant and fail safe infrastructure. This bring functionally two benefits: one is the awareness that models and data are not owned by a single organization, the other is that in this way there is no “big brother syndrome”. This one of the major concerns in current Internet evolution: it was Microsoft for operative systems, now it's Google in the Internet age.

In any case these two aspects are related and represent the two faces of the same coin: this is how negotiations happened in the real world. The application of a p2p topology is a natural consequence of the implementation of a decentralized community. It's since the late 60's that Conway has postulated a law² that takes it's name that states that “Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations” on other words “Any piece of software reflects the organizational structure that produced it”. It'd have been a gross error to adopt a centralized implementation in a context where the p2p topology is in the essence of the problem ONE is addressing.

The Factory environment on the other hand is where models are created: a single self contained system devoted to the creation of models, as such the topology is a single node where the only connection is with the Distributed Knowledge Base to store and retrieve models.

The factory is a rich client application, even if in the early stage of the project a thin client technology based on SVG was evaluated.

4 The Factory Environment

The Factory environment is implemented by a MDA based modelling editor, we envisioned three options to create it:

- customize an open source UML editor (e.g. Argo UML);
- develop it from scratch;
- generate it from a meta model.

In order to have a quicker development life-cycle and to be more consistent with the MDA approach, it was decided to generate, as much as possible, a graphical modelling editor from the ONE meta modelling language.

Based also on previous projects experiences, the Eclipse framework (<http://www.eclipse.org>) has been considered as the primary sources for MDA related frameworks for the construction of the Negotiation Model Editor.

Since UML is a general purpose language and too generic for the needs of ONE, a DSL had to be created ad-hoc. There are essentially three ways in MDA to specify a language:

1. by using a subset of UML;

² Melvin E. Conway“How Do Committees Invent?” Copyright 1968, F. D. Thompson Publications, Inc.Reprinted by permission of Datamation magazine, where it appeared April, 1968..

2. by using UML Profiles;
3. by creating a meta model from scratch using a modelling language (meta meta-model), in the same way UML is defined; hence using a meta language (M3).

In order to have the best flexibility, it was decided to adopt the third approach. Two kind of reference languages can be used when defining a meta model:

1. MOF;
2. Ecore (part of EMF).

MOF is generally considered more formal and, to some extent, academic. As a matter of fact, there are not many production based systems which adopt it; the most relevant critic is that it is too complex at the time being.

EMF has been conceived in the Eclipse community and it is more essential and close to the pragmatic needs of developers. For this reason it has been adopted as part of the Technical architecture of ONE, for encoding the meta model.

To support the MDA approach of the software life cycle in the construction of the editor, the Eclipse EMF [3] (Eclipse Modelling Framework), Eclipse GEF [6] (Graphical Eclipse Framework) and Eclipse GMF [7] (Graphic Modelling Framework) plug-ins will be used, their dependency is shown in Figure 4 below.

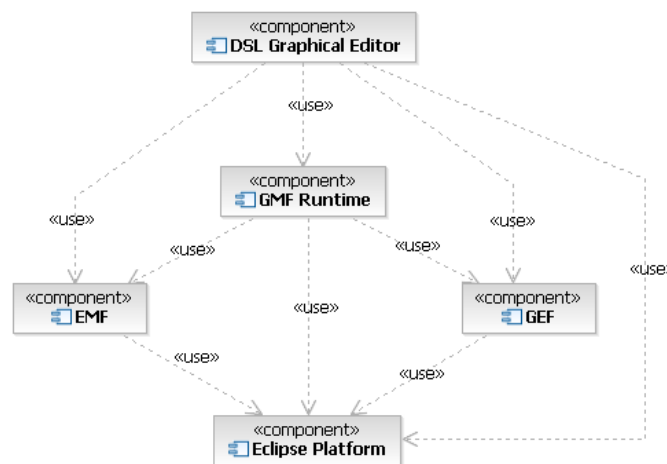


Figure 4: Languages dependencies in the Model Editor

Inside those frameworks, the input ONE Ecore meta model is used in a series of transformations (see Figure 5 below) in order to obtain other intermediate models and then, the code generation will produce the corresponding graphical editor, which could also be customized in order to fit special requirements which can not be expressed in the input meta model, for example for creating wizards for adding Variables, Protocol Rules, Criteria, Action Language support, access to the repository etc.

