Esprit Project 22703
SYnchronous Reactive Formalisms
SYRF

FINAL REPORT

Contents

Foreword

This final report aims at summing up the results achieved during the three years of the project, and to draw some conclusions and perspectives. For each work package and task, we try to emphasize the main success, but also to describe the encountered problems or insufficiencies, and finally, to consider the foreseeable uses of the results, both from scientific and industrial points of view.

Work Package 0

Management and Dissemination

In this chapter we only list the project meetings, and summarize the dissemination actions and the publications.

Meetings and Reviews

Dates	Place	Type	Organizer
Jan. 30–31, 1997	Antibes (France)	"kick-off" meeting	Inria Meije
Jun. 5–7, 1997	Stokholm (Sweden)	meeting	Prover Tech.
Jan. 29–31, 1998	Grenoble (France)	meeting and 1st review	Inria Spectre
Jun. 8–9, 1998	Skt Augustin (Germany)	meeting	GMD
Feb. 14–16, 1999	Rennes (France)	meeting and 2nd review	Inria EpAtr
Jun. 20–22, 1999	Visby (Sweden)	meeting	Linköping U.
Jan. 18, 2000	Grenoble (France)	final review	Inria Spectre

Workshops

The project sponsored the 4th, 5th and 6th Workshops on Synchronous Programming, which were held, respectively, in Roscoff (France, Dec. 15–19, 1997) , in Gandia (Spain, Dec. 7–11, 1998), and in Hyères (France, Nov. 29 – Dec. 3, 1999). Many of the talks in these workshops were given by members of the Project and concerned results of the Project.

FTH'98, the Workshop on Formal Techniques for Hardware and Hardware-like Systems (June 19, 1998, Marstrand, Sweden, Jun. 19, 1998) was co-chaired by a one of the Project partner, and had a high participation of SYRF partners.

Tutorials and Publications

Tutorials

1. Nicolas Halbwachs. "Synchronous programming of reactive systems, a tutorial and commented bibliography." *Tenth International Conference on Comp uter-Aided Verification, CAV'98*, Vancouver (B.C.), July 98. LNCS 1427, pp. 1–16.

2. Mary Sheeran and Gunnar Stålmarck. "A tutorial on Stålmarck's proof procedure for propositional logic." *International Conference on Formal Methods in Computer-Aided Design (FMCAD'98).* Palo Alto (Ca), November 98.

3. Mary Sheeran and Nicolas Halbwachs. "System design and verification using Lustre." *Forum on Design Languages*, FDL'99, Lyon, France, Aug. 30 - Sept. 3, 1999.

4. François Pilarski, Albert Benveniste and Nicolas Halbwachs. "Civil avionics systems – Problem and formal approaches" *World Congress on Formal Methods, FM'99*, Toulouse, France, September 20-24, 1999.

Articles

1. N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.

2. S. Nadjm-Tehrani and J-E. Strömberg. Verification of Dynamic Properties in an Aerospace application . *Formal Methods in System Design*, 14(2):135–169, March 1999.

3. A. Poigné, O. Maffeis, M. Morley, R. Budde and L. Holenderski. The Synchronous Approach to Designing Reactive Systems. *Formal Methods in System design*, 12, 163-187, 1998.

4. A. Benveniste, B. Caillaud and P. Le Guernic. Compositionality in dataflow synchronous languages: specification and distributed code generation. *Information and Computation*, to appear.

Conference and Workshop Papers

1. O. Åkerlund and S. Nadjm-Tehrani and G. Stålmarck. "Integration of Formal Methods into System Safety and Reliability Analysis." 17th International Systems Safety Conference, ISSC'99, Orlando, August 99.

2. S. Bensalem, P. Caspi, C. Dumas, and C. Parent-Vigouroux. A methodology for proving control programs with Lustre and PVS. *Dependable Computing for Critical Applications, DCCA-7.* San Jose, January 99. IEEE Computer Society

3. A. Benveniste, P. Le Guernic, and P. Aubry. Compositionality in dataflow synchronous languages: specification and code generation. Malente Workshop on Compositionality, W.P. de Roever, A. Pnueli Eds., September 1997, LNCS 1536.

4. A. Benveniste , S. Gaubert, and C. Jard. Monotone rational series and max-plus algebraic models of real-time systems, *Wodes'98*, August 1998.

