# The ProjectIT-Studio, an integrated environment for the development of information systems

Alberto Silva, Carlos Videira, João Saraiva, David Ferreira, Rui Silva

INESC-ID, Instituto Superior Técnico,
Rua Alves Redol, nº 9 –1000-029 Lisboa, Portugal
{alberto.silva, cvideira, joao.saraiva, david.ferreira, rui.silva}@inesc-id.pt

**Abstract.** Despite the efforts made to overcome the problems associated with the development of information systems, it is still an immature activity with negative consequences in time, budget and quality. One of the main causes for this situation is the fact that many projects do not follow a structured, standard and systematic approach, like the methodologies and best practices proposed by the Software Engineering community. In this paper, we overview an innovative approach for the development of information systems, called ProjectIT, and how it is supported by an integrated set of tools, called "ProjectIT-Studio". This approach benefits from the ideas of some of the most referred initiatives, such as the Microsoft's Dynamic Systems Initiative (DSI) or OMG's Model Driven Architecture (MDA).

**Keywords:** ProjectIT, Software Requirements, MDA/MDD, CASE Tools.

## 1 Introduction

The development of information systems is a complex process usually initiated with the identification and specification of the requirements of the system to be developed, and in particular of its software components. Requirements describe what the system should do, which is obviously critical for the success of the whole development process. Several surveys and studies (such as The Chaos Report, available at http://www.standishgroup.com) have emphasized the costs and quality problems that can result from mistakes in the early phases of system development, such as inadequate, inconsistent, incomplete, or ambiguous requirements [5]. We consider in our research work that the emphasis in software development projects should be stressed in the project management, requirements engineering and design activities, and consequently the effort in production activities, like software programming, should be minimized and performed as automatically as possible.

As a result of the experience gathered from previous research and practical projects, the Information Systems Group of INESC-ID (http://gsi.inesc-id.pt/) started an initiative in the area of requirements engineering and model-driven development named ProjectIT [6]. The main results of this project are: (1) a new requirements specification language, called ProjectIT-RSL ([7] and [8]); (2) a UML profile, called XIS (short name for "eXtreme modeling Interactive Systems"), which was proposed and validated in previous work [9,10] and is now in its second version; and (3) a set of integrated tools, called ProjectIT-Studio [11], that supports the activities of the software development life cycle, from requirements specification to the application of generative programming techniques.

Interactive systems are a sub-class of information systems that provide a large number of common features and functionalities, such as user-interfaces to drive the human-machine interaction, databases to keep the involved information consistent, and role-based access control to manage end-users and related permissions [25]. ProjectIT promotes a platform-independent design for interactive systems: starting the process by specifying and validating requirements by using ProjectIT-RSL, then refining the obtained high-level model by using the XIS profile which allows the design of interactive systems at a PIM level ("platform independent model", according to the MDA terminology), and finally through model-to-code specific transformations, different platforms can be supported, such as Web, desktop or mobile specific platforms.

This paper is focused on the description of the ProjectIT-Studio, which is a modular and extensible CASE tool where we are testing our research ideas. Section 2 presents an overview of the ProjectIT initiative, in particular how the different roles involved in the project contribute to the development of information systems. Section 3 gives an overview of the set of tools that together compose the ProjectIT-Studio workbench. Sections 4 and 5 specifically address the main components of ProjectIT-Studio: the requirements tools (ProjectIT-Studio/Requirements) and the modeling and code generation tools (ProjectIT-Studio/MDD). Section 6 describes how the ProjectIT-Studio tools are combined to work together. Finally, section 7 presents related work of other initiatives and section 8 concludes this work, justifying our perception that this proposal has innovative contributions for the community.

## 2 The ProjectIT Initiative

The Information Systems Group of INESC-ID has been involved for some time in the development of research projects in the area of software engineering and the software development process, applying the results achieved in the daily projects in which it is participates. We started a project, called **ProjectIT** [6], to provide a complete software development workbench, with support for project management, requirements engineering, and analysis, design and code generation activities. This section introduces the main functional components and the proposed approach of the ProjectIT.

### 2.1 Main Functional Components

The functional architecture of the ProjectIT research program is represented in Figure 1. Among others, ProjectIT has two main components that are currently being developed, and have already reached a significant maturity level:

1. **ProjectIT-Requirements** [12] is the component of the ProjectIT architecture that deals with requirements issues. The main goal of this project is to develop a model for requirements specification, which, by raising their specification rigor, facilitates the reuse and integration with development environments driven by models. Our vision is to build a tool for writing "requirements documents", like a word

processor, and as we write, it will warn us of errors violating the requirements language rules.

2. After a validation process, the requirements information can be used as input to the **ProjectIT-MDD** component [9,13], which applies different techniques and mechanisms to accelerate and improve the software engineering activities. These techniques are aligned with model transformation techniques, and in particular with the so-called XIS UML profile [9], a set of coherent UML extensions that allows a high-level, visual modeling approach to design interactive systems.

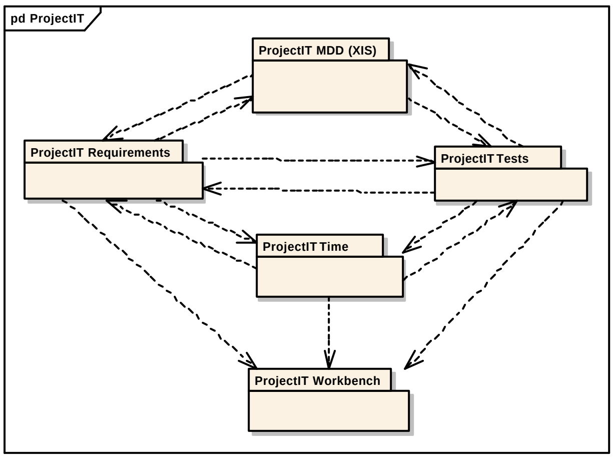


Figure 1. ProjectIT functional architecture.

## 2.2 ProjectIT Approach

The ProjectIT approach involves several tasks performed by different roles as suggested in Figure 2, in two main moments:

1. The definition of the artifacts that are the core of the ProjectIT approach, which are later used in specific projects; these tasks are usually executed by the **Architect**.
2. The use of these artifacts to develop specific projects, involving the other roles.

Generically, we can say that the application of the ProjectIT approach receives system requirements (e.g., functional, non-functional and development requirements, from different stakeholders) as its main input, and produces a set of artifacts (e.g., validated requirements, source code, configuration scripts or data scripts) as its main output.

The tasks performed by the software architect are critical to the operation of the ProjectIT approach. The Architect is responsible for the following tasks: (1) define/adapt the linguistic terms and the syntactic and semantic rules of ProjectIT-RSL which, as described in [8], can be defined on a project basis (thus providing a true support for a Domain Specific Language [8]); (2) define a suitable and easy-to-use UML profile (in the scope of interactive systems the XIS profile could be used; nevertheless, the ProjectIT approach is profile-independent, which means that different UML profiles can be used); (3) develop model transformations features, such as “Model2Model Transformation Templates” to produce new models; and finally (4) develop model-to-code template features, such as "Model2Code Transformation Templates", to produce software and documentation artifacts from models, using generative programming techniques.

