ARCHIVIO

Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE
DELLA INFORMAZIONE

PISA

SOME IDEAS ABOUT AN INTERACTIVE SYSTEM TO SUPPORT SPECIFICATIONS DEVELOPMENT

P. Asirelli

Nota interna B82-01

Marzo 1982

ARCHIVIO

Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

SOME IDEAS ABOUT AN INTERACTIVE SYSTEM TO SUPPORT SPECIFICATIONS DEVELOPMENT

P. Asirelli

Nota interna B82-01

Marzo 1982

ABSTRACT

The importance of specifications in quality software production is nowadays widely recognised. It is also recognised that conventional programming languages are not suitable for talking about properties of programs, packages or, more in general, objects which we have to deal with when building and maintaining complex systems. Lot of research is going on on the subject. Techniques and formal languages are studied and compared in order to find the best suitable ones to express specification.

The importance of integrated systems to support software development is also widely recognised. Other considerations arise when observing the process that start from the original idea of a project and ends with its implementation, and observing how it nowadays evolves. All this leads us to consider the case that the 'integrated software development system' of the future will be based around specification languages. Three principles are individuated which we believe are fundamental to a system to support interactive development of specification, and which affect both the basic specification language (BSL) of the system and its integrated tools.

1. Introduction

The problem of quality software production is widely recognised. A first important step which has to preceed imp lementations has been individuate as the for malization (specification) of the problem that has to be solved and implemented.

To specify means to deal with properties of objects and their interactions in the world being created. Properties of objects and their interactions can only be expressed in languages that make them easy to understand (the language must have a clear semantic, little syntactic constraints) and so be easy to modify and formal enough to avoid ambiguity. It is recognised that those are not characteristics of "conventional programming languages" [Liskow & Zilles (1975), Backus (1978), Winograd (1979)] and that other more suitable languages must be found.

Formalization techniques (surveys can be found in : Liskov & Zilles(1975), Cleaveland(1980) and languages have been designed such as PROLOG [Colmerauer et al.(1972), Kowalski (1974), Szeredi(1977),Clark & McCabe (1979)] ML [Gordon et al.(1978), Gordon et al.(1979)],NPL [Burstall(1977), Burstall & Goguen (1977), Burstall & Darlington (1977)] or CLEAR [Goguen & Burstall (1980)], OBJ [Goguen & Tardo (1977), Goguen(1978)], HOPE [Burstall et al. (1980)], Z [Abrial et al. (1979), Abrial (1980)], GIST [Balzer (1979)] to establish formalisms in which specifications could be expressed and tested as well. System for the formal proof, based on such languages, have been implemented, such as Stanford LCF [complete references can be found in Gordon et al. (1978)], its developments in PPC [Aiello et al.(1977)] and Edinburgh LCF [Gordon et al.(1978], and TELL [Levi & Sirovich (1975)].
Moreover formal languages also allow specifications to be easily modified and proved again and finally transformed and synthetized [Darlington (1975),(1977); Burstal & Darlington (1977); Clark & Sickel (1977)], in order to derive efficient algorithms, that then can be implemented, in whatever programming languages, via compilation.ystem to transform and synthesize programs have been built, Burstall & Darlington(1977) and se also Guiho & Gresse (1980) in reference to other systems can be found. Interactive systems forthe development of specifications still being object of intense research are the CAT system [Burstall & Goguen (1980)] and [(Balzer (1908a), (1980b)].
Specifications need to be written, proved correct, modified, tested again etc., this process is analogous to the process of software development, whatever programming language is used. All the problems related with methodologies and tools for the development of software applications, apply to the process of specifications development as well.

For conventional programming languages it has been shown that a strong integration of language and tools in an interactive program development environment is essential to support methodologies in the design, development, test and maintenance of large software applications [Cheatham & Townley (1974), Cheatham (1977), Degano et al.(1979), Asirelli et al.(1979), DOD (1980)].
The same strong integration of language and tools in an interactive specifications development environment is essential since we believe that

the future will present the same phases of design, development, test and maintenance of software applications via specifications. In other words we believe that software applications will be developed interactively at the specification level [whyle their correspondent implementations (automatically produced) will be considered to be what nowadays is the "compiled version" of a program written in a conventional programming language] and that for that a suitable environment is necessary.

