Integration of RE and MDE paradigms: the ProjectIT approach and tools

A.R. da Silva, J. Saraiva, D. Ferreira, R. Silva and C. Videira

Abstract: The suggestion that in software development projects the emphasis must be on the project management (RE), requirements engineering, and design activities, and consequently efforts in production activities – such as traditional software programming and testing – should be minimised and performed as automatically as possible is discussed. The ProjectIT approach that integrates contributions from the RE and model-driven engineering communities is also discussed. The goal with requirement specification is not just in managing textual specifications, but also to obtain a consistent requirements document that is in conformance with a domain-specific language, and that can be re-used to increase the design and development activities in the context of model driven and code generation techniques. Furthermore, the feasibility and benefits of this approach by presenting a proof-of-concept case study are discussed, in which the orchestration of the concepts and concrete components related with the ProjectIT approach, the PIT-RSL, XIS and PIT-TSL languages and the ProjectIT-Studio CASE tool is emphasised. A practical demonstration of the approach including the description of the system requirements, the design of the system, the use of code generation techniques, and how they integrate to improve and accelerate the software engineering lifecycle is presented.

1 Introduction

The development of information systems is a complex process, usually initiated with the identification and specification of the system requirements and 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 emphasised the costs and quality problems that can result from the mistakes in the early phases of system development, such as inadequate, inconsistent, incomplete or ambiguous requirements [1]. These facts emphasise the need to improve higher-level tasks of the software development projects, such as project management and requirements for engineering and design activities to reduce the efforts and automate production activities, such as software programming and testing.

As a result of the experience gathered from previous research and practical projects, we started an initiative called 'ProjectIT' [2], which involves contributions from both the requirements engineering (RE) and the model-driven engineering (MDE) communities. RE recognises the relevance of socio-technical approaches and best practices for the requirements elicitation, specification, validation, re-use and management [3, 4]. RE needs to be sensitive to how people understand the world around them, how they interact and how the sociology of the workplace affects their actions. That is why RE goes beyond the usual concepts related exclusively with information systems, and receives contributions from different areas, such as cognitive psychology, anthropology, sociology and linguistics [5]. This unique combination leads a wide range of RE techniques, such as stakeholder interviews, JAD and workshop sessions, prototype design, ethnographic or analysis of existing documentation [3, 4].

On the other hand, MDE considers the systematic use of models as primary engineering artefacts throughout the engineering lifecycle, where models are considered as first class entities [6]. The best known MDE initiative is model-driven architecture (MDA), which was proposed by the Object Management Group [7, 8]. This initiative clearly identifies models at different abstraction levels, namely, the computational-independent model (CIM), platform-independent model (PIM) and platform-specific model (PSM). MDE also involves the use of generative techniques to support model-to-model transformations as well as model-to-code (or textual artifacts in general) transformations to accelerate the development process and, consequently, to improve the productivity of software development processes [9–11]. A number of significant MDE-based projects can be identified, such as WebML [12], OOHDM [13], AndroMDA [14], Jamda [15], JET [16], OptimalJ [17], ArcStyler [18], XDE [19] and Codagen Architect [20].

In the past, many research initiatives have been developed independently by the RE and MDE communities, even if they address common issues and problems. However, ProjectIT proposes a tight integration between RE and MDE; we claim that RE and MDE approaches should be addressed complementarily with real benefits, in particular: (1) rigorous and formal requirements specification; (2) traceability between system artifacts, namely between requirements, models and source code; (3) automatic production of software artefacts and prototypes; and (4) improved productivity.

doi:10.1049/iet-sen:20070012

Paper first received 5th February and in revised form 22nd October 2007

The authors are with INESC-ID, Instituto Superior Técnico, Rua Alves Redol, No. 9 –1000-029 Lisboa, Portugal

E-mail: alberto.silva@acm.org

ProjectIT approach is focused upon interactive systems, which are a sub-class of information systems that provide a wide range of common features, as pointed out by Preece *et al.* [21]: (1) user-interfaces, to drive human–machine interaction; (2) databases, to keep the manipulated information consistent; and (3) role-based access control, to manage end-user profiles and related permissions.

To support the activities of the entire software development life cycle, from requirements specification to the use of generative programming techniques, we have defined and developed the following languages and tools:

- PIT-RSL: a textual-controlled natural language for requirements specification;
- XIS: a UML-based language (or UML profile) for modelling interactive systems at a platform-independent level;
- PIT-TSL: a script-based language to define software architecture templates; and
- ProjectIT-Studio: an extensible, modular plugin-based CASE tool that supports the proposed approach.

ProjectIT proposes a platform-independent process for specifying and designing information systems, following a structured sequence of steps: (1) use PIT-RSL to specify and validate requirements; (2) refine the obtained higher-level model using the XIS profile, which allows the design of interactive systems at a PIM level; and finally, (3) generate different versions of the system for different platforms, such as Web, Desktop or Mobile specific platforms, through model-to-code-specific transformations, as defined in PIT-TSL.

This paper describes the ProjectIT approach and highlights the inherent benefits of its use in real projects in terms of productivity and quality of the achieved software development process and products. To achieve this goal, we also present ProjectIT-Studio, which is a modular and extensible CASE tool that we have developed to support and test our research ideas, and MyOrders2, a proof-of-concept case study used to evaluate and consolidate the theoretical groundwork of the ProjectIT approach.

This paper is structured as follows. Section 2 overviews the ProjectIT approach, in particular how the different roles involved in the project contribute to the development of information systems, and all the supporting groundwork concepts grouped by the three major software development process phases, namely system requirements specification, system design and system development. Section 3 introduces a set of tools that together composes the ProjectIT-Studio workbench, addressing its main components related to the three phases above mentioned: ProjectIT-Studio/Requirements, ProjectIT-Studio/UMLModeler and ProjectIT-Studio/MDDGenerator. Section 4 presents a case study to demonstrate the practical application of the ProjectIT approach and the benefits offered by ProjectIT-Studio tools. Finally, Sections 5 and 6, respectively, present related work of other initiatives and conclusions of our research work, justifying our perception that this proposal makes innovative contributions to the community.

2 ProjectIT approach

This section introduces the conceptual view of the ProjectIT approach, namely the roles and tasks involved, as well as the techniques and languages used, according to a phase-oriented perspective.

2.1 Overview of the ProjectIT approach

The ProjectIT approach can be defined using a role/task-oriented perspective, in which a set of tasks is performed by different roles: Architect, Requirements Engineer, Designer, Programmer, Tester and Integrator. As suggested in Fig. 1, these tasks can be grouped into two main moments: (1) definition of the artefacts that are the core of the ProjectIT approach, and which are later used in specific projects; these tasks are usually performed by the Architect; and (2) use of these artefacts to develop specific projects; these latter tasks are performed by the other roles. Generically, we can say that the ProjectIT approach receives system/business requirements as its main input, and produces a set of artefacts as its main output, such as validated requirements, source code and configuration scripts or data scripts.

The Architects have a critical role according to the ProjectIT approach, as they are is responsible for the initial tasks of the process. First, they define or adapt the linguistic terms and the syntactic and semantic rules of ProjectIT-RSL [22] which can be defined on a project basis, thus providing true support for a domain-specific language. Afterwards, they are also responsible for the definition of a suitable and easy-to-use UML profile (in the scope of interactive systems the XIS profile can be used); nevertheless, the ProjectIT approach is language-independent, which means that different UML profiles can be used). They can develop model-to-model features, such as 'Model2Model Transformation Templates' to produce new models, and model-to-code transformations features, such as 'Model2Code Transformation Templates', to support the automatic generation of software and documentation artefacts through generative programming techniques.

The Requirements Engineers are responsible for gathering different system requirements using well-known requirement elicitation techniques, such as meetings, interviews, JAD sessions among designers, clients, end-users and other stakeholders [3, 4]. These requirements are then specified in ProjectIT-RSL and validated using a set of parsing, knowledge extraction and inference tools [23]. The ultimate goal of ProjectIT-Requirements is to allow a non-technical user to actively participate in the specification and validation of the software requirements.

The Designer receives the validated requirements and is responsible for producing an integrated set of models (the 'Design System' task). At this stage, the Designer can apply automatic model transformations, the 'Apply Models Transformations' task, based on the model-to-model transformation templates previously developed by the Architect. This task can be useful in certain situations to simplify or accelerate the design task. The correctness and quality of these models are essential to obtain good results in the subsequent tasks.

After the Designer's participation, the Programmer applies model-to-code transformations (the 'Apply Model2Code Transformations' task), which means the Programmer applies generative code techniques, based on the model-to-code transformation templates previously provided by the Architect. Although this task is performed almost automatically, the Programmer's intervention is still required for implementing specific system features or behaviour, because it is not possible to capture and design all the system requirements. For instance, using the XIS language, it is not possible to capture all business rules or the non-functional requirements during the previous phases. Consequently, programmers are required to

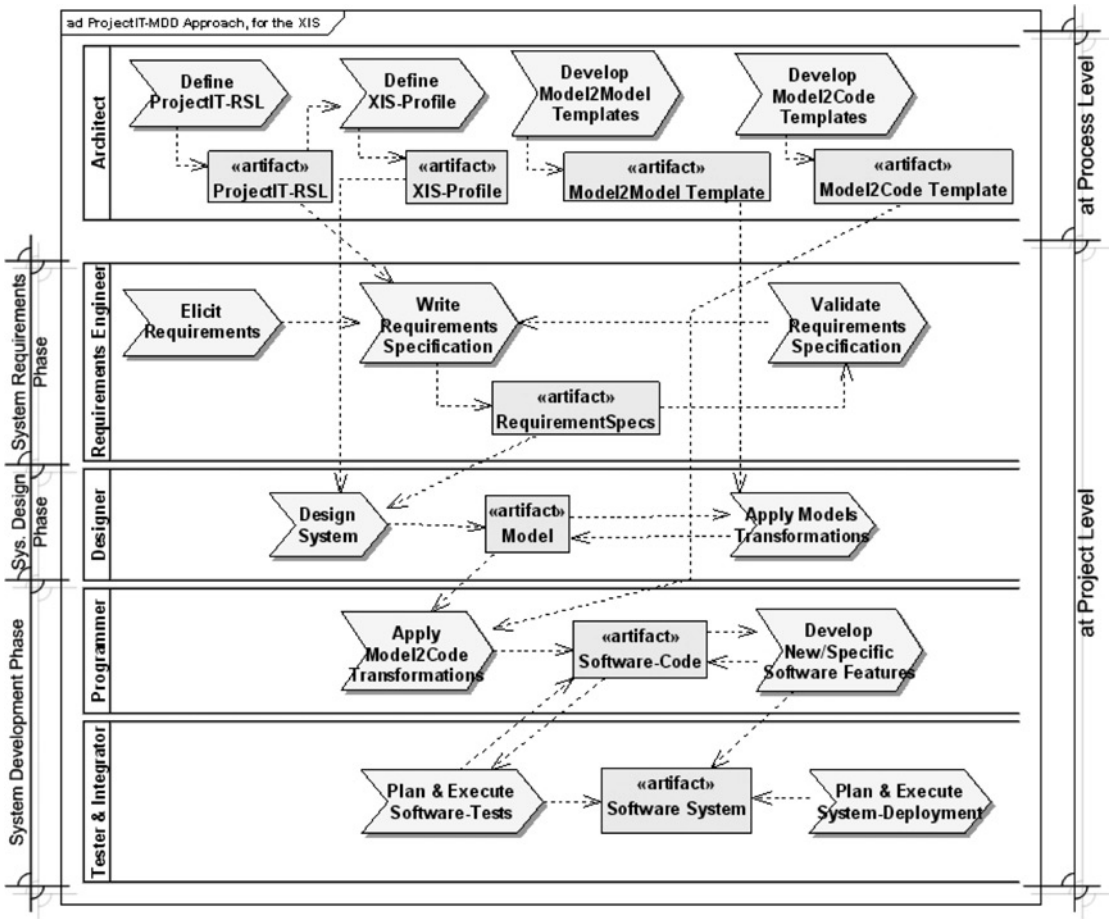

Fig. 1 *Roles and tasks involved in the ProjectIT approach*

produce specific components, typically helper source codes, such as facades, adapters, controllers, and business logic. These activities are suggested in Fig. 1 by the 'Develop New/Specific Software Features' task.

Finally, the intervention of the Testers and Integrators is necessary to prepare and perform different tests to guarantee system quality. These activities are suggested in Fig. 1 by the 'Plan & Execute Software-Tests' and 'Plan & Execute System-Deployment' tasks. Currently the ProjectIT approach does not provide any specific contributions regarding these tasks.

Besides these roles and a task-based perspective, the ProjectIT approach can also be analysed according to software development project phases, namely: (1) system requirements phase, for which the Requirements Engineer is the main responsible party; (2) system design phase, covering all the activities performed by the Designer; and (3) system development phase, assigned to the Programmer, Tester and Integrator roles. The concepts and detailed coverage of each of these phases are described in the following subsections.

2.2 System requirements phase

Regarding the system requirements phase, the task performed by the software architect is mainly the definition of convenient requirements specification language. This language is then used and applied by requirements engineers, domain experts and other stakeholders, who participate in the requirements elicitation and specification tasks. In many cases, this language benefits from previous projects, and reuses the same linguistic patterns. We called this language 'ProjectIT-Requirements Specification Language' or PIT-RSL, for short.

To define PIT-RSL, we analysed the format and structure of the requirements gathered from several projects in which group members were involved in the past. Afterwards, we identified a common set of linguistic patterns for the requirements of interactive systems and, finally, we derived a corresponding metamodel of requirements that adequately represents the patterns identified [22].

The linguistic pattern analysis leads us to the identification of the main concepts of the PIT-RSL language, as suggested in Fig. 2: (1) Actors are active resources that perform operations on entities; (2) Entities are static resources affected by operations; (3) Properties are entities' attributes that represent and describe their state; (4) Operations are composite workflows that modify entities and their properties; operations are further specialised in (5) Activities that are composite operations and (6)

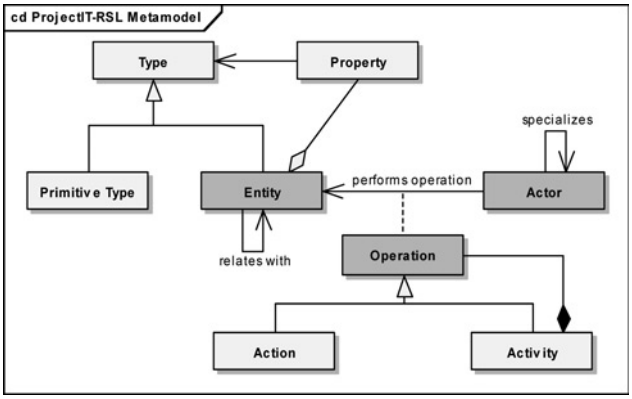

Fig. 2 *ProjectIT-RSL's Metamodel*

Actions, that are atomic and primitive, and with direct support by code generation tools.

