An Architecture and Methodology for the Design and Development of Technical Information Systems*

R. Capobianchi[1] and M. Mautref[1] and M. van Keulen[2] and H. Balsters[2]

[1] Alcatel Alsthom Recherche, Route de Nozay, 91460 Marcoussis, France,
{capobian;mautref}@aar.alcatel-alsthom.fr
[2] University of Twente, Dept. of CS, P.O. Box 217, 7500 AE Enschede, The Netherlands,
{keulen;balsters}@cs.utwente.nl

Abstract. In order to meet demands in the context of Technical Information Systems (TIS) pertaining to reliability, extensibility, maintainability, etc., we have developed an architectural framework with accompanying methodological guidelines for designing such systems. With the framework, we aim at complex multi-application information systems using a repository to share data among applications. The framework proposes to keep a strict separation between Man-Machine-Interface and Model data, and provides design and implementation support to do this effectively.
The framework and methodological guidelines have been developed in the context of the ESPRIT project IMPRESS. The project also provided for "testing grounds" in the form of a TIS for the Spanish Electricity company Iberdrola.

1 Introduction

The software engineering discipline advances both in the direction of better efficiency and effectiveness, as well as in the direction of expanding its application area. A significant part of these advancements can be attributed to the availability of increasingly powerful methodologies and accompanying tools. This has made it possible to build larger and more complex information systems (IS). Examples of these are CASE systems (some even with facilities for cooperative work), GIS's, CIM systems, etc. However, the design and development of these systems demands more from methodologies and tools than they currently often provide. For example, ever more strict requirements for reliability and extensibility emerge.

In the IMPRESS project (ESPRIT 6355), we have focussed on Technical Information Systems (TIS) which is a generalizing term for the aforementioned systems. A TIS can be defined as follows [BCMM94]:

> A Technical Information System (TIS) is an active repository of all the information describing a product (e.g. a physical system) or process: design models (structure, functions, behaviour), logistic support data (maintenance plans, operation procedures), documentation of the product and components, experimental knowledge issued from and supporting its operation, repair pieces and so on.

* This work has been conducted within the ESPRIT project IMPRESS (Integrated, Multi-Paradigm, Reliable and Extensible Storage System), ESPRIT Nº 6355

These data are typically produced by the various processes occurring during the product life-cycle: preliminary studies, design, manufacturing, production, deployment and exploitation. Each process retrieves the available information on the product required for its own treatments, and feeds the TIS with results that may be used afterwards by other processes.

In the project, we have built an object-oriented DBMS with special facilities for things that are needed in TIS's like multi-media. To build a TIS, of course, more is required than an OODBMS. We have devised a common architecture for TIS's and special methodological guidelines for the design and development of TIS's for this architecture. The IMPRESS system is accompanied by an extensive array of tools (the IMPRESS support toolkit) supporting this methodology, ranging from general database and user-interface design tools to TIS special purpose tools.

Because TIS's are generally distributed systems, we have paid special attention to this aspect. We have chosen for a repository-based approach meaning that the system is divided into a (possibly distributed) repository containing all shared data, and various independent applications performing specific tasks making use of the shared data in the repository. This architecture is further detailed in section 2. Section 3 gives a brief review of the properties of existing OO methodologies. This section will concentrate on those properties that are especially useful or problematic for the design and development of TIS's. Section 4 will then explain the specific methodological guidelines for IMPRESS. These guidelines work specifically towards the aforementioned architecture. It also describes the accompanying tools.

2 A common architecture for TIS

An IS like a TIS can be functionally decomposed in many parts, however, two major components could be identified: the management of the dialogue between the user and the system, and the management of the real world system. Those elements correspond to the various functions expected by the system end-user.

From the design point of view, we can define the *Model* part of a TIS as the gathering of all the necessary data and functions needed to provide the requested functionality of the application, which is mainly decision-making support based on access to a set of data. Typically, the model is a representation of what is usually called the UoD (Universe of Discourse). Its description should be directly based on the analysis of the real system and should only use terms relevant to the UoD.

One of the interests of designing an extensive model of a real-world system is that it allows the development of many different applications based on the same UoD. In order to make full use of this interest, the Model component may be split into two parts: a shared Model part (reused and shared by all the applications, managed by a database to function as a persistent data repository) and an application-specific Model part (extension of the shared Model to meet the specific demands of a particular application).