5. A. Benveniste, C. Jard, S. Gaubert. Algebraic techniques for timed systems. *Concur'98*, Nice, France, Sept. 98, LNCS 1466.

6. A. Benveniste, B. Caillaud and P. Le Guernic. "From synchrony to asynchrony." In J.C.M. Baeten and S. Mauw, editors, CONCUR'99, Concurrency Theory, 10th International Conference, LNCS 1664. 1999.

7. L. Besnard, P. Bournai, T. Gautier, N. Halbwachs, S. Nadjm-Tehrani, A. Ressouche. "Design of a multi-formalism application and distribution in a data-flow context: an example." 12th International Symposium on Languages for Intensional Programming, ISLIP' 99, NCSR Demokritos, Athens, Greece, Juin 1999.

8. P. Caspi and M. Pouzet. A Co-iterative Characterization of Synchronous Stream Functions. Proceedings of the Workshop on Coalgebraic Methods in Computer Science, Lisbon. *Electronic Notes in Theoretical Computer Science*, volume 11, March 1998.

9. T. Gautier and P. Le Guernic Code generation in the SACRES project. *Safety-critical Systems Symposium (SSS'99)*, Huntingdon, 1999, to appear in LNCS.

10. N. Halbwachs and P. Raymond. "Validation of Synchronous Reactive Systems: from Formal Verification to Automatic Testing." *Asian Computing Science Conference, ASIAN'99.* Phuket (Thailand), December 1999.

11. B. Jeannet, N. Halbwachs, and P. Raymond. "Dynamic Partitioning in Analyses of Numerical Properties." *Static Analysis Symposium, SAS'99.* Venice (Italy), September 1999. LNCS 1694.

12. O. Kouchnarenko and S. Pinchinat. Intensional Approachs for Symbolic Methods. *MCFS Workshop on Algorithms and Tools. Electronic Notes in Theoretical Computer Science*, vol.18, 1998.

13. A. Kountouris, C. Wolinski. "Combining Speculative Execution and Conditional Resource Sharing to Efficiently Schedule Conditional Behaviors." ASP-DAC'99, Hong Kong, January 1999.

14. A. Kountouris, C. Wolinski. "High-level Pre-synthesis Optimization Steps using Hierarchical Conditional Dependency Graphs." EUROMICRO'99, IEEE Computer Society Press, Milan0, Italiy August 1999.

15. J.C. Le Lann, C. Wolinski. "Load Balancing and Functional Unit Assignment in High-Level Synthesis." SCI'99/ISAS'99, Orlando, Florida, August 1999.

16. M. Lin, J. Malec, and S. Nadjm-Tehrani. On Semantics of Reactive Rule-based Systems. *Advances in Computing Science, proceedings of ASIAN'97.* Katmandhu, Nepal, December 1997. LNCS 1345.

17. F. Maraninchi and Y. Rémond. Mode-Automata: About Modes and States for Reactive Systems. *European Symposium on Programming, ESOP'98.* Lisbon, Portugal, April 1998.

18. H. Marchand, M. Le Borgne. Partial Order Control of Discrete Event Systems modeled as Polynomial Dynamical Systems. *1998 IEEE International Conference On Control Applications*, Trieste, Italy, September 1998.

19. H. Marchand, P. Bournai, M. Le Borgne, P. Le Guernic. A Design Environment for Discrete-Event Controllers based on the SIGNAL Language. *1998 IEEE International Conf. On Systems, Man, and Cybernetics*, San Diego, California, October 1998.

20. H. Marchand, M. Samaan. "On the Incremental Design of a Power Transformer Station Controller using Controller Synthesis Methodology." World Congress on Formal Methods (FM'99), LNCS 1709, Toulouse, France, October 1999.

21. S. Nadjm-Tehrani. Time-Deterministic Hybrid Transition Systems. In *Hybrid Systems V, Proceedings of the Fifth international workshop on hybrid systems*, LNCS 1567. 1998.

22. S. Nadjm-Tehrani. Integration of Analog and Discrete Synchronous Design, Extended abstract. *Hybrid Systems: Computation and Control.* Nijmegen, March 1999.

23. S. Nadjm-Tehrani and O. Åkerlund. "Combining Theorem Proving and Continuous Models in Synchronous Design." FM'99, World Congress on Formal Methods, Volume II, LNCS 1709. September 99.