Besides these core concepts, we have also identified higher-level concepts because requirements documents, despite being written in natural language, should follow a predefined structure [24]. Thus, we considered the existence of the Section element, a higher-level construct that groups classes of related requirements, and the System element, which is a coherent collection of sections that define a specific software component. Sections have a type for specifying the category of the enclosed requirements (see Section 4.2 for more details).

2.3 System design phase

Regarding the system design phase, the task performed by the software architect is primarily the definition of a suitable design or modelling language. Nowadays, UML and MOF are the most popular mechanisms to create domain-specific languages, and so we used them in our prototypes [10, 11].

We have defined the XIS language [25] ('eXtreme modeling Interactive Systems') as a UML profile oriented towards the design of interactive software systems in a way that should be as simple and efficient as possible. There are several advantages, but have drawbacks of using UML for defining new DSLs through the 'profile' mechanisms as we did in XIS. Other alternatives could be adopted, for example MOF or even simpler meta-metamodel features, which could provide in theory a clean and easy process to define new languages/DSLs. However, these alternatives require specific CASE tools, and presently there is not an adequate kind of metamodelling CASE tool.

To achieve this goal, the XIS language follows some key ideas [25]:

- Modularisation, by allowing the definition of 'business entities' with any level of granularity;
- Separation of concerns, through the definition of multiple views that handle different aspects and are relatively independent among themselves;
- Use-case-driven approach, through the identification of actors and use cases, to manage the main functionalities of the system; and
- Model transformations, by defining a series of possible approaches based on different kinds of model transformations.

XIS Multiviews: XIS proposes the modelling of interactive software systems through multiple views, which are illustrated in Fig. 3. The Entities View specifies the domain of the system in various levels of granularity. This view can be described by the following views: Domain View and BusinessEntities View. The Domain View specifies the relevant entities of the system from the problem-domain perspective and using simple class diagrams with classes, attributes, and creates relationships such as associations, aggregations and inheritance. The BusinessEntities View specifies the system's business entities as aggregations of domain entities or even other business entities. The BusinessEntities View allows the specification of entities of any level of granularity because they can be of a coarser or finer granularity simply by aggregating other business entities or domain entities.

The Use-Cases View specifies actors and use cases, and establishes the corresponding permissions. This view involves the Actors- and UseCases-specific views. The Actors View specifies the system's actors (i.e. the roles

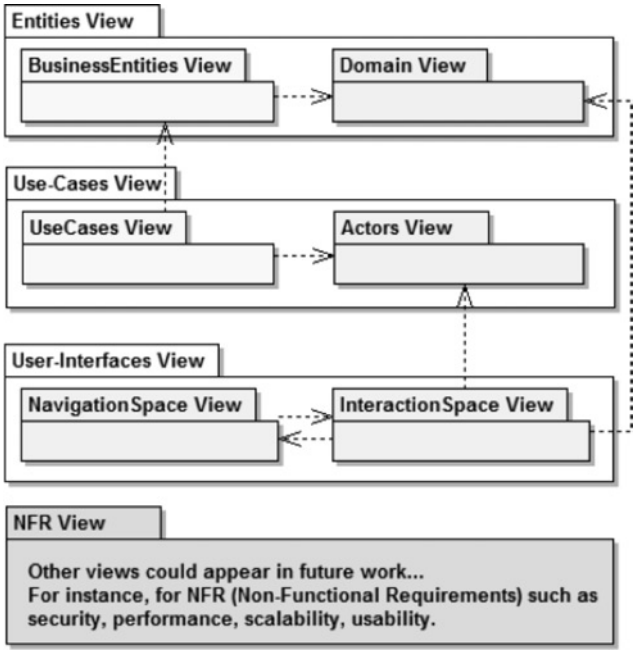

Fig. 3 *Multiview organisation of XIS*

that the end-users can assume) that can perform operations and inheritance relationships between those actors. The UseCases View specifies the relationships between actors and the operations; those actors are allowed to perform on business entities.

Finally, the User-Interfaces View specifies the aspects related to the user interface of the system, and is based on the application of typical interaction patterns [25, 26]. This view involves the following views: the InteractionSpace View and NavigationSpace View. The InteractionSpace View is an abstract 'screen' that receives and presents information to end-users during their interaction with the system. It employs sketching techniques to visually define user–interface interaction elements that are contained in each interaction space, and to specify access control between actors and user–interface elements. The NavigationSpace View defines the navigation flow between any of the interaction spaces, in a way similar to a directed graph, where the nodes are InteractionSpaces and the edges are the navigation transitions.

Design approaches: Fig. 4 shows two different approaches for designing interactive systems according to the XIS language: the classic and the smart approaches. Following the 'classic' approach (shown in the left section of Fig. 4), the Designer should develop the Domain, Actors, NavigationSpace and InteractionSpace views. The BusinessEntities and UseCases views are optional but can also be produced, as they could be useful for documentation purposes. However, these views are useless from the perspective of model-to-code transformations. It should be noted that the classical approach is time-consuming because the NavigationSpace and the InteractionSpace views usually take a long time to produce, but can be necessary if model-to-model transformations are not available.

On the other hand, following the 'smart' approach (shown in the right section of Fig. 4), only the Designer has to produce the Domain, BusinessEntities, Actors and UseCases views. The smart approach speeds-up the modelling of the system because it is supported by 'model-to-model' transformations, which automatically generate time-consuming user–interface models. These generated user–interface

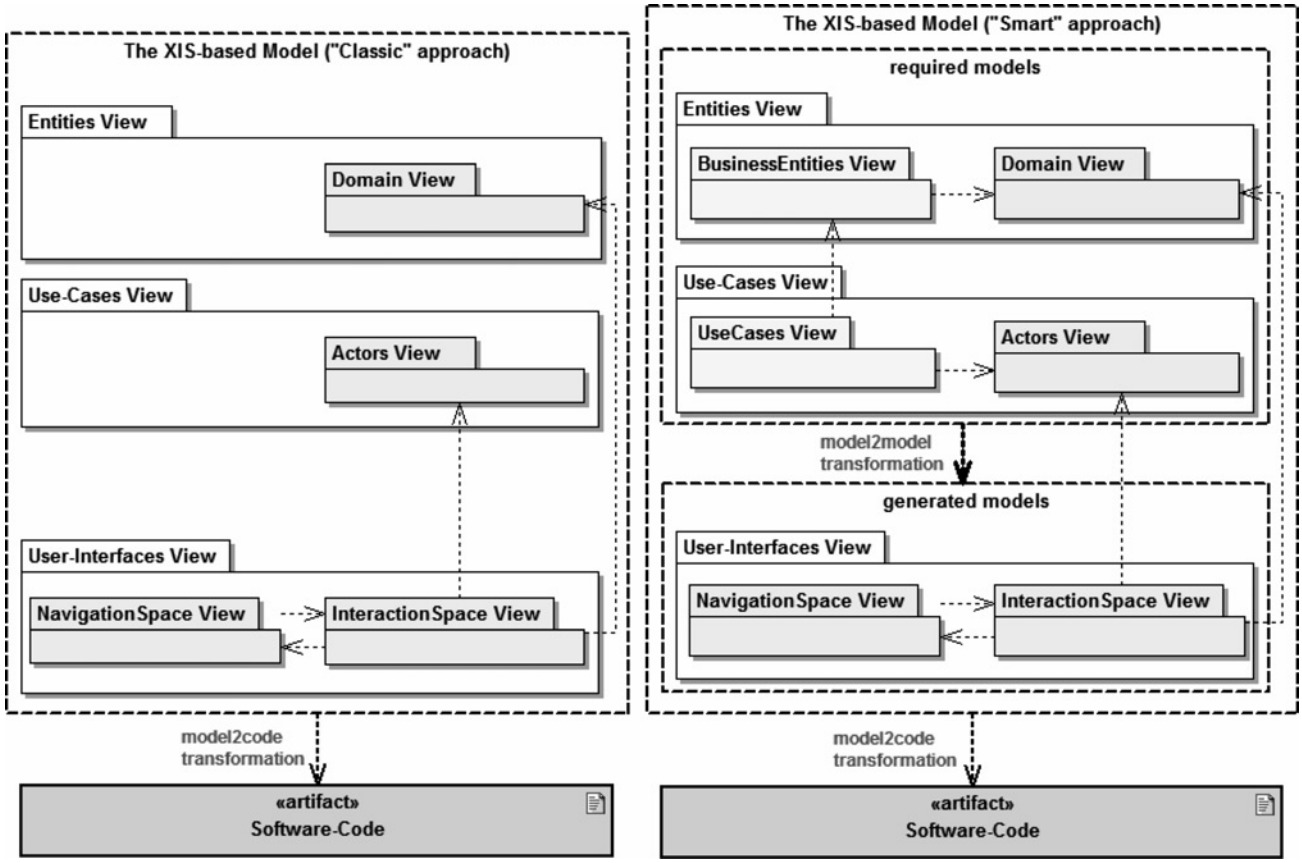

Fig. 4 *'Classic' and 'smart' design approaches defined by XIS*

models can still be customised or changed to support specific requirements, and not well captured by the model-to-model transformation templates, such as issues concerning the UI layout and behaviour.

2.4 System development phase

Regarding the system development phase, the task performed by the software architect is primarily the definition of specific model-to-code transformation templates to be used by code generative processes to produce the source code and deployment files of the application.

The software architect uses a Template Specification Language, PIT-TSL for short, to define these templates. The PIT-TSL has several elements to tag the text that guide the generator engine. These tags are evaluated at generation-time to calculate expressions and control the generation process. Moreover, these tags can control the template engine to include other code-templates, specify comments sections or even mark zones in the generated files that should be preserved in future regenerations. The dynamic evaluation of these tags depends on the information provided by the model and always results in a string that is output to the source code file in the exact point it was defined in the template. The text outside the tags is written to the output code file as-is; it mainly contains statements of the programming language that provides the functionality of the application for the target platform.

Using these model-to-code transformation templates, the programmer triggers the generative processes to create the source code files. After this generative process, the programmer can develop specific functionality on top of the generated code, in places previously determined by the software architect and marked with specific tags, to preserve these changes in future generations.

The programmer should also manage files supporting system compilation and deployment, to produce the application executables for the target platform. Whenever necessary, he performs debugging tasks and applies corrections on the code. The bugs occurring in the generated code outside the marked places should be reported to the software architect and must be fixed directly in the code-templates, to guarantee correctness of future generations.

3 ProjectIT-Studio CASE tool

ProjectIT-Studio is an integrated environment that supports some of the most important tasks of the software development life cycle, such as requirements specification, architecture definition, system design and modelling and code development. 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 profile definition.

ProjectIT-Studio consists of an orchestration of plugins for Eclipse.NET as illustrated in Fig. 5. 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 [27]. The ProjectIT-Studio tool provides three main plugins, which are discussed in the following subsections.

3.1 ProjectIT-Studio/Requirements

The ProjectIT-Studio/Requirements plugin involves two main components. The first one, PIT-RSL Parser, contains all of the logic responsible for the early stages of natural language-processing activities, such as the Structural

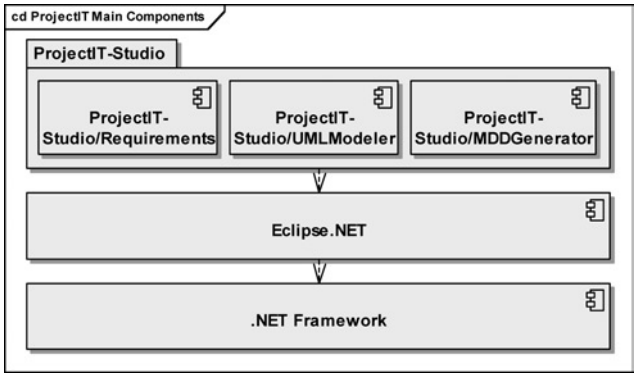

Fig. 5 *ProjectIT-Studio main components*

Parser (SP) and the Fuzzy-Matching Parser (FMP), which can operate in standalone mode. The second component, PIT-RSL Editor, aggregates all of the logic related with the Eclipse.NET plugin, namely all the functionality associated with the text editor and GUI components that provide on-the-fly feedback for assisting the requirements text editing activity.

The ProjectIT-Studio/Requirements plugin encapsulates text editor behaviour and the base classes responsible for the plugin initialisation, logging and termination. It contains the PIT-RSL Editor and a comprehensive set of tree views which provide a coherent set of IDE-like features that extend the base platform with specific behaviours to support the PIT-RSL requirement specifications and validation activities.

The PIT-RSL Editor supports the vision of a tool for writing 'requirements documents', like a word processor that detects and gives feedback of errors violating the requirements language and grammar rules [22]. The PIT-RSL Editor implements all of the required features regarding the requirements specification process, namely: syntax-highlighting, auto-complete, on-the-fly error checking with text annotation and other kinds of graphical annotations such as tasks and bookmarks, giving instant feedback on most syntactic and semantic problems or avoids them entirely. As illustrated in Fig. 6, the ProjectIT-Studio Requirements Perspective offers a wide set of potentially useful views to support requirement engineering activities [28]. The main element at the middle area corresponds to the PIT-RSL Editor, which offers common text editor features. This main area is surrounded by several views, namely Navigator, Outline, RSL Concepts, Tasks, Properties, Bookmarks, RSL Console, RSL Import Templates, RSL Optimal Parsing Trees and RSL TS Rules views.

The Navigator View allows the user to access and manage all the contents of the current project, through a tree view; exploring the structural hierarchy of organisation and storage of the artefacts available (such as files and folders).

At the bottom, the Outline View enables the user to navigate and have instant access to all requirement document sections through a simple click, since this view presents the requirements document structural hierarchy captured by the Structural Parser component.

The RSL Concepts View presents all concepts captured, grouped according to the PIT-RSL metamodel (Fig. 2), namely: Actors, Entities and Operations. This tree view also emphasises inheritance relations (i.e. generalisation–specialisation) between concepts of the same metaclass. The lower pane of the same view presents detailed information about the selected concept in the upper pane, such as entities' properties.