In order to distinguish such different Model parts, we have chosen to define the TIS *Database* component as the shared Model part and the TIS *Model View* component as the complete Model needed for a specific application. The Model View will be composed of the relevant subset of the Database, enhanced with specific data needed by the

application and not present in the Database, in other words, it defines an enhanced view over the Database [AB90, Ber91].

One of the main functions of the TIS will be to allow the consultation of Model View data by the end-user. Such interaction with the end-user is handled by another TIS component which we call the *Man-Machine Interface* (MMI). The aim of this component is to display the information to the operator in an efficient and useful way. It should also provide ways for the user to act on and modify the displayed information (e.g. actions on buttons or graphical representations of real objects, text editing, ...). More generally, all the multi-media information and the hyper-media navigation could be seen as part of the MMI. In other words, its aim is to translate end-user actions into computer actions by analyzing the requests and triggering the proper functions provided by other software components of the system. Reciprocally, it should interpret display requests, coming from the system, in order to communicate information to the end-user.

In systems designed for novice computer users, the metaphor of direct manipulation of objects by their graphical representations is more and more used [Hud87]. Clearly, this kind of MMI, based on user-driven interaction, seems to be more natural and easier to learn and use for an end-user of the system who is not a computer expert. In terms of computer implementation, this type of interaction needs a relatively complex architecture, mainly due to the huge number of end-user possible actions [Mye89, Hud87].

The main issue encountered while designing and implementing a TIS with Database, Model and MMI parts is the danger to mix together the different parts of the system without taking care of the distribution of responsibilities. This approach always leads to a very tightly-coupled design which raises lots of problems such as software complexity, reusability, evolutivity, maintenance or reliability of the system [HH89].

In the field of MMI design, one of the proposed answers to this problem is to impose a strict separation between the MMI and other parts of the application. Following the Seeheim model [Pfa85], the Model-View-Controller metaphor introduced by Smalltalk [Gol84, PK88] or the multi-agent architecture PAC [Cou90], the MMI and application parts should be separated by a dialog controller responsible for the communication and translation between them. An advantage is the identification of strict responsibilities. Unfortunately, much confusion exists regarding the responsibilities of the Controller.

Ensuring a clear separation between Model part and MMI part and clarifying the position of the Database, we propose to adopt an architectural decomposition of a TIS, that extends the controller approach to multi-application systems like TISs. This decomposition identifies four main parts (see figure 1) :

- *MMI*. This part is composed of objects like windows, buttons, and menus and is in charge of the interaction with the end-user of the application. The provided functionality only deals with the graphical aspect and user event handling. The part only uses “MMI terms”” (e.g. colors, pixels, mouse events, etc.)
- *Model View*. This part describes the functionality of a specific application. The functionality provided by the objects in this part only reflect the ones found in the real system and only use “Model terms” (e.g. substations, voltage, switch state ‘on’ or ‘off’). The data is split in two parts : application-specific objects and shared objects.
- *Translator*. This part synchronizes MMI events with Model View functions to ensure that the Model always faithfully represents the information displayed in the

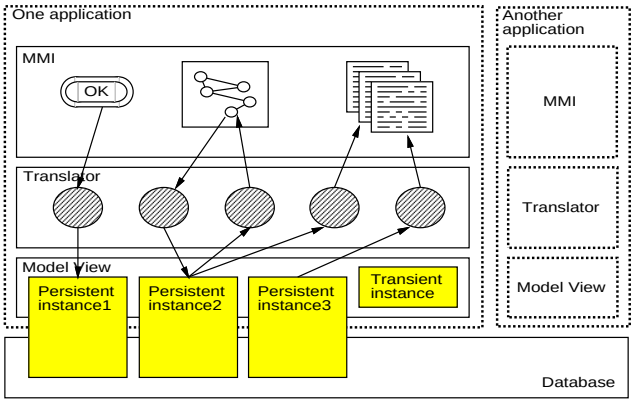

Fig. 1. Example of TIS decomposition

MMI, and the output of the MMI always faithfully represents the Model. To achieve this, the part is responsible for the translation of MMI modifications into Model modifications, and vice versa. (e.g. : if model-object.voltage = 220 KV then MMI-object.color := red).

