Tool Support for Helping the Use of Frameworks

Ricardo Pereira e Silva Eng., M.Sc.
Universidade Federal de Santa Catarina - Depto. de Informática e de Estatística
Florianópolis - SC - Brazil (Ph.D. Student at Universidade Federal do Rio Grande do Sul - Instituto de Informática), e-mail: ricardo@inf.ufsc.br

Roberto Tom Price Eng., M.Sc., D.Phil.
Universidade Federal do Rio Grande do Sul - Instituto de Informática
Porto Alegre - RS - Brazil, e-mail: tomprice@inf.ufrgs.br

Abstract

Frameworks promote design and code reuse, at a higher level of granularity. The use of frameworks is a hard task though, because usually they lack documentation and instructions on how to use them, thus being, in general, very difficult to understand. This paper addresses some mechanisms to help framework usage. It presents SEA-Preceptor, a hyperdocument based tool that sets up the possible paths to be followed by framework users to construct applications. SEA-Preceptor hyperdocuments also allow navigating through design or code of frameworks (or applications), as a way of helping on framework understanding. Frameworks and applications are specified in SEA environment (that hosts SEA-Preceptor and other tools) as sets of diagrammatic models that are views of a semantically checked metamodel repository, which is semi-automatically translated into source code.

1. Introduction

An object-oriented framework is a structure of interrelated classes, that is an incomplete implementation of a set of domain-applications. This class structure must be adapted to generate specific applications [9].

Design reuse is the main benefit in using frameworks: a framework has defined the control flow of the applications that can be generated from it. Besides that, there is also code reuse: a set of framework classes is included in applications, and framework users need to know only part of them. So, framework promotes reuse of code and design [20].

Frameworks must present the features of generality, flexibility and extensibility. The main feature is generality, because a framework must be a general abstraction of a domain. Flexibility regards the possibility of changing the framework functionalities, according to the needs of specific applications. Framework evolution can occur at any time of its life cycle, because the use of a framework may lead to new needs and new abstractions of its domain. So, to construct an extensible framework, a developer must foresee its future uses, as well as the possibility of extending the limits of the treated domain.

The framework approach has two kinds of disadvantages: complexity to develop and complexity to use. Complexity to develop regards the need of supplying framework design with the features above mentioned, as explored in a previous work [18]. Complexity to use regards the required effort for a user to learn how to develop applications from a framework.

This paper presents SEA-Preceptor, a tool that aids in the usage of frameworks to build applications. At first, the requirements to make someone able to use a framework are presented. Also approaches to attain this goal, and existing tools to support framework usage, are discussed. SEA, a framework development environment that supports SEA-Preceptor tool, is very briefly discussed. Next, the SEA-Preceptor tool is described.

SEA-Preceptor drives application development by means of a hyperdocument, which establishes the steps to be followed by framework users. Different paths may be followed, according to the needs of the application in development. Applications are specified by means of modelling techniques that describe their static and dynamic aspects. Resulting design specifications are translated into programming language. During application development, SEA-Preceptor allows design specifications or code to be browsed (specifications and code of frameworks or existing applications).

2. Considerations on using frameworks

The first requirement for generating an application from a framework is to discover how to adapt its structure, according to the application needs.

2.1. Modes of framework using

White-box frameworks are designed to generate applications by subclassing their abstract classes, through the specification of the body of their abstract methods or overriding of their concrete methods. In black-box frameworks, which present concrete classes, customisation occurs by means of object composition - different behaviour of applications are obtained with different object combinations. White-box frameworks require more effort to learn how to use them, but they allow for a wider range of applications. Grey-box frameworks blend the two approaches, allowing customisation by means of object composition, and by means of subclassing [20].

2.2. What a user must know to use a framework

The users need to know the framework structure to handle either white-box or grey-box frameworks. To use these frameworks, the following three key-questions must be answered.

⌖ **Question 1 - *What classes?***: the concrete classes of an application have two possible origins: new classes created by the users or framework concrete classes. So, what classes should be developed to generate an application and what concrete classes may be reused?

⌖ **Question 2 - *What methods?***: A method of an abstract class can be classified as: abstract, template or base [8]. When a template method is executed, the call of at least one hook method occurs, and the hook method can be an abstract, template or base method [13]. Thus, when producing applications:

- *abstract* methods must have their body specified (that is, their implementation in concrete classes);
- *template* methods may have their behaviour defined. This can be done by means of the redefinition of the called hook methods. The evaluation of the need or convenience of producing methods should be extended to the hook methods.
- *base* methods can be overridden, but carefully. According to framework design, they do not need be overridden.

So, for creating a concrete class for an application, what methods must be created and what inherited methods may be reused?