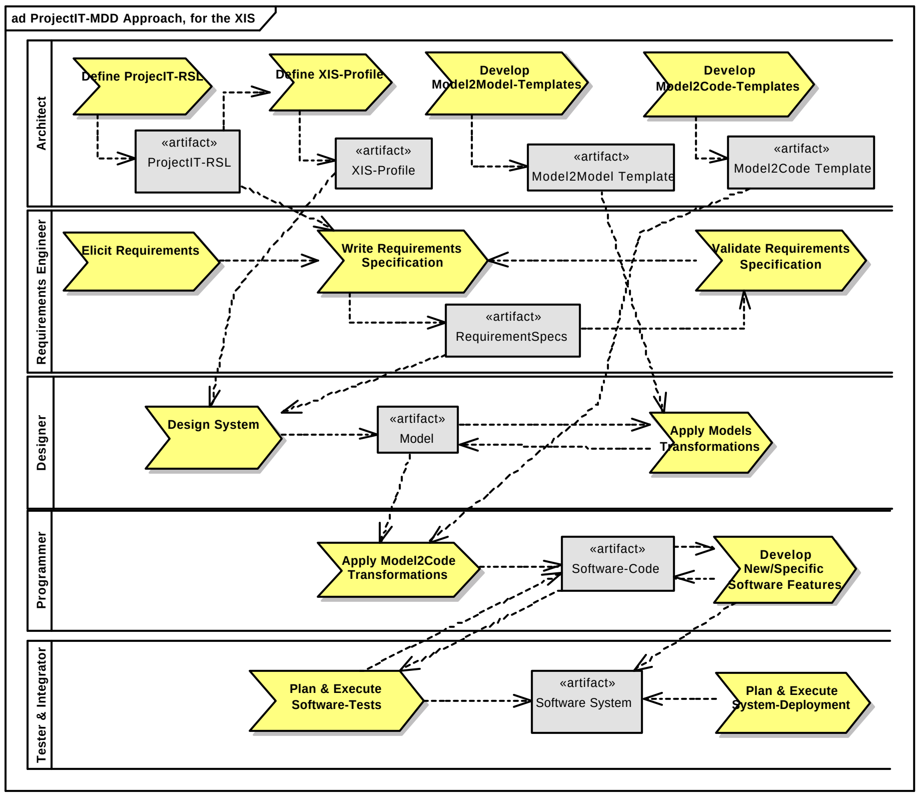

ad ProjectIT-MDD Approach, for the XIS
Architect
Requirements Engineer
Designer
Programmer
Tester & Integrator
Define ProjecIT-RSL
Define XIS-Profile
Develop Model2Model-Templates
Develop Model2Code-Templates
«artifact» ProjectIT-RSL
«artifact» XIS-Profile
«artifact» Model2Model Template
«artifact» Model2Code Template
Elicit Requirements
Write Requirements Specification
Validate Requirements Specification
«artifact» RequirementSpecs
Design System
«artifact» Model
Apply Models Transformations
Apply Model2Code Transformations
«artifact» Software-Code
Develop New/Specific Software Features
Plan & Execute Software-Tests
«artifact» Software System
Plan & Execute System-Deployment

Figure 2. Roles and tasks involved in the ProjectIT approach.

The Requirements Engineer is responsible for gathering different system requirements using well-known requirements elicitation techniques (such as meetings, interviews, JAD sessions among designers, clients, end-users and other stakeholders). These requirements are then specified in ProjectIT-RSL, and validated using a set of parsing, knowledge extraction and inference tools [14]. The ultimate goal of ProjectIT-Requirements is to allow a non-technical user to specify the system requirements by his own. The validated requirements are then passed on to the Designer, whose responsibility is to produce an integrated set of models (the "Design System" task). Still, the Designer can apply model transformations automatically according to the model-to-model transformation templates (represented by "Model2Model Template") developed previously by the Architect. This task can be useful in certain situations in order to simplify or accelerate the design task. The correctness and quality of the produced models are essential to obtain good results in the subsequent tasks.

After the Designer's intervention, the Programmer applies model-to-code transformations ("Model2Code Template"), which means the Programmer applies generative code techniques to models, based on the templates provided by the architect. Because it is not possible to capture and design all the system requirements (for instance, business rules or non-functional requirements), at least using the XIS profile, Programmer intervention is still required. Consequently, programmers are required to produce specific components, typically helper source code, such as facades, adapters, controllers and business logic. These activities are suggested in Figure 2 by the "Develop New/Specific Software Features" task.

Finally, the intervention of Testers and Integrators is necessary to prepare and perform different tests in order to guarantee system quality. These activities are suggested in Figure 2 by the "Plan & Execute Software-Tests" and "Plan & Execute System-Deployment" tasks.

## 3 The ProjectIT-Studio Overview

The focus of this paper is to discuss the productivity tool of this initiative, ProjectIT-Studio. This section overviews its basic architecture and main usage scenarios. The next two sections analyze some relevant details and its main features.

ProjectIT-Studio is an integrated environment that supports the central tasks of the software development life cycle, such as requirements specification, architecture definition, and system design. In addition, and because it promotes productivity, ProjectIT-Studio provides innovative features such as requirements-to-models, models-to-models, models-to-code transformation techniques; template managing; and UML profiles definitions.

The research follows two parallel lines, one more oriented towards the requirements issues, the other more directed to the study of model driven development techniques. The requirements line of research has developed an initial prototype [15] using the Visual Studio .NET extensibility mechanisms. On the other hand, the XIS project, developed in preliminary versions, applied model driven development techniques, particularly at the level of model-to-code transformations [9]. Besides minor issues, these versions showed that we lacked a common base platform for all our research: the work-around for this issue was accomplished with the port of the Eclipse platform to the .NET environment, called Eclipse.NET (at http://www.sourceforge.net/projects/eclipsedotnet ) [11,13].

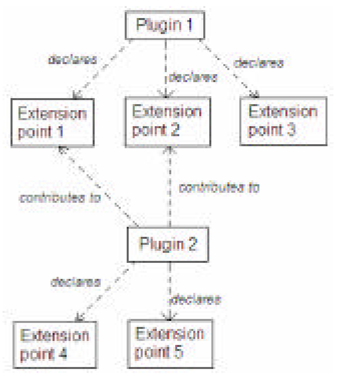


Figure 3. Relations between Eclipse.NET plugins and extension-points.

Eclipse.NET is an integration platform with a plugin-based architecture that provides an elegant and efficient mechanism to allow plugins to communicate, without making them depend on each other [11]. Plugins can have two kinds of relations: (1) “imports” and (2) extension points. If Plugin A imports Plugin B, then Plugin A *requires* that Plugin B is present and functional, otherwise Plugin A will be disabled. Extension points are conceptually similar to interfaces, as they allow communication between plugins but a plugin is not disabled if no extensions are similar; plugins can declare extension points and they can also contribute to extension points. For example, if Plugin A declares Extension Point A (E.P.A) and Plugin B contributes to E.P.A, then Plugin A and Plugin B will be able to communicate; however, if there are no contributions to E.P.A, Plugin A will still be functional (instead of disabled). Figure 3 illustrates the possible relations between plugins and extension points: plugins can declare extension points, and they can also contribute to extension points; it is also quite common for a plugin to contribute to an extension declared by that same plugin (e.g. for supplying a default information processor).