Three principles are individuated which we think are fundamental for a system to support interactive development of specifications:

1. The ideator of a project should be allowed and helped to develop the specification of his own project.

2. The specifications should be given at different levels of detail ("vertical combination" - CAT-system [Burstall & Goguen (1980)]).

3. The automatic generation of alternative models for incomplete specifications should be provided ("horizontal combination" of CATSystem [Burstall & Goguen (1980)] and see also automatic generation of alternative models in Balzer(1980b)).

We will analise how those principles arise looking at the present situation of the process of devloping whatsoever project and how those principles would affect language and tools of the future system to support interactive development of specifications.

2. <u>The Present Projects Development</u>

Let us start considering in <u>Fip 1</u>, the PL (Programming Language) modelling as it appears in [introduction of CACM (1980)] suggested by Mary Shaw.

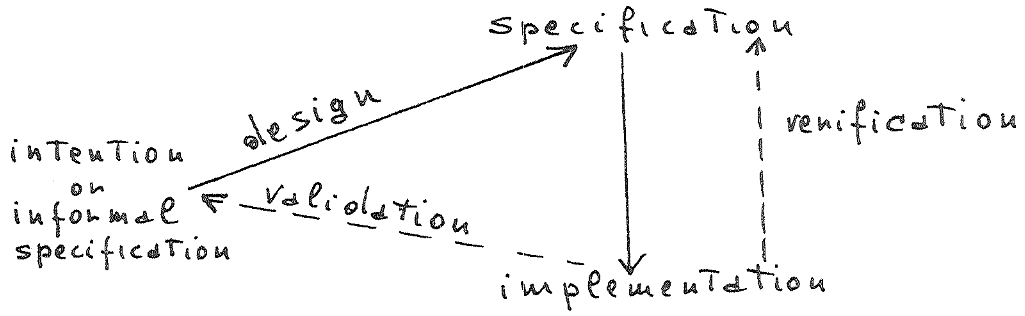


fig. 1

We believe that reality could be represented more completely as in Fig.2

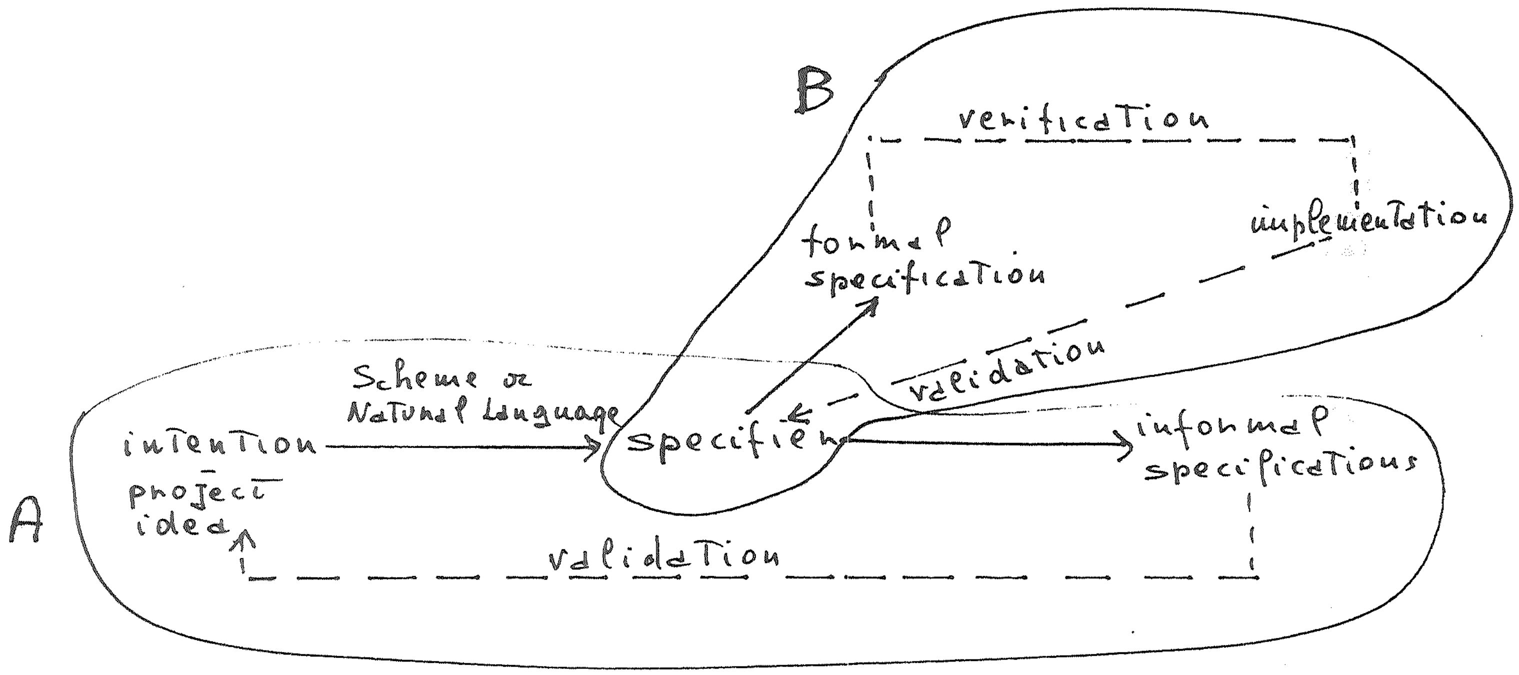


fig. 2

We observe that there are two main phases in the process thaat start from the iea of a project and ends with its implementation.

2.1 First phase (A)

The part A of the process shown in fig.2 is related with the process that starts with the first idea of the project and ends with an informal specification of it.

Generally the person that has got the idea of the specific application to be implemented is different from the one that will carry on all the process of developing the applications, and generally those two people have different knowledges and backgrounds. The former has the sort of knowledge related to the project he wants to be realised, in the sense of past experience, technicality and behaviour, he is, in other words, the specialist of the sort of problems the application will have to deal with. The latter is the specialist of specification languages, programming languages, computer science in general. Between those two people there has to be a flow of information that can easily lead to misunderstanding, so that the project that is developed may be different from the one meant, just from the beginning. There has to be a process of validation of the informal specifications written from the specifier, this validation can lead to a better understanding between the two people and to modification of the informal specification. This process of validation goes on until the two people are sure to have understood each other and the informal specification as signed.

The process of validation previously mentioned would avoid getting a final product different from the one thought of at the beginning. It doesn't avoid another problem,though, the problem of people that change their minds. In other words, the final project could be exactly what was, at the beginning, wanted but that time the committant could have changed his mind, may be because he himself has achieved a better understanding of the problem (perhaps caused by the real behaviour of the final product), so that the project asked this time is, slightly or completely, different from the previous one.

