MetaMeth
A Tool For Process Definition And Execution

Roberto Caico [2], Massimo Cossentino [2,3], Luca Sabatucci [1], Valeria Seidita [1], Salvatore Gaglio [1,2]

(1) DINFO - Dipartimento di Ingegneria Informatica, Università degli Studi di Palermo - Viale delle Scienze, 90128 Palermo, Italy
(2) Istituto di Calcolo delle Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche;
(3) SET - Université de Technologie Belfort-Montbéliard - 90010 Belfort cedex, France
robertocaico@libero.it; sabatucci@csai.unipa.it; cossentino@pa.icar.cnr.it; gaglio@unipa.it

I. Introduction

Nowadays, several different tools are used in Software Engineering; in this work we are mainly interested to those supporting the design phases. These are usually classified in three categories: CASE, CAME, CAPE tools. MetaMeth is a CAME and a CAPE tool at the same time.

A CAME tool is a computerized tool that supports the method engineer in the construction of its own methods that is principally based on reuse so it aids in storing the reusable part of existing methodologies (*method fragments*) and in providing the interface for a useful and easy retrieval and assembly of fragments. MetaMeth allows a method engineer to interface with a repository of method fragment in order to retrieve them for creating his own methodology.

A CASE tool as an automated tool devotes to help the designer in the software development process by providing a software support for a reliable development of activities, lowering the risk of errors and enhancing the productivity; this definition even implies activities like planning, and management, administrative and technical aspects of a project. MetaMeth allows to manage the process through the workflow engine, the agents devoted to design activities and the expert system.

II. MetaMeth – System Requirements

Our work started with the identification of the requirements for the tool to be built. The first part of these requirements refer to the CAME functionalities:

1. **Fragment Repository**: the tool supports a fragment repository collecting methods coming from several design processes; each fragment is an extension of an existing repository we built according to the work done within FIPA, extended with the proper set of expert system rules and software components (agents) used to support the specific GUIs required in the fragment; the repository of fragments already exists and we are working for integrating it in the tool. We also considered important to plan an easy extensibility of our tool supporting several different design processes, but at the moment only the fragments used to implement our own methodologies (classic and agile process) are already fully integrated in the tool.
2. **Process Definition**. Starting from the repository of method fragments the tool allows the composition of new processes. We decided to adopt a standard by OMG, the Software Process Engineering Metamodel (SPEM) for modeling our methodologies. Once the process is modelled in SPEM we use a graphical tool (JaWE) to produce its XPDL translation (XPDL is the process specification language adopted by WFMC for describing workflow processes).