⌖ **Question 3 - *What do the methods do?***: the methods that must be created will produce the particular behaviour of specific applications. A framework user must understand the semantics of these methods, that is, what are their responsibilities, in what running situations do act these methods, and how do these methods implement collaboration between different objects?

3. Related work

Different ways of describing a framework will require more or less effort to learn how to use it. There are some proposals of framework description formats and tools that use them, aimed for reducing the required effort to use a framework, as the following described.

3.1. Ways of learning how to use a framework

The answer to the three questions above may be searched in the framework description. During the framework design the classes and methods that must be built by the users have been defined, with their intended semantics. Information sources to understand it, may be documentation and source code. Documentation provided to understand a framework could be classified in two different types: description of the framework design (how a framework works) and description of how to use a framework.

Source code analysis

The simplest way of helping the framework users on understanding a framework is to make its source code available. It can be its complete code, only the code of some of its classes or code of framework based applications. The tasks of a framework user when developing applications includes code analysis. It leads to understanding the framework design and therefore, to discover what must be developed to construct an application. This corresponds to a Reverse Engineering activity, by witch a framework user must recover the framework design, because at the beginning of the code analysis there are no descriptions at a high-level of abstraction, but just code. That activity in some cases requires a lot of effort, which makes infeasible the use of a framework.

Understanding framework design by means of code analysis is the only possible way when framework documentation is not available. No one of the three key-questions above mentioned is answered without some effort in code analysis. In most cases, source code analysis is the hardest way of learning how to use a framework.

Design framework documentation

Current approaches of expressing framework design structures are originated from framework development methodologies that do not use modelling techniques like that used by traditional OOAD methodologies. And so, design documentation is mainly based on textual description, references to source code and use of devices for describing discrete aspects of framework design - nearly always to emphasize flexibility on framework structure, as the highlight of design patterns [6].

Pree proposes the highlight of metapatterns used in a framework, by means of a diagram similar to object model of OOAD methodologies. That notation includes the classes that take part in metapattern and an iconic representation of this metapattern, showing the hook and template methods (see figure 1, that presents the use of metapattern based notation to describe Publisher/Subscriber design pattern) [13]. The advantage of this approach is its capacity of highlighting the points in design structure that framework developer kept flexible, that is, where should start the search of the key-question answers. This description way otherwise, does not describe the entire framework structure nor instructs how to use the framework to generate applications. It needs be complemented with some other description technique.

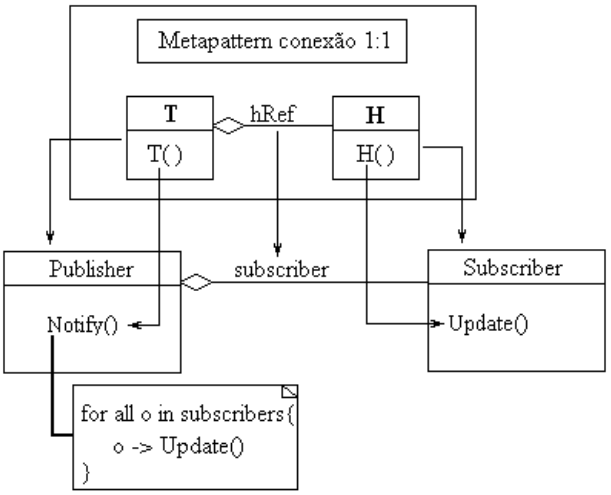

Figure 1 - Publisher/Subscriber design pattern using metapattern based notation

Contracts [7] describe collaboration between groups of objects, by means of a formal language which represents:

- the participants - instances or sets of instances that take part in the contract;
- the contractual obligations - responsibilities of the participants in their interactions.

Contracts can describe discrete collaboration aspects from a framework design. It has the advantages of modelling object interaction in a formal manner, and, as contracts can be reused as basis or part of others, they have the capacity of describing complex behaviour as composition of simpler ones. Contracts can be used to highlight the flexible parts of a framework structure, similarly to metapatterns. The main disadvantage of contracts is that it corresponds to an extensive and very low-level of abstraction description (classes, methods, attributes). It makes difficult to understand a framework design described by means of contracts [13].

"How to use" framework documentation

Documentation aimed at teaching how to use a framework explains the steps that must be followed by a framework user, to generate applications. This kind of documentation gives little emphasis on design aspects. Its main goal is to describe shortly as possible, how to produce applications. It presents textual format and often includes some other way of description, like examples using source code.

Cookbooks, as used for framework MVC [12], are sets of textual recipes that describe how to use a framework to solve problems in application development. Its main advantage is the capacity to answer the key-questions directly, and reduce the time spent to produce applications. The main disadvantage of the cookbook approach is that it covers a very limited range of application categories: they let users helpless when their needs do not match the cookbook contents. Another problem with cookbook approach is that nearly always the process of learning about a framework includes low level activities: once answered the key-questions (*what to do*), the user often needs to do code analysis to learn "*how to do*".