All these considerations lead us to conclude that, the best person entitled to deal with all the process of developing specification is the one that has the idea, that is, because:

(i) He knows quite a lot of the project he has in mind, he has the understanding of the world the project is simulating or working on. (Object involved, actions, laws, behaviour), and he would not need to explain it to anybody else with all the complication that follow.

(ii) He could change his mind while the process goes on. Then it should be possible to avoid to start the process from the beginning, and make some differentiatiions from the original idea by some alternative tentative solutions to the realization of the project.

(iii) He is the only one that can judge whether or not the final product corresponds or satisfies the needs for which the product was developed.

We can, at this point, fix the first principle of a system to develop specifications that is: "the ideator should be allowed to develop the specification of his own project, and he should be given tools which help and guide him all the way throughh the process".

Let us now consider the other points of the specifications development process.

2.2 Second phase (B)

Part B of the process shown in fig.2 deals with the process of starting and carrying on the specifications of a project until the implementation of the prototype. What does the specifier need, during allthe process of developing the specification in so that it will be easy for him:

(a) To modify specifications;
(b) To think of the project or of its components;
(c) Try different, alternative, specifications of the same object, solutions or law?

First of all, specifications are easy to express, to modify and reconsider if they can be expressed at different levels of detail.

Moreover, not all the information needed during the development are available at the beginnig, that is, the specifier does not always know, at the right time, all the information that at that time are necessary to carry on the development. This methodology of stepwise refinement applies to specifications as well as to any other programming acitivity. So, here it comes the second principle.

"Specifications should be given at different levels of detail".

One other aspect of the development of specifications which is similar to the development of programs, is that sometimes the problem may have alternativesolutions that could be built combining together different versions of the same component parts. This possibility is very important and would allow to carry on the development of slightly different models of the same project, giving a better insight of the possible alternative realizations and a better knowledge usefull to judge the final chosen model. So, here it comes the third principle.