The reader can also note the Eclipse.Net typical Tasks, Properties and Bookmarks views, which were adapted to the ProjectIT-Studio/Requirements plugin, allowing the user to list all reported error/warning debug messages (with direct links to their source location), view properties of the current file, and list all available bookmarks and navigate through them.

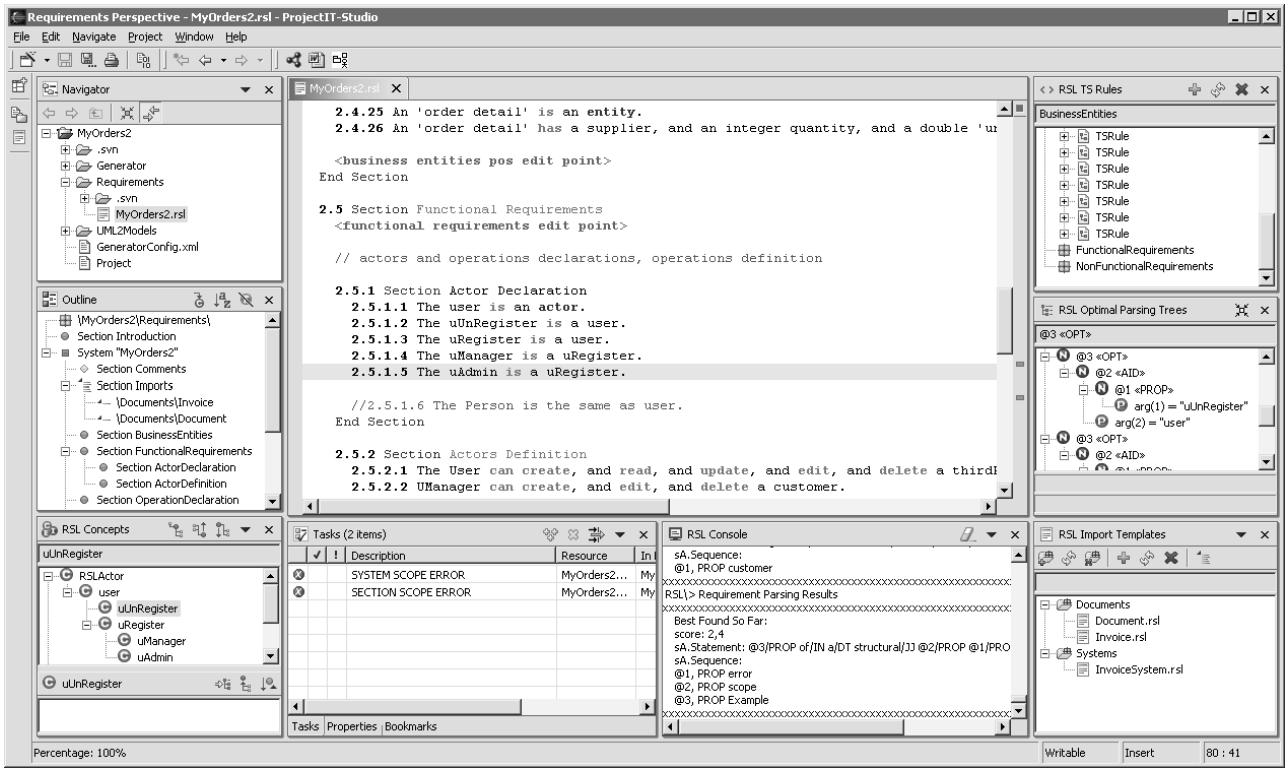

Fig. 6 *ProjectIT-requirements screenshot*

The RSL Console View continuously displays the data from the background parsing log, which provides relevant information and metrics for the advanced user, the RE.

At the bottom right corner, the RSL Import Templates View allows the management of the common repository of reusable artefacts. Each template available in this library can be easily included in the current requirements document by a simple click of a button. Moreover, all file system changes are immediately reflected in this view, allowing the user always to have the most recent view against the shared repository.

At the top, the RSL Optimal Parsing Tree View presents the Abstract Syntax Trees (AST) captured by the Fuzzy Matching Parser for each statement, according to the available set of Template Substitution (TS) rules. It presents the terminal (surface) elements and the non-terminal (abstract) elements identified during the parsing process.

In the last view, at the upper right corner, the RSL TS Rules View, allows the Architect to manage the available set of TS rules, namely it offers a window form to add and edit existing TS rules.

Finally, the ProjectIT-Studio/Requirements plugin can be customised through a specific Preferences page.

The ProjectIT-Studio/Requirements plugin uses a standard compliant UML 2.0 implementation to create a model that is later passed to the ProjectIT-Studio/UMLModeler. This means that this plugin provides the requirements-to-model transformation, with the following mappings: (1) entities are mapped into UML classes, preserving their inheritance relationship; (2) entity properties are mapped into the respective UML class's attributes; and (3) actors are mapped into UML actors' hierarchies. All of these UML elements can be later redesigned in one or more UML class or use case diagrams, respectively.

3.2 ProjectIT-Studio/UMLModeler and ProjectIT-Studio/MDDGenerator

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

ProjectIT-Studio/UMLModeler: As Fig. 7 illustrates, the ProjectIT-Studio/UMLModeler is a tool that allows designers to create visual models using the UML 2.0 language [29]. This tool also presents two relevant features: (1) UML Profile definition mechanism; and (2) model-to-model transformation mechanism.

A profile definition consists of two separate parts: the syntax and the semantics. The syntax is defined by associating stereotypes with images, which is the typical mechanism recommended by OMG and by which most UML-modelling tools designed. This association between stereotypes and images is done entirely within the tool. For the semantics definition, the tool supports the graphic

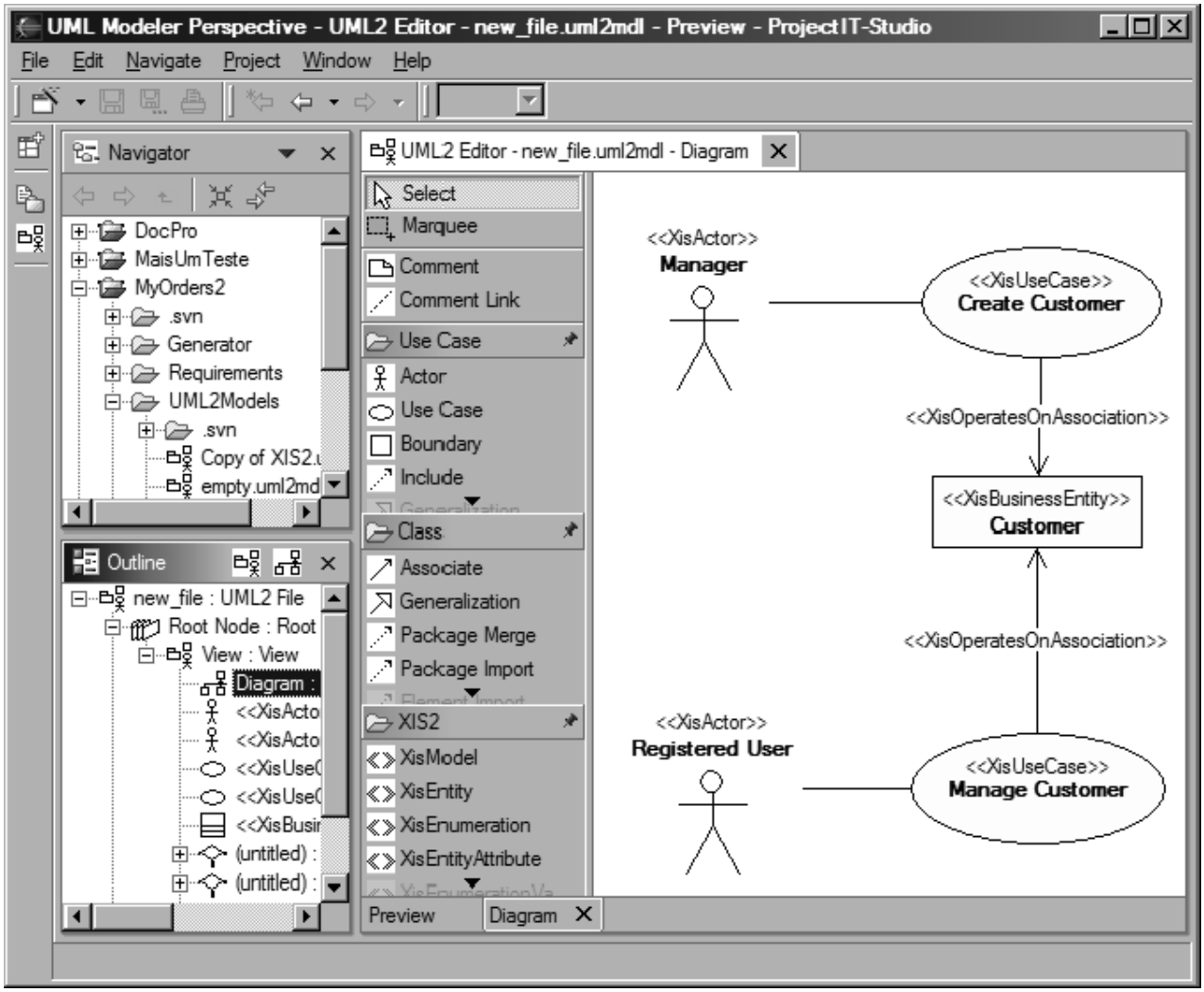

Fig. 7 *Usage scenario of the ProjectIT-Studio/UMLModeler*

specification of extensions between Stereotypes and UML metaclasses, using a mechanism similar to the one provided by the Enterprise Architect [30]. The tool also supports additional profile semantics through external .NET assemblies, containing code to validate the model in terms of the profile. This allows the tool to assure continuous validation of the model being created, according to the UML semantics as well as the profile semantics.

The designer creates the models based on a given UML profile, for example the XIS UML profile. These models can be used in generative processes (specified by the programmer, as the next subsection describes) to generate system artefacts according to specific software architecture. Fig. 7 illustrates a usage scenario of ProjectIT-Studio/UMLModeler, with some XIS stereotypes applied.

The model-to-model transformation mechanism allows the manipulation of a UML model by any ProjectIT-Studio plugin. This mechanism can be used to provide a UML model as input to a plugin that can process it, and to alter the UML model itself (e.g. by adding new UML elements to the model, based on its current contents). These model transformations are currently specified in C# source-code, involving three steps of model manipulation: (1) receiving a model; (2) parsing it; and (3) changing it, if necessary. Any changes to the model are detected by ProjectIT-Studio/UMLModeler, which immediately updates the visual representation of the model in the ProjectIT-Studio graphical interface.

The user applies model-to-model transformations to the currently open model in ProjectIT-Studio/UMLModeler by selecting the appropriate item from the model's context menu. This mechanism also allows developers to provide validation methods which are used to determine whether a certain transformation can be applied to the model; ProjectIT-Studio/UMLModeler only displays in the model's context menu the transformations in which validation methods return a successful response, which helps to avoid potential problems in transformations.

ProjectIT-Studio/MDDGenerator: ProjectIT-Studio/MDDGenerator is the plugin responsible for the generation of system artefacts. This plugin receives a generative process and produces the corresponding software artefacts, for example source code or documentation files. A generative process is a configuration file that specifies the name, model and software architecture of the final application (see Fig. 8).

A model is an abstract representation of a software system, created by the designer based on a UML profile, such as the XIS profile. Subsystems are divisions of the model classified by a category, such as entities, actors or interaction spaces.

Software architecture is defined by an integrated set of model-to-code transformation templates specifically developed for a concrete target platform, such as ASP.NET, Java/JSP, Windows Forms, Java/J2ME. These templates belong to a specific category such as database, data services, user interfaces and others. Fig. 9 shows a part of a template, written in the PIT-TSL language, which specifies the transformations from XIS models into SQL-DDL scripts.

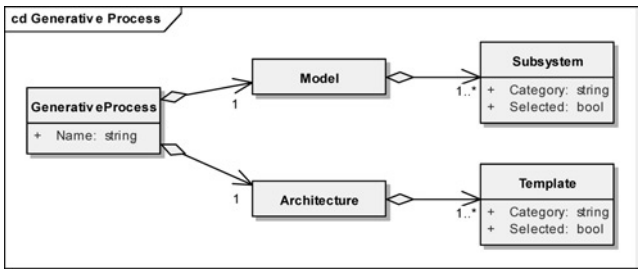

Fig. 8 *Overview of generative process concepts*

PIT-TSL stands for 'ProjectIT Template Specification Language' and allows the definition of textual templates which resembles closely the code to be generated. The template engine transforms templates into source code, compiles and executes them at runtime, and finally collects the output into the specified file artefact (see Fig. 10). The developed template engine is an extension of the 'TemplateMaschine' framework (see http://www.stefansarstedt.com/templatemaschine.html). It supports new directive operations that help specify richer templates in a scripting language style with syntax similar to ASP.NET. Templates promote iterative development because they can be easily derived from examples or refined as needed. Additionally, templates are independent of the target language and simplify the generation of any textual artefacts, including documentation [9].

Fig. 11 shows the ProjectIT-Studio/MDDGenerator editor that allows the definition and the configuration of generative processes. Through this editor it is possible to individually select models, subsystems and templates to allow different generation approaches, for example complete, partial or incremental generations.

The ProjectIT-Studio/MDDGenerator works as suggested in Fig. 10: it reads the generative process, loads the model (with selected subsystems), then passes it as input to each selected template which is then processed by the template engine to produce the source code or documentation files.

4 MyOrders2 case study

This section presents a practical application of the concepts previously discussed in the paper, emphasising the benefits of the ProjectIT approach and its support given by the ProjectIT-Studio CASE tool. We guide this proof-of-concept discussion through a simple case study, called MyOrders2, which is a simple management information system, specified and designed in a platform-independent way, and developed for two concrete platforms, desktop and web-based.

4.1 Informal description

MyOrders2 is an information system that allows the storage of relevant information for any hypothetical organisation. Its purpose is to support the typical interaction activities between clients and suppliers, in particular the activity of ordering products.