Documentation patterns proposed by Johnson and used by him to describe how to use the framework HotDraw [9], constitute an alternative to cookbook format. Its documentation patterns are structured recipes that describe how to do something using the framework resources, and that contain pointers to other documentation patterns, like links in a hyperdocument. Its main advantage in relation to conventional cookbooks is that the link between documentation patterns makes easy to browse the documentation through a path defined according to the framework user needs.

3.2. Cookbook tools for helping framework usage

The frameworks MVC [12] and ET++ [13] have tools, that supply on-line access to the contents of their cookbooks. MVC cookbook includes textual description and short examples in Smalltalk. ET++ cookbook is based

on the hyperdocument approach, what allows to define the navigation sequence according to the user needs. Moreover, ET++ cookbook includes parts of design specification using the *metapattern* notation [13], besides textual description and examples in C++ code. These tools present the advantage of the cookbook approach: direct indication of sequence steps to be followed, to solve a design problem. Unfortunately, they also present the cookbook disadvantages.

Meusel [11] produced a cookbook for HotDraw, joining the documentation pattern [9] and hyperdocument approaches. Besides documentation patterns, the tool includes tutorials and design patterns, to describe some punctual aspects of the framework design. Relative to the common cookbook approach, Meusel's cookbook contributes mainly with the use of hyperdocument. The adoption of the documentation pattern approach produced recipes in a structured form.

The active cookbook approach, proposed by Pree and by him implemented in a prototype [14], establishes that a cookbook tool, besides describing the steps to be followed, automatically carries out implementation actions. So, according to this approach, an active cookbook tool can also produce part of the application code.

Nautilus is a tool for producing active cookbooks that describe the steps to produce applications by means of the use of a framework, allowing the inclusion of automatic actions for the creation of application source code (in Smalltalk) [21]. Nautilus is coupled with MetaExplorer [3], which is a tool for dynamic analysis of applications. MetaExplorer produces a description of application dynamic behaviour, by means of metaobjects, which observe message flow between application objects. The main advantage of Nautilus is that its integration with MetaExplorer contributes to framework learning process, supplying some help in code analysis. Reverse Engineering tools (as code browsers or cross-reference tools) can help the code analysis activity. Dynamic analysis should use tools, like MetaExplorer.

The mentioned cookbook tools have common features. They must be constructed by someone that knows the framework design; they describe the steps to develop applications based on frameworks; and they rely on code analysis for further understanding of the framework design. Except for Nautilus, none of the mentioned tools help on code analysis. They do not manipulate design specifications. They only hand two types of information: cookbook contents (textual description, which in some cases includes source code examples and diagrams to describe specific design aspects) and source code.

4. SEA, an environment for the development and the use of frameworks

SEA is an environment that supports the development and the use of frameworks. SEA-Preceptor, the tool for helping application development, is part of SEA environment.

4.1. Framework environment requirements

SEA environment is being developed, considering the following requirements.

Support for editing graphical models

Nearly all framework development methodologies, as well as tools based on them, ignore graphical OOAD methodology notations, aiming directly at code in programming language. To make the obtained framework useful to many people, the use of auxiliary devices, like cookbooks and the highlight of design patterns is recommend [13].

On the other hand, current OOAD methodologies and their supporting tools are directed chiefly for application development. Notations of OOAD methodologies do not represent completely some concepts required for framework design, like essential classes and redefinable classes [17]. For example, the Unified Modelling Language, UML [15], that is going to be a *de facto* standard notation for OOAD [4], does not adequately represent framework constructs [1] as its basic features. However, through the use of stereotypes one may extend UML to support the representation of most framework concepts.

The use of graphical modelling techniques can increase the comprehensibility of framework (and application) descriptions [5], helping to make the development and use of frameworks a comfortable activity. In the SEA environment, frameworks and applications are specified as sets of graphical models with textual complements. These specifications are translated into Smalltalk.

Support for semantic edition

Code generation demand that an environment assures the semantic consistency of the specification, instead of acting simply as graphical editor. The MVC approach is adequate for separating conceptual elements from their views and it was adopted to allow semantic treatment at conceptual level. SEA adopts the following ways of supplying semantic consistency for design specifications:

- there is a metamodel that establishes the

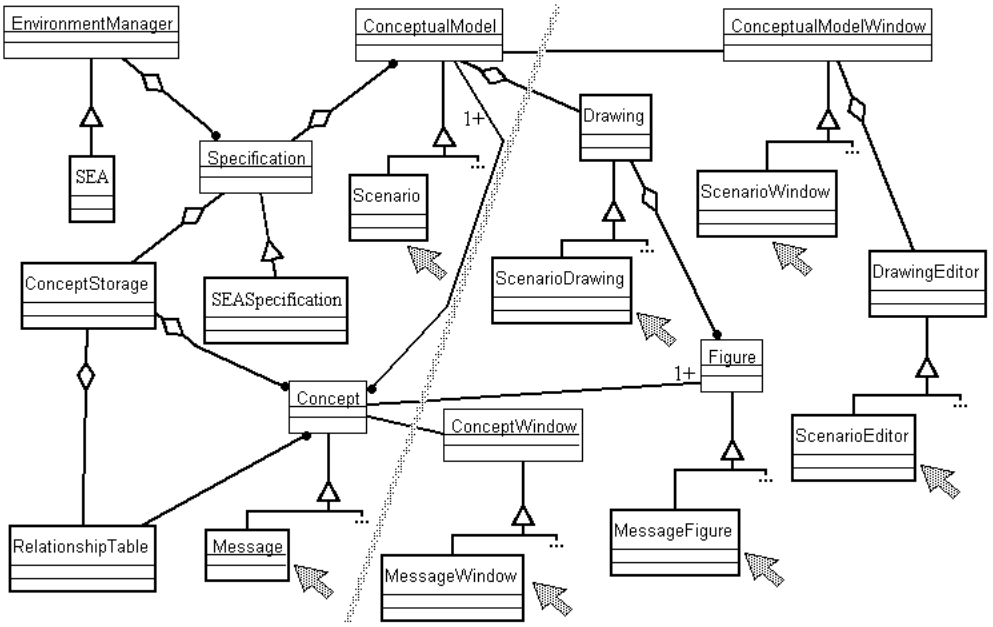

Figure 2 - SEA environment classes that define part of the specification structure, detaching as example, part of the classes of a scenario (*Drawing*, *DrawingEditor* and *Figure* are HotDraw classes)

specification elements (models and their components) and the semantic relationship between these elements. The metamodel is implemented in the environment class structure defining the specification repository;

- environment editors are driven by the metamodel constraints;

Support for reusing software artefacts

It is possible to increase the development productivity by means of software reuse in different levels, through facilities and tools to promote: Reverse Engineering; to import external software artefacts, like *components* [19]; to access and modify libraries of existing software specifications and libraries of design pattern structures.

Flexibility

Flexibility is required to allow the evolution of an environment, that is, to include new tools and new functionalities, with less effort as possible, and without unexpected side-effects over the environment structure. To produce a framework for environments, instead a single environment, is an adequate approach to obtain flexibility. SEA environment presents flexibility in many aspects, for instance:

- modify the model types that define a specification (change the metamodel);
- include or change tools that access specifications;
- use different storage devices (DBMS, files etc.);
- change the permissions of accessing specifications.

4.2. SEA, structure and functionalities

To obtain an environment that supports the above requirements, SEA is being built by means of a framework architecture. SEA reuses the frameworks MVC [12] and HotDraw [2].

Figure 2 presents part of the specification repository structure of the SEA environment and some tools that handle it. All components (instances of *Concept* subclasses) referred by specification models (instances of *ConceptualModel* subclasses) are kept by a single storage (instance of *ConcepStorage*). A class that appears simultaneously in more than one object model diagram, or in others diagrams, for example, is a same specification component. Specification models have a drawing that corresponds to their visual presentation. Similarly, the specification components are associated with figures, present in the model drawings. Figure 2 left side classes refer to conceptual elements and right side classes, to visual elements, according to the MVC approach. This figure highlights (with arrows) some of the classes of the scenario model, the message concept, used in scenarios, and the respective editors.

Figure 3 shows the overall structure of the SEA environment framework, which includes the above described specification structure. It is based on the toaster approach [16]. Independent tools, accessed through the environment manager window, share the same specification repository. So, due to the low