– *Database*. The objects in this part will be shared among one or more different applications. Such objects will be persistent (i.e. they will survive sessions). The functionality provided by the Database is going to be used to build each specific Model View (common data part).

The expected advantages of this approach are the enhancement of the quality of the design on the following points :

– *Evolutivity* and *maintainability*. Modification of the MMI design (often required to integrate the final user requirements) will have less impact on Model design, and the MMI is be protected against modification of Model object implementation by the Translator.
– *Reusability of Database*. The Model View specializes the Model in the Database allowing some disparity with the fixed database schema and the use of the same database for one or more applications.
– *Modularity*. The separation between the different parts allows the use of different tools (or languages) for Model and MMI development. It also allows to deal with the distribution of the system and the use of communication protocols.

Figure 1 shows an example of TIS decomposition. In this example, two applications have been defined on the same Database. Each Model View is composed of persistent objects, coming from the Database, and transient objects specific to each application. The presentation to the end-user is handled by the MMI component and the coherency management between Model View and MMI is assumed by the Translator part. The arrows represent possible links between objects belonging to different layers.

The decomposition of a TIS in MMI, Translator, Model View and Database objects results in the definition of an *object-oriented framework* [FJ88]. A framework is com-

posed of classes with predefined relationships between each other and predefined behaviour that can be specialized to finally build a specific application. In fact, the primary goal of a framework is to be reusable and refinable for other application developments. In order to ease the transition between design and implementation, an implementation framework (i.e. a set of reusable classes written in a programming language), corresponding to this design framework, has been developed.

3 Brief review of existing OO-methodologies

In order to guide designers and programmers through the development of a TIS, methodological guidelines should be provided. The first step of this study was to look at general OO design methods and to identify how they can be reused and integrated in a proper methodology for TIS development within the IMPRESS environment.

Many methodologies have recently appeared to help a designer of object-oriented (OO) software. Those methodologies adopt more or less different approaches concerning design policies or decomposition in phases. In general, however, they all provide different models for the various views on a system one can have:

- a *static model*, representing the structural aspect of the system, i.e. the attributes and static relationships between objects (e.g. composition, inheritance, instantiation, grouping).
- a *collaboration model*, describing the cooperative side of the system, i.e. the static properties of object interactions (client-server dependencies, the data flow, etc.)
- a *behavioural model*, describing at object level the actions objects can perform, and at system level inter-object behaviour (state transition diagrams, scenarios, etc.)

While studying the different methodologies, we learned that, although these models are general, there are basically two approaches concerning modelling. The first approach, which we have called *data-driven*, focuses on a "natural" data model first, and then enhances the model with, from the UoD's point of view, logical behaviour [CY90, Boo91, RBP+91]. We have called the other approach *responsibility-driven*, because it pays attention to the responsibilities (i.e. behaviour) first and then focuses on the information an object needs to perform the required functionality [Mey88, WBWW90]. Also, not every model seems to be as fruitful or even wanted for the different parts in our architecture. For example, a collaboration model for application objects is perhaps the most interesting and important model, while such a model doesn't clarify much for database objects.

The data-driven approach is an adaptation of ADT (Abstract Data Type) design methods for object-orientation. Firstly, the structure of the "real world" is modelled as closely as possible. Attention is paid to identifying classes, attributes, relationships that exist between those classes (e.g. aggregation, subtyping), and which of the possible states of the objects are valid. Based on this model, one tries to find the operations that the objects must logically be able to perform. We have observed that the resulting model is quite "natural", i.e. it reflects the UoD, and is therefore easily readable. Its high task independence makes the need for change as a consequence of changing demands low. The weak

points are, for example, that it may contain more functionality than is actually needed, and the stress on the data results in a model with little large scale functionality.

The responsibility-driven approach is almost an anti-pole. It focuses firstly on structuring the task of an application by dividing responsibilities among the objects. It uses the same modelling primitives (i.e. classes, subtyping, etc.). The attributes of an object follow naturally from the information that is needed for the object to fulfill its responsibility, and, as a result, encapsulation of these attributes is quite natural. An advantage of this approach is that the design is geared towards providing the necessary functionality with the least effort possible. However, as a result, the design is rather specific for the target application and, therefore, changing demands almost always call for changes in the design. On the other hand, changing the design can be done without much effort because of the high data independence.