The MyOrders2 system includes the following main entities: (1) ThirdParty, which represents an abstract entity corresponding to the concept of a typical enterprise having a name, a list of Affiliates, and a list of Orders; (2) Affiliate, which corresponds to an enterprise's representative member, and has a list of attributes, essentially contact (address, phone and fax) and the category of affiliation; (3) AffiliateType, which specifies the relationship between the ThirdParty and its Affiliate; (4) Customer is a ThirdParty that represents the typical buyer role (an entity that requests an order) and it is characterised by a list of Market types, importance and flexibility attributes; (5) Market, which matches the real market concept representing the place, real or virtual, where buyers and sellers interact to exchange goods and

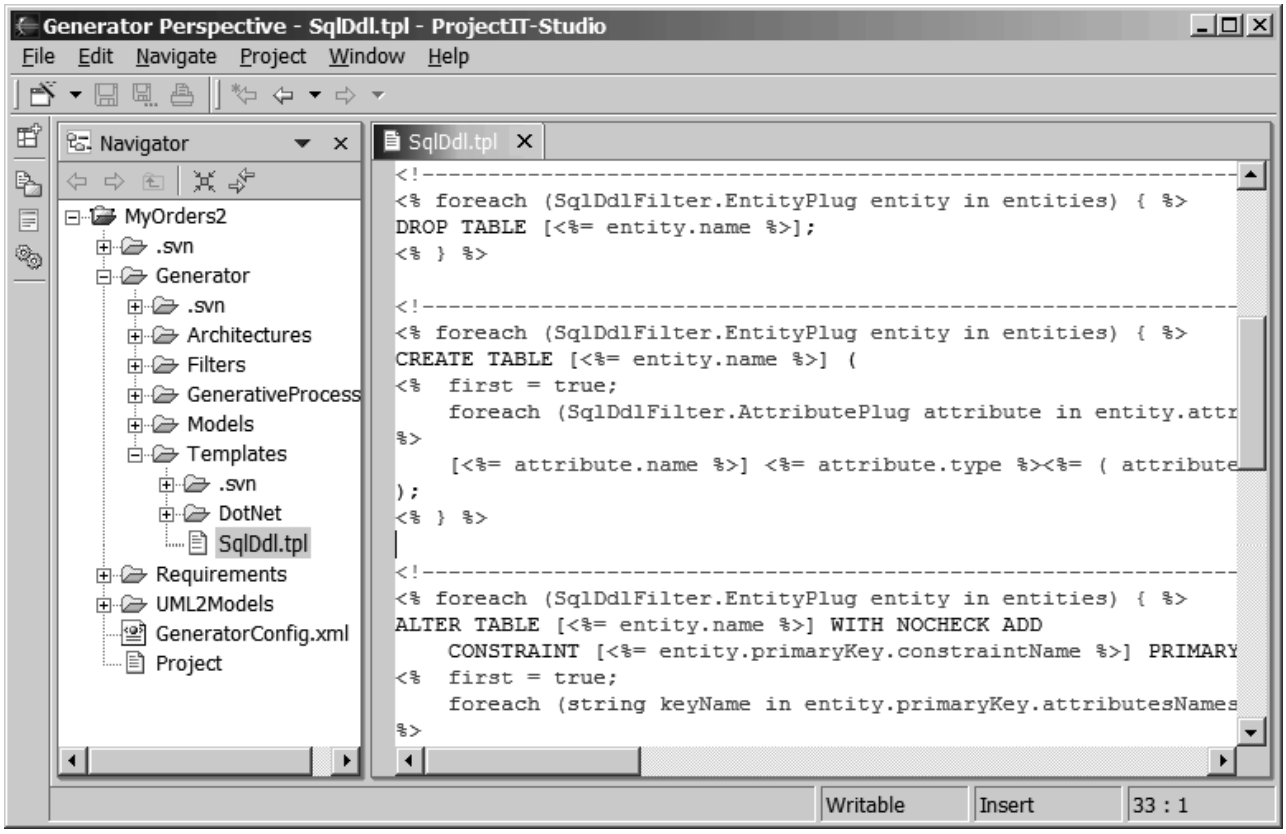

Fig. 9 *Template editor*

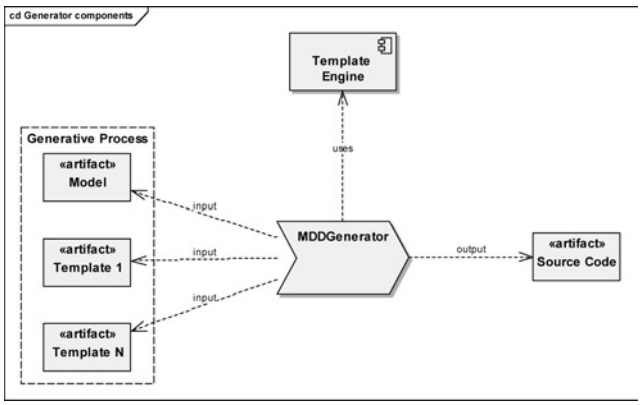

Fig. 10 *Template Engine overview*

services; (6) Supplier is a ThirdParty and clearly maps to the entity that provides Products, being described by a Products catalogue and two evaluation metrics: quality and responsiveness; (6) Product, which represents a good or service, and is described by attributes such as name, price, stock items and production state; and (8) Order, which represents a Product's request and is described by an identifier, a set of dates, values and order details.

The MyOrders2 system supports different roles, which are associated with several use cases, in accordance with different permissions (e.g. query, create, delete and edit business entities). For example, the system should support the following roles: (1) registered user, who can be either a Customer or Supplier (or even both), is able to create, query and cancel orders or only view requested orders and update Product lists; (2) manager, who manages information about the Customers, Suppliers and Orders entities; and (3) administrator, who manages user accounts.

4.2 System requirements phase

This section presents the textual specifications used to capture the MyOrders2 system from the requirements informally described in the previous section. Each frame presented in the following reveals an important requirements specification fragment, showing different aspects of the PIT-RSL language.

From a user point of view, the conversion between the previous textual specification into a PIT-RSL specification can be achieved using two different approaches. The first option involves specification of a process without the support offered by ProjectIT-Studio/Requirements, using an ordinary text editor. The second possibility corresponds to a powerful and interactive process of requirements specification, in which the user is assisted by ProjectIT-Studio's error reporting mechanism and a comprehensive set of views while writing, since the input is constantly validated against the specification model, thus providing support for a better and iterative requirements specification and validation process.

The frame below presents the typical structure of a PIT-RSL requirements specification document, revealing several types of sections and the hierarchical relations between them (Fig. 12).

The first specification's section is used to provide an overview of the requirements document's purpose and contents. To achieve this goal the user must use the specific introduction section, beginning with the tokens SECTION INTRODUCTION (Fig. 13). It can be very useful for providing a brief description during a document's search and cataloguing operations, to facilitate its retrieval and contents reuse, or even to include a document's overview in an automatically generated report.

After defining the SECTION INTRODUCTION, the user must specify the main application unit region, called the SYSTEM

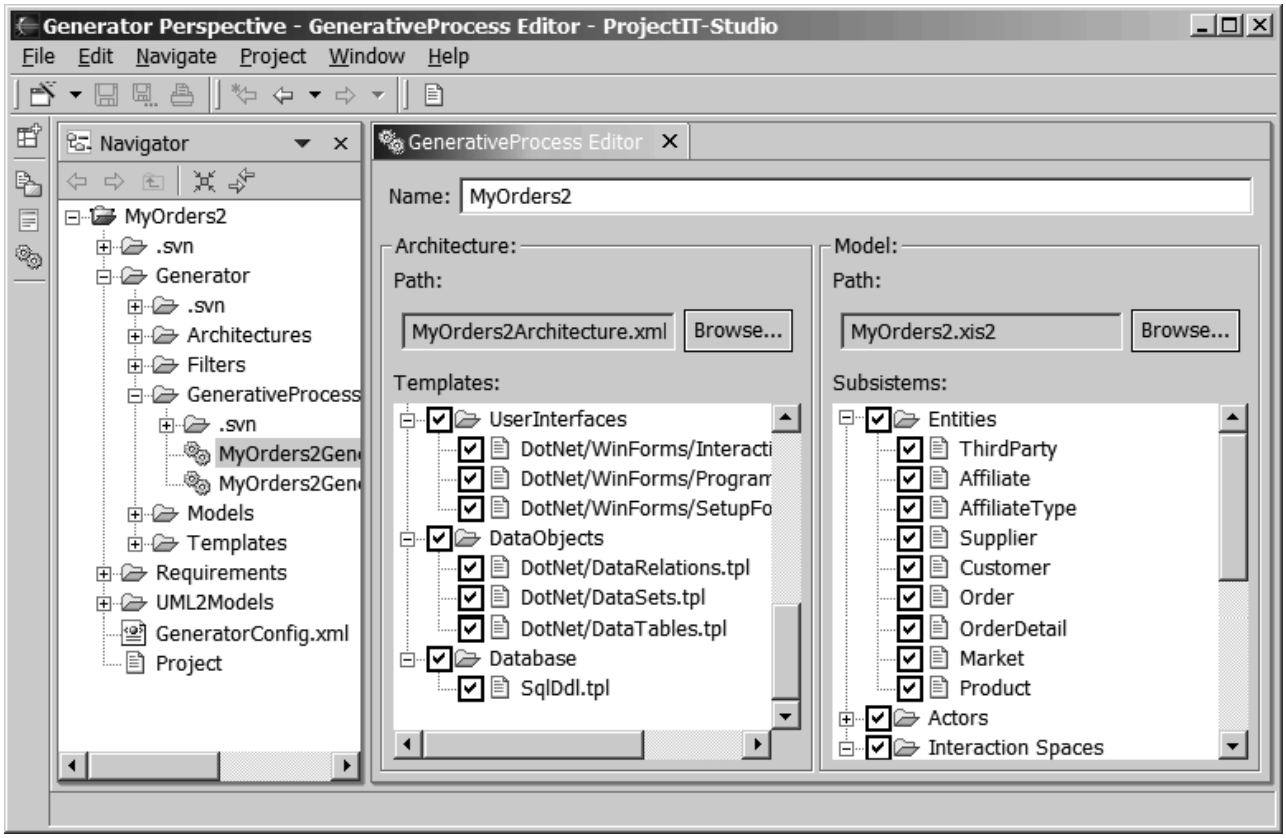

Fig. 11 *Usage scenario of the ProjectIT-Studio/MDDGenerator*

```
1 Section Introduction (...) End Section

2 System "SystemName"
        2.1 Section Comments (...) End Section
        2.2 Section Imports (...) End Section
        2.3 Section Business Entities (...) End Section
        2.4 Section Functional Requirements
                2.4.1 Section Actors Declaration (...) End Section
                2.4.2 Section Actors Definition (...) End Section
                2.4.3 Section Operations Declaration (...) End Section
                2.4.4 Section Operations Definition (...) End Section
        End Section
        2.5 Section Non-Functional Requirements
                2.5.1 Section Security (...) End Section
        End Section
End System
```

Fig. 12 *Structure of PIT-RSL requirements specification document*

context, followed by the system's name (in this case 'MyOrders2'). At this level, as well as in other inner nested levels, users can specify as many SECTION COMMENTS as they want because they are neither parsed nor stored, since they only provide an auxiliary annotation mechanism for enriching the specifications description (Fig. 14). Besides the SECTION COMMENTS, at the SYSTEMS scope level, the user can specify a SECTION IMPORTS (only one per SYSTEM region). This specific section type is used to declare import entries, which, when parsed by the FMP, will make it invoking the SP import mechanism to include its contents in the document's underlying concepts model. For example, if the 'use' keyword is used, then the imported system or sub-system will be available as a monolithic component. Otherwise, if the 'import' keyword is used, then the imported specifications will populate the system's model, allowing the user to able to override them by simply writing a definition for the same concept.

The SECTION BUSINESS ENTITIES is of extreme importance because it is where one can define the domain model of the system currently under specification.

The next frame presents the textual entries (surrounded by the '<' and '>'characters) that can be used to introduce variability points to requirements documents (implementing a template-based reuse mechanism with edit points), hence providing specification placeholders like a template form (Fig. 15). These edit point entries, typically located at the beginning (pre-edit point) and at the end (post-edit point) of each relevant section, allow the extension of requirements document sections in a controlled way, while keeping the rest of the specifications unchanged.

The next frame presents the definition of user roles involved in MyOrders2. They are specified as a hierarchy of actors (an actor can specialise or generalise other actors), with the possibility of specifying for each role the set of authorised features regarding the involved business entities (Fig. 16). Currently these include a set of predefined actions, such as create, read, update and delete.

When the RE finishes the specification of the requirements document, a corresponding UML 2.0 model can be produced automatically. Currently, only the domain model (class diagram) and actors hierarchy (a part of use case diagrams) can be generated. This process is triggered by the 'Generate UML2 Model' button, available in the ProjectIT-Studio/Requirements Toolbar. This action subsequently invokes the ProjectIT-Studio/UMLModeler tool for further refinement of the generated UML model (more detail in the next section).

4.3 System design phase

The initial tasks of the 'System Design Phase' vary according to the approach followed. The first one is based upon the work of the System Requirements Phase, described in the previous section, which resulted in a preliminary version of the model, produced automatically through PIT-RSL-to-Model transformation techniques. The second

```
1 Section Introduction
       Description of a simple order management software information system.
       Main objective: to provide a simple case study for proof-of-concept of the ProjectIT approach and
       tools.
End Section
```

Fig. 13 *Tokens section introduction*

```
2 System "MyOrders2"
       2.1 Section Comments
              The will describe the domain view, corresponding to the static/structural view of the
              system.
       End Section
       2.2 Section Imports
              - use \Documents\Document.
              - use \Documents\Invoice.
       End Section
(...)
```

Fig. 14 *Section comments in MyOrders2*

```
2.4 Section Business Entities
    <business entities pre edit point>
    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 'affiliate type', and a 'contact name', and an address, and a 'contact
          title'.
    2.4.5 An address is composed by a city, and a 'postal code', and a country, and a phone, and a fax.
    2.4.6 An 'affiliate type' has a description.
    2.4.7 A supplier is a thirdParty.
    2.4.8 Each supplier has a list of 'order details', and an integer quality, and an integer
          responsiveness.
    2.4.9 A customer is a thirdParty.
    2.4.10 A customer is the same as a client.
    2.4.11 Each customer has an integer importance and an integer flexibility.
    2.4.12 Each customer has one or more markets.
    2.4.13 Each market has a list of markets.
    2.4.14 A market is an entity.
    2.4.15 A market has a name.
    2.4.16 Each supplier provides a list of products.
    2.4.17 Each product is provided by one or more suppliers.
    2.4.18 A product is an entity.
    2.4.19 A product has a 'product name', and a double 'unit price', and an integer 'order units', and an
           integer 'stock units', and boolean 'discontinued'.
    2.4.20 A customer emits a list of orders.
    2.4.21 Each order is emitted by one customer.
    2.4.22 An order is an entity.
    2.4.23 An order has an 'order code', and an integer 'order number', and a date 'order date', and a
           date 'required date', and a date 'shipped date'.
    2.4.24 An order is composed by a list of "order details".
    2.4.25 An 'order detail' is an entity.
    2.4.26 An 'order detail' has a supplier, and an integer quantity, and a double 'unit price', and a
           double discount.
    <business entities post edit point>
End Section
```

Fig. 15 *Textual entries for requirements documents*

possibility involves more modelling efforts, as all models have to be designed from scratch, with the information gathered from stakeholders. The first alternative is also desirable because it allows the acceleration of the design task, and effectively supports the goal of requirements-to-models traceability.