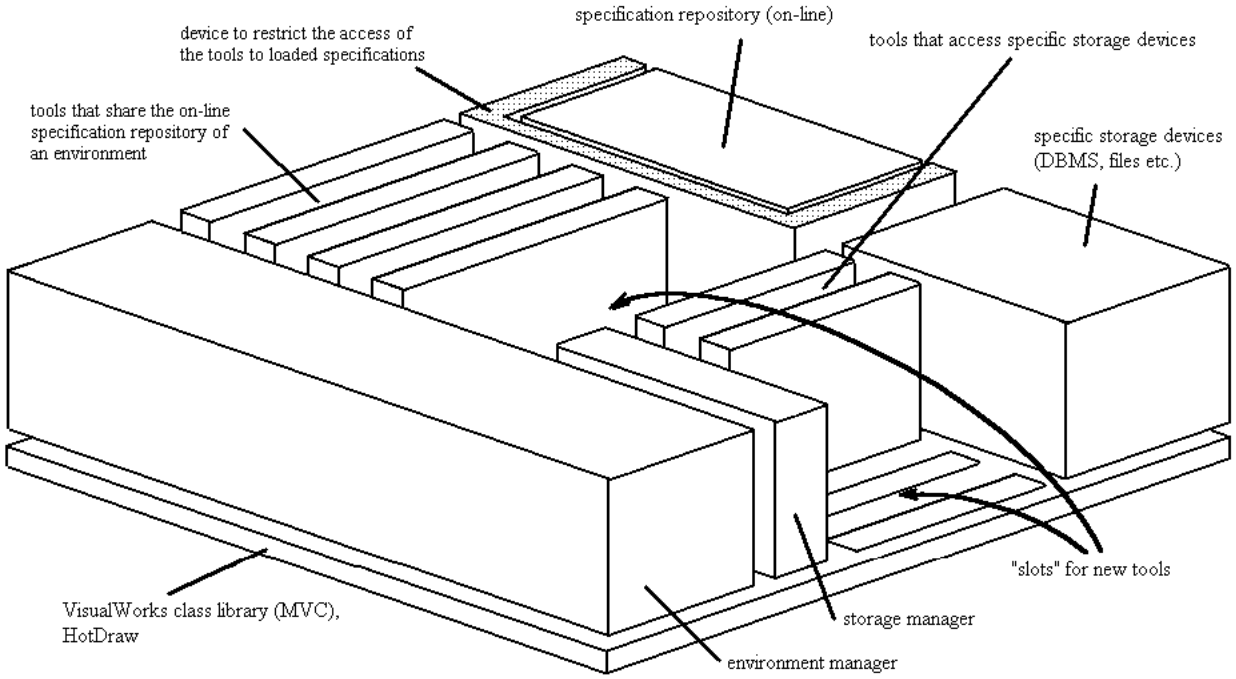

Figure 3 - The architecture of the framework environment, based on the toaster approach

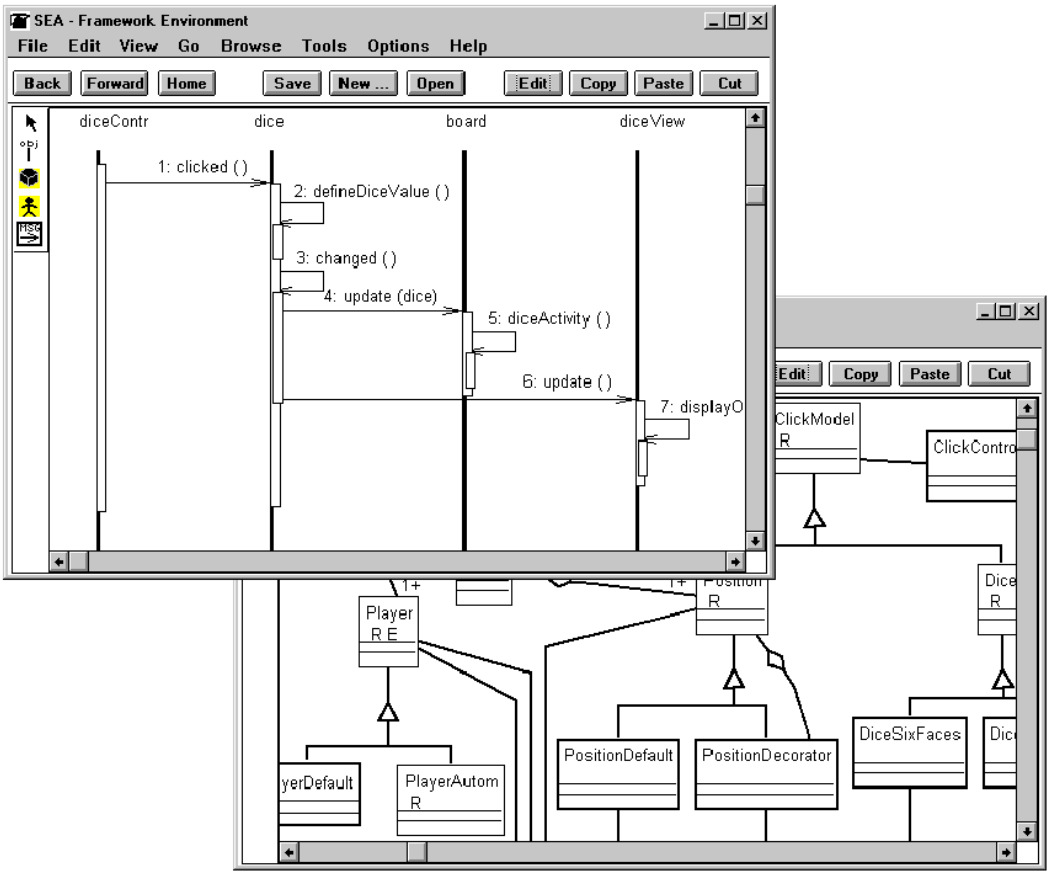

Figure 4 - Scenario and object model editors, from the SEA environment

coupling between tools, it is possible to modify, include or exclude a tool, without affecting the behaviour of other tools. Moreover, flexible aspects are untied: different specification structures can use the same storage device, or the same specification structure can be used with different access restriction devices, for example. Figure 4 presents some SEA editors.