Although both approaches appear to exclude each other, one need not make an absolute choice. For example, in the responsibility-driven approach, one may decide to refrain from deviating too much from the logical structure of the UoD. It depends on the purpose of the design which of both approaches is likely to work best. We advise to use the data-driven approach for the design of the repository (i.e. database design), because it is often not known what future applications may use it for and, therefore, task independence is a useful property. However, for designing the applications that use this repository, the responsibility-driven approach is likely work well, because the application's task is often clear and the less effort needed is a nice feature. The methodology described in the following section reflects this advise.

4 Methodological guidelines for TIS design

As explained earlier, we have directed our attention concerning methodology towards the specifics of developing TIS's. The common architecture of section 2 forms the basis of our methodological guidelines. We have thought about what guidelines and tools we could give for developing the architectural components. Another issue is a logical order of design and development of these architectural components. In dividing the development of a TIS into phases, we also took notice of the fact that in technical environments, people tend to consider instantiation (i.e. data entry) as part of IS development.

To illustrate the phases in the proposed global development process, the demonstrator application of the IMPRESS project is used as a running example. The demonstrator concerns a TIS for the Spanish electricity company Iberdrola. The users of the system are operators of an electrical network. Their task is to control the flow of electricity inside the system and efficiently and effectively react on malfunctions of electrical components in the network. To be able to do that, the operators need facilities to simulate the effects of possible actions on electricity flow before actually performing these operations in reality. Their tasks also involve retrieving information about the structure of the network, maintenance information, procedures to be followed, and other textual and schematic documentation. Because of the large amounts of information available, advanced querying facilities are required [BCMM94].

Figure 2 shows the proposed global development process. The design phase has been subdivided into the design of each of the four architectural components. Although there

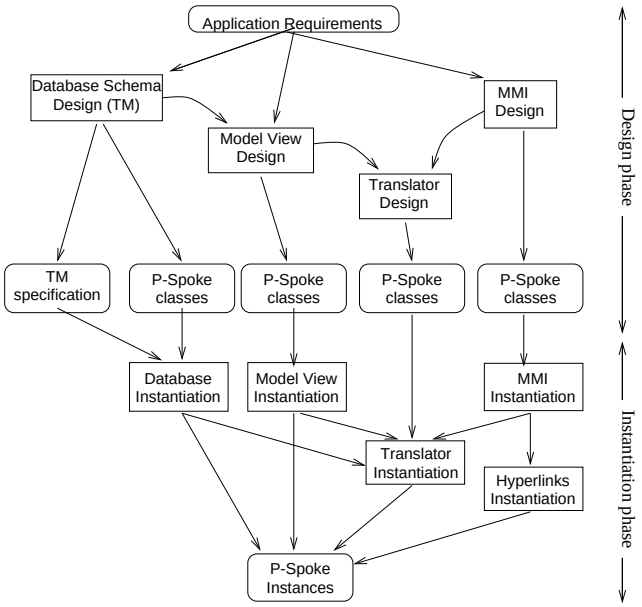

Fig. 2. Global development process

is much freedom to do things in parallel, there are some input/output dependencies between them (see Figure 2). In the sequel, all phases pass the review explaining what each phase is for, which guidelines apply there, and what tool support we provide.

Reliability is in the context of TIS's an important issue. Therefore, we have chosen to base database design on the formal specification language TM [BBZ93, BBdB+95]. The ability to do more advanced kinds of verifications and validations of the schema other than simple syntactical checks is a big advantage. The gentle force imposed on the designer to be more exact also adds to the quality of the resulting schema. By offering a designer high-level OO language constructs and embedding the language in a toolset, he is shielded against the burdens a more formal treatment brings with it [FKS94a, FKS94b]. In the project, we have, together with Iberdrola, specified a database schema, using the data-driven approach, for storing the network configuration and its state.

Based on the database schema of the repository, the *model view*, which is like an external schema in the ANSI/SPARC architecture, can be designed for a particular application. It contains a subset of the classes of the database schema supplemented with application dependent classes for both persistent and transient data. We have decided to do application programming in the programming language PSPOKE, which is a SmallTalk-like "pure" OO language with facilities for persistency (it uses the object server GEODE for this [BHPT]). To achieve a smooth connection between the schema of the repository designed in TM and the model views, we have developed a TM-to-PSPOKE translator, that translates both the structural part of the schema (i.e. the class specifications) as well as the behavioural part (i.e. the constraint and method specifications).

