INFORMATICA, 1996, Vol. 7, No. 3, 389–405

ANALYSIS OF WELL-UNDERSTOOD DOMAIN FOR SOFTWARE TOOLS BUILDING

Vytautas ŠTUIKYS

Kaunas University of Technology
Studentų 50, 3031 Kaunas, Lithuania
E-mail: vytas.stuikys@if.ktu.lt

Abstract. This paper presents the framework for well-understood domain analysis as a decisive stage for the successive process of domain-specific software tools building. Specific features of the well-understood domain analysis are formulated. Initial model of the tools to be built and the analysis process are described. Analysis is performed with the reusability concept in mind and as a result essential domain knowledge is extracted. The latter is defined by a term "domain knowledge template".

Key words: domain analysis, reuse, VHDL, domain-specific program, program generation, knowledge template.

1. Introduction. Domain analysis is a widely discussed topic in the software engineering area aiming to introduce the available framework for domain knowledge extraction and it successive use in software systems building. The benefit of this process can be significantly enhanced if its execution is performed with the reuse concept in mind. By domain knowledge we mean domain assets, i.e., components in the form of requirements, models, specifications, designs, templates, source codes, etc. If these components are gained from a domain in such a manner that their most general features and characteristics are defined at the stage of the process performed, then this asset can be transferred to a system designer and reused in the same software system or relative applications of that domain.

Building of the tool or system is our aim. The tool should provide facilities for domain specific program generation. By a domain specific program in this context we mean the behavioural or/and structural description of electronic hardware pieces in VHDL (VLSIC Hardware Description Language, IEEE std., 1988, i.e., the standard high level language created on the ADA concept basis).

Such a tool is called the domain-specific Program Generator (PG).

Our intention is to use the PG for experimental validation of some concepts of the interactive program generation and students teaching at the postgraduate level as well. To build such a tool, the system designer is to have essential knowledge about the domain concerned. We suppose that at the primary stage an initial system model is to be built, i.e., initial requirements and specifications introduced by a system analyst or transferred from the higher level (external) system. Then additional detailed knowledge is to be extracted from the domain through domain analysis (DA). As a result the more exact model is built.

In this context, DA is regarded as a key ingredient part of the system building process. DA is not a trivial problem: various approaches and tools are proposed for it solving. Our approach is orientated towards those aspects of the problem which are related with DA when a domain can be treated as the well-understood one. We will provide a more exact definition of this term later.

The content of this paper is as follows. The relevant papers related to our work are analyzed briefly in Section 2. The main differences between analysis of a domain and the well-understood domain are formulated in Section 3. The initial system model is described in Section 4. The rest material presents the details of our approach – the framework and the process for well-understood DA with the main results gained from DA are presented in Sections 5 and 6, respectively.

2. References related to our work. The references analyzed are categorized into two groups: 1) those that are related to a program generation as an independent problem; 2) those in which a program generation is dealt with in the context of software reuse and domain analysis.

The main concepts presented in works of the first group are as follows. In Teitelbaum and Reps (1981) the program synthesizer was described for generating programs in PL/M from templates. The ATLAS program generator as a menu-based system which produces test programs for the digital units under test (UUT) was presented in Gross and Gerg (1983). The program generator's use in modelling area was discussed and some generalized concepts for Program Generator building are proposed in Luker (1987). The papers (Phelps, 1987; Terry, 1987) deal with the program generating problem from the standpoint of a commercial use. The various aspects of the knowledge-based approach for a high-level language programming automation was analyzed in Hindin (1986).

The domain – specific program generation and design in adapted ATLAS subset for testing of the analogues UUT was described in Štuikys and Toldin (1989), Štuikys and Toldin (1993), Štuikys (1993).

Papers of the second group have been widely discussed in numerous conferences, workshops and special issues are published on this theme. Here we present only those references which were analyzed during this paper preparation.

These papers are subdivided into two groups:

a) papers in which a direct relationship between a program generation and software reuse can be found and estimated through the composition and generative concept: Biggerstaff and Richter (1990), Griss (1993), Jarzabek (1995);

b) papers dealing with the approaches that correspond to the scheme – "from domain analysis to program generation through reuse concept": Yin, *et al.*, (1987), Prieto-Diaz (1990a, 1990b), Iscoe (1990), Lubars (1990), Isco (1991), Maiden and Sutcliffe (1992, 1993 a, 1993 b), Kanapeckas *et al*, (1995).