Thus, depending on the choice above, the Designer efforts are substantially different, but in any case they must always customise the model to reflect issues that were not specified in the requirements. For instance, for interactive systems, nothing was defined from the UI design and navigation point of view. To edit this model, the Designer opens it with the ProjectIT-Studio/UMLModeler. Besides the common modelling features which are usually found in traditional UML modelers, the Designer can also apply one or more UML profiles (such as XIS) to the model to facilitate its interpretation by ProjectIT-Studio/MDDGenerator's generative templates.

In the MyOrders2 case study, we can use the XIS profile to further customise the model obtained from the ProjectIT-Studio/Requirements. Fig. 17 presents the UML diagram of the MyOrders2 system's domain model. It shows the core concepts and the respective relationships designed according with the UML class diagram and the XIS stereotypes. Fig. 18 presents the UML diagram of the MyOrders2 system actors' model, after applying the XIS profile. It shows the roles that a user can assume, by means of an authentication mechanism, while interacting with the MyOrders2 system.

Still, according to the XIS, the Designer should define the Navigation Space with all Interaction Spaces that represent screens by which the system's user will interact. The

```
2.5 Section Functional Requirements
    // actors and operations declarations, operations definition
    2.5.1 Section Actors Declaration
        2.5.1.1 The User is an actor.
        2.5.1.2 The URegistered is a User.
        2.5.1.3 The UManager is an URegistered.
        2.5.1.4 The UAdministrator is an URegistered.
        2.5.1.5 The Person is the same as User.
    End Section
    2.5.2 Section Actors Definition
        2.5.2.1 UManager can create, and edit, and delete a customer.
        2.5.2.2 URegistered can create a customer.
        2.5.2.3 The User can create, and read, and update, and edit, and delete a thirdParty.
    End Section
End Section
```

Fig. 16 *User roles involved in MyOrder2*

Navigation Space is responsible for representing all the possible navigation flows between the system's Interaction Spaces; Fig. 19 presents the diagram of the MyOrders2 Navigation Space, in which all the MyOrders2 Interaction Spaces are connected by navigation flows (modelled as UML associations with an applied stereotype, whose tagged values provide further information about that navigation).

Interaction Spaces themselves are modelled using Classes to which XIS stereotypes are applied. However, unlike other traditional modelling approaches, this takes advantage of the graphical information provided by the diagram, by using the UML Diagram Interchange specification [31], allowing the Designer to have a more realistic idea of the generated screen appearance. For a more detailed description of how to model a system using the XIS UML profile, please refer to [25, 26].

Fig. 20 shows the diagram of the system's main screen model. Unlike most of the other Interaction Space models, this diagram does not specify the layout of a screen *per se*,

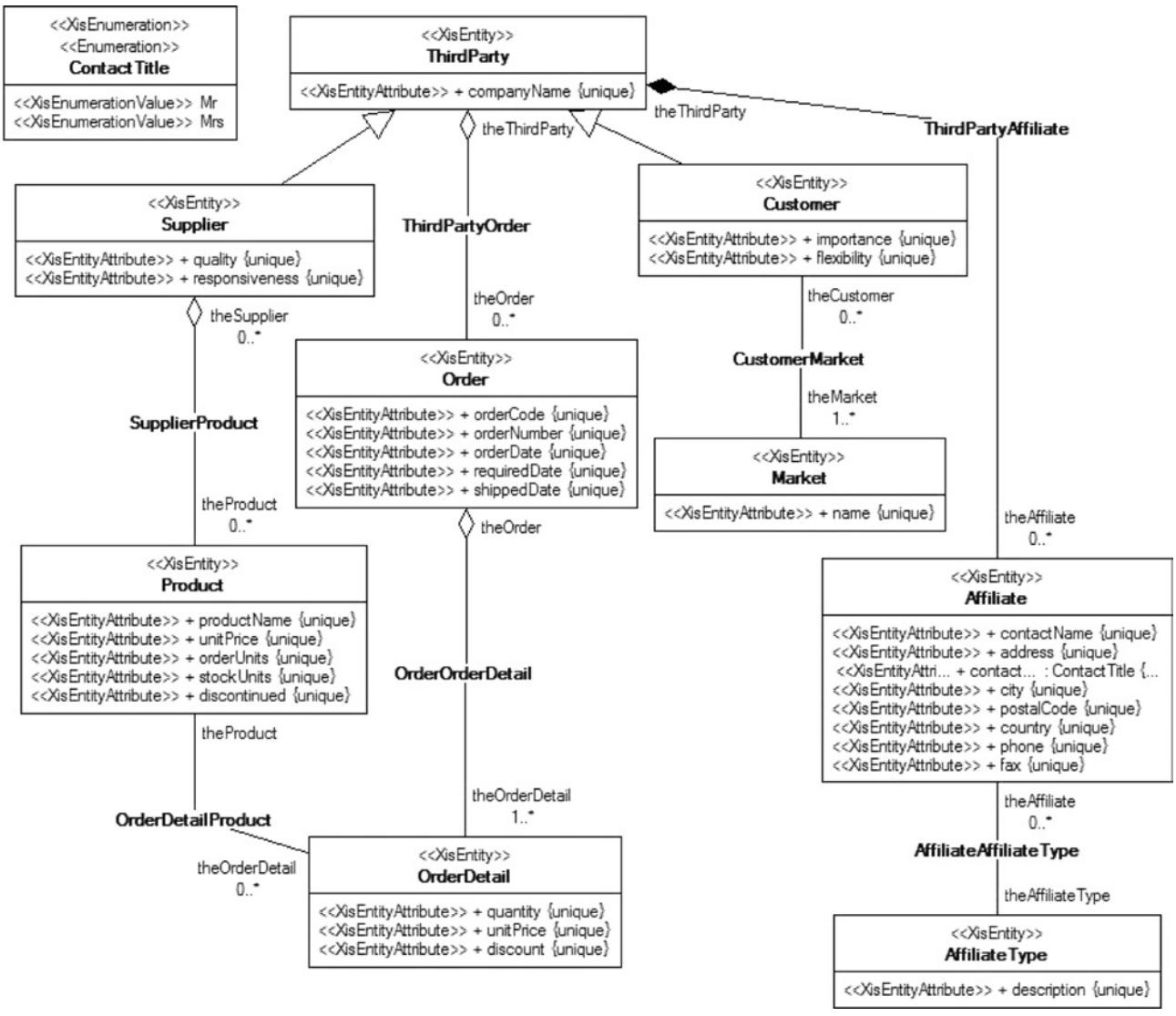

Fig. 17 *Domain view of MyOrders2 (using the XIS profile)*

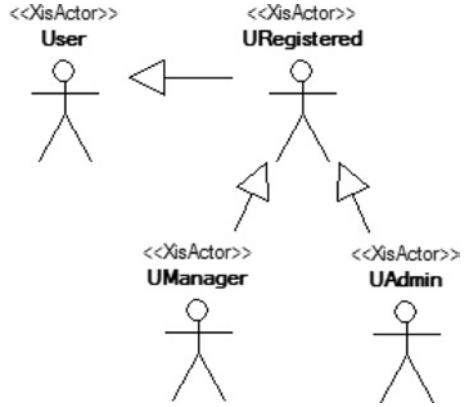

Fig. 18 *Actors view of MyOrders2 (using the XIS profile)*

but rather the generic layout with a set of menus that will be presented to the user on the generated application's first screen.

Fig. 21 shows the 'Customers' Interaction Space model, which involves the management of a list of Customers. Besides listing the existing Customers, this Interaction Space also allows the creation, editing and deletion of Customer entities.

Finally, Fig. 22 illustrates the 'Order' Interaction Space, which presents the details of an Order according the Master–Detail interaction pattern.

4.4 System development phase

Following the ProjectIT approach, the 'System Development Phase' is mainly performed automatically, based on code generative techniques. Nevertheless, even when applying MDD and code generation techniques, the Software Architect, Programmer and Tester roles are still necessary and crucial for the final deployment of the system.

After modelling the various views of the MyOrders2 system, the Programmer can generate the corresponding artefacts for a specific deployment platform. This task involves the following steps: (1) select the architecture that represents the deployment platform; (2) define a generative process; and (3) trigger the code generation process.

The architecture specification is supported by the ProjectIT-Studio's Architecture Editor, as illustrated in Fig. 23. It involves the creation of an architecture file, and the addition of specific software templates previously defined by the Software Architect. Obviously, architectures can and should be re-used through different projects.

After the architecture's definition, the Programmer defines a generative process, and edits it through the ProjectIT-Studio's Generative Process Editor (illustrated in Fig. 24). This generative process will establish an association between an architecture (e.g. Windows Forms for the .NET Framework) and the involved model, in this case study, the MyOrders2 model.

After the definition of the generative process, the corresponding artefacts can be generated by clicking the 'Generate' button supported by ProjectIT-Studio's Generative Process Editor.

Finally, the Programmer can extend and or customise the generated code to develop features and requirements not well captured at the design phase, such as time constraints or business rules not addressed by the used language. These code extensions can be developed by using programming language mechanisms such as inheritance, stub methods, or C# 2.0's partial classes. Figs. 25–27 show

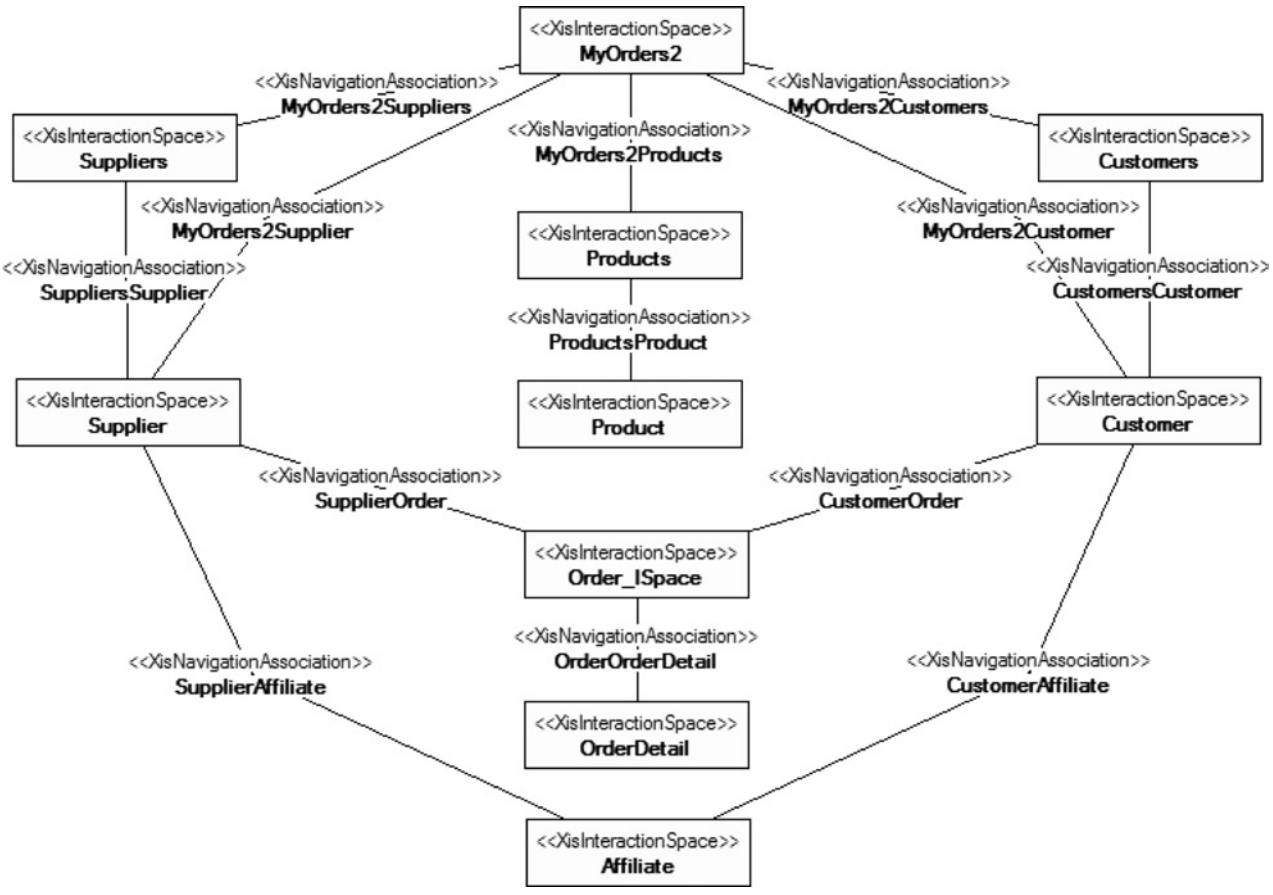

Fig. 19 *Navigation view of MyOrders2 (using the XIS profile)*

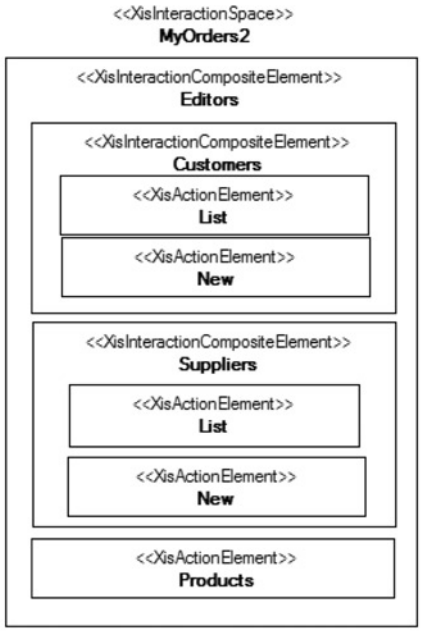

Fig. 20 *MyOrders2 main screen as a XIS Interaction Space*

some screenshots of the MyOrders2 case study corresponding, respectively, to the models illustrated in Figs. 20–22, deployed to two different platforms, a desktop and a Web-based platform.