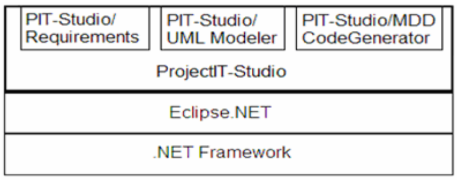


Figure 4. ProjectIT-Studio main components.

Figure 4 presents the generic architecture of the ProjectIT-Studio environment as an orchestration of a set of components developed on the top of the Eclipse.NET framework. Therefore, ProjectIT-Studio should be considered an extensible, modular and plugin-based environment.

On the other hand, Figure 5 presents some usage scenarios for ProjectIT-Studio. Basically, it can be configured and used in two situations: stand-alone (single-user) and collaborative work (multi-user) scenarios. It should be stressed that, independently of these scenarios, ProjectIT-Studio is used differently based upon the interacting role (Architect, Requirements Engineer, Designer or Programmer) as discussed previously in Section 2.

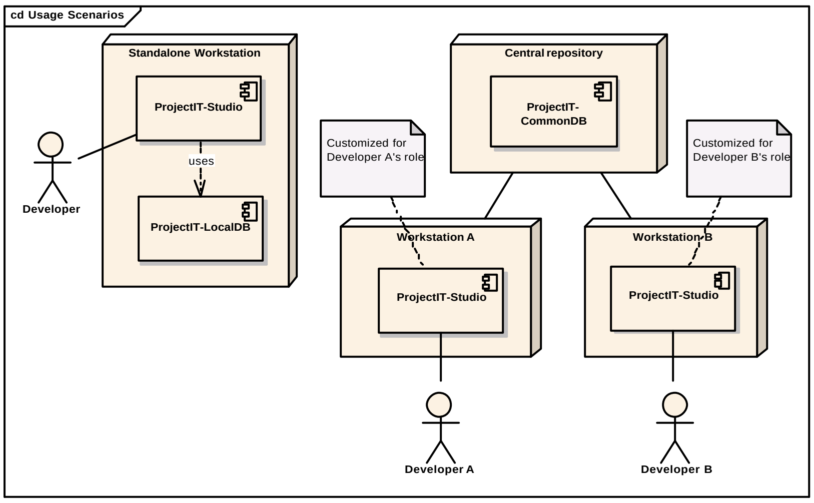


Figure 5. ProjectIT-Studio usage scenarios.

## 4 The ProjectIT-Studio/Requirements

The ProjectIT-Studio/Requirements component follows an innovative approach concerning the capture and management of requirements. Its goal is to support natural language free-form text requirements processing, by providing: (1) a flexible and robust approach to requirements elicitation, where requirements engineers, as well as non-technical stakeholders, can write the specification of the system to be developed; and (2) activity guidance assisted by a rich set of GUI components, namely on-the-fly feedback during the specification activity. Besides the tool, we have developed a new requirements specification language, called ProjectIT-RSL (or PIT-RSL for short), defined after the identification of the most common linguistic patterns used in requirements specification [8]. The supporting concepts of PIT-RSL are aligned with our MDD component metamodel, allowing a clear mapping between them; thus, the produced artifacts can be passed as input to the MDD component, which in turn becomes responsible for initializing the internal generative process and produce the source code of the specified system.

The ProjectIT-Studio/Requirements component follows the normal project creation cycle: (1) project creation; (2) requirements edition; (3) requirements analysis and validation; and finally (4) the possibility to export the captured information to other formats, persistent database storage or, eventually, proceed to further software product's life-cycle stages, specifically modeling and automatic code generation activities both supported by the ProjectIT-Studio/MDD component tools.

In terms of text edition, the specific PIT-RSL text editor supports all the typical IDE features [17] [18] [19], such as on-the-fly syntactic verification and full syntax highlighting, which means that as we write, all expressions are validated by the language checking component; in case there is an error, it is immediately detected and highlighted. The auto-complete feature is also always available, providing the user with hints of how to complete each PIT-RSL sentence. Upon the creation of a syntactic and semantically correct, consistent requirements repository, the integration with the ProjectIT-Studio/MDD's plugins (ProjectIT-Studio/UMLModeler and ProjectIT-Studio/MDDGenerator, presented in the next section) allows further refinement and automatic code generation of the captured information. This process thus covers all the stages of the software lifecycle, supporting the entire development process, from specification to delivery.

## 4.1 ProjectIT-Requirements Specification Language

To define ProjectIT-RSL, we analyzed the format and structure of the requirements gathered from previous projects that we have been involved, which were mainly interactive systems. Afterwards, we identified a common set of linguistic patterns for the requirements of this category of systems, and finally derived a metamodel of requirements that can adequately represent the patterns identified [8].

The linguistic patterns' analysis leaded us to the identification of the main concepts of our language, the PIT-RSL, namely: (1) *Actors*, which are active resources that perform operations on entities; (2) *Entities*, which are static resources affected by operations; (3) *Properties*, which are entities' attributes that represent and describe their state; (4) *Operations*, consisting on a composite workflow that modify entities and their properties; operations are further specialized in (5) *Activities*, which are composite operations, and (6) *Actions*, which are atomic and primitive, and with direct support by the code generation tools.

Besides these core concepts, we have also identified more high-level concepts because requirements documents, despite being written in natural language, should follow a predefined structure. We identified these high-level constructs that group classes of requirements, the *Section* element, and the coherent collection of sections that define a specific software component, the *System* element. Sections have a type for specifying the category of the enclosed requirements.

The proposed context-free grammar serves only the purpose of defining the structural elements of the requirements document, such as systems and sections declarations, and also the requirements statements structure. All syntactic and semantic constructs of PIT-RSL are specified in the form of Template Substitution (TS) rules that are used by the Fuzzy-Matching Parser as described in the next section. The complete analysis of all the linguistic patterns identified is beyond the scope of this article. However, we present here, in EBNF notation, some of the high-level tokens that are used as reference when elaborating the previously mentioned TS rules.

```
<Entity Definition> : <Entity Inheritance Definition> | <Entity
    Property Definition> | <Entity Equivalence Definition> | <Entity
    Association Definition>
<Entity Inheritance Definition> : <Entity> is a <Entity>
<Entity Equivalence Definition> : <Entity> is the same as <Entity>
<Entity Property Definition> : <Entity> has <Property Definition>*
<Property Definition> : [ a | an | the ] [<Primitive Type>]
    [<Quantifier>] <Property>
<Quantifier> : <number> | at least <number> | at most <number> | a list
    of | each | …
<Property> : Name | <Entity>
<Entity Association Definition> : <Entity Active Association Definition>
    | <Entity Passive Association Definition>
```

**<Entity Active Association Definition>** : [<Quantifier>] <Entity> <Active Verb> [<Quantifier>] <Entity>
**<Entity Passive Association Definition>** : [<Quantifier>] <Entity> <Passive Verb> [<Quantifier>] <Entity>

