INFORMATICA, 1990, Vol.1, No.1, 167–182

VILNIUS: A SYSTEM FOR THE CONSTRUCTION OF INTELLIGENT APPLICATIONS

Albertas ČAPLINSKAS, Vilius MATULIS, Viktor TONKICH

Institute of Mathematics and Cybernetics,
Lithuanian Academy of Sciences,
232600 Vilnius, Akademijos St.4, Lithuania

Abstract. This paper presents an overview of the system VILNIUS, its design goals and general architecture along with some brief remarks on the experience of work with the system. VILNIUS is both an application shell and program development environment. It is used to create intelligent applications and to combine several different applications into one application system. The intended primary application domain is the planning and scheduling of large research and development projects.

Key words: intelligent application shell, program development environment, life cycle model.

Introduction. In the last years numerous efforts were made to create various software development and programming environments in order to improve the quality of products and the productivity of programmers. A key objective of such environments is to support all aspects of software production through its life cycle, encompassing the project management and technical development. At the same time numerous efforts were made to achieve domain-specific automatic programming. A principle behind it is that user's needs can be

expressed in some machine independent specification language and automatically transformed to programs. Software tools that perform this task are called generators or synthesizers.

The system VILNIUS, developed at the Institute of Mathematics and Cybernetics of the Lithuanian Academy of Sciences, is both a domain-specific application shell and a program development environment. VILNIUS, as an application shell, can be filled with data, input, output and processing formats and may be used to create intelligent applications, such as the systems for planning and scheduling large research and development projects. A target system is created from a given set of application modules. Required linkages among the modules are generated automatically. VILNIUS, as a program development system, supports the development of application modules.

Target systems are intelligent, problem-oriented application systems, which provide fill-in -the-form input languages. These languages allow a computationally naive user to describe problems in the terms and concepts of application domain. Here a user is offered a very limited set of means to achieve a rigidly defined set of objectives. Having only a limited set of things to understand, one can concentrate on the problem itself, rather than on the special procedures required to solve it.

This paper presents an overview of the VILNIUS general architecture, along with some brief remarks on the experience of work with this system. To place the VILNIUS architecture in a context, we shall first present the design goals, history and environment in which VILNIUS is to function. Then, we shall briefly sketch the model of the software life cycle upon which we have based VILNIUS and the techniques by which we intend to design and implement the target software.

Design goals and history. Our goal was to allow the development of medium- sized (50 000-500 000 source code

lines) complex software projects, involving a team of up to twenty members. The software constructed with VILNIUS may be developed over a period of several years and must have sufficient performance to be usable in a production environment.

There was a number of design goals and features that had an impact on the architecture of the system. Nevertheless, the basic design goals are:

—to support modular programming in IBM 360/370 Assembler H language,

—to accommodate a programming language, a design language and requirement specification language,

—to simulate a target software run-time environment and allow the execution and debugging of application modules before generating a framework of target software,

—to produce target software automatically from a given set of application modules and a given design specification,

—to allow the integration of several different applications into one package,

—to support the configuration management and installation of target software,

—to support a dynamic documentation of target software,

—to provide a collection of implemented tools being so highly integrated that the system could be considered as a single tool.

The design and implementation of the system VILNIUS was begun in 1975 (Vaičiulis and others, 1976) and a demonstrable prototype version was in operation under DOS/ES on ES-1020 computer in 1978. The first commercially available version runs under OS/ES MVT 6.0 since 1980.A version VILNIUS-2 (Čaplinskas and Matulis, 1981;Čaplinskas, Matulis and Tonkich, 1982) was the first version that could reasonably be called an integrated environment. This version

runs under OS/ES MVT 6.0 and under VM BOS since 1985. At present we are developing a more elaborate version VILNIUS-3, which is to be implemented on ES-1045 computer and IBM PC/AT compatible jointly. ES-1045 is used as a mainframe and IBM PC/AT as a workstation. Further we describe the version VILNIUS-3.