"The creation of alternative models for the same project should be allowed".

This is in someway related to:
(a) "horizontal combination" of the CAT system [Burstall & Goguen (1980?)] in which different specification versions of the same component are chosen and combined together to build an alternative model of the same project or part of it; and with:
(b) the automatic generation of alternative models in [Balzer (1980a),(1980b)].

3. The Complete System

Our view of a system to support the development of specification is the following.

There is Kernel Specification System (KSS) which is an interactive system of tools built around a basic specification language, (BSL). KSS alows one to give specification:

(a) Expressed in BSL at different level of detail, driving the user down one to give specification expressed in BSL at different levels of detail, driving him down the levels, asking for new information when needed and generating alternative models for the same project, when possible and when the details are not completely specified.

The user is allowed:

(1) To choose some of the alternative models generated to carry on for further development and reject others.

(2) To go back to upper level, modify specification and start the process again.

(3) To choose one final model that then will be transformed, synthesized and finally compiled in some target, programming language or machine language.

Special configuration of KSS can be built, extending KSS with tools and with extensions of BSL to make specifications suitable to express particular kind of problems.

4.1 <u>Fundamental aspects of BSL</u>

The specification language should be the kernel language in which it should be possible to define extensions of it as application oriented specification languages or specialised speifications languages. Those specialised languages are not different languages, they just correspond to the kernel language <u>plus</u> some "set of abstractions" problem oriented. That is:

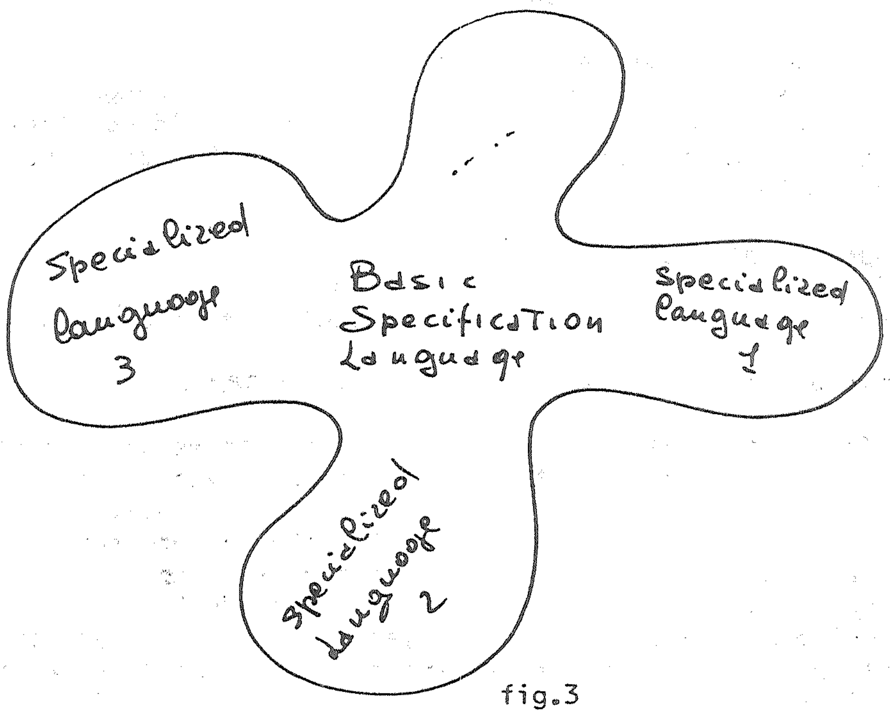


fig.3

So that, for example, extensions of the BSL will be built to form an MSL (Mathematical oriented specification language), or BKSL (Banking oriented spec. language) etc. so that, in MSL it will be possible to deal with functions domain and codomain, sets etc., if those concept were not already given by BSL, while in BKSL it will be possible. To deal with Bank accounts and customs etc. Because those specialized language are only extention of the BSL, all the tools applicable to BSL will also be applicable to specializations of it.

This characteristic of BSL or better this view of the world make it possible to satisfy the 1st. principle previuosly seen, so that it would be possible for the mathematician, the Bank manager or the nuclear engineer to design their own projects. All this seem to be quite fair if we consider that they are the best experts and judges of the final product; They are the peoplethat change their mind about the project, they are the people that could benefit in ideas or could gain a better understanding of their problems in a system which is supposed to work interactively with the user.