The process is iterative (the meta model might changes, especially in the early phases of the project) so it is important to ensure that the customizations made to the generated editor are preserved between different iterations (the complete description

of the designing and development of the Negotiation Model Editor is the subject of a dedicated deliverable of the ONE project [2]).

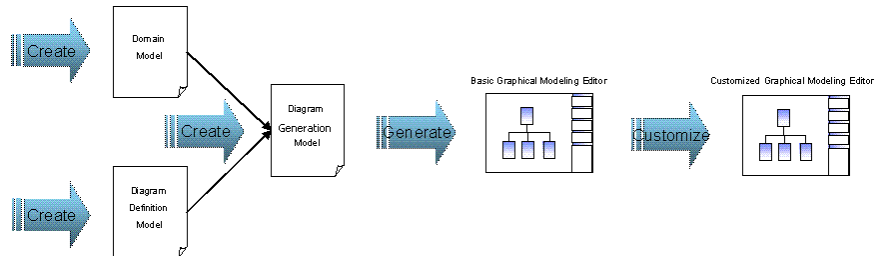


Figure 5: Model Editor development process streamlined

ONE of the key feature of adopting the MDA approach in the Negotiation Model Editor tool is that it depends on the Negotiation meta model and as a consequence the tool just need only to be updated/regenerated if the Negotiation meta model changes. This is a benefit, since changing just the model assures a tight control over the changes, the code generation is automatic and the previous customizations of the editor will be preserved by the framework across regenerations. Anyway during the production phase of ONE, the meta model is not supposed to change very often because not all the components, especially those in the Execution Environment are not generated from the ONE meta model.

In case the ONE meta model is changed, not only the Editor must be updated to generate new compliant models, but also all the other components that depend on it such as the Setup and the Negotiation Engine. In addition, the environment shall be able to process both kind of models: the one made with the previous version of the meta model and the new one: this is not a trivial effort. ONE is enabling this scenario by storing the version of the meta model used by models and by implementing a Distributed Knowledge Base which is meta models agnostic. Addictive changes in the negotiation meta models during the production phase are to be implemented with caution because a significant effort is required: a destructive changes on the other hand like removing or renaming an entity will jeopardize the entire architecture.

The resulting editor will allow users to create negotiation models which will be executed by an engine component. In line with the MDA approach, these models should not be specific to any underlying technical platform (they are a kind of PIM's, Platform Independent Models). The editor is a rich client application that does not runs inside a web platform: before being run, the models can be stored locally in the file system or in the remote public distributed knowledge base (DKB); the models can be reused by other users to fit their needs.

The negotiation Setup and the Engine component will also benefit from the PIM nature of the negotiation models, since they will be capable of running it on different platforms and to execute every model produced by the editor, no matter in which platform it is going to run. The ONE model does not make assumption about the Execution Engine, Jboss or Ilog or others, not even of course it assumes the participants of the negotiations are humans using Web based interface or software components using Web Services. As a matter of fact the Runtime environment could

be executed as a PHP, Java or .Net application; ONE is supporting a Java implementation using a thin client user interface.