A software specification (of framework or application) is produced in SEA environment by means of the building of graphical diagrams that describe static and dynamic features [17]. Static description is based mainly on object model diagrams. Different diagrams allow to describe the dynamic behaviour: the overall software behaviour, by means of description of its transactions (use cases), the refinement of this behaviour by means of scenario and state diagrams; and the description of the body of the class methods. The description of the methods is based on a logical structuring notation (similar to action diagrams [10]). Skeletons of method specifications, generated by the analysis of other dynamic diagrams, can be extended by the users with statements in the programming language syntax.

SEA environment offers functionalities for framework development, as the following:

- refactoring actions - for example, moving attributes and methods from a class to another, merging classes, changing the order, the origin or the target of messages etc.;
- semantic edition - the removal of a class from a specification, for example, produces (among other effects) disconnecting the instances of this class from the scenario diagrams, automatically;
- design pattern usage - it is possible to search design pattern structures in a library (the contents of which can be expanded) and use them in a specification.

5. SEA-Preceptor tool

SEA-Preceptor tool is aimed to driving and monitoring the actions of the framework users, when they use a framework to generate applications. Furthermore, SEA-Preceptor helps to understand a framework design by means of aiding the browsing of framework design specifications.

5.1. Requirements for supporting the development of applications based on frameworks

As general requirements, a tool that supports the development of applications based on frameworks must:

- **drive the actions of framework users to produce applications** - the well thought driving of users steps may reduce the time they spend discovering what must be done to generate applications. A hyperdocument is an adequate way of conducting the user tasks, because it allows different paths of development, according to the specific application needs.
- **reduce the effort to understand framework design** - the support tool must allow access to high-level design descriptions of the used framework, as well as of existing applications. This is mainly helpful when the user needs do not match exactly with the instructions of the driver hyperdocument. In this situation source code analysis should not be the only alternative.
- **liberate users of low level activities** - mandatory actions (like creating some specification components) can be semiautomated. For example, if a framework design establishes the need of creating a subclass of one of its classes, the tool should ask the name of the new class and include it in the application design specification, instead of waiting for the user to do it himself. Furthermore, translating specification design to source code should use specification-translators. The user fills in specification voids, instead of producing the full application code.
- **check the satisfaction of the requirements to produce an application based on a framework** - framework design establishes requirements for applications developed under it, like the need of producing subclasses of some of its abstract classes, or the definition of abstract methods of these classes. A tool that manages the development of applications must check the accomplishment of these requirements, as part of the task of driving user activities.

The established requirements differ of existing approaches of aiding framework users, chiefly on the emphasis on high-level specification.

5.2. SEA-Preceptor functionalities

SEA-Preceptor tool carried out the following set of functionalities.

Framework requirement analysis

SEA-Preceptor searches for classes and methods that must (or can) be defined. This information is searched in framework specifications. In the framework requirement analysis, SEA-Preceptor searches and lists the following situations, that require user attention:

- subclasses must be created (and their abstract methods must be defined) for the classes that are

abstract, redefinable, essential, and show no specified subclasses;

- for the classes that are abstract, redefinable, essential, and that show at least a subclass in the framework specification, there is no obligation to create subclasses, but if a user chooses to create a subclass of one of these classes, all of their abstract methods must be defined;
- for the classes that are abstract, redefinable, inessential, and that may or not show subclasses specified, there is no obligation to create subclasses, but if a user chooses to create a subclass of one of these classes, all of their abstract methods must be defined;

Each situation above described of creation (mandatory or not) of classes and methods will originate an *action link* in the hyperdocument, that executes the creation action, besides to load the adequate model editor in SEA environment workspace.

Hyperdocument edition

It is necessary to support the creation and changing of a navigation driver hyperdocument. F*ramework requirement analysis* must supply the relation of the action links to be included in this hyperdocument.

At the moment of writing, the functionality of hyperdocument edition is not fully implemented. VisualWorks is used to build hyperdocuments, while the inclusion of the action links is done by the user himself.

Application requirements checking

It is the verification if all mandatory specification elements are included in an application specification (based on a framework). This verification uses the list of mandatory and optional specification elements obtained by means of a *framework requirement analysis* and it checks:

- if all the mandatory classes have been created;
- if there are undefined methods in the newly created concrete classes.