Almost completely parallel to the design of the database and an application's model

view, one can design the MMI. For this, the IMPRESS support toolkit provides the Motif GUI design tool CAID which, among other things, takes care of generating almost all PSPOKE code for the GUI. Because a lot of useful results have already been achieved in the area of GUI design, we have not substantial methodological guidelines for GUI design. We did, however, develop a tool (the Synoptic Editor) for designing diagram languages, because in TIS's a lot of schematics is used (e.g. a global view of the network structure, electrical schemas of various levels of detail).

As explained earlier, the greater part of the results of research towards methodology done in IMPRESS, has been in providing a good connection (the Translator) between the MMI and the model view while still leaving them as independent as possible. Because of the way we have defined the translator component, almost all of its design can be taken care of by a tool called POTOMAC. For example, one can define graphically that instances of class Powerplant are represented in the GUI by a specific symbol and that the specific states of a power plant (e.g. operational/nonoperational, hydraulic/nuclear/thermal) are represented by several variations of the symbol (e.g. using color). POTOMAC takes care of generating the code for this including support for automatic propagation of changes in either the GUI or the Model View to the other one.

The result of these design activities is on one hand a TM specification of the database schema and, on the other hand, a PSPOKE program of the application. However, in case of a TIS, this is not considered to be a complete system. Before a TIS can be made operational, it must already contain a significant amount of information. For example, (digital scans of) manuals, detailed schematics of the various electrical circuits, the structure of the electrical network, etc. all being hyperlinked together and supplemented with (cross-)indexes to enable an operator to retrieve all, for his particular situation, relevant information. In the IMPRESS support toolkit, tools for the various kinds of instantiation are provided, for example, a multi-media instance editor for creating multi-media documents, a hyperlinks editor, and many more. Some of the tools aid both in the design and instantiation phases, for example, POTOMAC and the Synoptic Editor. We've also developed an experimental tool for information retrieval called COLORADO, because, as mentioned before, one of the most prominent difficulties for an operator is how to obtain relevant information about a particular crisis situation in the least time possible in order to be able to react to the situation before too much damage has been done.

As a final remark, we note that, of course, database design is not always necessary. Usually, only a new application for an existing repository is required and therefore database design has already been done. Sometimes however, the repository may need some extra functionality which means that some database design has to be performed.

5 Conclusions

Cooperation between end-user companies and research institutions in the context of an ESPRIT project has proven to be fruitful for testing and refining ideas. In the IMPRESS project, this cooperation has led to a well-balanced design and implementation framework for developing Technical Information Systems. The framework proposes an architecture and provides design and implementation support, as well as methodological guidelines to effectively develop TIS's using this framework.

The expected results of maintainability, reusability, and modularity as listed in section 2 have been achieved quite satisfactorily with readily available technology[2]

In the project, we focused on practical usability of the framework and methodological guidelines. There has already been an abundance of research done in the realm of methodology [Boo91, CY90, Mey88, RBP+91, WBWW90], and it has been our incentive to apply this acquired knowledge to the particular application domain of TIS. As a result of this investigation, we came to incorporate database instantiation as part of the global development process, because for a TIS to be useful, it must already contain a lot of information, and because of the crucial nature and the amount of work involved in filling the database, this activity should be addressed during development.

Another issue that was thoroughly investigated for this purpose was the controversy between data- and responsibility-driven design. It turned out that the one is not better than the other, but that both are suitable for different purposes; the data-driven approach is more suitable for the design of the shared Model, while the responsibility-driven approach is more suitable for application-specific design.

To summarize the above, the IMPRESS project offered us the opportunity to bring ideas about methodology into practice in the specific domain of TIS. In the process of developing an architectural framework for building TISs, the subjects of database instantiation and data- vs. responsibility-driven design proved to be major issues. Also, by working towards a common architecture for TISs, it became possible to provide good support for the development process in the form of a design and implementation framework combined with practical methodological guidelines.

Acknowledgements

We would like to thank our partners[3] for a successful project. Concerning the topic of this paper, we'd like to thank in particular Rolf de By (University of Twente), Enrique Burguera (Iberdrola), and Florence Ardorino (Alcatel Alsthom Recherche) for their contributions.