To provide a brief insight on the nature and aspect of the previously mentioned TS rules, we supply part of the PIT-RSL built-in TS rules XML file, where the simpler entity definition TS rules are presented. The description XML element allows the reader to easily establish the relation between the above EBNF notation rules specification and the rules exhibited below. Except for the particular PIT-RSL concepts specific tags, we have adopted the Brown Corpus tag-set notation [20] [21]. The capitalized words in the template XML elements represent keywords of our specification language, whereas lower-case words in the substitution XML element, prefixed with $ symbol, represent a variable binding with the value matched by the respective word in the template during the parsing process.

```
<BusinessEntities>
    <!-- ENTITY DEFINITION -->

    <TSRule>
        <description>Entity Inheritance Definition</description>
        <template>new_entity/PROP IS ENTITY</template>
        <substitution>$NODE/EID</substitution>
    </TSRule>
    <TSRule>
        <description>Entity Equivalence Definition</description>
        <template>new_entity/PROP IS SAME AS entity/ENT</template>
        <substitution>$NODE/EED</substitution>
    </TSRule>
    <TSRule>
        <description>Entity Property Definition</description>
        <template>new_entity/PROP HAS property/PROP</template>
        <substitution>$NODE/EPD</substitution>
    </TSRule>
    <TSRule>
        <description>Entity Property Definition</description>
        <template>new_entity/PROP HAS property_list/PROP/LIST</template>
        <substitution>$NODE/EPD</substitution>
    </TSRule>

 (...)

<BusinessEntities>
```

### 4.2 ProjectIT-Studio/Requirements Architecture

The ProjectIT-Studio/Requirements component is composed of two main packages. The first package, PIT-RSL Parser, contains all the logic responsible for the early stages of the natural language processing, such as the Structural Parser (SP) and Fuzzy-Matching Parser (FMP), which can operate in standalone mode. The second package aggregates all the logic related with the Eclipse.NET plugin; it encompasses all the functionality associated to the text editor and GUI components that provide on-the-fly feedback for assisting the requirements text editing activity. Figure 6 overviews the high-level ProjectIT-Studio/Requirements' architecture.

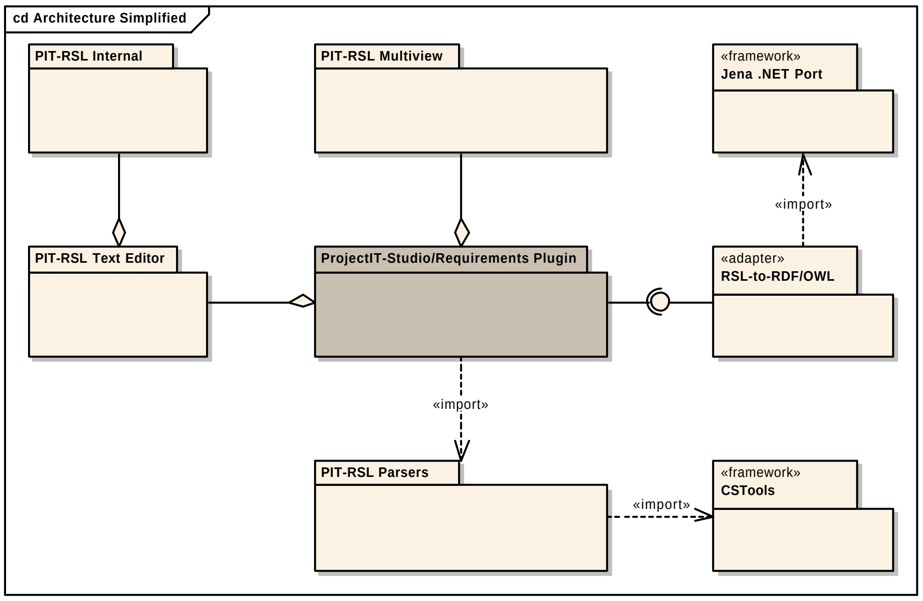


Figure 6. ProjectIT-Studio/Requirements architecture's overview diagram.

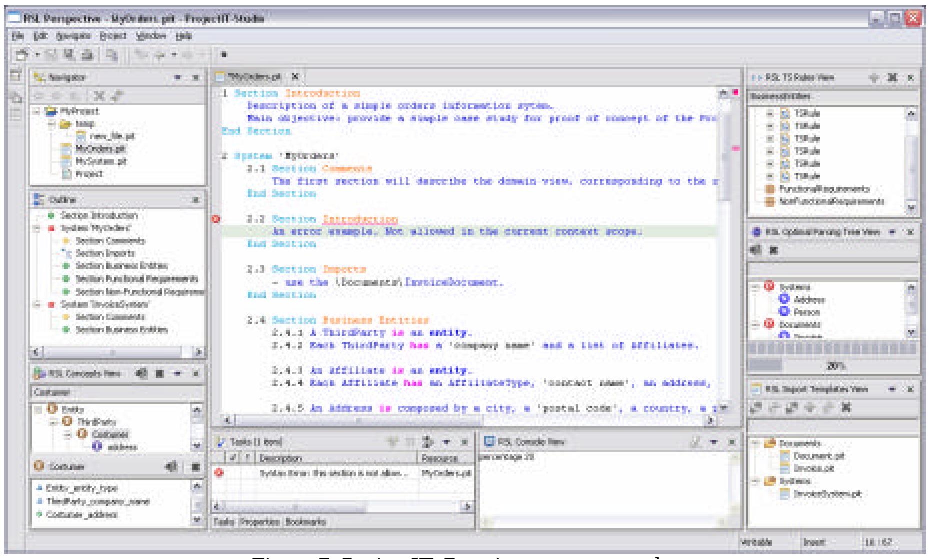
The **Structural Parser** (SP) is a part of the PIT-RSL Parser package and deals with the early format/typographical normalization and document structural analysis stages of the parsing process. It is a LARL (1) parser generated by CSTools (an object-oriented compilers' generation tools framework with a similar Lex/Yacc scripting language, available at http://cis.paisley.ac.uk/crow-ci0/), after processing a lexical and a grammar files, which contain the lexical and context-free grammar rules, respectively.

The **Fuzzy Matching Parser** (FMP) processes natural language free-form text requirements by implementing a rule-based fuzzy matching algorithm for Natural Language Processing (NPL) [22] [23]. It provides a robust and flexible approach because it does not follow a rigid syntax approach as classical programming languages do. The core concept is to take advantage of the intrinsic adherence of text semantic to its sentence syntax, which can be applied to requirements' textual representation when considering their specific features, which allow us to make some simplifications.

The Fuzzy Matching Parser's main goal is to find the optimal parsing tree, by successive comparisons between natural languages patterns (defined by a set of rules designated as TS rules) and the written requirements; every time a valid match occurs, the parser performs the respective rules substitutions in a controlled recursive process, until no more TS rules can be applied.

The ProjectIT-Studio/Requirements Plugin is the root package, built upon Eclipse.NET framework, which provides a powerful extensibility mechanism and plugin architecture. It encapsulates specific text editor's behavior and the base classes responsible for the plugin initialization, logging, and termination. It contains the PIT-RSL Text Editor with several other tree-view components, and provides a coherent set of IDE features that extend the base platform with specific behavior to support the PIT-RSL requirements specification process.