Environment. The environment in which we intend VILNIUS to be used is composed of software developers and software managers who have several years of experience in constructing software. Several different applications can be developed by different teams together. The system VILNIUS administrator function is provided. This function includes various tasks, such as the system installation and maintenance, applications cataloguing, creation of data bases and program libraries, etc.

Life cycle model. We follow some variation of the structured life cycle model. Our life cycle concept (Čaplinskas and Pamedienė, 1983) is that application development consists of the following four phases and their primary activities:

1.Requirement evaluation.

2.Requirement analysis and logical design (requirements analysis, object and relationship identification , action identification , operation identification , user interface design, domain dictionary preparation).

3. Structured detailed design (evaluation of detailed design specification, identification of modules and interfaces between modules, algorithm design).

4. Top-down implementation (generation of target system kernel, module coding, documentation preparation, test design, kernel acceptance test, module acceptance test, application integration, application acceptance test).

Our approach to a logical design is fundamentally based on the object concept(Shlaer and Mellor,1989). Its purpose is:

— to capture the domain-specific knowledge of application in a form that lends itself to verification by domain experts;

— to represent the requirements in a form that is easily mapped into a non-object-oriented design;

— to provide, through a formal model of the problem domain, a foundation upon which detailed design decisions can be made;

— to transfer the domain knowledge to the low level designers and programmers.

Our view is that application can be broken up into entities and relationships, which are meaningful to a user. The conceptual entities of the problem domain must be identified and represented by objects that have properties (attributes). Objects are abstractions of like instances of any concept in the real world. A property represents a fact about the corresponding real-world entity. Each object consists of some state (its properties) and a set of operations that determine its external behaviour. A relationship is an abstraction of a systematic pattern of association that exists between the corresponding real-world entities.

Problem domain information is expressed as much as possible in terms of objects, attributes and relationships. After a structural model has been produced, a designer must turn his attention to the dynamic behaviour of the conceptual entities and associations. A life cycle of an object or relationship must be represented as a number of states. Each state represents a condition of the object. An object is moving from one state to another by some operation. An application must be represented as a number of actions, which have been created to control operations on objects. Actions are abstractions of the corresponding tasks in problem domain. An action can be made of any number of actions.

A user interface is represented as a set of forms. The forms in our approach can be much more general and elaborate than business paper forms. They are considered as general mappings,which, with the attribute values given , generate the appropriate input or output messages. The forms represent a structured approach to these messages and guide a user to filling a request for information.

The results of the logical design work are represented in the form of an application domain dictionary. The requirement specification language is used for this purpose.

The next step in our approach is the structured detailed design. In this step, the focus shifts from the real-world application problem to the solution of this problem in an automated system. During a detailed design phase,the structure of the target system should be defined in detail, the inputs, functions and outputs for each module being specified. The design language is used for this purpose. The results should be represented in a form of detailed design specification. A framework of this specification is generated automatically from a domain dictionary.

In top-down implementation step, the kernel of the target system from a given detailed design specification is generated automatically. The application modules must be coded in IBM Assembler H, Fortran IV or PL/1 language. The kernel and a set of application modules are integrated into the framework of the ultimate system.

Architecture. VILNIUS consists of two main parts: a framework and a set of tools. The VILNIUS framework supports the approach described in the previous section by maintaining an independent database and independent set of libraries for each individual project. When a new project is initiated, a database and libraries by a system administrator are created. Work on each project can be proceeded autonomously, using the full resources of VILNIUS. The re-

sources available to work on each individual project have the following main components: a database; a program library; a task library; a test library; a documentation library; and a working file store.

At present the system VILNIUS consists of the following tools:

—requirement specification language processor;
—design language processor;
—programming language processor;
—debugger;
—document preparation system;
—data management tools;
—configuration management tools;
—skeleton of kernel;
—scheduling module;
—dynamic document editing tool;
—user teaching system.