The domain knowledge analyzed can be found in: Armstrong (1992), IEEE Standard ATLAS (1981), IEEE Standard VHDL (1988), IEEE Standard 1076 (1989), Baker (1993), Ashenden (1995).

The encouragement to write this paper has been received mainly from two sources. The first was the fact that there is a poor number of works in which the well-understood domain is analyzed independently. The second stimuli was the concept mentioned in Prieto-Diaz (1990): "If a domain language exists that can acceptably describe the objects and operations of required system, then the systems analyst has a framework to hang the new specification. This is the *reuse of analysis of information*, and in our opinion it is the *most powerful sort of reuse*".

3. Differences between a domain analysis and the well-understood domain analysis. What is DA can be understood from the definition given in R. Prieto-Diaz (1990): it "can be conceived of as an activity occurring prior to systems analysis and whose output (i.e., a domain model) supports systems analysis in the same way that systems analysis output (i.e., requirements analysis and specification document) supports the systems designer's tasks". It must be noted that there are several others definitions (see, for example, Rolling (1994) :

DA "refers to the process of identifying, collecting, organizing and representing the relevant information in a domain. It involves analysis and study of existing systems, using knowledge captured from domain experts', underlying theory, and perspectives on technology that is relevant to the domain".) According to the first definition the main goal of DA is a domain model building. By a domain model many authors (Prieto-Diaz, 1990; Iscoe, 1990) understand a domain language by which the domain objects and their relationships (properties) are described (an objected-oriented approach to DA). There are domains, however, in which a domain model (or models) already exists. So, domains can be subdivided into two main categories according to the domain model existence:

- a domain (case 1)
- the well-understood domain (case 2).

The term "well-understood domain" means that the domain's high cognition level is achieved and it can be measured. The measuring means (units) of this level could be: the number of models and the level of abstractions used for their representation, the existence of standards, the number of standards, the standard's status (local, international); the number, status and characteristics of the tools developed for domain specific tasks solving, *et al.*

The case 2 means that a domain at least once was analyzed. If the international standards for a certain domain exist (as in our case), it must be emphasized that a very wide and thorough examination of that domain has been done before creating the standard. The complexity of this problem can be conceived of from the examples described in ATLAS, IEEE (std. 1978, 80, 84, 89) or in VHDL, IEEE std. (1987). No matter, however, what a case we have, the case 1 or 2, DA can be performed again and again if the goals for analysis are changed. Especially it is true if the teaching aspects are in mind.

It is not ours intention to provide the differences between DA in the case 1 and case 2 in general. The differences formulated below should be accepted as our experience in this area.

1. For well-understood domain the domain problems' space can be identified, i.e., the domain scope defined. For example, the problems' space for domain analyzed is as follows: structural, behavioural description of domain entities, signal flow description between inputs and outputs, configuration of system architecture, modelling time delay and event sequencing, and data modelling (VHDL tutorial, 1989).

2. It should be supposed that in the case 2 a domain can contain several more narrow domains (an application areas) which, in turn, could be treated as the independent ones. So, for well-understood domain a domain hierarchy can be defined in which domain objects, their properties (characteristics), domain task classes, task specifications, and tasks models are included. For example, the domain analyzed consists of hardware design (synthesis), modelling, testing. The latter, for example, can be treated as an area of the digital, analogue or hybrid circuits testing.
3. As a result of the statement 2, *an hierarchical approach* can be introduced, namely:
   - object classes hierarchy and their properties analysis,
   - task-oriented analysis,
   - specification-oriented analysis,
   - model-oriented analysis at each level of objects hierarchy.
4. The main task is not to create new models for well-understood domain but to *find, understand* and *select* the existed ones which at the most degree correspond to the aims formulated. This procedure in the following Sections (5 and 6) are formulated as a problem of the *basic domain knowledge* elicitation.
5. In case 1 special tools for DA are needed and used (Iscoe *et al*, 1991). In case 2 the special tools are not so highly required, because of the fact that the general model (standard language) and domain-specific tools can be used not only for reaching the goals for which they have been developed but they can serve to support DA tasks as well. This concept is argued in DA model presented in Section 5.
6. The basic domain knowledge extracted from the well-understood domain through DA can be expressed by such a term as the "knowledge template ". A wide range of knowledge templates can be found for well-understood domain. It is important to note that these templates can be defined not only at the source program (subprogram) description level, but at the tasks and task specification level, as well. Furthermore, they have at least two representation forms: the textual and visual (pictorial).
7. If a domain can be treated as well-understood the various aspects of the domain should be analyzed and expert knowledge are highly needed.
8. A team should be created for DA which would operate under close co-

operation between a domain expert, system analyst and system designer. Some ideas so far discussed are summarized in Fig. 1.