The PIT-RSL Text Editor implements our vision to build a tool for writing "requirements documents", like a word processor that detects and gives feedback of errors violating the requirements language and grammar rules, as described in [8]. This component implements all the required features to assist the stakeholder during the requirements specification process, encompassing the code responsible for extending the available text editor extension point of the Eclipse.NET platform. It provides syntax-highlighting, auto-complete, on-the-fly error checking with text annotation, and other kind of graphical annotations such as tasks and bookmarks, giving instant feedback on most syntactic and semantic problems, or avoids them entirely. Figure 7 presents a screenshot of the PIT-RSL Text Editor.


```
1 Section Introduction
   Description of a simple orders information sytem.
   Main objective: provide a simple case study for proof of concept of the Pro
End Section

2 System 'MyOrders'
   2.1 Section Comments
      The first section will describe the domain view, corresponding to the r
   End Section

   2.2 Section Introduction
      An error example. Not allowed in the current context scope.
   End Section

   2.3 Section Imports
      - use the \Documents\InvoiceDocument.
   End Section

   2.4 Section Business Entities
      2.4.1 A ThirdParty is an entity.
      2.4.2 Each ThirdParty has a 'company name' and a list of affiliates.

      2.4.3 An affiliate is an entity.
      2.4.4 Each Affiliate has an AffiliateType, 'contact name', an address,

      2.4.5 An address is composed by a city, a 'postal code', a country, a z
```

Figure 7. ProjectIT-Requirements screenshot.

One of our goals is to keep our tool compliant with OMG's standards such as UML 2.0 and the XMI format. The possibility to export to the XMI format provides an industry-standard, widely accepted way to exchange information, captured during the requirements natural language processing stage, to other modeling and MDD third-party tools.

Despite of the existence of a modular Structural Parser for format/typographical normalization and document structural analysis, in parallel and overlapping some of this standalone parser responsibilities, we had to create a similar internal parser mechanism to take advantage of the full out-of-the-box potential provided by the Eclipse.NET framework. This **Internal Parser** (IP) processes all the information required to support the annotation model features (namely the error reporting mechanism) and feeds the outline page view. When working with the PIT-RSL Text Editor, the Structural Parser assumes a secondary role since it is only used to load and parse (eventually, in a recursive manner) the requirements documents and templates files, and their respective systems specifications referred in the requirements document's import section.

Finally, there are two packages crucial in this architecture, the RSL-to-RDF/OWL and a Jena.NET port. The latter package represents a .NET port of the Jena framework (available at http://jena.sourceforge.net/), created by using IKVM.NET (available at http://www.ikvm.net). This framework endorses the ProjectIT-Studio/Requirements Plugin with the knowledge-base and inference-engine capabilities typical of a natural language parsing tool [26]. The former package contains the adapter pattern code that provides a clean C# API for using the Jena.NET without the necessary traces of Java syntax code.

The Jena framework stores all the gathered concepts bundled by each individual optimal parse tree, as a tree forest that constitutes an ontology, where concepts are related to each other in a graph, which allows navigation through class and properties lattices [31]. The inference engine offers specialized reasoners (RDF and OWL engines) and, in addition, a generic rule-based inference engine, which supports forwards, backward, and hybrid chaining reasoning engines. Jena is essential for extracting requirements' implicit knowledge through flexible queries.

## 5 The ProjectIT-Studio/MDD

The ProjectIT-Studio/MDD component supports the modeling and the transformation tasks according to the ProjectIT approach. This component aggregates two plugins: (1) a standard UML visual modeling tool (ProjectIT-Studio/UMLModeler) and (2) a template-based code generator (ProjectIT-Studio/MDDGenerator).

### 5.1 ProjectIT-Studio/UMLModeler

The ProjectIT-Studio/UMLModeler plugin consists of a tool for standard UML modeling in the context of the ProjectIT approach. The tool allows the designer to create visual models of the system using the UML 2.0 modeling language [27], as Figure 8 illustrates. The designer creates the models based on a given UML profile, e.g. the XIS UML profile [10]. These models are used in the definitions of generative processes (specified by the programmer, as the next subsection describes) to generate the system artifacts according to specific software architectures.

In addition to the creation of UML models, the tool also features a simple Profile definition mechanism, which allows the further customization of UML; this allows the designer to adapt the system modeling language to the templates that the subsequent generative process (in the ProjectIT-Studio/MDDGenerator) will use.

A profile definition consists of two separate components: (1) the syntax and (2) the semantics of the profile. The syntax is defined by associating stereotypes with images, which is the typical mechanism that the UML specification provides [27] and that most UML modeling tools support; this association between stereotypes and images is done entirely within the tool. As for the definition of profile semantics, the tool supports the graphic specification of extensions between Stereotypes and UML metaclasses (similarly to Enterprise Architect [28]). The tool also supports additional profile semantics, using external .NET assemblies containing code to validate the model in terms of the profile; this allows the tool to assure the continuous validation of the model being created, according to both the UML semantics and the profile semantics.

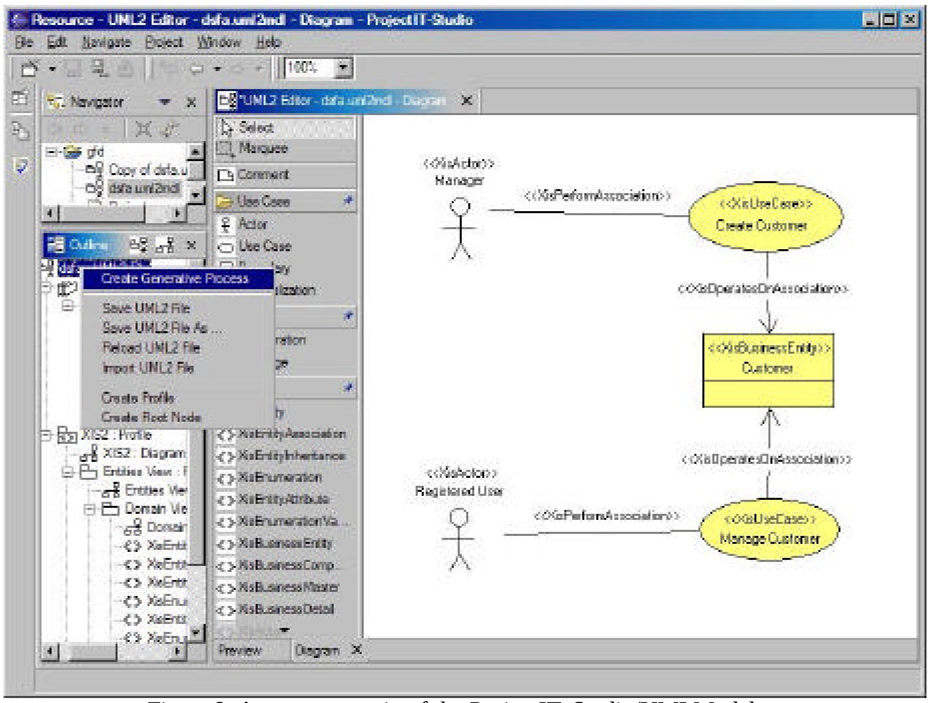


Figure 8. A usage scenario of the ProjectIT-Studio/UMLModeler.

## 5.2 ProjectIT-Studio/MDDGenerator

ProjectIT-Studio/MDDGenerator is the plugin responsible for the generation of system artifacts. As input, it receives a generative process, which is a configuration file that specifies the name, model, and software architecture of the final application. Figure 9 shows an overview of generative process concepts which we describe next.