The VILNIUS requirement specification language is an interactive form-oriented language, which allows to describe objects, relationships, operations, actions and forms. The application designer defines requirements in a form of domain terms dictionary. Each entry in this dictionary is a description of some object, operation, relationship, action or form. Each description consists of two main parts: a conceptual description and a design decision description. The requirement specification language processor supports the capturing, organization and analysis of user requirements. In addition, this processor generates automatically a framework of a detailed design specification from user requirements. The processor runs on IBM PC/AT.

The VILNIUS design language is a form-oriented batch language.This language allows to describe the schemes of data bases, computational schemes and their implementation. The designers define computational schemes in the form of mul-

tilevel controlled production systems (Georgeff,1982). These schemes describe how procedural task execution plans can be produced from a non-procedural task descriptions (user's messages) . The plan is used for interconnection of given modules. Our approach to module interconnection is similar to the approach reported in (Prieto- Diaz and Neighbors,1986). The design language processor performs a detailed design specification analysis and transforms these specifications into internal representation (semantic model of a given application domain) (Čaplinskas and Matulis,1981).

Though PL/I and FORTRAN IV as the application module implementation languages are provided in VILNIUS, VILNIUS has its own application module implementation language (MIL) (Čaplinskas and Jazukevičius ,1986), too. MIL is a macroextension of IBM 360/370 Assembler H language. It supports modular programming and allows separate compilation of application modules. MIL also provides interaction means between application modules and run-time environment of a target system . MIL processor is implemented as IBM 360/370 Assembler H preprocessor. A special purpose checker is used to implement a guard,which is a filter for the values of a certain parameter type and within a specific subrange. This checker was developed at the Institute of Computers in Minsk as a single autonomous tool (Margolin,1982).

VILNIUS-debugger simulates a target system run-time environment and allows the execution and debugging of application modules before generating a framework of this system.

A documentation preparation system is a special purpose word processor. It was developed at the Computing Center,the Siberian branch of the USSR Academy of Sciences, as a single autonomous tool and was incorporated into VILNIUS.

Data management tools allow to create and reorganize data base files, to add or change data, to filter data with a given query, and to print or display reports. A data base in

VILNIUS is thought of as a segment hierarchy. The individual data items are grouped together into segments. The segments are associated together in parent-child relationships. For data manipulation and modification, query and reporting language (QRL) is provided. QRL is a non-procedural linear keyword-oriented language,which is embedded as a sublanguage in MIL. QRL data retrieval facilities allow a multiple entity target and relational operators as qualification expressions. The reporting facilities allow an automatic column alignment, placement of headings and titles, data editing and other output formatting functions. All required views of data base must be preliminary described in a detailed design specification as data base subschemes.

Configuration management tools provide some facilities for the creation and maintenance of various libraries and for a target software integration and installation.

The design language processor, MIL processor,debugger, documentation preparation system, data managements tools and configuration management tools run on mainframe computer.

The architecture Δ_p of any target software P is defined as follows:

$$\Delta_p =< K, M_p >,$$

where K is the kernel of a system P, M_p is a semantic model for a given application domain.

The kernel K is a built-in component of any target software. Currently it includes the following components: a task statement translator T_1 , an action scheduler T_2 , a problem solving plan constructor T_3 , a problem solving plan interpreter T_4 and a VILNIUS-virtual-machine μ . For details about these components see (Čaplinskas and Matulis,1981; Čaplinskas and Tiešis ,1986).

A semantic model M_p provides the following levels of application domain description: a communication level σ_p ,a

functional level φ_p , a static properties description level ω_p , an algorithmical level α_p and an application modules level γ_p.

The main way, in which the components of K and M_p interact in a system P , is illustrated by Fig. 1. T_5 in this figure denotes a set of data management tools. For details about the semantic model M_p and target system's architecture see (Matulis,Tonkich and Čaplinskas,1989).

The set of schedule modules includes 11 modules. These modules can be used as the basis for building network analysis and critical path scheduling tools in target softwares.