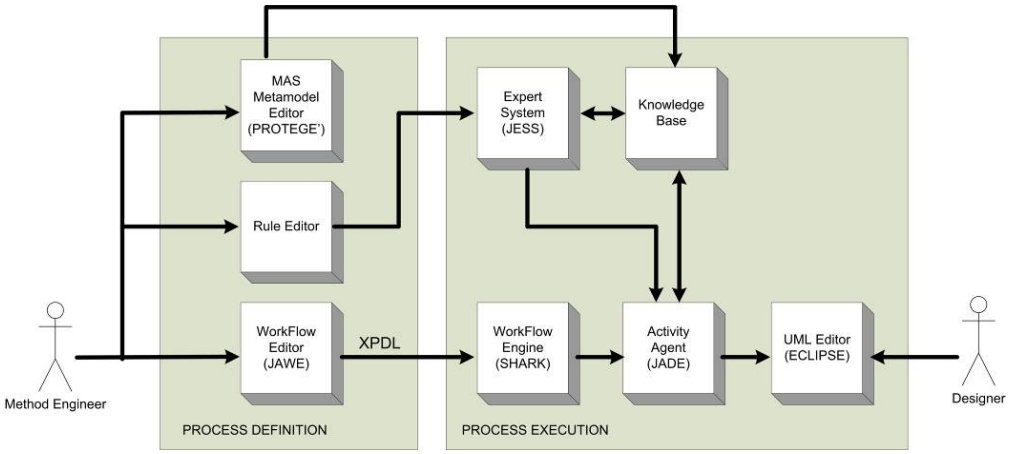

Figure 1 – Architecture of the MetaMeth application

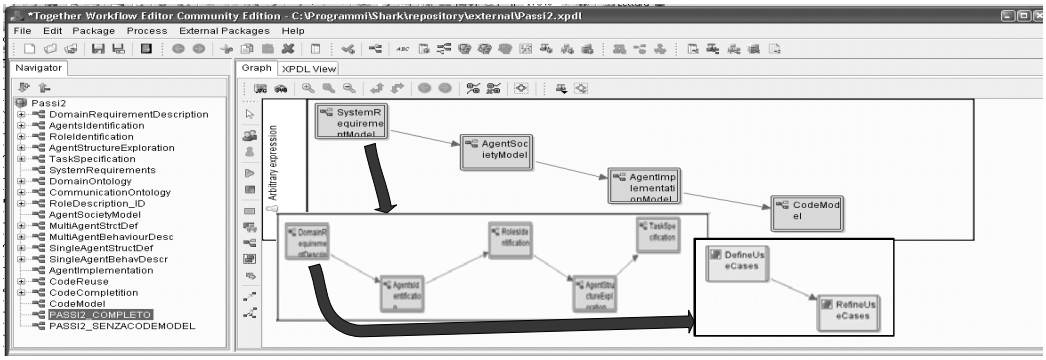

Figure 2 - A screenshot of the JaWE tool used for design the PASSI methodology

3. **Process Lifecycle.** The tool should support iterative and incremental design processes composed of several activities (eventually organized in some kind of hierarchy) with several different possible iteration paths.
4. **Notation, Syntactic and Semantic Rules**. In a methodology almost all work products have to respect three types of rules: notational, syntactic and semantic. The CAME tool have to allow the introduction of syntactic and semantic rules (expressed in a first order logic language) and to assign a graphical notation among available for working with fragments coming from different methodologies (a set of editors, developed ad-hoc for this scope, will be available).
5. **Process Roles**. During the composition of the method the method engineer may assign activities to perform to different human roles involved in the process.

The second list of requirements is related to the CASE functionalities:

1. **Process Execution**. The first requirement of this section is to instantiate a methodology built using the MetaMeth/CAME (received as XPDL specifications and a set of rules for notation, semantic and syntactic verification) and to orchestrate all the CASE services in order to design a system.
2. **Team Work**. Our tool support distributed design processes (involving several designer working on different phases at the same moment in different locations)
3. **Automatic Composition**. The tool keep in consideration dependencies among work products and include the possibility of automatically compose all (or portion of) diagrams that allow such an help.
4. **Automatic Verification**. Syntax check on notational aspects of the project and the consistency check on some design aspects like the correct instantiation of the most important elements of the MAS meta-model
5. **Reuse and Code Generation**. The tool integrates a reuse technique based on design patterns; these are collected in a repository and may be used during design. The tool has a code generator that uses an MDA approach to transform the design view in a implementation view and finally in a code view; using this approach different coding languages or implementation frameworks could be adopted. The tool supports also the production of an adequate documentation that, starting from the work products, is able of creating complex documents merging diagrams, text, tables and so on.

III. Software Architecture and Tecnologies

The prototype we developed is based on several technologies, standards and existing tools, that are independent software for enacting the process definition and the process execution; this is shown in Figure 1.

The most relevant open source components we reused in our Metameth tool are:

1. JaWE (by Enhydra) adopted as a graphical workflow editor to design the process (it exports the process using the XPDL format);
2. Shark (again by Enhydra) adopted as a workflow execution engine (it is able of reading XPDL files and also to interact with our Java-based Activity Agents); this tool ensures the design process instantiation and allows the distributed and asynchronous execution of the different activities.
3. Jade is the platform we used to develop our Activity Agents; this is the most diffused FIPA-compliant agent development platform.
4. Jess is Java-based rule engine we used to build our expert system; such a system is the 'intelligent' part of the Metameth tool; a relevant portion of its services are required by the Activity Agents that need reasoning capabilities in order to assist the designer in his duty.
5. The MAS meta-model required by the adopted methodology is depicted in form of an ontology using the tool Protégé by the Stanford University, California.
6. IBM Eclipse is the IDE we used to develop our UML editors.

IV. Example.

Now we are going to illustrate with an example the main steps of the construction of a new methodology and its enactment with the MetaMeth tool.

The scenario starts with JaWE session used to define the new methodology; this tool offers a graphical interface to model the process as a flow of sub-processes, and activities; each activity may be atomic or be decomposed in sub-activities. In Figure 2 we report a screenshot of the tool showing three different boxes each one related to a piece of the methodology at a different level of abstraction. The first box (the top one)

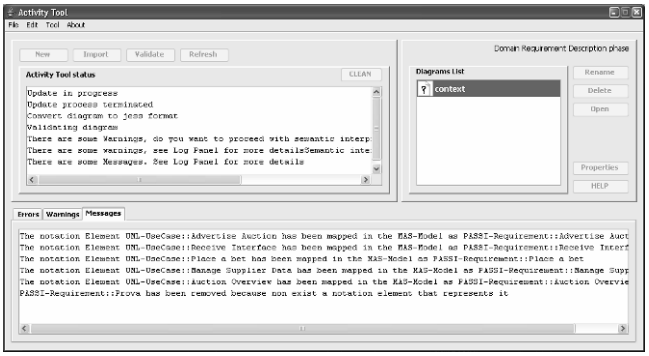

Figure 3 - An example of activity tool for the Domain Requirement Description phase of PASSI

describes the main phases of our methodology (*system requirements*, *agent society*, *agent implementation* and *code model*). In the second box there is an exploitation of the system requirement phase in its composing activities (*domain description*, *agent identification*, *role description*, *agent structure exploration* and *task specification*); finally, in the third box (the lowest one), the *domain description* activity is decomposed in two atomic operations (*define use case* and *refine use case*).