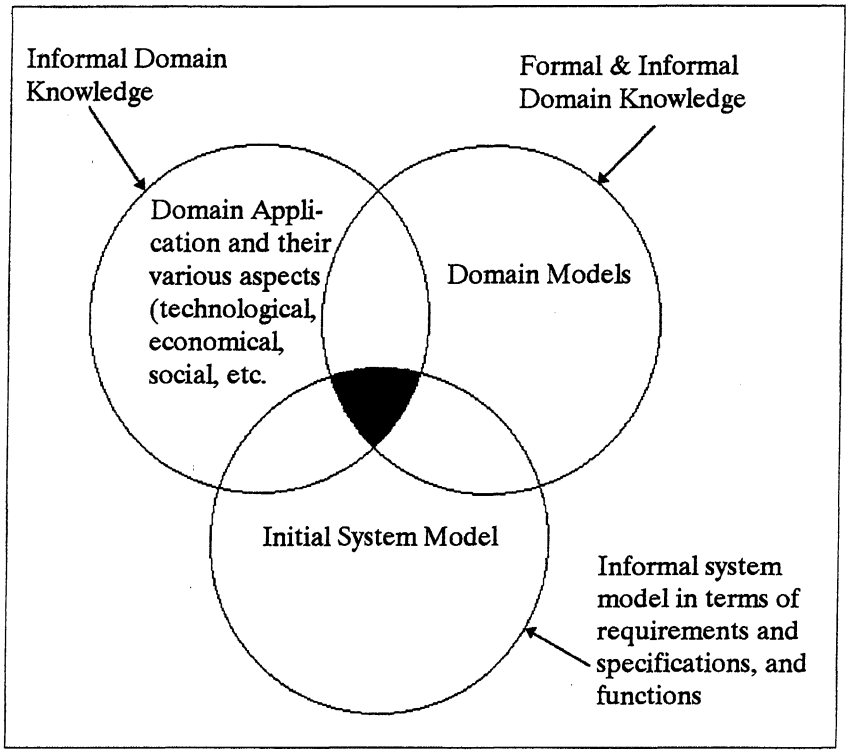

Fig. 1. Interrelationship between domain applications, domain models and domain system which is to be built.

4. System initial model. The system initial model is presented as a set of requirements and specifications formulated at the primary stage of the design. These requirements are as enumerated below.

1. The PG will not be a commercial tool. It is orientated for students teaching at the postgraduate level in VHDL programming and interactive program generation. It ought to have capabilities for examinations of models used for the program generation.

2. The PG is an independent system but it could be integrated and connected with other CAD systems (which support the VHDL) through the external interfaces (files).
3. The tool is to perform the following main functions which are enumerated in the priority decreasing order:
   - program creating (generating);
   - teaching;
   - editing;
   - gluing.
4. The system is to deliver the behavioural and structural descriptions of hardware pieces in VHDL (IEEE std. 87, 93). But not the full language capabilities are intended to be implemented, only those which are frequently used and reveal the essential features of the language.
5. Hardware descriptions generated by the tool ought to be started from the gate level. The higher levels (register, microcell and system) will be also comprised but after an experience gained from the lowest, i.e., the gate level.
6. It is not required that the tool would produce the formal description in VHDL of the entire system. Hardware pieces will be initially produced by the tool using the program creating (generating) function. Then those pieces will be connected by gluing facilities to form a more complex system.
7. The syntactic and semantic correctness of hardware descriptions generated by the tool is to be guaranteed in software engineering sense. But the formal approval of the correctness is not required. It will be based on expert knowledge.
8. Interactions between the tool and its user are to be minimized, i.e., the effective user's interfaces should be built. A special emphasis is to be given to the pictorial objects representation.
9. The program's creating is described as:
   - a composition function by which a formal description is produced from the reuse library components; it must be noted that we intend to implement the more powerful reuse mechanisms than simple generic of VHDL or those that are used in the existing tools of that class;
   - a generating function by which a VHDL description is produced from

the domain problems specifications.

10. The system is to be open for extensions and modifications.

We suppose that through DA the initial model will be detailed, added and changed, i.e., the more exact model synthesized as it generalized in Fig. 2. The model presented can be assumed as a higher level stage of the system development process and that this model should be intuitively incorporated into the entire cycle of the tool design.