The dynamic document editing tool allows to fill up a skeleton of documents, prepared using the WORDSTAR 2000 system, by the current options of target software, retrieved from a detailed design specification.

The users learning system provides an introduction to VILNIUS command language.The system runs on mainframe computer.

All VILNIUS tools that run on mainframe computer are integrated into a single system. The inclusive integration style is used. The inclusive integration is based on the existence of a VILNIUS-framework, which is considered as the dominant component. All other components are subservient.A user can work with a subservient component only through or in conjunction with the dominant component.The user interface is based on a user-driven interaction through a linear keyword-oriented command language (Laurinskas and Tonkich,1984).

Experience with VILNIUS. The intended primary VILNIUS application domain is the development of systems for the planning and scheduling of large research and development projects.

Fourteen various applications, such as multi-project analysis by time, cost evaluation, short-term and long-range planning, scheduling, control of research and development projects and activities in an enterprise or its divisions was developed

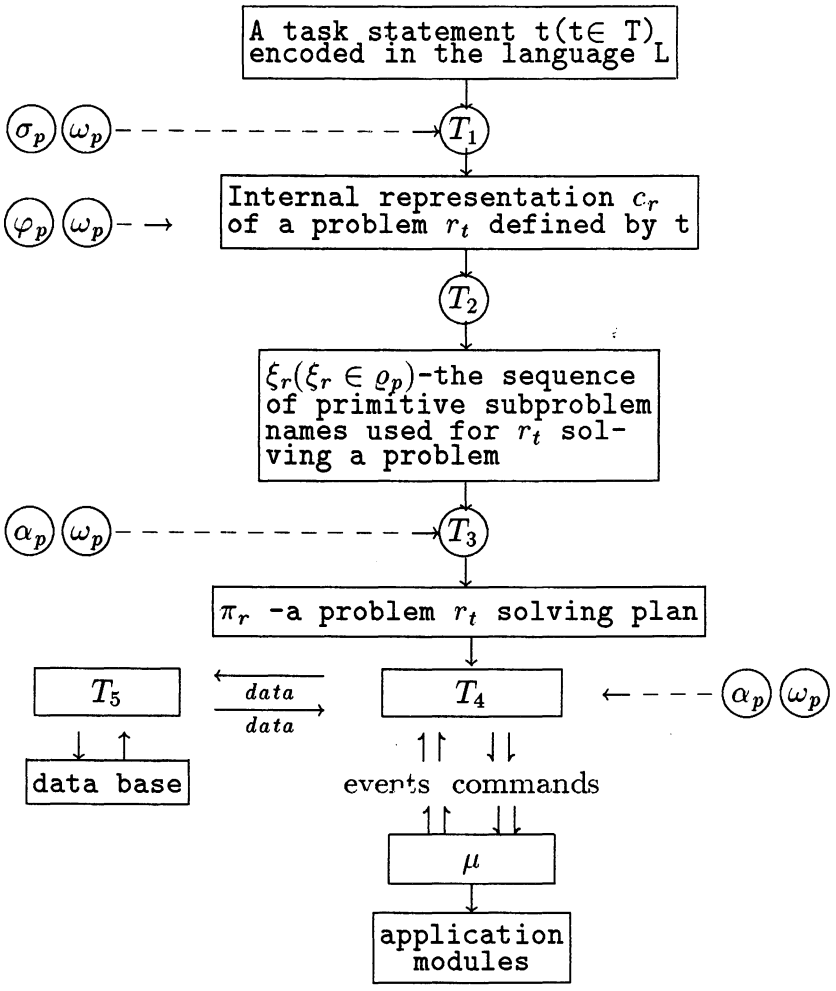

Fig. 1 The interaction of components of the system VILNIUS-2 target systems

under the contracts with government departments of the USSR and Machine Construction Bureaux in Leningrad, Novosibirsk, Kiev, Kharkov and Vilnius. Most applications were integrated into two management information systems, NIOKR and TOPAZ, that are based on the arrow-oriented network model and the node-oriented network model, respectively. Depending on the disc storage available, up to 5 000 activities in each project and up to several hundred projects in both systems can be processed.