It is also clear that to encourage those people we have to find a way of expressing and developing specifications so that it is easy for them to deal with their own view of the problem. In other words specification languages based on set theory might cause some problem to a person that has not a very strong mathematical background, on te other hand the mathematician wouldn't like to have bank accounts in the way.

These specialised languages can be seen just as particular applications made up as special congifurations of modules written in BSL.

From all the previous considerations, the following features can be individuated:

(a). BSL must provide abstractions mechanism.

(b). Modules [Wirth (1979)] are an essential structuring concept for specification languages as well as for PLS (Programming Languages) in general. Modularization of information allows information hiding other and then providing with a mechanisms suitable to satisfy the 2nd and 3rd principle, that is modularization make it possible to:

i Decompose of specifications in parts;
ii Express specifications at different levels of detail;
iii To put together modules to build particular configurations of the same applications;
iv To express different specifications for the same problem.

(c). From (a) and (b) it follows that BSL will provide mechanisms for users types definition together with facilities for a strong type checking. We believe that the use of types should be as simple as possible, avoiding syntactic complications and constraints that make a program quite complicated to read and to understand. In some ways types should be present in the language in the form of objects the user deals with [LCF, GIST] other than obliging the user to have objects that have a type and a value. The view of objects types and relations between these types actions and demons, constraints on states and transitions, as it is in GIST, should be considered, although the feeling is that less details should be specified at each level. In the end it should be possible to define:

<u>specification level 1</u>

a specification of objects, actions and relations existing in the world

<u>specification level 2</u>

specification of objects subtypes (of objects in spec 1) with detailed specifications of actions and relations of spec 1.

<u>specification level n</u>

specification of objects subtypes (of objects in spec. n-1) with detailed specification of actions and relations od spec. n-1.

That is it should be possible, level by level, to have to fully understand the part of the project the level is specifying, following a step-wise refinement methodolofy of development of specifications. So here is another characteristic of BSL:

(d). It should provide syntactic constructs easy to use and meaningful. The structure of a specification at a given level should be easy to understand, with little syntactic constraints, and little concepts specified at each level.

(e). BSL must be an executable language.

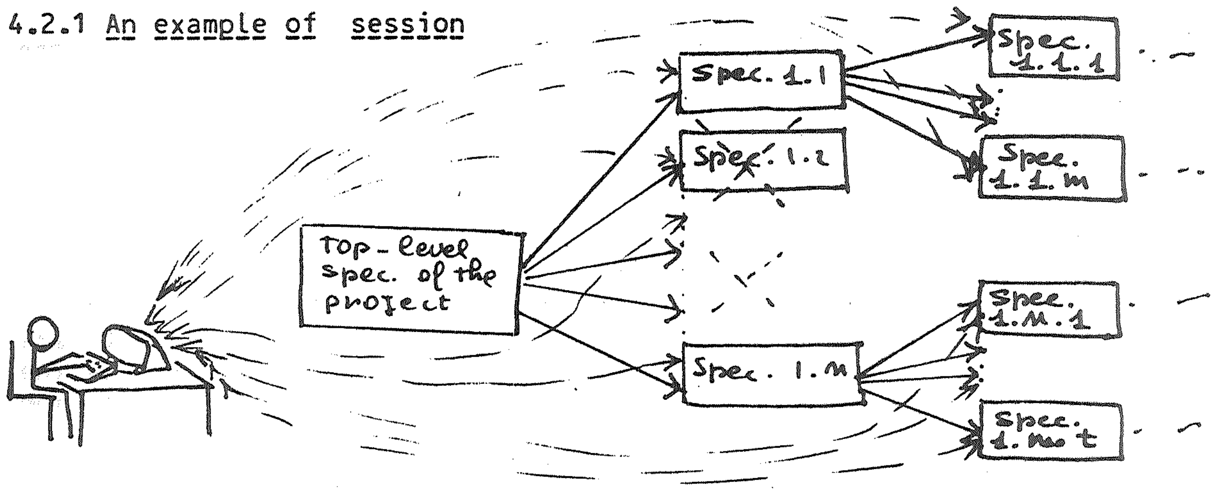
Other characteristics are left to a more detailed study of other existing specification languages, that should be compared having in mind the previously mentioned characteristics.