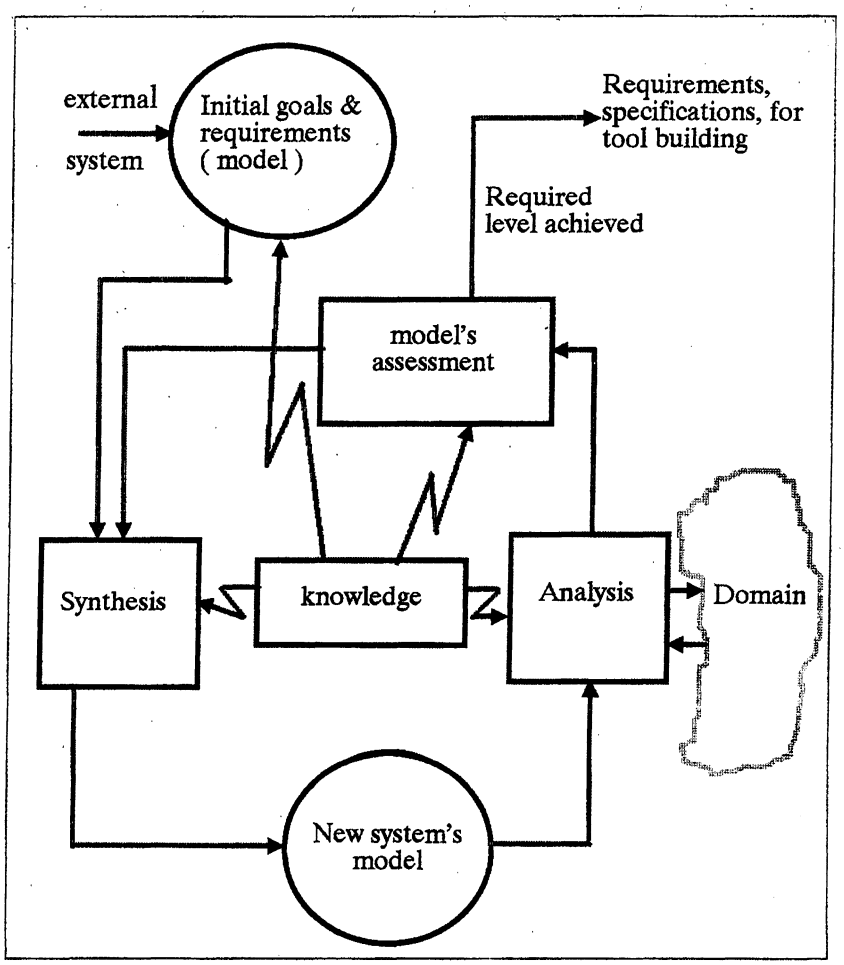

Fig. 2. General framework for DA.

5. The framework and process for well-understood domain analysis. The general model for the well-understood DA is presented in Fig. 3. This model is based on the strict cooperative work of the team members performing the analysis. The team consists of a system analyst, programmer(s) and domain expert. A programmer or programmers are the domain specific tools designers. But the leading role in team activity belongs to the system analyst because of the fact that he (or she) is responsible for the initial system model creation, understanding of goals for domain specific tools needed. It must be noted that for reaching goals of the DA the initial knowledge about domain analyzed is highly eligible not only for a domain expert (it is obligatory), but for system analyst and tools designers, as well.

According to the proposed model, the activity performed can be described as an iterative process. To perform this process, several ways can be used. These are listed as the cases a, b, c and d below:

a) direct DA by a separate team member,

b) direct DA by a separate team member with interviewing with other team members,

c) DA with the domain models and tools use,

d) a mixed way as a combination of cases a, b, and c.

The case a) is indicated by a single break lines in Fig. 3. The process should be conceived of as a knowledge elicitation from the domain via direct study of books, text-books, tutorials, documentation, lecture courses, etc.

In case b) (see double break lines in Fig. 3.) there is an additional possibility of interviewing with other team members. But it is not anticipated neither in case a) nor in case b) to use the domain models and tools for DA.

The case c) is denoted by two-orientated lines in Fig. 3. This way can ensure the more systematized knowledge extraction. The case d) could be treated as the most powerful mechanism in the process.