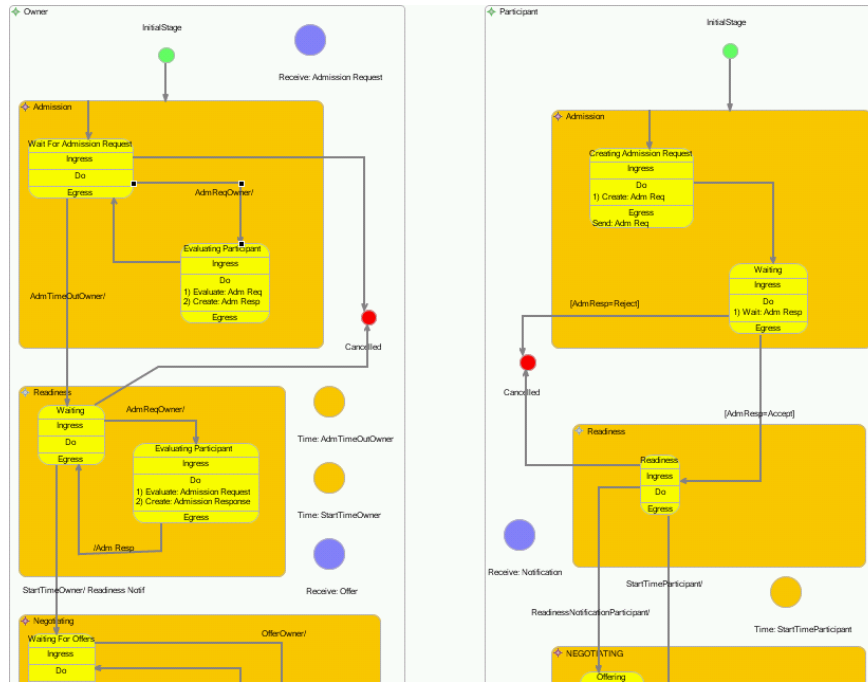


Figure 6: Negotiation Editor, two state machines

The notation used is based on UML State Diagrams. In ONE it was decided to draw two state diagrams, one represents the Owner (left side) the other the Participant (right side).

Each state diagram has 4 Super Stages: Admission, Readiness, Negotiation and Acceptance that regulate the way negotiation are managed at runtime. Each Super Stage has one or more Stages (states) connected with Transitions that are regulated by the usual Trigger, Action and Condition.

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 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, it's straightforward to adopt a subset of the Java Lang[JLAN] in order to avoid useless transformations between the Editor and the Engine.

The model editor provides an Action Script editor based on the open source project Bean Shell [BSHELL]: a Java interpreter (Java script) augmented with the ability to access global negotiation variables as defined in the meta model like list of sent

messages, list of participants, start time of the negotiations, list of receive messages, date and time and the model specific attributes.

In order to avoid reading sensitive information or gaining control of the negotiation, it was decided not to enable the Participant side to define an Action Script.

5 The Runtime Environment

The Runtime is the Web base environment where negotiations' instances are created from models and executed; in addition this environment provides a set of side functionalities 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 [13]: it can be said that, with the exception of the Model Editor, the run time provides all the resources needed to expose all the functionalities as specified in Deliverable "D2.1 Functional Architecture". 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 (Figure 2 "ONE simplified standard stack" in page 3).

As specified in the project deliverable D2.1, the Runtime environment supports the ONE Web Portal and the MyONE. While the Web Portal is publicly accessible, the MyONE requires authentication; in general is used by users that were involved in negotiations either as a participant or as an owner.

The most relevant functionalities of the ONE Web Portal are:

- to allow users to configure a negotiation in a section of the Portal which is called the "Setup";
- to execute a negotiation in a section of the Portal which is called the "Negotiation Console";
- the workspace, where contacts, profile, past negotiations are managed.

Thanks to the Runtime architecture, the ONE Web Site is not implemented as a centralized server, even if from an outside perspective it looks like a classical web site like Ebay[EBAY]: the information (like models, negotiations, history, contacts...) are stored in the DKB (Distributed Knowledge Base) that adopts a decentralized peer-to-peer network of nodes.