24. S. Nadjm-Tehrani. "Integration of Analog and Discrete Synchronous Design." Second international workshop on Hybrid Systems: Computation and Control, LNCS 1569. March 99.

25. D. Nowak, J.P. Talpin, P. Le Guernic. "Synchronous Structures." Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99), LNCS 1664, August 99.

26. A. Poigné and L. Holenderski. On the Combination of Synchronous Languages. *Workshop on Compositionality*, Malente 1997, LNCS 1536.

27. P. Raymond, D. Weber, X. Nicollin, and N. Halbwachs. Automatic Testing of Reactive Systems. *19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

28. M. Sheeran and A. Boraelv. How to prove properties of recursively defined circuits using Stålmarck's method. *Int. Workshop on Formal Techniques for Hardware and Hardware-like Systems* (associated with Mathematics of Program Construction Conference), Marstrand, June 1998.

Reports

1. M. Allemand, F. Bodin, A. Kountouris, P. LeGuernic, J-C. Le Lann, A. Seznac, and J-C. Wolinski. A synchronous approach for hardware design. IRISA Research Report nr. 1131, October 1997.

2. M. Bourdellès. The steam boiler controller problem in Esterel and its verification by means of symbolic analysis. Research report Nr. 3285, Inria, October 1997.

3. P. Caspi and M. Pouzet. A co-iterative characterization of synchronous stream functions. Research report nr. 97-07, Vérimag, October 1997.

4. L. Holenderski. Translating *LEA* to synchronous components. Arbeitspapiere der GMD, Nr. 1090, 1997.

5. M. Kerboeuf, D. Nowak, J.-P. Talpin. "The steam-boiler problem in SIGNAL-COQ." Research report Irisa, No 3773, October 1999.

6. F. Maraninchi and Y. Rémond, "Mode-Automata: About Modes and States in Reactive Systems", Research Report, Verimag, September 1997.

7. H. Marchand, M. Le Borgne. "The Supervisory Control Problem of Discrete Event Systems using polynomial Methods." Research report Irisa, No1271, October 1999.

8. D. Nowak. "Spécification et preuve de systèmes réactifs." Thèse de l'Université de Rennes 1, IFSIC, October 1999.

9. S. Pinchinat, H. Marchand, M. Le Borgne. "Symbolic Abstractions of Automata and their application to the Supervisory Control Problem." Research report Irisa, No 1279, November 1999.

10. J.-P. Talpin, A. Benveniste, P. LeGuernic. "Asynchronous deployment of synchronous transition systems." Research report Irisa, No 1269, October 1999.

Work Package 1

Case Studies

Assessment of the experiments based on the Schneider case study

Within the SYRF project, three experiments have been performed on the Schneider case study (i.e., the SRIS):

- Verification of SRIS properties with XLESAR,
- Automatic testing of the SRIS software with Lurette,
- Reingeniering of SRIS with Mode-automata.

These experiments are detailed in the corresponding deliverables.

In this section, Schneider assesses these experiments from an industrial point of view: are the methods mature enough to be transferred in an operational context? Can they be applied cost-efficiently? What is the usability and the suitability of the tools? How can they be integrated in a coherent development process?

In the first paragraph, we briefly present Schneider Electric practices and recall our expectations from the SYRF project. Then, we focus on the three experiments conducted on the SRIS case study we have provided to the SYRF partners. As a conclusion, we present the next steps in order to exploit some results coming from SYRF.

Schneider Electric practices and expectations from SYRF

Schneider Electric develops safety critical instrumentation and control systems for the nuclear field. The software part of these systems are developed according to the following requirements:

- a strict adherence to a life cycle (i.e., the V life cycle),
- the definition of a validation procedure at each stage,
- a verification and validation team independent from the development team.

These guidelines meet the IEC 880 standard “Software for computer in the safety systems of nuclear power stations”.

As early as 1985, Schneider Electric adopted the Lustre data flow language and developed the SAGA tool, the first software development environment entirely based on Lustre. Lustre is based on a clean mathematical semantic and is well adapted to critical fields.

Moreover, it is close to the traditional know-how of Schneider Electric and has been easily adopted by the development engineers. Since then, the SAGA tool (now commercialised by Verilog by the name of SCADE) has been used in the large for the development of our systems. Concurrently to this industrial use, Schneider Electric has been continuously involved in several research project, in particular in collaboration with the Verimag laboratory (e.g., Eureka SYNCHRON, Esprit SYRF, Esprit CRISYS).