As a result of the process performed, knowledge from a domain is extracted. This knowledge is called as *basic domain knowledge* (BDK) because it is impossible to comprise all aspects of the domain. By BDK we understand that part of domain knowledge which relates to the initial goals and initial system model formulated for tools building. The concrete content of BDK will be presented in Section 6. It must be noted that BDK is to be assessed. The assessment is presented in our model as a final interviewing between the system's analyst

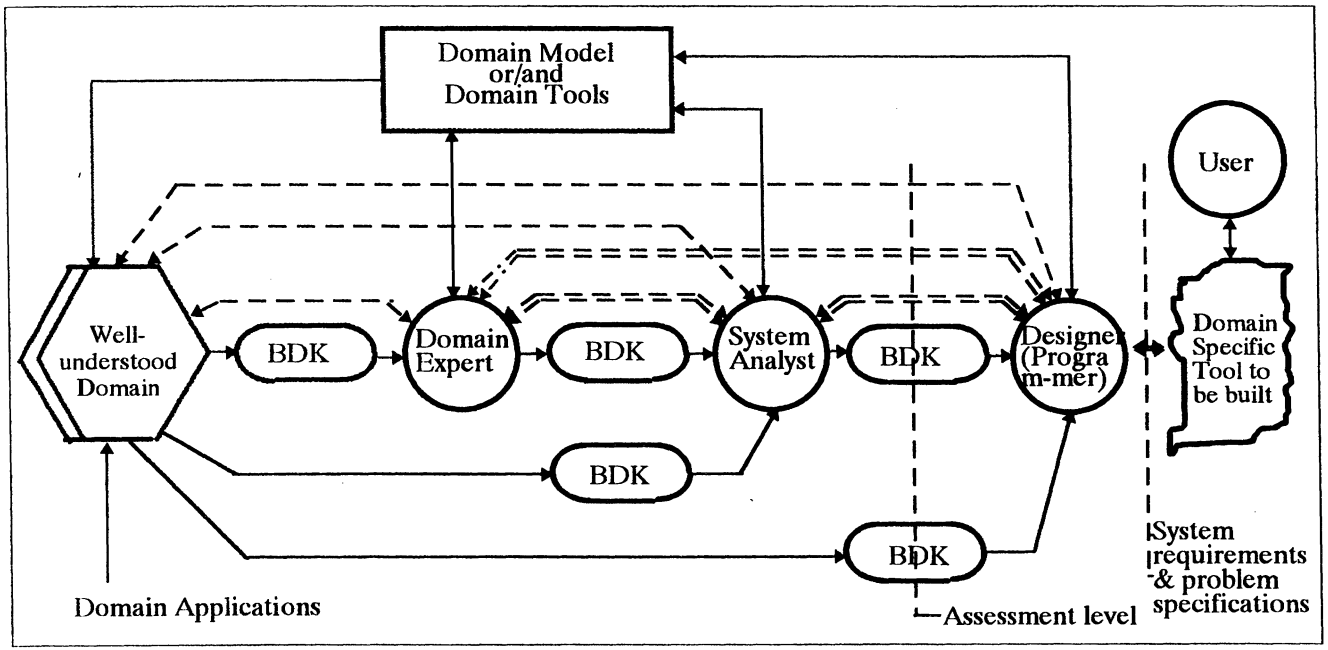

Fig. 3 The framework for the well-understood domain analysis.