Those languages should include:
- HOPE
- GIST
- CLEAR
- LCF
- PROLOG

4.2 The System

We will first try to individuate the way in which the specifications of a project should proceed and then we will the functions (tools) the system should support

4.2.1 An example of session



The designer sits and, interacting with the system, gives a first specification of his project. Starting from there and interacting with the user (particularly asking him information about objects or relations not completely specified yet) the system will try to derive a second level of specifications. This second level consists of a set of alternative models that are shown to the user which, at that level and interacting with the system, can reson about the generated models.

The user should be allowed at this point to:

(a). Modify the top-level specifications of the project and start the process again;

(b). Test the behaviour of the generated models. This possibility has a great advantage because it allows the user to get a better idea of what the final project should look like;

(c). Choose some/all of the generated alternative models to be further developed (and in this case should be allowed for each model, to give more detailed information); Repeat the process for every chosen model.

(c). Choose the model/models he believes satisfactory and start a process of:

1. Transforming the algorithm/s abtained to get efficient ones

2. Start a process of implementation (compilation) in the host programming language - and get, that way the final implemented product.

4.2.2 Tools (functions) of the system

We will now try to individuate what sort of tools the syntem should provide to assist the user during all the process.

1. The user, first of all, should be allowed to enter the system and choose the language (a set of BSL extensions) specialised for the kind of problem he is going to deal with.

Two \approaches are possible:

(a) The specialized language is given in a particular configuration of the system for developing specification, tailored for the particular user. This means that, as for BSL and its extensions, we can see a first configuration of the system in which the extensions of BSL are implemented. A copy, or a particular combination of the tools of the systems and new tools are put together with a particular BSL extension to form a "specialised" specification system as a particular application built for a user depending upon the sort of problem he has to deal with. The first configuration of the system can be seen as the Kernel system from which the derived systems are build and is, of course, used from the specialist of the specification system.

(b) The specialised language is given as an alternative among others in the same specification system environment.

In both cases (the Kernel system in the first case, "the" system in the second) there must be tools to allow the creation of a particular configuration of the system environment in the sense of putting together a specialised language and tools (they can be copies of the original tools or particular linking configuration of the original tools).
In the first case the user, when entering the system finds himself in an already suitable (for his needs) environment in which he can begin to

work assisted in accordance to the methodology previously described.

In the second case the user finds himself in a more wide and powerful environment in which he can do lots more than he might need, so there must be a first environment from which to choose a suitable one. In this top level environment it must be possible to deal about "what is the most suitable language and tools configuration for a problem"; It involves the ability of talking about the various extensions of BSL, tools suitable for those extensions, goals of the particular configuration etc. Note that in this case, the Kernel system is only a particular suitable environments suitable to build extended environments.

Let's suppose now that the user is in his suitable environment, what he needs is:

2. A tool to guide him to give the specifications of his project. It should be a tool that helps the user to understand more his problem in general and to think of it in a more structured way; for example asking him to individuate in the world he is going to deal with, the objects involved, the sort of movement that is present in that world (actions), what objects are active and what are passive in what actions (relations), whether there are laws and what those laws are (constraints).Of course this tool depends on the language (BSL or its extensions) chosen, it reflects the concept and philosophy of the language and we believe that the previous should be considered as a good philosophy a language to give specification should follow.

3. Tools to test the behaviour (may be a symbolic interpreter [Balzer (1980c).

Specification should be given at different levels of detail. At every level alternative models should exist, so, the system should provide features to investigate a model and derive new alternative models from it.

3. An interpreter of the language should be provided which exercise the behaviour of a model running the given models and looking for inconsistency, errors of all sorts and which derive new level of specifications when possible.

The system should keep track of all the models created, that it has keep the generation tree of the models so that the following should be possible:

4. To decide the elimination of some models and of their derivations;

5. To eliminate all the generated models and start the process again from the beginning;

6. To choose a particular model in the generation tree to operate on it.

After the process has gone on for a while the user may have got more detailed ideas or information about his project. This new information can be such that a model should be run with the new information added to it as a particular example of behaviour of the prototype - in this case the interpreter of the language should be given this information before running a model - no problem arise if no successor of that model exists. New information can be more fundamental and effect all the generation of models already done or part of it, it should then be possible:

7. To retrieve the level affected from the new information and start all the generation again, scratching that old part of the generation tree. It would be auspicable a tool to find all the model affected by those modifications even up to the root of the tree whenever the new information create inconsistency with information already existing in the model. This way, the process of finding models that have to be modified as a consequence of new knowledge, would not be so desperate while it would preserve congruence of all the alternative modules generatd from the same basic knowledge.