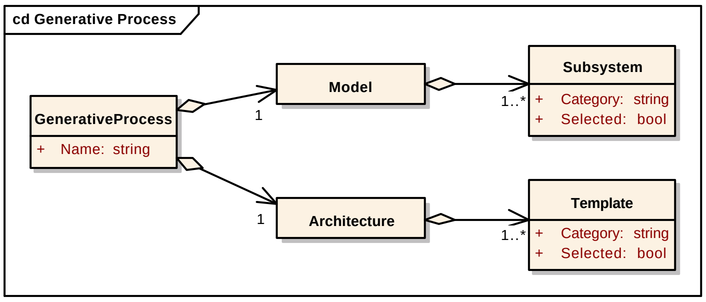
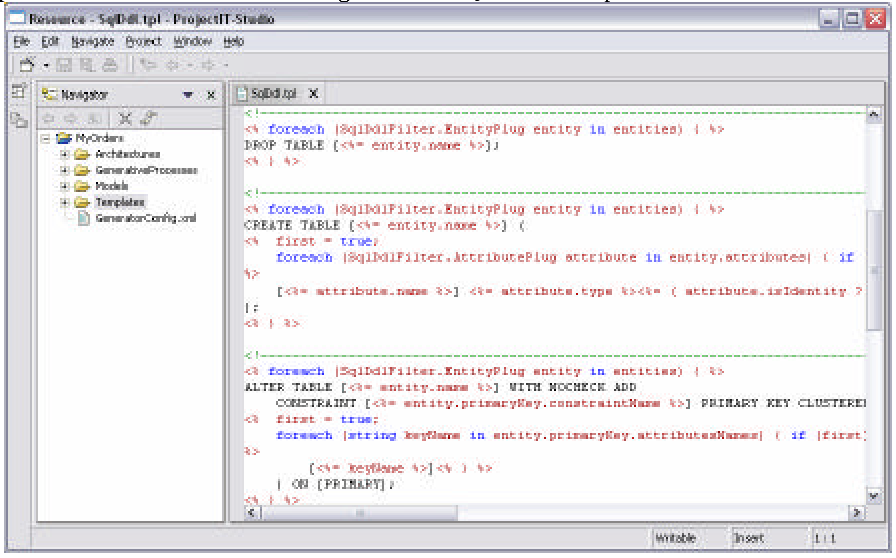


Figure 9. Overview of generative process concepts.

According to the ProjectIT approach, a model is an abstract representation of a software system, created by the designer and based on a given UML profile, e.g. the XIS UML profile [9]. Subsystems are divisions of the model that pertain to a category such as entities, actors or interaction spaces.

According to the same approach, a software architecture is a generic representation of a software platform, created by the architect by developing templates for that target platform. Templates are generic representations of software artifacts to support the "Model2Code" transformations, they pertain to a category such as database, data services, user interfaces, and others. Figure 10 shows a part of a template that specifies the transformation that generates SQL-DDL scripts to create database tables.



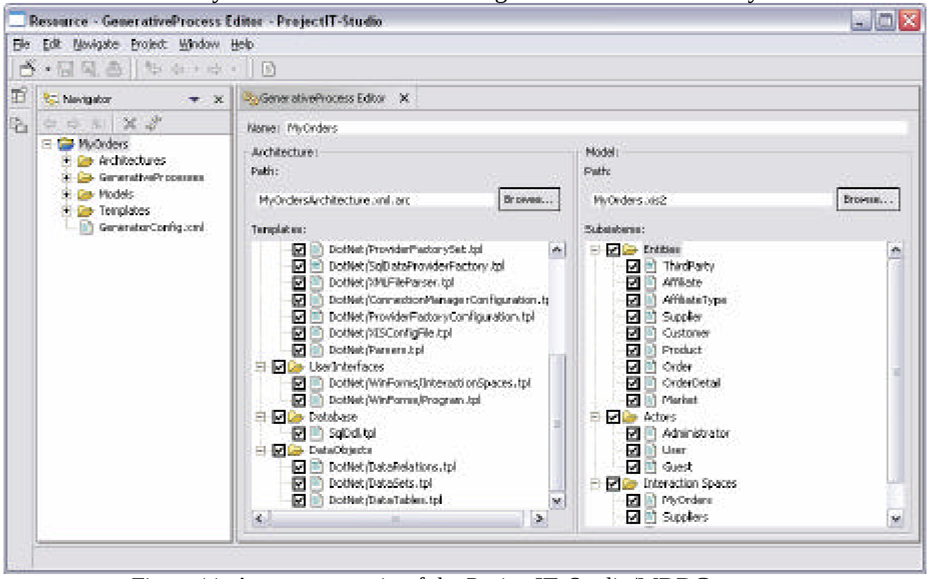
```
<!-----------------------------------------------------------------------------------
<% foreach (SqlDdlFilter.EntityPlug entity in entities) { %>
DROP TABLE [<%= entity.name %>];
<% } %>

<!-----------------------------------------------------------------------------------
<% foreach (SqlDdlFilter.EntityPlug entity in entities) { %>
CREATE TABLE [<%= entity.name %>] (
<%   first = true;
     foreach (SqlDdlFilter.AttributePlug attribute in entity.attributes) { if
%>
     [<%= attribute.name %>] <%= attribute.type %><%= ( attribute.isIdentity ?
);
<% } %>

<!-----------------------------------------------------------------------------------
<% foreach (SqlDdlFilter.EntityPlug entity in entities) { %>
ALTER TABLE [<%= entity.name %>] WITH NOCHECK ADD
     CONSTRAINT [<%= entity.primaryKey.constraintName %>] PRIMARY KEY CLUSTERE
<%   first = true;
     foreach (string keyName in entity.primaryKey.attributesNames) { if (first)
%>
          [<%= keyName %>]<% } %>
     ) ON [PRIMARY];
<% } %>
```

Figure 10. Template editor.

Figure 11 shows the editor provided by ProjectIT-Studio/MDDGenerator that allows the configuration of generative processes; subsystems and templates can be selected individually to allow the incremental generation of the final system.



Name: MyOrders

Architecture:
Path: MyOrdersArchitecture.xml.arc

Templates:
- ☑ DotNet/ProviderFactorySet.tpl
- ☑ DotNet/SqlDataProviderFactory.tpl
- ☑ DotNet/XMLFileParser.tpl
- ☑ DotNet/ConnectionManagerConfiguration.t
- ☑ DotNet/ProviderFactoryConfiguration.tpl
- ☑ DotNet/XISConfigFile.tpl
- ☑ DotNet/Parsers.tpl
- ☑ UserInterfaces
  - ☑ DotNet/WinForms/InteractionSpaces.tpl
  - ☑ DotNet/WinForms/Program.tpl
- ☑ Database
  - ☑ SqlDdl.tpl
- ☑ DataObjects
  - ☑ DotNet/DataRelations.tpl
  - ☑ DotNet/DataSets.tpl
  - ☑ DotNet/DataTables.tpl

Model:
Path: MyOrders.xis2

Subsystems:
- ☑ Entities
  - ☑ ThirdParty
  - ☑ Affiliate
  - ☑ AffiliateType
  - ☑ Supplier
  - ☑ Customer
  - ☑ Product
  - ☑ Order
  - ☑ OrderDetail
  - ☑ Market
- ☑ Actors
  - ☑ Administrator
  - ☑ User
  - ☑ Guest
- ☑ Interaction Spaces
  - ☑ MyOrders
  - ☑ Suppliers

Figure 11. A usage scenario of the ProjectIT-Studio/MDDGenerator.

The ProjectIT-Studio/MDDGgenerator woks as follows: it reads the generative process and loads the model with the selected subsystems, then passes the result as input to each selected template, which is then processed by the template engine.