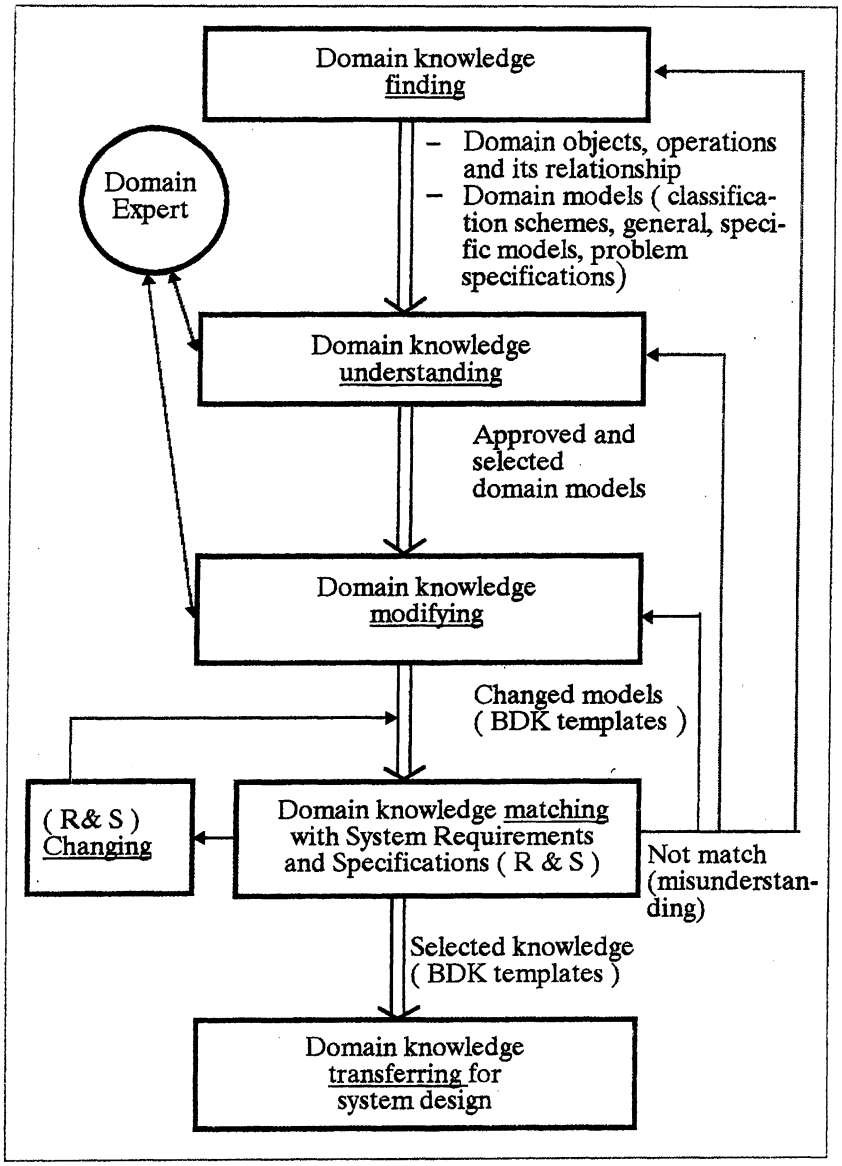

Fig. 4. The process of analysis for well-understood domain.

and system designers.

The processes for DA through the model proposed are detailed in Fig. 4. The general procedure which starts within domain analysis and ends within a tools implementation via the intermediate stages is shown in Fig. 5.

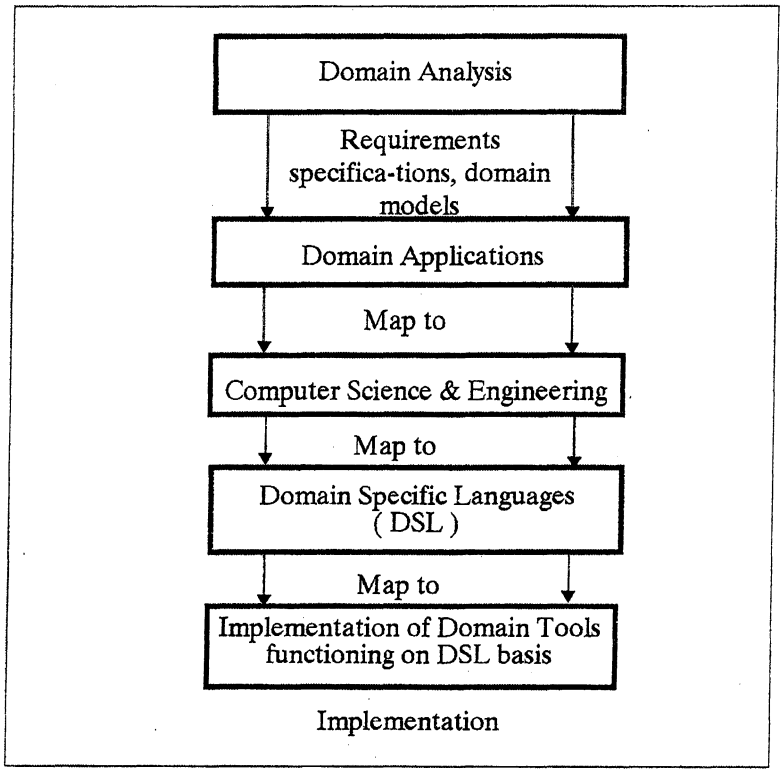

Fig. 5. A general tool buiding procedure: from analysis to implementation.

6. The main results gained from DA. The main results achieved from DA are formulated as BDK (see Fig. 3.). By BDK in this context we mean the following:

- classification schemes of
  - domain objects, i.e., hardware circuits (or circuit pieces),
  - object characteristics (parameters),

  - main problems (tasks) related to the domain,
  - problem specifications,
  - models used;
- models itself;
- domain problems (tasks) itself;
- problem specifications itself;
- object model's examples in VHDL for their reuse in PG.