8. A transformation tool should be provided which, once the user has individuated and chosen a model/s to be implemented, generates an efficient version of the specification of the model.

9. There should be a compiler from BSL to the target language or machines, so to get a final implementation of the model.

Conclusion

Some ideas of an interactive system to develop specifications have been discussed. Those ideas have been originated from the importance of specifications in the production of reliable software and from other considerations such as:
- Problems of validation of projects and understanding between the specifier and the person that ask for a project to be realized;
- Problems related with validation of software with the specifications;
- Problems related to the development and modification of specifications and their impact on all the other parts of a specification not directly (but indirectly) affected by modifications.

A Kernel specification system (KSS) is described that consist of a system of tools built around a Basic Specification Language (BSL), which together should allow to develop specification following a methodology based on 3 good principles:

1. The ideator of a project should be allowed and helped to develop the specification of his own project.

2. The specifications should be given at different levels of detail ("vertical combination" - CAT-system [Burstall & Goguen (1980)]).

3. The automatic generation of alternative models for incomplete specifications should be provided ("horizontal combination" of CATSystem [Burstall & Goguen (1980)] and see also automatic generation of alternative models in Balzer(1980b)).

We have analised how those principles affect language and tools of an integrated and interactive system to support specifications development which we believe is the future of the actual integrated system for software development.

ACKNOWLEDGEMENTS

Much of the ideas collected in this paper were generated during a period of study spent at Imperial College of Science and Technology of London.

I would like to thank Dr. J. Darlington for encouraging me to work on those ideas. Thanks also to all people and Institutions that made it possible for me to go at I.C.S.T. especially Prof. M.M. Lehman, Head of the Department of Computing of I.C.S.T. and Prof. F. Denoth, Director of I.E.I.

I am particularly grateful to Dr. R.A. Kowalski and Prof. G. levi whose support has been essential to make my stay at I.C.S.T. profitable.

BIBLIOGRAPHY

Abrial, J.R. (1980) - The specification Language Z : Syntax and "Semantics" - Int. Rep., Oxford University, Computing Laboratory.

Abrial, J.R., Schuman, S.A. and Meyer, B. (1979) - Specification Language - Proc. Summer School on the Construction of Programs, Belfast, Cambridge University Press, 1980.

Aiello, L. et al. (1977) "PPc (Pisa Proof Checker): a tool for experiments in theory of proving and mathematical theory of computation", Fundamenta Informaticae 1.

Asirelli et al. (1979) - "A Flexible Environment For Program Development Based on a Symbolic Interpreter" - Proc. of 4th. Int. Conference on Soft. Eng.,Munich, September 17-19, pp. 251-263.

Backus, J. (1978) - 'Can programming be liberated from the von Newman style? A functional style and its algebra of programs'. Turing Lecture, CACM22,7, pp. 391-401.

Balzer, R.M. (1979?) - GIST - Int. Rep. I.S.I. ,U.S.C.,Marina del Rey, California

(1980a) - "An implementation Methodology for Semantic Data Base models" - Proc. Int. Conf. in Entity-Relationship approach to System Analysis and Design, North-Holland, Amsterdam.

(1980b) - 'Dynamic System Specification' - Proc. on the workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park, Colorado, June 23-26, ACM special issue-SIGART/SIGMOD/SIGPLAN Notices, 16,1, January 1981, pp. 95-97.

(1980c) - 'Validating operational sofware specification'- Int. Rep. I.S.I., U.S.C., Marina del Rey, California.

Burstall, R.M. (1977) - Design consideration for a functional programming language - Proc. of Infotech State of the Art Conference, Copenhagen, pp. 54-57.

Burstall, R.M. and Darlington, J. (1977) - A transformation system for developing recursive programs - JACM, pp. 44-67.

Burstall, R.M. and Goguen, J.A. (1977) - Putting theories together to make specifications - Invited paper in Proc. of Fifth Int. Joint Conf. on A.I., Cambridge, Mass., pp. 1045-1058.

(1980?) - CAT, a System for the Structured Elaboration of Correct Programs from Structured Specifications - Int. Rep Edinburgh University.