The template engine transforms templates into C# source code, compiles and executes it in runtime, and then collects the output into the specified file artifact (see Figure 12). This engine is derived from TemplateMaschine (more information in http://www.stefansarstedt.com/templatemaschine.html), which we adapted to support new directive operations that help the architect to specify richer templates in an artifact-oriented perspective, using a syntax similar to ASP.NET.

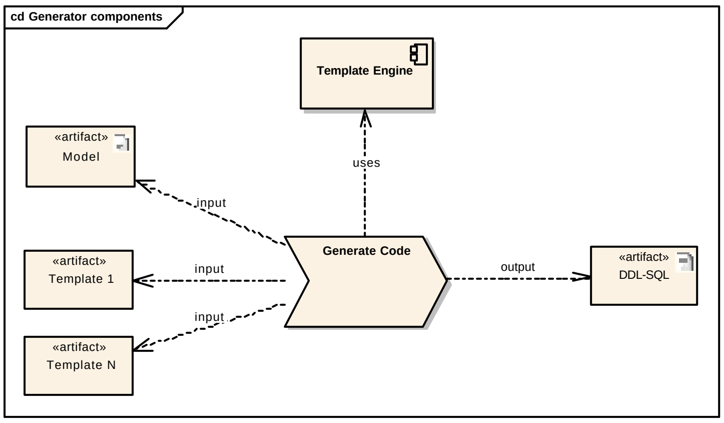


Figure 12. Template Engine overview.

## 6 The ProjectIT-Studio – Putting all together

One of the requirements we had in mind when designing the architecture of ProjectIT-Studio was the ease of installing/removing components, which requires a certain degree of independence between these components. However, these components are required to cooperate (when deployed), which also requires a certain dependency between them. The underlying platform (Eclipse.NET) features a plugin architecture with an extension-based collaboration mechanism for plugins, allowing the fulfillment of these requirements in an effective and elegant manner.

ProjectIT-Studio is implemented as a set of plugins for the Eclipse.NET platform: ProjectIT-Studio/Requirements, ProjectIT-Studio/UMLModeler, ProjectIT-Studio/MDDGenerator, ProjectIT-Studio/Kernel, and ProjectIT-Studio/CommonDB (for text simplicity, we will omit the "ProjectIT-Studio" prefix in the remainder of this section). The Kernel and CommonDB plugins are base plugins, whose function is to support the development-oriented top-level plugins: Requirements, MDDGenerator and UMLModeler. Figure 13 illustrates how the ProjectIT-Studio plugins are working together.

The Kernel plugin provides basic framework facilities, like the UML2 metamodel, to the top-level plugins. The CommonDB plugin allows the top-level ProjectIT-Studio plugins to serialize and deserialize information to a persistent storage mechanism (such as a relational database), decoupling persistence details from those plugins.

Serialization to a persistence medium used by multiple ProjectIT-Studio instances allows those instances to be synchronized with each other.

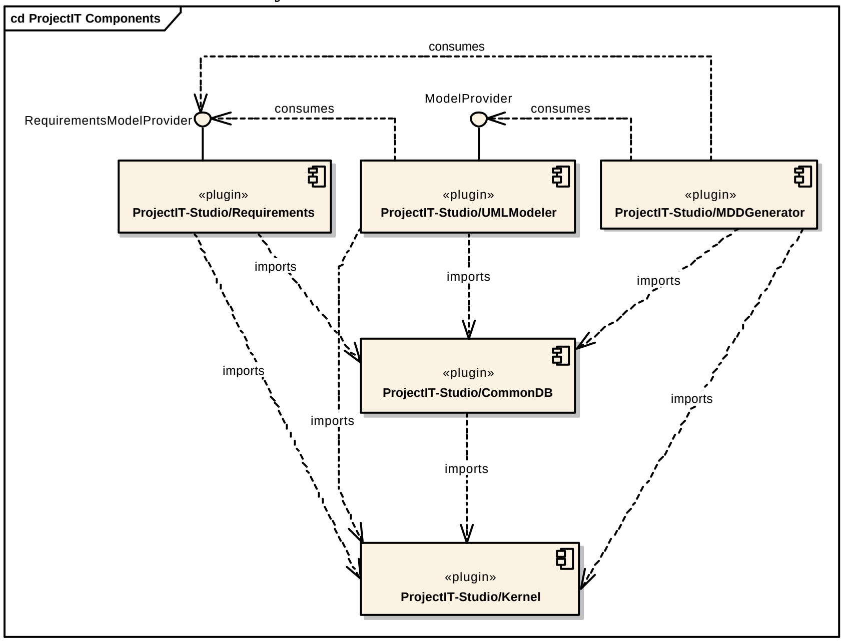


Figure 13. ProjectIT-Studio's plugins and relations between them.

The top-level plugins (Requirements, UMLModeler and MDDGenerator) also interact with each other, through Eclipse.NET's extension-point mechanism.

Requirements declares an extension-point, called "RequirementsModelProvider", which means that the plugin can provide models; any plugins that can process (i.e. consume) models produced by Requirements should declare themselves as consumers of this extension-point.

UMLModeler also declares an extension-point, called "ModelProvider", with a purpose similar to Requirements' "RequirementsModelProvider" extension-point. Additionally, since the ProjectIT approach includes obtaining models from requirements, UMLModeler declares itself as a consumer of Requirements' "RequirementsModelProvider" extension-point.

MDDGenerator, as the last tool to be used in the ProjectIT approach, does not declare any extension-points, since there would be no consumers. The plugin does declare itself as a consumer of both UMLModeler and Requirements' extension-points ("ModelProvider" and "RequirementsModelProvider", respectively). This way, MDDGenerator can receive models from both Requirements and UMLModeler, allowing the developer to: (1) directly generate code from Requirements; or (2) generate code from a model specified using UMLModeler.

All the plugins that supply extension-points (i.e. Requirements and UMLModeler) are configured to present the developer with a set of options, allowing the choice of the consumer that should receive the model. Figure 12 previously illustrated this, with the "Create Generative Process" context menu entry meaning that the model currently being specified would be supplied to MDDGenerator (that declared itself as a consumer of UMLModeler's "ModelProvider" extension-point).

## 7 Related Work

The idea of providing a complete software development workbench, throughout the entire life cycle is not new, but only recently some attempts had some success. Since the first CASE tools (many of which have already disappeared from the market) that we have been pursuing the idea of code generation, but the best we reached was the generation of database scripts. In terms of modeling, many tools have been created, but they concentrate of the visual issues, and not on the automation of tasks of the development process. For example, many requirements tools deal with requirements as information that is managed like any other concept managed by an information system; requirements are thus inserted, updated and viewed as records in a table of a relational database.

Even in terms of research there are not many proposals, like that we propose, that start in the requirements specification task and end with the application of code generation techniques. Ambriola and Gervasi proposed a "lightweight formal method" approach, supported by the use of modeling and model checking techniques to produce a formal validation of the requirements written in natural language [16]. The project CIRCE uses NL as the specification language and provides feedback to the user with a multiple-views approach. They also use fuzzy matching domain-based parsing techniques to extract knowledge from requirements documents, which is later used to provide different views and models to analyze this knowledge. Although Circe and ProjectIT-RSL have some similarities, there are between them many differences, namely in the architecture, concepts and algorithms used, and above all, in the strategy: the goal of CIRCE has been requirements validation, and only recently moved to research about the integration with model driven approaches, whereas our goal with requirements specification is to obtain a consistent requirements document that is in conformance with a metamodel, which also enables the use of model driven techniques and code generation.