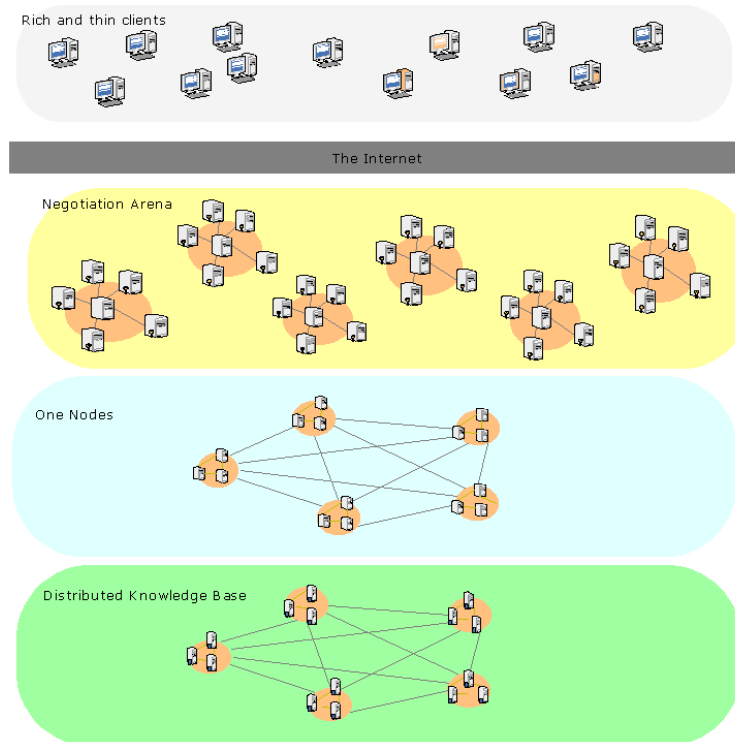


Figure 7: Multi-topology architecture

The implementation is not straightforward because the topology of a negotiation environment is multidimensional as the Figure 7 above shows. In the ONE architecture we have defined three p2p architectures each -potentially- with a different topology:

1. Negotiation Arena: essentially a p2p networks of star based networks;
2. One Nodes: a p2p network;
3. DKB: another p2p network.

As it was said, the ONE Runtime components will not reside in a single central server but will make use of participants' resources as in a pure peer-to-peer network: the community owns both the environment as well as the data that makes it valuable. There can be several ONE 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 hence a stateless application server, it does not own a local storage of information; it stores and retrieve information from the Distributed Knowledge Base.

Nevertheless from the usability perspective there is not much difference respect to a traditional Auction or Negotiation web site. This is a unique feature for this type of system that represents an advancement in the state of the art; this strategy greatly hardens the architecture and the implementation.

6 The Negotiation Arena

There are essentially three kinds of topologies in ONE (reference Figure 7 above):

1. ONE Nodes: the network of ONE Nodes which represent the Community;
2. DKB: the trusted network of DBK nodes which, in relation to a Community, provides the storage support;
3. Negotiation Arena: the network that represents all the participants when executing a negotiation. With respect to the other two topologies, the Negotiation Arena adopts a centralized architecture, where -for the sole duration of a negotiation- the owner hosts the central node of a star based network topology where participants are the vertexes that are connected through their own ONE Node.

The One node is technically shown in Figure 8 below, it's built on top of the Spring [21] service container, where a ONE specific framework has been developed. 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 [23] of the Portlet specification [22].

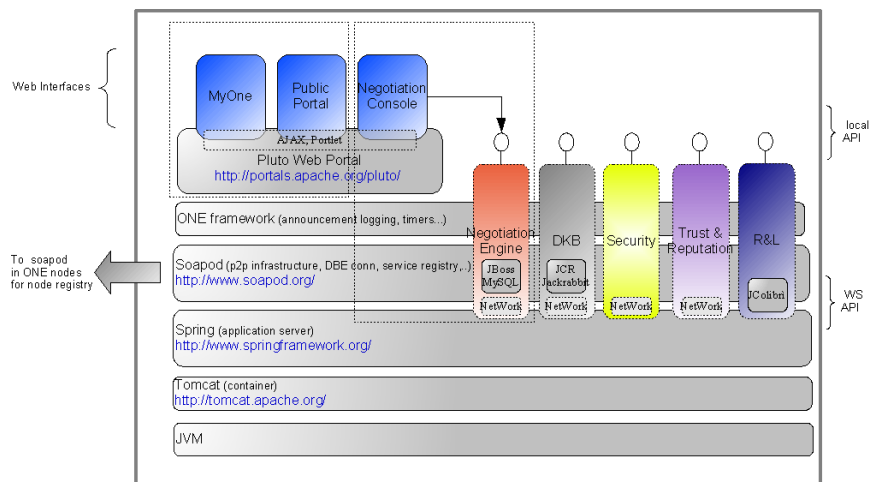


Figure 8: One node, layers of technologies

In a Negotiation Arena, a negotiation starts in the node that has been defined by the owner at setup time; the central node is assumed to remain active for the entire duration of the negotiation despite the fact that the owner is logged in or not. In this scenario, and in ONE in general, it is assumed that participants have a preference for a trusted node through which they access a ONE instance. This node, as it was introduced in the previous section, is responsible for its traffic and for the activity of their users, as such there is a cross trusted relation that will be properly implemented by the Security Component of ONE. In other words, all the nodes in ONE are the trustees of the users coming by their connection and hence the prime subject of trust is a node. As such the most important trusted entities in a Negotiation Arena are essentially the nodes of the participants and of the owner.