5 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) our community has been pursuing the idea of code generation, but the best that we reached, for a long time, was the generation of database scripts. In terms of modelling, many tools have been created, but they concentrate on visual issues, and not on the automation of tasks of the development process. For example, many requirements tools just deal with requirements as information that is managed like any other concept managed by an information system. In these tools, such as Telelogic DOORS, IBM Rational RequisitePro, or Borland Caliber, requirements are just 'pieces of text', managed as records on top of some relational database, without any syntactic or semantic validation or introspection as it is proposed in ProjectIT. Another example is provided by the OMG SysML, which is a general-purpose graphical modelling language-oriented towards complex systems, covering several software development process activities and artefact types [32]. SysML is a UML 2 enhanced subset: on one hand it reuses part of UML 2 and, on the other hand, it provides some extensions to it. SysML introduces new diagram types, namely Requirements Diagram and Parametric Diagram types. Moreover, SysML modifies UML 2 diagram types, such as Activity Diagram, Block Definition Diagram and Internal Block Diagram. The four SysML core viewpoints are Structure, Behavior, Requirements and Parametrics. SysML adopts OMG standards, namely XMI, to ensure maximal interoperability between modeling CASE tools. Despite the fact that the SysML final specification was only recently approved, there was already an available tool support for SysML from several renowned industry vendors, namely ARTISAN, IBM, Sparx Systems and Telelogic. One of SysML's main benefits derives from the enhancement of the UML notation with semantic for disregarded modelling aspects, specially requirements and behaviour concerns [32]. It introduces a graphical element for representing text-based requirements and relates them to other model elements, through traceability mechanisms [33]. These requirement diagrams capture requirement hierarchies and requirement derivation, and satisfy and verify relationships, thus providing a bridge between the typical requirement management tools and the system models. However, this approach still suffers from an important drawback: it treats requirements as black-box elements [34], neglecting its natural language semantics in favour of enhanced traceability. This clearly is in contrast with our approach, which seeks to capture each requirement meaning and validate requirement specifications accordingly. Despite traceability issues not being our current major concern, we recognise that it is an extremely important feature to include in ProjectIT-Studio tools, in a crosswise manner.

Even in terms of research, there are few proposals, like ours, 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 modelling and model-checking techniques to produce a formal validation of the requirements written in natural language [35]. 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 are later used to provide different views and models to analyse this knowledge. Although Circe and

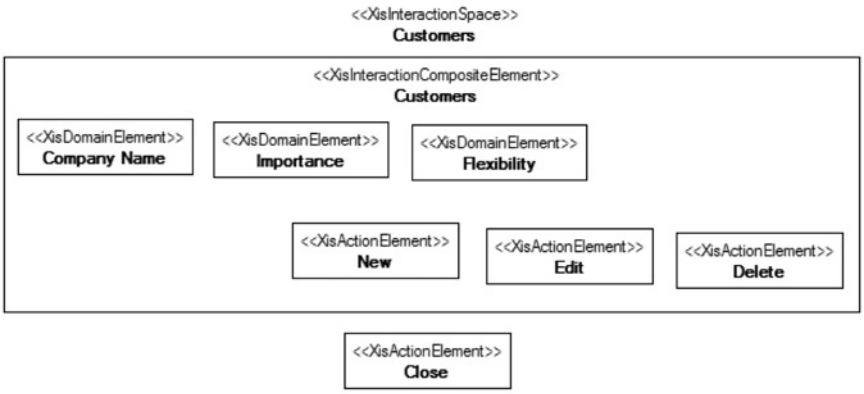

Fig. 21 *XIS model of the MyOrders2 'Customers' Interaction Space*

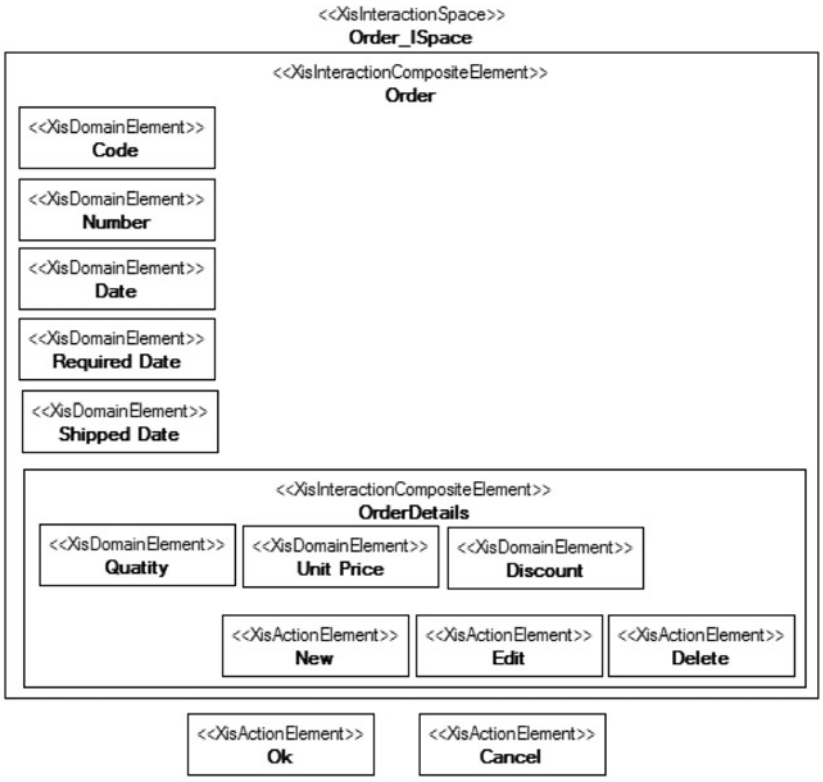

Fig. 22 *XIS model of the MyOrders2 'Order' Interaction Space*

ProjectIT-RSL have some similarities, there are many differences between them, namely in the architecture, concepts and algorithms used, and above all, in the strategy: the goal of CIRCE is requirements validation, and only recently moved to research about 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. The number of research projects in the area of requirements engineering is quite significant, and some of them have been under development for a while. For example, Attempto Controlled English (ACE), first described in [36] uses a controlled natural language to write precise

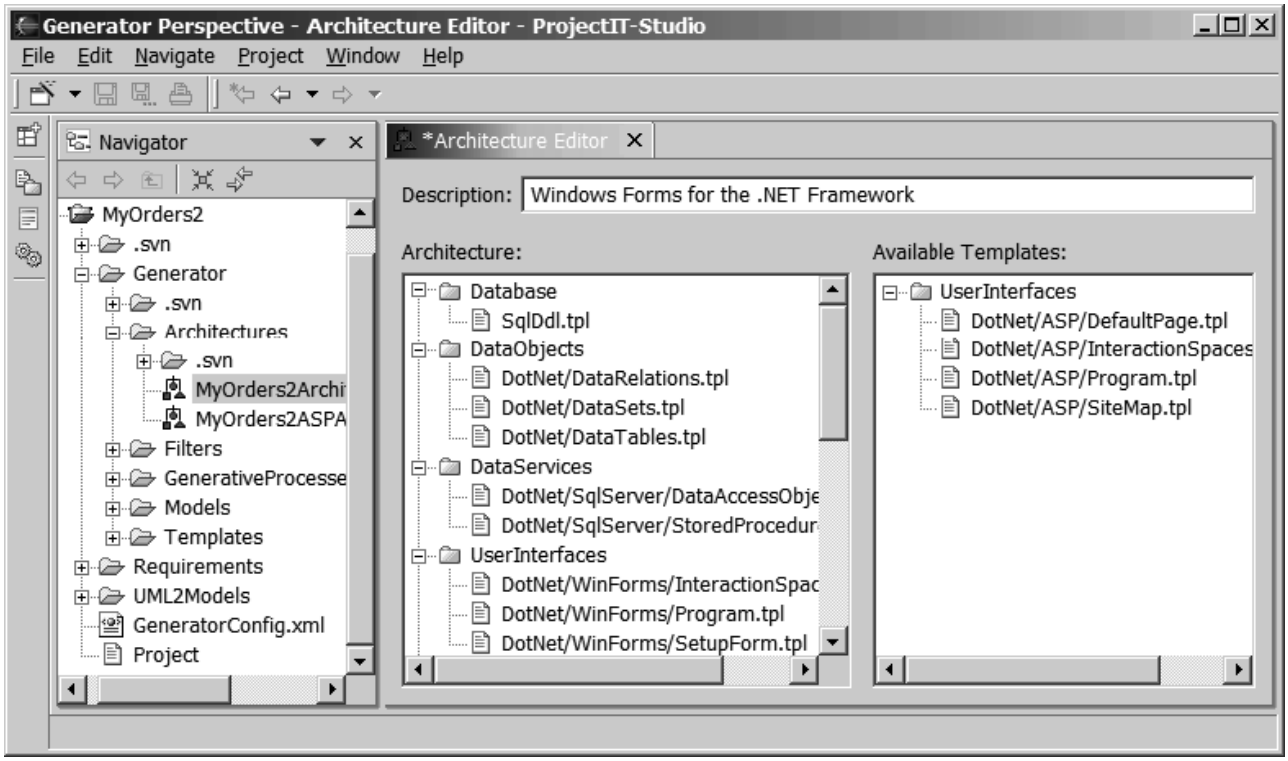

Fig. 23 *Screenshot of the ProjectIT-Studio Architecture Editor*

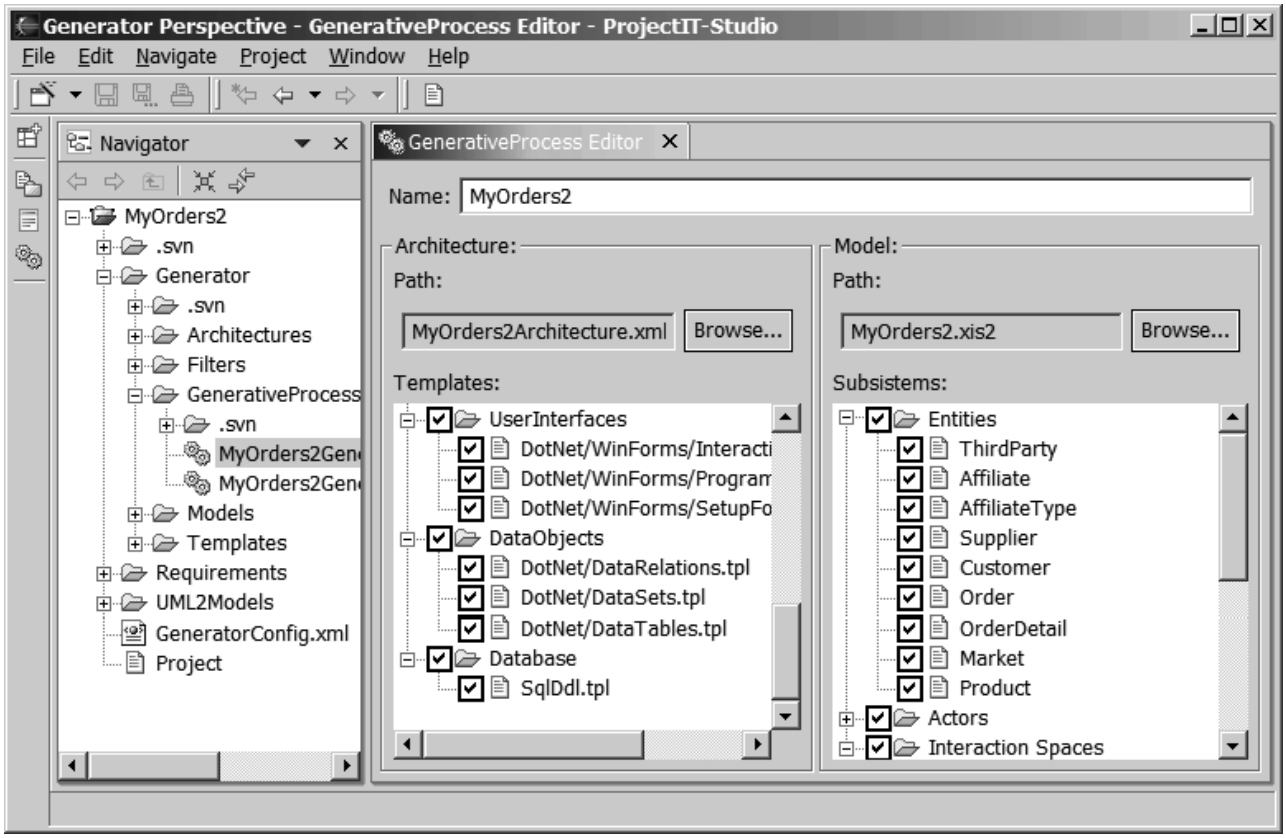

Fig. 24 *Screenshot of the ProjectIT-Studio Generative Process Editor*

specifications that, for example, enable their translation into a first-order logic similar representation (called DRS). Approaches as in [37] and [38] reduce the level of imprecision in requirements by using a limited number of sentence patterns to specify a requirement for a particular domain. Denger has also identified natural language patterns used to specify functional requirements of embedded systems, from which they developed a requirements statements metamodel