Conclusions. In this paper we have described the main components of the system VILNIUS. The design of VILNIUS addresses two distinct sets of concerns. First, we are seeking for tools to support the development of intelligent applications. The second set of concerns arises from our interest in the application of formal design methods to practical large scale software development. A particular challenge is the combining of formal and informal approaches in the development of a large system. The positive aspects of our approach are:

1) VILNIUS is designed from a coherent software engineering philosophy and is not a collection of ad hoc tools;

2) it provides the integration of traditional data base facilities as well as problem-solving system facilities.

Our effort is not only a theoretical proposal, but also an actual tool for constructing intelligent applications. Our experience of work with VILNIUS has been favorable. The applications were developed and integrated with less effort, few bugs have been reported and fixed easily.

The organization of VILNIUS has benefited from the lessons of several rounds of implementation. On the other hand, some other systems , including DISUPPP (Perevozchikova and Jushchenko, 1986), MEMO (Pruuden, 1983), PRIZ (Tyugu, 1984), VIKAR (Karatuev, 1982), (Dixon and others, 1989), (Hagino and others,1983), also influenced the VILNIUS architecture.

Many issues are touched upon in this paper, but we have only attempted to sketch our approach. More details can be found in(Čaplinskas and Matulis, 1981; Čaplinskas and Pamedienė, 1983; Čaplinskas, 1988).

Acknowledgement. Since 1976 over thirty persons have spent time working on the VILNIUS project. VILNIUS is primarily an outcome of their work and enthusiasm. We wish to thank everyone,who has contributed to the design and implementation of this system.However, we feel it is important to acknowledge the major contributors. E. Tiešis implemented much of the kernel and a part of design language processor; V. Jazukevičius – MIL processor; Z. Pamedys, A. Bražūnis, B.Bagdonavičienė, R .Pamedienė – data management tools; R. Jonušas, V. Kučas – configuration management tools; and R. Siniuvienė, L. Čikotaitė, A. Janavičiutė – a requirement specification language processor. Our thanks also to B.Pikšrienė for her constructive comments.

The development of VILNIUS has been partly supported by a grant from the USSR State Committee for Science and Technology.

REFERENCES

Čaplinskas, A. A., and V. A. Matulis (1981). *The System VILNIUS. Conception, Structure and Using Technology.* Inst. Math. and Cybern. Acad. Sci. Lith. SSR , Vilnius. 146pp. (in Russian).

Čaplinskas, A. A. , V. A. Matulis and V. D. Tonkich (1982). VILNIUS-2 -the problem-oriented software development system. In V.Matulis and A. Čaplinskas (Eds.), *Avtomatizatsija protsesov planirovanija i upravlenija,* Vol.9. Inst. Math. and Cybern. Acad. Sci. Lith. SSR , Vilnius. pp. 108-120 (in Russian).

Čaplinskas, A. A., and R. J. Pamedienė (1983). *The System VILNIUS. Logical Design of Target Software Packages.* Inst.

Math. and Cybern. Acad. Sci. Lith. SSR , Vilnius. 180pp. (in Russian).

Čaplinskas, A. A., and A. E. Tiešis (1986). VILNIUS-2 monitor as a virtual machine. In V. Matulis and R. Kanopa (Eds.), *Avtomatizatsija protsesov planirovanija i upravlenija,* Vol.11. Inst. Math. and Cybern. Acad. Sci. Lith. SSR , Vilnius. pp. 18-25 (in Russian).

Čaplinskas, A. A. (1988). *Design Principles of Problem- Oriented Software Systems.* Inst. Math. and Cybern. Acad. Sci. Lith. SSR , Vilnius. 174pp. (in Russian).