As a consequence, users who intend to participate to a negotiation are to be redirected, together with their trusted/owning node, to the central node where the negotiation will be hosted and run. Participants are not required to log in directly to the central node, they can use any ONE Web Portal to access the community but when “Joining” a negotiation a proper forwarding mechanism will be executed in order to connect the user to the central node via his trusted node (the information about where is the right node is specified in the negotiation instance stored the Distributed Knowledge Base). It is important to enforce the concept that a node becomes “central” only for the duration of the negotiation: this is not an absolute property of a node. When the negotiation will terminate the node will return “generic”, with no specific role.

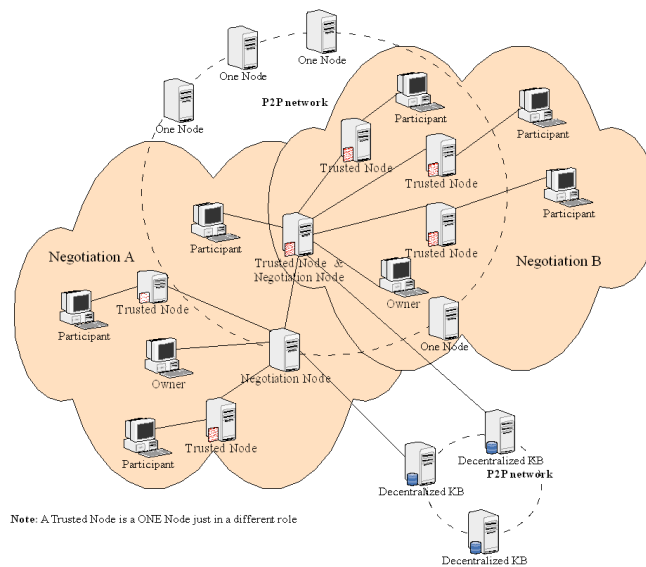


Figure 9: Overlapping Negotiation Arenas

A consequence of this approach is that a node might run, hence host, several negotiations at the same time; it could be a central node for many negotiations. 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 other nodes. In Figure 3 in page 5 is represented an ideal scenario where two separate Arenas do not share any Node and is represented also their relation with the Distributed Knowledge Base. Figure 4 above represents the overlapping scenario when there is a node which is both hosting a negotiation and is the trusted entry node of a different one.

Another alternate scenario which is supported is the case where an user wishes to participate to a negotiation without having a trusted node, in this case the architecture will allow the user to join the negotiation using the owner's central node; this assumes that the owner when setting up the negotiation has allowed this feature. The pros will

be a probable increase in the number of participants, with the cons of the risk of accepting non trusted participants.

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. Whenever a central node is unavailable the “Join” of a negotiation will forward the request to the backup node. This and other scenarios are to be implemented by the Execution Engine.

7 Recommender and Learner

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 [18] that plays a key role in supporting users during the negotiation processes, it recommends on how to reply or respond in a negotiation. It's 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 behaviour and the actions of other partners in past negotiations.

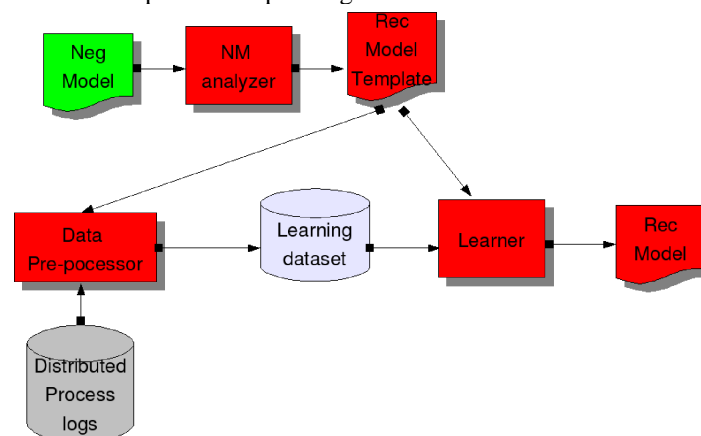


Figure 10: The design of the learner component (from D4.1 Recommender and Learner [18])