The next step in the definition of the methodology is the specification in terms of Jess rules of the semantic of the MAS meta-model elements and the (work products) composition rules of the process. This is done using Protégé to draw the ontology and a Rule editor tool we built to describe Jess rules starting from some templates.

When the methodology has been entirely described using the XPDL language (process aspects) and Jess rules (semantics and composition rules), the process administrator may instantiate it using the process execution module. This has been developed using a multi-agent system composed by:

1. A Controller agent (that is interfaced with the workflow execution engine).
2. One or more Stakeholder agents (one for each designer that uses it to accept, start, decline the activities assigned to him from the process definition). After a log in session, the user may verify his activity list and can start/ refuse or delegate an activity.
3. An Expert System agent (used to wrap the Jess engine).
4. One or more Activity agents.

When the designer chooses of performing an activity, an agent becomes responsible for coordinating all the operations related to the specific activity (also in collaboration with the Expert agent and the UML editors). Activity agents offer several services to the designer: i) auto-composition used when a work product can be automatically modified/created or updated; ii) notation interpretation, used to map notational elements (use cases, classes, activities, …) into elements of the MAS meta model (requirements, agents, behaviours, …), iii) semantic validation used to verify the semantic consistence of the whole project.

Figure 3 shows the user interface of the Activity agent associated to the domain requirement description phase of our methodology; semantic interpretation and validation have been already done with the result that use cases have been mapped to requirements. In Figure 4 the first three work products of the methodology are reported (domain requirements description, agent identification and role identification).

V. Future Works

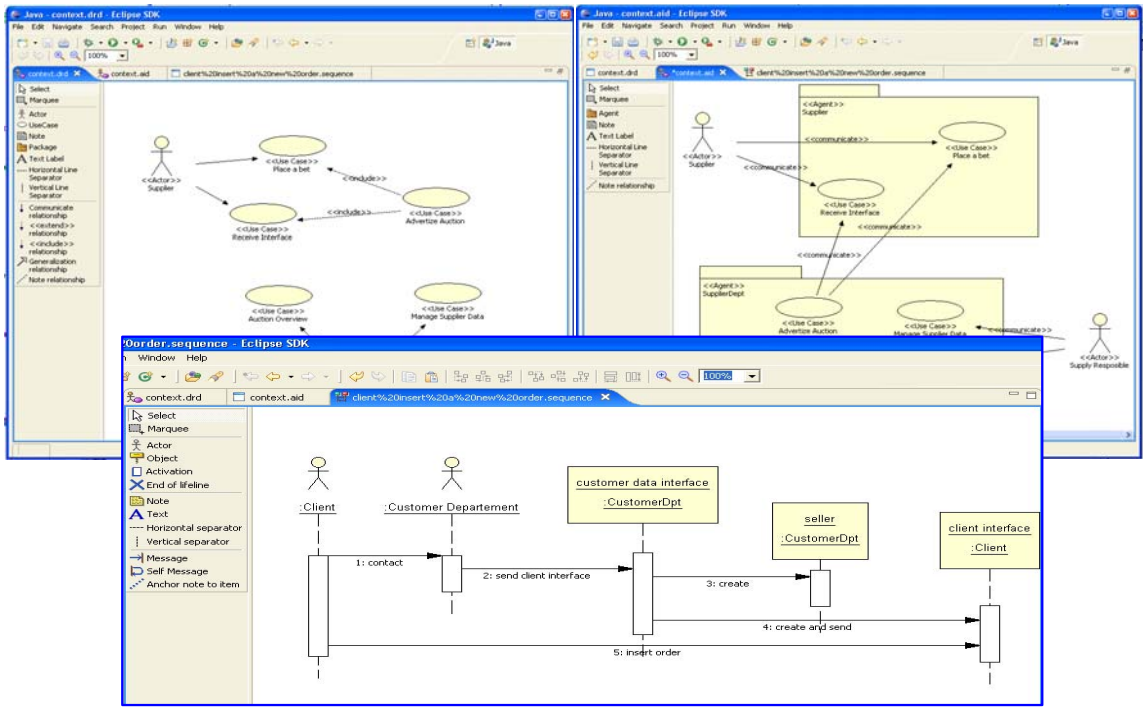

Figure 4 - Some screenshots of the UML editor developed as a plug-in for Eclipse

In the future we are going to complete this tool with more features, specifically we are going to interface it with an agent-oriented pattern reuse tool that also allows code generation for one of the most diffused agent development platforms (Jade). The production of an extensive and well-formatted documentation from the design artifact is also scheduled and will be obtained through a society of agents, each one specialized for the composition of one specific kind of document.

Another improvement, we are working on, is the population of the fragment repository, extracting methods from the existing agent-oriented methodologies.