It must be stated that various classification schemes contain the valuable information for PG interfaces building. The problem specifications and object model examples in VHDL are used for the PG main function implementation (see Section 4). The problem specifications include the following forms: Boolean equations, truth tables, structural diagrams, circuit schematics, state tables for finite automata models (Baker, 1993) timing diagrams and process model graphs (Armstrong, 1992).

The templates for BDK are regarded as the most essential result in the context of our goals achieved. The examples of these templates are as follows:

- generalized textual descriptions in VHDL of hardware pieces such as gates, registers, multiplexers, counters, etc. (see Fig. 6 for details);
- visual frames for hardware entities to form schematics of a circuitry required;
- at the task level the basic domain tasks such a behavioural, structural, signal (data) flow descriptions, signal event sequencing and data type descriptions (VHDL tutorial, 1989) are regarded the domain task templates;
- at the specification level the truth and state tables' representation forms, the frames for Boolean equations, etc., are instances of templates at this level.

7. Discussion. Domain analysis is regarded as a key ingredient part of the software system building process. It is assumed that domain analysis is to be performed with the reusability in mind. To achieve this goal, the domain analysis model is proposed. This model comprises only those aspects of the domain which can be treated as the well-understood one. The cognition level of the well-understood domain can be measured by the number of the models, standards, tools, their status, etc. existing in that domain. The domain analysis model is described as a procedure of the *basic domain knowledge* extraction by a team providing the analysis. The team consists of a system analyst, domain

```
entity Gated_@p1 is
        Generic ( Delay_Time: TIME: = @p5 ns );
        port ( @g21: in BIT; @g22: out BIT );
end Gated_@p1;
Architecture DataFlow_@p1 of Gated_@p1 is
        begin
                @g22 <= @p4 @g23 after
            Delay_Time ;
        end DataFlow_@p1;
```

Fig. 6. The textual template in VHDL for gate level with parameters @xx that are to be defined before the template instantiation.

expert and tool designer (or designers). The leading role in the team belongs to the system analyst. The starting point for analysis is creation of the initial model which can be produced by the system analyst or introduced from the higher level system. By this model the design goals and initial requirements are formulated. The domain analysis result is the detailed initial model which is expressed by the domain objects' classification schemes, domain problem specifications, domain objects models in VHDL and summarized by the term "knowledge template". The knowledge templates could be treated as the most reusable items for software tools building.

By a domain expert and system analyst we mean the University teachers (professors) who are engaged in teaching of the computer hardware related courses. The tool designers are postgraduates who are seeking for M.Sc. or doctoral thesis preparation. By a domain model and tools for domain analysis we mean the standard VHDL language, ALLIANCE (ASIMUT) and CADENCE tools, respectively.

The analysis should be regarded as a successful activity with the results achieved mainly due to the fact that all participants involved have had back-

ground knowledge about the domain concerned.

8. Acknowledgments. I would like to thank the associate professors P. Kanapeckas, E. Kazanavičius, S. Maciulevičius and postgraduates V. Rastenytė, A. Strukov, M. Terleckis, G. Ziberkas for the numerous discussions on this topic and associate professors V. Reklaitis, A. Mikuckas and G. Palubeckis for this paper's draft review.

REFERENCES

Armstrong, D. (1992). *Chip-level Modelling with VHDL.* Mir, Moscow. 174 pp. (in Russian).

Ashenden, P.J. (1995). *The Designer's Guide to VHDL.* Morgan Kaufmann Publisher. 688 pp.

Baker, L. (1993). *HDL Programming with Advanced Topics.* John Wiley and Sons. 365 pp.

Biggerstaff, T., and C. Richter (1990). Reusability framework, assessment and directions. In W. Tracz (Ed.), *Software Reuse: Emerging Technology (Tutorial).* IEEE Computer Society Press. pp. 3–11.

Fisher, G., S. Henninger and D. Redmiles (1991). Cognitive tools for locating and comprehending software objects for reuse. In *13th International Conference on Software Engineering.* Austin, Texas. pp. 318–328.

Griss, M.L. (1993). Software reuse: from library to factory. *IBM Systems Journal,* **32**(4), 548–566.

Gross, O.B., and J.S. Gerg (1983). Automatic ATLAS program generator (AAPG) for the advanced electronic warfare test set. *AUTOTESTCON,* **4**, 286–291.