It is als o checked if the created application-specification differs from the framework specification through the inclusion of additional elements. For instance, subclasses of unredefinable classes, or classes without a framework superclass. SEA-Preceptor only warns the application users; it does not hinder the creation of these specification components, because they might be unforeseen requirements of the application domain.

5.3. SEA-Preceptor usage

SEA-Preceptor provides support for two types of users: framework developers and framework users. For framework developers, SEA-Preceptor supports the development of hyperdocuments that guide the use of frameworks, and for framework users, it supports the handling of these hyperdocuments to produce applications.

A framework developer initially do a framework requirement analysis, to list the specification elements that must or may be created (each one will correspond to an action link). Next, he produces an hyperdocument that must contain all the defined action links. The hyperdocument must include the definition of the possible paths, as well as the links to navigate through the hyperdocument pages, design specifications or code

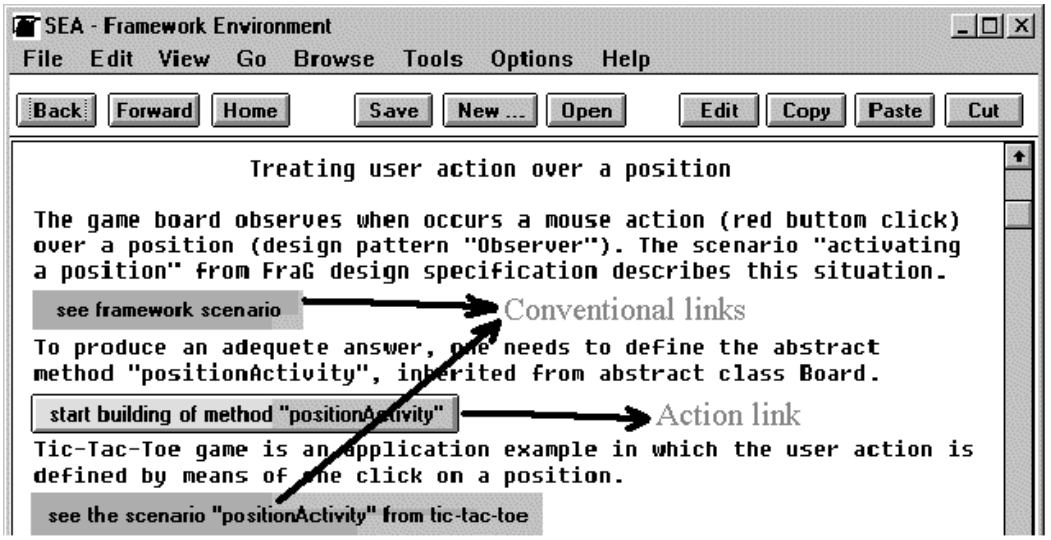

Figure 5 - FraG hyperdocument page, loaded in SEA environment

of the framework (or of existing applications).

The front-end of SEA-Preceptor to framework users is the hyperdocument, handled in SEA environment workspace. A framework user will follow a navigation path according to his needs. Activation of action links will cause edition actions. For example, the activation of a link that creates a class will open an object model editor and a dialog box to ask the class name; once this is supplied, the class will be included in the diagram. Figure 5 shows an example of the hyperdocument that drives game development using FraG, a framework for board games [17].

Specification documents produced in the SEA environment are treated as hyperdocuments (all model and component descriptions can include links). So, when a conventional link leads to a framework specification model - to describe some design detail, for example - a new path may be defined by the framework user, simply by link activation, in this model and in following documents. By means of the activation of the SEA environment *back button*, it is possible to return to the driver hyperdocument. So, this possibility of navigation available in SEA-Preceptor hyperdocuments, besides to drive application development, allows the browsing of design specifications and code, as a way of learning about a framework.

Once the framework user considers the application specification development finished, he will use SEA-Preceptor to do an *application requirement checking*. Besides SEA-Preceptor, the framework users need the SEA environment functionalities to produce an application based on a framework (as graphical model edition, semantic consistency checking and code generation).

6. Conclusion and future work

This article presents SEA-Preceptor, a tool supporting the steps to produce applications based on frameworks. SEA-Preceptor emphasizes the use of high-level models for framework design and the guidance for the production of the application specification through a hyperdocument based cookbook. In fact, an integration between OOAD notations and existing framework development techniques is proposed.

In the SEA environment, an application is produced extending a framework design specification. The design notations are based on diagrammatic OOAD modelling languages, slightly adapted to framework design needs. These specifications are translated into source code. SEA environment has tools that support the preparation of the framework specification, do the semantic checking of the metamodel repository, translate specifications into source code, and SEA-Preceptor, the tool that aids user for application development.

SEA-Preceptor offers the following functionalities: *framework requirement analysis*, *hyperdocument edition* and *application requirement checking*. The requirement analysis produces a list of classes and methods from the framework specification, that must (or can) be created for the application. The hyperdocument edition creates the hyperdocument that will guide the application development activity. The application requirement checking verifies if the application requirements are complete.