The recommender system might also automate certain type of actions, i.e., to reply on behalf of the participant. The strategy learning algorithms will be based on the observation and storage of real interactions that happens in the Execution Environment.

The Learner is a component that infers Recommendation Models from logs of past negotiations, it is used by the Recommender. Figure 10 above shows a sketch of the internals of the Learner component.

8 Summary and Conclusions

MDA is the founding principle that drives the entire architecture, this allows the solution to functionally scale, to have software components that are not bind to any specific negotiation and are adaptable to custom need. The consortium has successfully and widely adopted the MDA Eclipse framework as a set of specifications and tools, this has provided also benefit in the daily organization of the work easing the interoperability issues among partners and software components.

The architecture adopts a decentralized approach to avoid having a single point of failure) this would jeopardize the adoption of the platform. The architecture is configured in order to behave like a self supporting organism that grow over time: nodes of participant organizations are attached to each other sharing resources.

This decision has put severe challenges in the project that required to rethink the model repository, the service registry so do the topology of the network. The technologies for the decentralization are built around p2p protocols.

The project, after about 18 months since its kick-off, is showing tangible results: negotiations models can already be modelled, published and executed.

References

1. Digital Business Ecosystem Project, www.digital-ecosystem.org
2. P. Cavalcante, P. Ferronato: Del3.3 Set of Software Design Tools. A ONE Deliverable(2008)
3. Eclipse Modeling Framework, <http://www.eclipse.org/modeling/emf/>
4. P.Ferronato: Architecture for Digital Ecosystems: Beyond Service-Oriented Architecture. Cutter Executive Summaries - Enterprise Architecture (2007)
5. Folksomy, <http://en.wikipedia.org/wiki/Folksomy>
6. Graphical Editing Framework (GEF), <http://www.eclipse.org/gef/>
7. Eclipse Graphical Modeling Framework (GMF), <http://www.eclipse.org/modeling/gmf/>
8. Mihaela Ion, Andrea Danzi, CreateNet: WP5: Identity Management and Reputation Framework for Trusted Negotiation, D5.2- Design of a peer-to-peer reputation model (2007)
9. Provides classes that are fundamental to the design of the Java programming language, <http://java.sun.com/j2se/1.4.2/docs/api/java/lang/package-summary.html>
10. Model Driven Architecture, www.omg.org/mda
11. OMG MDA Guide, <http://www.omg.org/docs/omg/03-06-01.pdf>
12. Meta Object facility, <http://www.omg.org/mof>
13. K. Stanoevska-Slabeva, R. Faust, V. Hoyer University of San Gallen: WP 1: Business Processes and Functional Requirements, D1.2–Software Requirements and Use Cases(2007)
14. OMG Negotiation Facility
15. Z.Boudjemil, J.Finnegan: D3.1–Languages, Models and Agreements. V1.1, Deliverable
16. P. Ferronato, SolutaNet: WP 2: Architecture, D2.1 – Functional Architecture (2008)
17. Pierfranco Ferronato, SolutaNet: WP 2: Architecture, D2.2 – Technical Architecture (2008)
18. Paolo Avesani, Andrea Malossini, Loris Penserini, Alessandro Serra, IRST – Fondazione Bruno Kessler: WP4: Recommendation and Learning, D4.1 - Interaction Design (2007)
19. UML www.omg.org/uml
20. OMG Document Number: formal/2007-12-01, “MOF 2.0/XMI Mapping”, V 2.1.1 (2007)
21. The Spring Framework, <http://springframework.org/>
22. JSRs: Java Specification Requests, JSR 168: Portlet Specification
23. Reference Implementation of the Java Portlet Specification, <http://portals.apache.org/pluto/>