Čaplinskas, A. A., and A. V. Jazukevičius (1986). The use of the implementation language in the development support system VILNIUS-2 for module interaction with the application run-time system. In V. Matulis and R.Kaňopa (Eds.), *Avtomatizatsija protsesov planirovanija i upravlenija,* Vol.11. Inst. Math. and Cybern. Acad. Sci. Lith. SSR , Vilnius. pp. 9-17 (in Russian).

Dixon, N., G. D. Parrington, S. K. Shrivastava and S. M. Wheater (1989). The treatment of persistent objects in Arjuna. *The Computer Journal,* **32**(4), 323-332.

Georgeff, M. P. (1982). Procedural control in production systems. *Artificial intelligence,* **18**(2), 175-201.

Hagino, T., M. Honda, A. Koga, K. Kojima, R. Nakajima, E. Shibayama and T. Yuasa (1983). *The IOTA Programming System.* In R. Nakajima and T. Yuasa (Eds.), *Lecture notes in computer science,* Vol.160. Springer-Verlag, Berlin- Heidelberg-New York- Tokyo. 217.pp.

Karatuev, V. G. (1982). A method for applications program package kernel construction. In V. M. Matrosov (Ed.), *Paketi prikladnich program i ich postroenie,* Nauka, Novosibirsk. pp. 161-179 (in Russian).

Laurinskas, J.V., and V. D. Tonkich (1984). *The System VILNIUS. An Input Language.* Inst. Math. and Cybern. Acad. Sci. Lith. SSR , Vilnius. 65pp. (in Russian).

Margolin, M. C. (1982). Parameter filtering of program objects.

Programmirovanie, **4**, 19-26 (in Russian).

Perevozchikova ,O.L., and E. L. Jushchenko (1986). *Systems for Computer Solution of Dialogue Problems.* Naukova Dumka, Kiev. 264pp. (in Russian).

Prieto-Diaz, R., and J. M. Neighbors (1986). Module interconnection languages. *The Journal at Systems and Software,* **6**, 307-334.

Pruuden, J. I. (1983). The MEMO metamonitoring systems family. In A. Vauglaid (Eds.), *Proceedings of the 2nd Conference "Automation of Applied Software Packages Production (Automation of Translators Production)".* Tallinn. pp. 86-88 (in Russian).

Shlaer, S., and S. J. Mellor (1989). An Object- Oriented Approach to Domain Analysis. ACM SIGSOFT, **14**(5), pp. 66-77.

Tyugu, E. H. (1984). *Conceptual Programming.* Nauka, Moscow. 255pp. (in Russian).

Vaičiulis, B.,V. Matulis, R. Siniuvienė and A. Čaplinskas (1976). Toward a design of application packages. In B. Vaičiulis (Eds.), *Avtomatizatsija protsesov planirovanija i upravlenija,* Vol.3. Inst. Math. and Cybern. Acad. Sci. Lith. SSR , Vilnius. pp. 9-47 (in Russian).

Received January 1990

A. Čaplinskas studied Mathematics at the University of Moscow, where he received his Diploma in 1966. From 1966-1970 he was working at the Vilnius University. Since then he is working at the Department of Mathematical Logic and Theory of Algorithms, Institute of Mathematics and Cybernetics of the Lithuanian Academy of Sciences. His professional interests concentrate on programming methodology, software engineering and knowledge based systems.

V. Matulis received the Dipl.-Math. Degree from the Vilnius University in 1956, and the Degree of Candidate

of Physical and Mathematical Sciences from the University of Leningrad in 1964. At present he heads the Department of Mathematical Logic and Theory of Algorithms at the Institute of Mathematics and Cybernetics. His professional interests concentrate on computer aided planning and control of projects, decision support systems, provability in formal systems, software system design methods and advanced programming environments.

V. Tonkich received the Dipl.-Math. Degree from the Vilnius University in 1972. At present he is the Managing Director of the Soviet -Austrian joint venture Baltic Amadeus. His professional interests concentrate on computer and programming tools for planning and control of software projects.