Navigation through the application-generator hyperdocument uses conventional links, to move between hyperdocument components as examples, text, diagrams, code and other ways of software documenting, but also employs action links that prompt the user for specific actions or activate tools to aid in the application specification process.

Currently the tool is still being tested and refined using small examples. The first application-generation hyperdocument being created is to support the development of games using the framework FraG [18]. This hyperdocument will be used in class to help students to develop board games using FraG. Other groups will develop games using the framework without the use of the hyperdocument. The comparison of the spent times to develop applications with equivalent complexity levels (with and without the use of Preceptor) will be used for a concrete evaluation of the contribution of the SEA-Preceptor approach. Besides SEA-Preceptor hyperdocument edition, other SEA functionalities must be included or completed, as the increasing of specification analysis, and the capacity of producing code in other programming languages, as Java.

SEA environment is being developed to support development and use of frameworks in high-level manner. According to this, SEA-Preceptor is pointed to make the development of applications based on frameworks, a comfortable activity.

7. References

[1] ARTIM, J. **UML: the language of blueprints for software?**. Panel session in: Object-Oriented Programming Systems, Languages and Applications Conference - OOPSLA, 1997, Atlanta. *It refers only to author position*.

[2] BRANT, J. **HotDraw**. Urbana: University of Illinois at Urbana-Champaign, 1995. *Master thesis*.

[3] CAMPO, M. R. **Compreensão visual de frameworks através da introspecção de exemplos**. Porto Alegre: UFRGS/II/CPGCC, mar. 1997. *Doctoral thesis*.

[4] COLEMAN, D. *et al*. **UML: the language of blueprints for software?**. Panel session in: Object-Oriented Programming Systems, Languages and Applications Conference - OOPSLA, 1997, Atlanta.

[5] FOWLER, M. Describing and comparing object-oriented analysis and design methods. In: Carmichael. **Object development methods**. New York: SIGS Books, 1994. p. 79-109.

[6] GAMMA, E. **Design patterns**: elements of reusable object-oriented software. Reading: Addison-Wesley, 1994.

[7] HELM, R. et al. **Contracts: specifying behaviour composition in object-oriented systems**. In: Object-Oriented Programming Systems, Languages and Applications Conference - OOPSLA, oct. 1990, Otawa.

[8] JOHNSON, R. E., RUSSO, V. F. **Reusing object-oriented designs**. Urbana: University of Illinois , 1991. Technical Report *UIUCDCS91-1696*.

[9] JOHNSON, R. E. **Documenting frameworks using patterns**. In: Object-Oriented Programming Systems, Languages and Applications Conference - OOPSLA, 1992, Vancouver.

[10] MARTIN, J. **Técnicas estruturadas e CASE**. São Paulo: Makron Books, McGraw-Hill, 1991.

[11] MEUSEL, M. *et al*. **A model for structuring user documentation of object-oriented frameworks using patterns and hypertext**. In: European Conference on Object-Oriented Programming - ECOOP'97, 1997.

[12] PARCPLACE. **VisualWorks Cookbook**. ParcPlace Systems Inc.1994.

[13] PREE, W. **Design patterns for object oriented software development**. Reading: Addison-Wesley, 1995.

[14] PREE, W. *et al*. **Active guidance of framework development**. Software - Concepts and tools v.16, n.3. 1995.

[15] RATIONAL. **UML notation guide**. Rational Software Corporation, 1997. (www.rational.com, uml/ad970805_uml11_Notation2.zip).

[16] SHAW, M., GARLAN, D. **Software architecture** - perspectives on an emerging discipline. Upper Saddle River: Prentice Hall, 1996.

[17] SILVA, R. P., PRICE, R. T. **O uso de técnicas de modelagem no projeto de frameworks orientados a objetos**. In: 26th JAIIO / First Argentine Symposium on Object Orientation (ASOO'97), Aug. 1997, Buenos Aires.

[18] _____. **A busca de generalidade, flexibilidade e extensibilidade no processo de desenvolvimento de frameworks orientados a objetos**. In: Workshop Iberoamericano de Engenharia de Requisitos e Ambientes de Software (IDEAS'98) abr. 1998, Torres.

[19] SZYPERSKI, C. **Component-oriented programming: a refined variation on object-oriented programming**. In: European Conference on Object-Oriented Programming - ECOOP, 1996, Linz.

[20] TALIGENT. **Leveraging object-oriented frameworks**. Taligent Inc. white paper, 1995.

[21] ZANCAN, J. **Nautilus - uma ferramenta de apoio à navegação sobre estruturas de frameworks**. Porto Alegre: UFRGS/II/CPGCC, 1998. *Master thesis in development*.