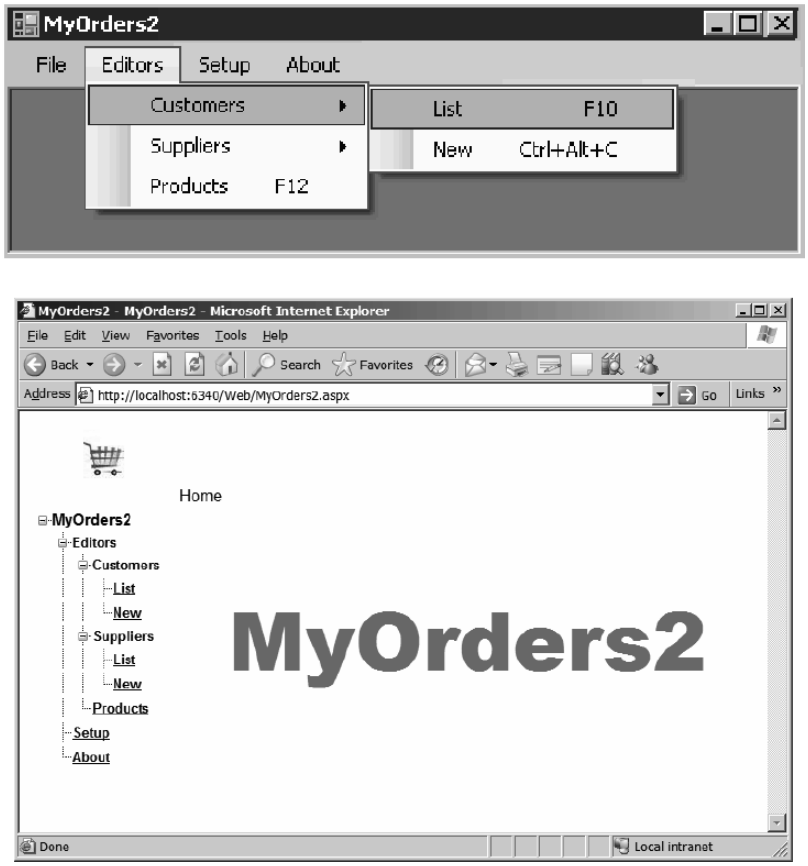

Fig. 25 *Screenshots of the MyOrders2 main screen*

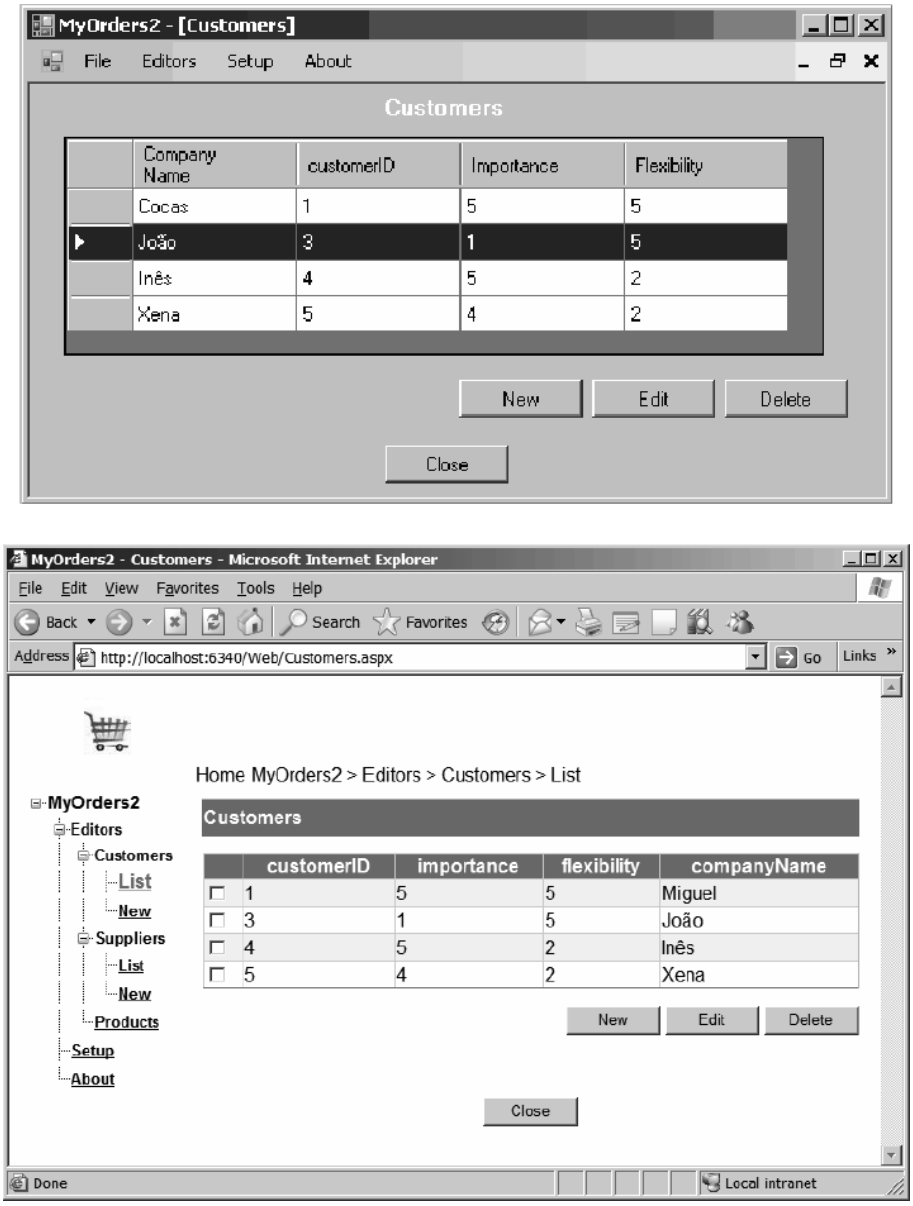

Fig. 26 *Screenshots of the MyOrders2 'Customers' screen*

[39]. Juristo *et al.* tried to formalise the analysis of natural language sentences to create precise conceptual model [40]. NL-OOPS [41] and LIDA [42] are systems that process natural language requirements to construct the corresponding object-oriented model.

In the area of MDE, there are some initiatives that propose a sequence of steps similar to the 'System Design Phase' and 'System Development Phase' of the ProjectIT approach, such as the MDA [7]. The ProjectIT approach itself is based on MDA, although the latter does not include natural language requirements specification, leading to the known gap between 'what the client wants/needs the system to do' and 'what the system really does'. Thus, the ProjectIT approach can be regarded as an instance of MDA combined with a domain-specific language that addresses requirements specification.

One of the advantages of the ProjectIT approach is that it is based on the UML standard, and allows the usage of any language derived from UML, such as the XIS UML profile, instead of using another Domain-Specific Language. In the context of interactive systems and user–interfaces modelling based on UML extensions, there are other initiatives besides XIS that should also be mentioned, in particular: WebML [12], OOHDM [13], OOWS [43], User-Experience (UX) [44], Wisdom [45], UMLi [46], UWE [47], OVID [48] and CUP [49]. The UX approach defines modelling elements for navigation design and discusses the transformations of UX UML models into code-level models, specifically for the Java Struts framework. The Wisdom and the UX approaches represent quite well navigation aspects with some similarities to the NavigationSpace View, but they don't define any models to represent each node of the user interface in an abstract way as in XIS. The Wisdom approach aims to maintain synchronisation between Wisdom and Canonical Abstract Prototypes [50], which represent each node of the user interface in an abstract way. However, Wisdom does not capture the relevant data of entities and business entities models, which are important in XIS to support model-to-model and model-to-code transformations. The UMLi approach proposes a profile to capture the conceptual, presentation and behaviour aspects of systems. The initiatives like UWE, WebML [12], OOHDM [13] or OOWS [43] have several presentation design similarities with the XIS's NavigationSpace and

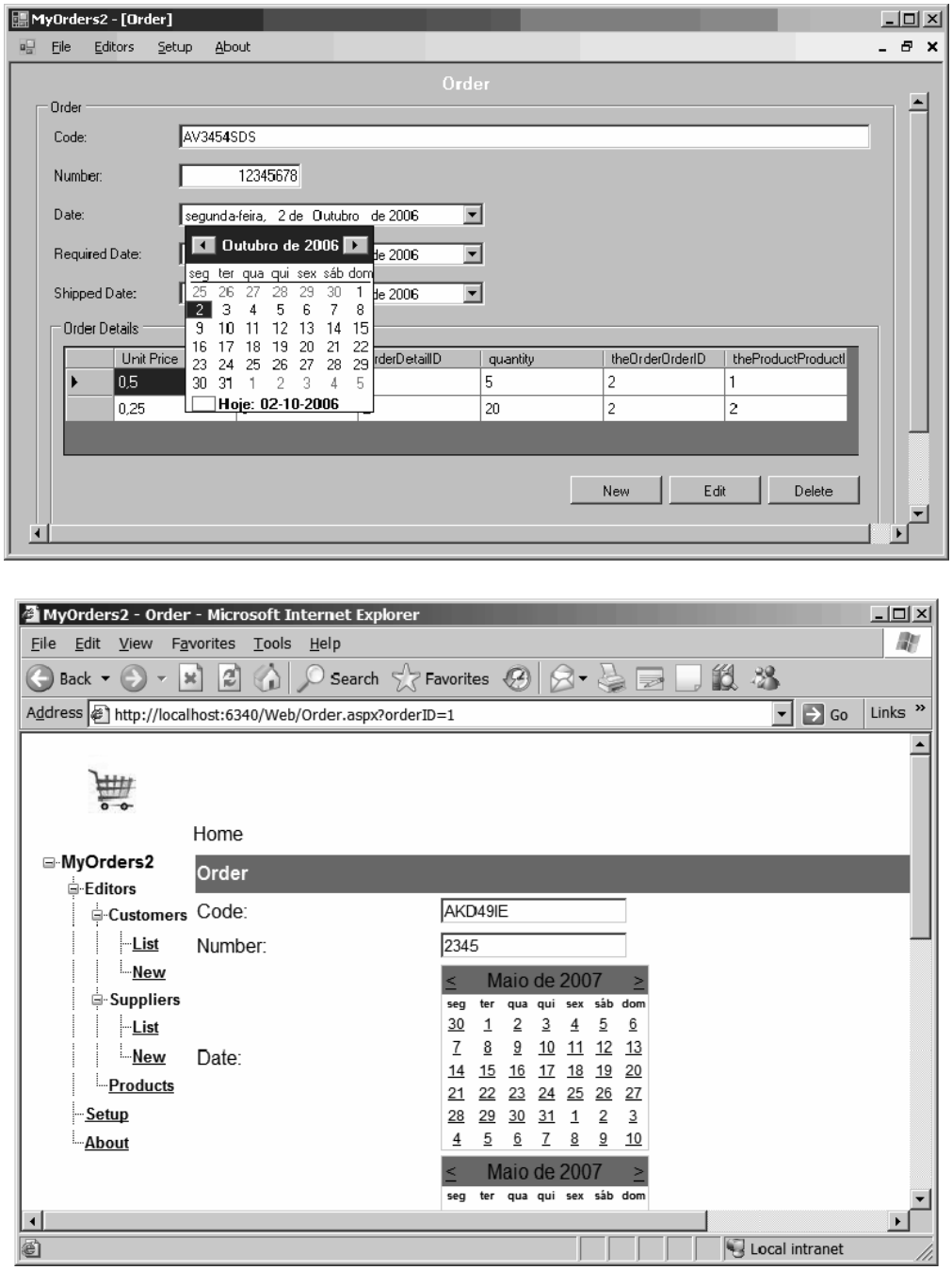

Fig. 27 *Screenshots of the MyOrders2 'Order' screen*

InteractionSpace views; however, they are particularly focused on modelling Web-based systems, and so do not support platform-independent models as proposed by the XIS language.

Microsoft itself announced the Dynamic Systems Initiative [51], an approach that integrates ideas and tools, with impacts at the hardware, systems, applications and process levels to ease 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 [52] that follows the ideas of software product lines and domain specific languages [53, 54]. 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 requirement specification at the beginning of the projects, whereas Software Factories typically do not address this issue in the requirements phase.

6 Conclusions and future work

This paper presents and discusses the innovative ProjectIT approach, and its main issues and challenges. ProjectIT integrates the contributions and best practices from two complementary areas: RE and MDE. Fig. 28 summarises the ProjectIT approach and emphasises the relationships between its languages, transformations and tools.

ProjectIT-Studio is an integrated environment with 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 most distinctive features of the ProjectIT initiative: requirements

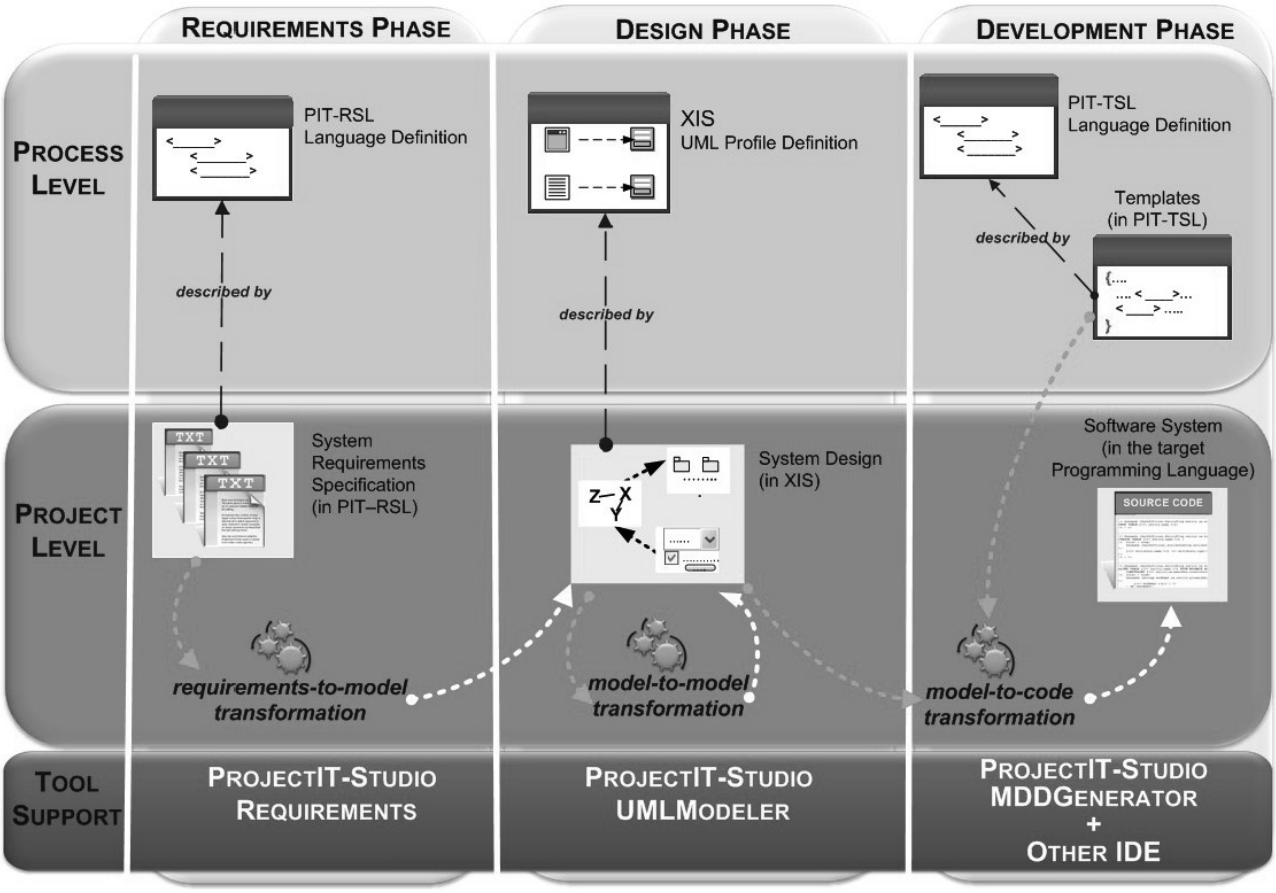

Fig. 28 *ProjectIT's languages, transformations and tools*

specification and management, requirements-to-models transformations, models definition, model-to-model and model-to-code transformations support.