Microsoft itself announced the Dynamic Systems Initiative, an approach that integrates ideas and tools, with impact at the hardware, systems, applications and process levels, in order to easy the development and management of information systems. Among other key ideas, it is supposed to automate some of the development tasks, thus resulting in increased productivity and reduced costs. Particularly important in this initiative is the Software Factories line of research [24], that follows the ideas of software product lines (it researches the key issues related with producing software like the traditional industrial product lines) and domain specific languages (it is concerned with the creation of specialized languages for the specification, design, modeling or even programming software systems) [29,30]. A more detailed comparison between DSI/Software Factories and ProjectIT results in the identification of many similarities, but still some important differences; for example, we strongly believe in the importance of getting a correct and validated requirements specification at the beginning of the projects, whereas Software Factories typically do not address the requirements phase.

## 8 Conclusions and Future Work

This paper introduces the ProjectIT research initiative, its main issues and challenges. In order to validate the proposed ideas, and contributions, we decided to design and develop our own workbench, called ProjectIT-Studio. This paper presents and discusses the most relevant architectural aspects of ProjectIT, as well as its innovative and distinct features.

ProjectIT-Studio is an integrated environment of a set of components developed on top of the Eclipse.NET framework, and so it is an extensible, modular and plugin based environment. ProjectIT-Studio currently supports the two most relevant and distinctive features of the ProjectIT initiative: (1) requirements specification and management and (2) models definition and model-to-code transformations. The results we have achieved until now show that the ambitious vision we had in the beginning of this project is possible. It is possible to produce software systems in a more productive way, by adapting and integrating techniques such as rigorous natural language based requirements specification, system modeling, code generation or models/artifacts transformation.

However, and in spite of the results already achieved, there are many issues, questions and research to be done in the near future. The first key issue at the stage of our research is results validation: the proposed features must be validated and tuned in real-world projects, which means that some of them could be applied by third parties, such as software houses. Second, the issue of components integration is also a mandatory one: we are currently lacking an adequate and integrated support for all our tools at the data representation and storage level. Frequently, each component manages its specific information in separate repositories, thereby implying a need to duplicate information. Third, the issue of model transformations is something relevant, namely to enhance the designer and the developer's productivity, and we are considering paying more attention to it in the near future.

## References

1. Baldonado, M., Chang, C.-C.K., Gravano, L., Paepcke, A.: The Stanford Digital Library Metadata Architecture. Int. J. Digit. Libr. 1 (1997) 108–121
2. Bruce, K.B., Cardelli, L., Pierce, B.C.: Comparing Object Encodings. In: Abadi, M., Ito, T. (eds.): Theoretical Aspects of Computer Software. Lecture Notes in Computer Science, Vol. 1281. Springer-Verlag, Berlin Heidelberg New York (1997) 415–438
3. van Leeuwen, J. (ed.): Computer Science Today. Recent Trends and Developments. Lecture Notes in Computer Science, Vol. 1000. Springer-Verlag, Berlin Heidelberg New York (1995)
4. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd edn. Springer-Verlag, Berlin Heidelberg New York (1996)
5. Bell, T., Thayer, T., Software requirements: Are they really a problem?, in Proceedings of the 2nd International Conference on Software Engineering, pp. 61-68, 1976
6. Silva, A., O Programa de Investigação "ProjectIT", Technical Report, October 2004, INESC-ID (available at http://berlin.inesc-id.pt/alb/uploads/1/193/pit-white-paper-v1.0.pdf)
7. Videira, C., Silva, A., Patterns and metamodel for a natural-language-based requirements specification language, in Proceedings of CaiSE'05 Forum, pp. 189-194, Porto, Portugal, June 2005

8. Videira, C., Ferreira, D., Silva, A., A linguistic patterns approach for requirements specification, in 32nd Euromicro Conference on Software Engineering and Advanced Applications, Dubrovnik, Croatia, August 2006
9. Silva, A., Lemos, G., Matias, T., The XIS Generative Programming Techniques, in Proceedings of the 27th Annual Int. Comp. Software & Application Conference, Dallas, USA, 2003
10. Silva, A., The XIS Approach and Principles, in Proceedings of the 29th EUROMICRO Conference, Turkey, Antalya, September 2003
11. Saraiva, J., Silva, A., Eclipse.NET: An Integration Platfom for ProjectIT-Studio, in Proceedings of the First Conference on Innovative Views on .NET Technologies (IVNET'05), Porto, Portugal, June 2005
12. Videira, C., Silva, A., ProjectIT-Requirements, a Formal and User-oriented Approach to Requirements Specification, Actas de las IV Jornadas Iberoamericanas en Ingeniería del Software e Ingeniería del Conocimiento - Volumen I - pp 175-190, Madrid, Spain, November 2004
13. Saraiva, J., Relatório Final de Trabalho Final de Curso – Desenvolvimento Automático de Sistemas, IST/UTL, July 2005
14. Videira, C., Ferreira, D., Silva, A., Patterns and parsing techniques for requirements specification, CISTI 2006, Esposende, Portugal, June 2006
15. Carmo, J., Videira, C, Silva, A., Using Visual Studio Extensibility Mechanisms for Requirements Specification, in Proceedings of the First Conference on Innovative Views on .NET Technologies (IVNET'05), Porto, Portugal, June 2005
16. Ambriola, V., Gervasi, V., The Circe approach to the systematic analysis of NL requirements, Technical Report TR-03-05, University of Pisa, 2003
17. What's New in Visual Studio 2005. http://msdn2.microsoft.com/en-us/library/88fx1xy0.aspx
18. Eclipse 3.2 - New and Noteworthy. http://download3.eclipse.org/eclipse/downloads/drops/R-3.2-200606291905/new_noteworthy/eclipse-news.html
19. NetBeans Features List. http://www.netbeans.org/products/ide/features.html
20. The Brown Corpus. http://clwww.essex.ac.uk/w3c/corpus_ling/content/corpora/list/private/brown/brown.html
21. Brown Corpus Manual. http://helmer.aksis.uib.no/icame/brown/bcm.html
22. Natural Language Processing. http://www.aaai.org/aitopics/html/natlang.html
23. Microsoft Research Natural Language Processing http://research.microsoft.com/nlp/
24. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, Greenfield, J., Short, K, Cook, S., Kent, S. Crupi, J., Hungry Minds Inc, U.S., 2004
25. Preece, J., et al., *Interaction Design*, Wiley, 2002.
26. Mich, L., Garigliano, R., NL-OOPS: a requirements analysis tool based on natural language processing, in Proceedings of Third International Conference on Data Mining Methods and Databases for Engineering, Bologna, Italy, 2002
27. Object Management Group, Unified Modeling Language: Superstructure – Specification Version 2.0, August 2005
28. Enterprise Architect – UML Design Tools and UML CASE tools for software development. http://www.sparxsystems.com/products/ea.html
29. Clements, P.and Northrop, L.. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley, August 2001.
30. Deursen, A., Klint, P. Visser, J., Domain-Specific Languages: An Annotated Bibliography, in http://homepages.cwi.nl/~arie/papers/dslbib/
31. Jena 2 Inference Support. http://jena.sourceforge.net/inference/index.html