Hindin, H. J. (1986). Intelligent tools automate high-level language programming. *Computer Design.* pp. 54–56.

IEEE Standard 1076 VHDL (1989). Tutorial CAD language systems inc. 77pp.

IEEE Standard ATLAS Language (1981). IEEE std. 416pp.

IEEE Standard VHDL (1988). Language reference manual IEEE std. 1076pp.

Yin, W., M.M. Tanik, D.Y.Y. Yun., T.L. Lee and A.G. Dale (1987). Software reusability: a survey and a reusability experiment. *IEEE Trans. Software Eng.,* 65–72.

Iscoe, N. (1990), Domain-specific reuse: an object-oriented and knowledge-based approach. In W. Tracz (Ed.), *Software Reuse: Emerging Technology (Tutorial).* IEEE Computer Society Press. pp. 299–309.

Iscoe, N., G.B. Williams and G. Arango (1991). Domain modelling for software engineering. In *13th International Conference on Software Engineering.* Austin, Texas. pp. 340–343.

Jarzabek, S. (1995). From reuse library experiences to application generation architectures. In *Proceedings of Software Reusability (SSR'95).* pp. 114–122.

Kanapeckas, P., S. Maciulevičius and V. Štuikys (1995). Domain analysis for the reusable domain-specific program building. In *Information Technology.* Technologija, Kaunas University of Technology. pp. 337–343.

Lubars, M.D (1990). Code reusability in the large versus code reusability in the small. In W. Tracz (Ed.), *Software reuse: Emerging Technology (Tutorial).* IEEE Computer Society Press. pp. 68–76.

Luker, P.A. (1986). Program generators and generation software. *The Computer Journal,* **29**(4), 315–321.

Maiden, N.A.M., and A.G. Sutcliffe (1992). Specification reuse by analogy. *Next Generation CASE Tools.* IOS Press. pp. 119–131.

Maiden, N.A.M., and A.G. Sutcliffe (1992). People-oriented software reuse: the very thought. In *Workshop on Software Reusability.* IEEE Computer Society Press. pp. 176–185.

Maiden, N.A.M., and A.G. Sutcliffe (1993). Requirements engineering by example: an empirical study. In *Proceedings of IEEE Symposium on Requirements Engineering.* IEEE Computer Society Press. pp. 104–112.

Phelps, R. (1987). New tools automate the application development cycle. Hardcopy, GNTB 1–26, Moscow. pp. 138–141.

Prieto-Diaz, R. (1990). Domain analysis for reusability. In W. Tracz (Ed.), *Software Reuse: Emerging Technology (Tutorial).* IEEE Computer Society Press. pp. 347–353.

Rolling, W.A. (1994). A preliminary annotated bibliography on domain engineering. *ACM SIG-SOFT, Software Engineering Notes,* **19**(3), 82–84.

Štuikys, V. (1993). *Software design for functional testing systems.* Kaunas University of Technology, Technologija. 169pp. (in Russian).

Štuikys, V., and E. Toldin (1993). Computer-aided test program design system (CATPDS) in ATLAS. *Informatica,* **4**(3–4), 384–398.

Teitelbaum, T., and T. Reps (1981). The cornell program synthesiser: a syntax-directed programming environment. *ACM,* **24**(9), 563–573.

Terry, Ch. (1987). CASE tools on an expanded range of computer systems. *EDN,* 221–228.

Tracz, W. (1990). Software reuse myths. In W. Tracz (Ed.), *Software Reuse: Emerging Technology (Tutorial).* IEEE Computer Society Press. pp. 18–21.

Received February, 1996

V. Štuikys received Ph.D. degree from Kaunas Politechnic Institute in 1970. He is currently associate professor at Computer Department, Kaunas University of Technology, Lithuania. His research interests include program generation for domain – specific systems, high level domain – specific languages, expert systems, digital signal processing and CAD systems.

GERAI APIBRĖŽTOS SRITIES ANALIZĖ PROGRAMINĖS ĮRANGOS SISTEMAI SUKURTI

Vytautas ŠTUIKYS

Straipsnyje pateikiamas gerai apibrėžtos srities analizės būdas, kuris traktuojamas kaip kuriamos programinės įrangos sistemos aukštesnysis lygmuo. Formuluojami gerai apibrėžtos srities analizės specifiniai bruožai, pateikiamas kuriamos sistemos pradinis modelis ir probleminės srities analizės procesas.