References

[AB90] S. Abiteboul and A. Bonner. Objects and views. Technical report, INRIA, 1990.

[ACM+94] F. Ardorino, R. Capobianchi, M. Mautref, R.A. de By, and M. van Keulen. Methodological guidelines for IMPRESS. Technical Report IMPRESS/AAR-TECH-W4-005-R0, Alcatel Alsthom Recherche, Route de Nozay, 91460 Marcoussis, France, November 1994.

[BBdB+95] R. Bal, H. Balsters, R. A. de By, A. Bosschaart, J. Flokstra, M. van Keulen, J. Skowronek, and B. Termorshuizen. *The TM Manual version 2.0 revision E*. University of Twente, Enschede, The Netherlands, June 1995.

[2] We'd also like to mention that the demonstrator application of the project won a prize in the Second Edition of the Prizes to Companies and Institutions in the Basque Country in the field of the application and development of information technologies [BMA95].

[3] Companies and universities participating in the IMPRESS project are: Alcatel ISR (France), EDS (France), Alcatel Alsthom Recherche (France), Iberdrola (Spain), Bureau Van Dijk (Belgium), University of Twente (The Netherlands)

[BBZ93] H. Balsters, R.A. de By, and R. Zicari. Typed sets as a basis for object-oriented database schemas. In *Proceedings 7th European Conference on Object-Oriented Programming (ECOOP), July 26–30, 1993, Kaiserslautern, Germany*, 1993.

[BCMM94] E. Burguera, R. Capobianchi, F. Marquinez, and M. Mautref. A technical information system for a dispatching control center. In *Proceedings of ISAP'94 (Intelligent System Application to Power systems), Montpellier, France*, September 1994.

[Ber91] E. Bertino. A view mechanism for object-oriented databases. Technical report, Universita di Genova, 1991.

[BHPT] J. Besancenot, L. Hammami, P. Pucheral, and J.-M. Thévenin. Geode, a multithreaded storage object library for advanced transactional applications.

[BMA95] E. Burguera, F. Marquinez, and I. Angulo. SITED: Systema de informacion tecnica de despacho de maniobras. *PC Week Magazine*, supplement 263, may 1995.

[Boo91] G. Booch. *Object oriented design with applications*. Benjamin/Cunnings series in Ada and software engineering. Benjamin/Cummings, 1991.

[Cou90] J. Coutaz. Interface homme-ordinateur - conception et réalisation. Technical report, Dunod, April 1990.

[CY90] P. Coad and E. Yourdon. *Object-oriented analysis*. Yourdon Press computing series. Yourdon Press, Prentice-Hall, 1990.

[FJ88] B. Foote and R.E. Johnson. Designing reusable classes. *Journal of Object-Oriented Programming*, June 1988.

[FKS94a] J. Flokstra, M. van Keulen, and J. Skowronek. IMPRESS database design tool: a high-level design toolset based on formal theory. In *Exhibition and Demonstration program of the 8th european conference on object-oriented programming (ECOOP), Bologna, Italy, July 4–8, 1994*, July 1994.

[FKS94b] J. Flokstra, M. van Keulen, and J. Skowronek. The IMPRESS DDT: A database design toolbox based on a formal specification language. In R.T. Snodgrass and M. Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994*, volume 23, page 506, 1994.

[Gol84] A. Goldberg. *Smalltalk-80: The Interactive Programming Environment*. Addison-Wesley, 1984.

[HH89] D. Hix and H.R. Hartson. Human-computer interface development: Concepts and systems for its management. *ACM Computing Surveys*, 21(1), March 1989.

[Hud87] S.E. Hudson. UIMS support for direct manipulation interfaces. *Computer Graphics*, 21(2), April 1987.

[Mey88] B. Meyer. *Object-Oriented software construction*. Prentice-Hall international series in computer science. Prentice-Hall, 1988.

[Mye89] B.A. Myers. User-interface tools: Introduction and survey. *IEEE Software*, January 1989.

[Pfa85] G.E. Pfaff. *User Interface Management Systems*. Eurographics Seminars. Springer-Verlag, 1985.

[PK88] S.T. Pope and G.E. Krasner. A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, August 1988.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premeriani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

[WBWW90] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object Oriented Software*. Prentice-Hall, 1990.

This article was processed using the LaTeX macro package with LLNCS style