The PIT-RSL language is the result of an empirical analysis of a series of requirements documents for interactive systems, culminating in a metamodel that captures both the most common natural language constructs (and their semantics) and the document structure. The PIT-RSL follows an innovative approach. It can be classified as a controlled natural language, since it seeks to gather the benefits from both worlds: it has expressive power, flexibility and ease of learning from natural language; it has the rigor and validation possibilities of formal approaches. Therefore our approach not only supports RE activities but also requirements specification and validation activities performed by non-technical stakeholders. This allows one to overcome the major challenges within the RE field: to minimise the conceptual distance between the model created in the mind of the stakeholders and the model perceived and represented by the requirements engineers.

XIS defines a language that promotes platform-independent design for interactive systems. This means that the XIS profile allows the design of interactive systems at the PIM level, so systems can be targeted, using specific model-to-code transformations, to different platforms, such as Web, Desktop or Mobile platforms, and source-code languages such as Java, C#, PHP, Perl or C++.

ProjectIT-Studio also supports a scripting template-specification language, PIT-TSL, with an ASP.NET like syntax. Templates are generic representations of software artefacts to support model-to-code transformations. The template engine transforms templates into source code, by compiling the template, executing it in runtime and then collecting the output into the specified file artefact.

To better explain the ProjectIT approach and respective support, the paper discussed the MyOrders2 case study. This case study showed how to apply and to use the described tools and languages in a simple and practical way to produce a concrete software system.

Our results show that the ambitious vision we had in the beginning of this project was correct. 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 modelling, model-to-model and model-to-code transformations.

However, in spite of the results already achieved, there are many other issues to tackle and research efforts to be done in the near future. The first key issue at the stage of our research is real-world 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 or research groups.

Second, the tool integration feature is also a mandatory one. We are currently lacking an integrated support for all the tools (i.e. ProjectIT-Requirements, ProjectIT-Modeler and ProjectIT-MDD) at the data representation and storage level. Frequently, each tool manages its specific information in separate repositories, thereby implying a need to duplicate information. Additionally, a common repository and data representation throughout the different tools would provide new advanced features among the elements managed by these tools, such as data querying and navigation, easy way to maintain traceability, automatic validations or intelligent inferences.

Third, the issue of models validation and model transformations is essential to enhance designer and developer productivity, and we are considering on focusing greater attention on it in the near future, for instance, exploring model consistency features as well as bidirectional model transformations through forward, reverse and round-trip techniques.

Fourth, there are many other management and business model issues that can be considering for future research in straight relation with this novel approaches to develop software systems. For example, regarding management, it is relevant to work on quality assurance at different levels, such as requirements, models, software architectures and even the generated code, and thus, working with a holistic perspective. On the other hand, regarding business models, it should be considered different from approaches to produce software, such as in-house, classical outsourcing or variants of offshoring.

7 References

1 Bell, T., and Thayer, T.: 'Software requirements: are they really a problem?'. Proc. 2nd Int. Conf. Software Engineering (ICSE'76) (Los Alamitos, CA: IEEE Computer Society Press), San Francisco, California, USA, October 1976, pp. 61–68

2 Silva, A.: 'O Programa de Investigação "ProjectIT"', Technical Report, INESC-ID, Lisbon, Portugal, October 2004, (available at: http://berlin.inesc-id.pt/alb/uploads/1/193/pit-white-paper-v1.0.pdf, accessed May 2007)

3 Kotonya, G., and Sommerville, I.: 'Requirements engineering: processes and techniques' (John Wiley & Sons, 1998)

4 Robertson, S., and Robertson, J.: 'Mastering the requirements process' (Addison-Wesley Professional, 2006, 2nd edn.)

5 Nuseibeh, B., and Easterbrook, S.: 'Requirements engineering: a roadmap'. Proc. Int. Conf. Software Engineering (ICSE'00) (ACM Press), Limerick, Ireland, June 2000, pp. 35–46

6 Matheson, D., France, R., Bieman, J., Alexander, R., DeWitt, J., and McEachen, N.: 'Managed evolution of a model driven development approach to software-based solutions'. OOPSLA & GPCE Workshop on Best Practices for Model Driven Development (OOPSLA'04), Vancouver, Canada, October 2004 (available at: http://www.softmetaware.com/oopsla2004/matheson.pdf, accessed July 2007)

7 OMG Model Driven Architecture (available at: http://www.omg.org/mda, accessed May 2007)

8 Frankel, D.: 'Model driven architecture: applying MDA to enterprise computing' (John Wiley & Sons, 2003)

9 Czarnecki, K., and Helsen, S.: 'Classification of model transformation approaches'. Proc. 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture (OPPSLA'03), California, USA, October 2003 (available at: http://www.softmetaware.com/oopsla2003/czarnecki.pdf, accessed July 2007)

10 Schmidt, D.: 'Model-driven engineering', *IEEE Comp.*, 2006, **39**, (2), pp. 25–31

11 Stahl, T., and Volter, M.: 'Model-driven software development' (Wiley, 2006)

12 WebML (available at: http://www.webml.org, accessed May 2007)

13 OOHDM (available at: http://www.telemidia.puc-rio.br/oohdm/oohdm.html, accessed May 2007)

14 AndroMDA, (available at: http://www.andromda.org, accessed May 2007)

15 Jamda (available at: http://jamda.sourceforge.net, accessed May 2007)

16 JET (available at: http://www.eclipse.org/modeling/m2t/?project=jet, accessed May 2007)

17 OptimalJ (available at: http://www.compuware.com/products/optimalj, accessed May 2007)

18 ArcStyler (available at: http://www.arcstyler.com, accessed May 2007)

19 Rational Rose XDE Developer (available at: http://www-306.ibm.com/software/awdtools/developer/rosexde, accessed May 2007)

20 Codagen Architect (available at: http://www.manyeta.com/en/Technology/codagen_architect_v3.2, accessed May 2007)

21 Preece, J., Rogers, Y., and Sharp, H.: 'Interaction design' (Wiley, 2002, 1st edn.)

22 Videira, C., Ferreira, D., and Silva, A.: 'A linguistic patterns approach for requirements specification'. Proc. 32nd EUROMICRO Conf. Software Engineering and Advanced Applications (EUROMICRO'06) (Washington, DC: IEEE Computer Society), Dubrovnik, Croatia, August 2006, pp. 302–309

23 Videira, C., Ferreira, D., and Silva, A.: 'Patterns and parsing techniques for requirements specification'. Proc. 1st Iberian Conf. Information Systems and Technologies (CISTI'06), Ofir, Portugal, June 2006, vol. II, pp. 375–390

24 Cockburn, A.: 'Writing effective use cases' (Addison-Wesley Professional, 2000, 1st edn.)

25 Silva, A., Saraiva, J., Silva, R., and Martins, C.: 'XIS – UML profile for eXtreme modeling interactive systems'. Proc. 4th Int. Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES '07) (Los Alamitos, CA: IEEE Computer Society), Braga, Portugal, March 2007, pp. 55–66

26 Martins, C., and Silva, A.: 'Modeling user interfaces with the XIS UML profile'. Proc. 9th Int. Conf. Enterprise Information Systems (ICEIS'07) (Springer), Funchal, Portugal, June 2007 (available at: http://berlin.inesc-id.pt/alb/static/papers/2007/cm-iceis2007.pdf, accessed July 2007)

27 Saraiva, J., and Silva, A.: 'Eclipse.NET: an integration platform for ProjectIT-Studio'. Proc. 1st Int. Conf. of Innovative Views of .NET Technologies (IVNET'05) (Instituto Superior de Engenharia do Porto and Microsoft), Porto, Portugal, July 2005, pp. 57–69

28 Ferreira, D.: 'ProjectIT-RSL', Graduation Thesis, Instituto Superior Técnico, Portugal, October 2006

29 Object Management Group: 'Unified modeling language: superstructure', version 2.0, August 2005 (available at: http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf, accessed May 2007)

30 Enterprise Architect – UML Design Tools and UML CASE tools for software development (available at: http://www.sparxsystems.com/products/ea.html, accessed May 2007)

31 Object Management Group: 'Unified modeling language: diagram interchange', version 1.0, April 2006 (available at: http://www.omg.org/cgi-bin/apps/doc?formal/06-04-04.pdf, accessed May 2007)

32 Object Management Group Final Adopted OMG SysML Specification 2006 (available at: http://www.omg.org/cgi-bin/apps/doc?ptc/06-05-04.pdf, accessed May 2007)

33 Balmelli, L., Brown, D., Cantor, M., and Mott, M.: 'Model-driven systems development' (IBM Systems Journal, 2006), (available at: http://www.research.ibm.com/journal/sj/453/balmelli.html, accessed May 2007)

34 Object Management Group: 'OMG systems modeling language tutorial'. Int. Council on Systems Engineering (INCOSE'07), San Diego, California, USA, June 2007 (available at: http://www.omgsysml.org/INCOSE-2007-OMG-SysML-Tutorial.pdf, accessed July 2007)

35 Ambriola, V., and Gervasi, V.: 'The Circe approach to the systematic analysis of NL requirements', Technical Report TR-03-05, University of Pisa, March 2003, (available at: http://circe.di.unipi.it/~gervasi/Papers/circe03.ps, accessed July 2007)

36 Fuchs, N., Schwertel, U., and Schwitter, R.: 'Attempto controlled english – not just another logic specification language'. Proc. 8th Int. Workshop on Logic Programming Synthesis and Transformation (LOPSTR'98) (Springer-Verlag), London, UK, 2003, pp. 1–20

37 Achour, B., and Guiding, C.: 'Scenario authoring'. Proc. 8th European-Japanese Conf. Information Modeling and Knowledge Bases (IOS Press), Vamala, Finland, May 1998, pp. 152–171

38 Rolland, C., and Proix, C.: 'A natural language approach for requirements engineering'. Proc. 4th Int. Conf. Advanced Information Systems (CAiSE'92) (Springer-Verlag), Manchester, UK, May 1992, pp. 257–277

39 Denger, C.: 'High quality requirements specifications for embedded systems through authoring rules and language patterns', MSc thesis, Fachbereich Informatik, Universität Kaiserslautern, 2002

40 Juristo, N., Morant, J., and Moreno, A.: 'A formal approach for generating OO specifications from natural language', *J. Syst. Softw.*, 1999, **48**, pp. 139–153

41 Mich, L., and Garigliano, R.: 'The NL-OOPS Project: OO Modeling using the NLPS LOLITA'. Proc. 4th Int. Conf. Applications of Natural Language to Information Systems (Springer), Paris, France, November 1999, pp. 215–218

42 Overmyer, S., Lavoie, B., and Rambow, O.: 'Conceptual modeling through linguistic analysis using LIDA'. Proc. 23rd Int. Conf. Software Engineering (ICSE'01), Toronto, Canada, May 2001, pp. 401–410

43 Pastor, O., Gómez, J., Insfrán, E., and Pelechano, V.: 'The OO-method approach for information systems modeling: from object-oriented conceptual modeling to automated programming', *Inform. Syst.*, 2001, **26**, (7), pp. 507–534

44 Kozaczynski, W., and Thario, J.: 'Transforming user experience model to presentation layer implementations'. 2nd Workshop on Domain-Specific Visual Languages (OOPSLA'02), Seattle, WA, USA, November 2002 (available at: http://se2c.uni.lu/tiki/se2c-bib_download.php?id=262, accessed July 2007)

45 Nunes, N., and Cunha, J.: 'Towards a UML profile for interaction design: the Wisdom approach'. Proc. 3rd Int. Conf. (UML 2000) (Springer), York, UK, October 2000, vol. 1939, pp. 101–116
46 Silva, P., and Paton, N.: 'User interface modeling in UMLi', *IEEE Softw.*, 2003, **20**, (4), pp. 62–69
47 Hennicker, R., and Koch, N.: 'Modeling the user interface of web applications with UML'. Workshop of the pUML-Group held together with the «UML»2001 on Practical UML-Based Rigorous Development Methods – Countering or Integrating the eXtremists (GI), Toronto, Canada, October 2001, pp. 158–172
48 Azevedo, P., Merrick, R., and Roberts, D.: 'OVID to AUIML – user-oriented interface modelling'. Proc. Workshop – Towards a UML Profile for Interactive Systems Development (TUPIS'00), York, UK, October 2000 (available at: http://xml.coverpages.org/TUPIS2000-Azevedo.html, accessed July 2007)
49 Bergh, J., and Coninx, K.: 'Towards modeling context-sensitive interactive applications: the context-sensitive user interface profile (CUP)'. Proc. ACM Symp. on Software visualization (SoftVis'05) (New York, NY: ACM Press), St. Louis, Missouri, May 2005, pp. 87–94
50 Constantine, L.: 'Canonical abstract prototypes for abstract visual and interaction'. Proc. 10th Int. Workshop DSV-IS 2003 (Springer, LNCS 2844), Funchal, Madeira Island, Portugal, June 2003, pp. 1–15
51 Microsoft Dynamic Systems Initiative (available at: http://www.microsoft.com/business/dsi/default.mspx, accessed May 2007)
52 Greenfield, J., Short, K., Cook, S., Kent, S., and Crupi, J.: 'Software factories: assembling applications with patterns, models, frameworks, and tools' (Wiley, 2004, 1st edn.)
53 Clements, P., and Northrop, L.: 'Software product lines: practices and patterns' (Addison-Wesley Professional, 2001, 3rd edn.)
54 Deursen, A., Klint, P., and Visser, J.: 'Domain-specific languages: an annotated bibliography' (available at: http://homepages.cwi.nl/~arie/papers/dslbib, accessed in May 2007)