Burstall, R.M., Mac Queen,D.B. and Sannella, D.T. (1980) - HOPE: An Experimental Applicative Language - Internal Report CSR-62-80- Dpt. of Computer Science, Edinburgh University.

CACM (1980) - Proc. of the workshop on Data Abstraction, Databases and Conceptual Modelling - Pingree Park, Colorado, june 23-26,SIGART/SIGMOD/SIGPLAN Notices 16,1, Jan.1981.

Cheatham, T.E. (1977) - New Direction in Software Development Tools - Proc. Congresso AICA '77, Pisa, Italy, pp.3-29.

Cheatham, T.E. and Townley, J.A. (1974) - A Proposed System for Structured Programming - Programming symposium (G. Goos and J. Hartmanis, Eds). Lecture Notes in Com. Science 19, Springer-Verlag, pp. 33-34.

Clark, K.L. and McCabe, F. (1979) - IC-PROLOG reference Manual, CCD Research Report, Imperial College, London.

Clark, K.L. and Sickel, S. (1977) - Predicate logic : a calculus for the derivation of programs - Proc. 5th Joint Conf. on A.I.

Cleaveland, J.C. (1980) - Mathematical Specifications - SIGPLAN Notices 15, 12, December.

Colmerauer, A. et Al. (1972) - Un systeme de communication homme-machine en francais - Rapport preliminaire, Groupe de Researcheen Int. Art., Universite' d'Aix, Marseille, Luminy.

Darlington, J. (1975) - Application of program transformation to program synthesis - Proc. of Int. Symp. on Proving and Imp. programs, Arc et Senans, France.

(1977) Program transformation and synthesis present capabilities - Rep. 77/43. Dpt. of Computing, Imperial College (to appear in Art. Int. Journal 1981).

Degano et al. (1979) - An integrated System to support Program Design Development and Analysis - Rivista di Informatica, ix, 4.

DOD (1980) U.S. Dpt. of Def. - Requirements for Ada programming support environment - "STONEMAN" - February.

Goguen, J.A. (1978) - Some Design principles and Theory for OBJ-O, a Language for Expressing and Executing Algebraic Specifications of Programs - Proc. int. Conf. on Math. studies of Inf. Processing (Kyoto, Japan), pp. 429-475.

Goguen, J.A. and Burstall, R.M. (1980) - Semantic of CLEAR, a specification language - working draft - Edinburgh University - to appear in Proceedings, 1979 Copenhagen Winter School on Abstract Software Specification.

Goguen, J.A. and Tardo, J.J. (1977) - An introduction to OBJ: A Language for Writing and Testing Formal Algebraic Program Specifications - Specification of Reliable Software Conf. Proc., Cmabridge, Mas., April.

Gordon, M.J. et al. (1978) - A Metalanguage for Interactive Proof in LCF - *Proc. of 5th ACM SIGACT-SIGPLAN Conf. on Principles of Programming Languages*, Tucson, Arizona.

Gordon, M.J., Milner, A.J. and Wadsworth, C.P. (1979) - Edinburgh LCF - *Lecture Notes in Comp. Science*, V.78.

Guiho,G. and Gresse, C.(1980) - Program ,Synthesis from Incomplete Specifications - *Proc. 5th Conf. on Automated Deduction*, les Arcs, France.

Kowalski. R.A. (1974) - Predicate Logic as Programming Language - *Proc. Information Processing 74*, North Holland Pub. Co., Amsterndam, pp. 556-574.

Levi, G. and Srovich, F. (1975) - Proving Program Properties Symbolic Evaluation and Logical Procedural Semantics - in Math. Foundation of Computer Science, *Lecture Notes in Comp. Science*, 32, Springer-Verlag, pp. 294-301.

Liskov. B.H. and Zilles, S. (1975) - Specification Techniques for Data Abstractions - IEEE Trans, on Soft. Eng., v. 1,SE - 1, n. 1, pp. 7-19.

Szeredi, P. (1977) " PROLOG - a very high level language based on predicate logic" , *2nd Hungarian Computer Science Conf.*, Budapest.

Winograd, T. (1979) - Beyond Programming Languages - CACM 22, 7,pp. 391-401.

Wirth, N. (1979) - The Module: A System Structuring Facility in High Level Programming Languages - *in* Language design and Programming Methodology, *Lecture Notes in Comp. Science 79*, Springer-Verlag.