Among the topics SYRF deals with (e.g., combination of formalisms, program verification, code distribution, hardware/software co-design, ...), Schneider Electric main interests concern:

- verification and validation techniques for Lustre programs. As our systems are highly critical ones, it is important to ensure that they behave as expected. Our current verification and validation process is mainly based on inspections of documents and testing (from unit testing to system testing). Formal verification with XLESAR and automatic generation of test cases with Lurette may be a way to improve this process.
- combination of formalisms. The design of our systems is mainly data-flow oriented and Lustre is well adapted to describe this type of behaviour. But it may happen that the design exhibits several running modes and in this case, Lustre is not a natural way to express this structure. The Mode Automata formalism is an interesting extension of the Lustre language.

Verification of SRIS properties with XLESAR

Schneider Electric has been interesting in formal verification for several years. Schneider Electric has already conducted several experiments with LESAR in collaboration with different companies and laboratories (Verimag, Renault, Elf Aquitaine,...). The conclusions resulting from these previous experiments were:

- the LESAR tool is powerful enough to deal with an industrial case study;
- but it is not mature enough to be integrated in an industrial context. It is difficult to use for a non-specialist.
- Moreover, the expression of the properties and the hypothesis is not an easy task.

One of the SYRF results is the development of the XLESAR user interface. This interface is a significant improvement of the tool: from a user point of view, this graphical interface eases the use of LESAR and of the associated methodology. This improvement corresponds to an industrial need: it allows a gain of time during the formal verification process and it is easier to convince and train engineers when a graphical tool is available.

The experience on the Schneider case study shows ambiguities and lacks in the requirements but no “real bug”: when the verification fails, this may mean that an assertion is missing due to unprecise requirements. Thus, although the formal verification did not exhibit real bugs, the benefit of such a method is to show problems in the requirements: lacks, inconsistency, ...

As a conclusion, Schneider is interested in the formal approach since it provides a high degree of confidence. But its integration in our industrial development process is still an open issue: how can we integrate it cost-efficiently? The development of the XLESAR interface is a significant step towards this transfer and Schneider intends to perform an internal assessment of this tool.

Automatic testing of the SRIS software with Lurette

The experiment with Lurette on the SRIS case study leads to convincing results. An internal assessment of the tool is currently conducted by Schneider Electric. The first steps of this internal project are the following:

- Methodological aspects. The automatic generation of test cases can be applied at different stages of our software development process.
  - First, it can be a way to improve the "dialogue" with our customer which provides us with the requirements of the system. The definition of the requirements (by the customer) and their implementation through a Lustre specification (by Schneider) is an iterative process and a close collaboration is needed during this phase. Till now, no simulation was used and this process was boring. The Lurette tool could be a way to simulate the specification developed by Schneider: it would be easier for the customer to understand the Lustre/Scade program by simulating its behaviour in some particular contexts.
  - Secondly, the automatic generation of test cases can help the independent V&V team. At each stage of the development life cycle (specification, preliminary design, detailed design, ...), this team is in charge of ensuring the consistency with the previous stage by means of inspections of documents and testing. Again, Lurette can be a way of simulating the Lustre/Scade program so that it is easier for the independent team to understand its behaviour.
  - Finally, automatic testing with Lurette is a way of validating the behaviour of the system. The idea would be to generate test cases at the specification level in collaboration with the customer and to run these test cases on the final system.
- Internal experiments. Concurrently to methodological aspects, several internal experiments have been performed on small case studies. They show the feasibility of the method and the usability of the tool.

The next steps of this internal project are twofold:

- a library of observers dedicated to Schneider needs is needed to express the assertion module.
- an experiment in an operational context. From a methodological point of view, it seems that the Lurette tool would be the most suitable to help the independent V&V team.

Schneider is convinced that the Lurette tool is of great interest from an industrial point of view. The tool is easy to use and the method can be integrated at different phases of the software life cycle. That's why Schneider aims at continuing the effort on this topic: a collaboration with the Verimag laboratory is in preparation within the "technical centre of software" project, funded by Région Rhône-Alpes.

Reingeniering of SRIS with Mode-automata

Introducing Mode-Automata in the Lustre data-flow language is a new research work which responds to an actual industrial need. The experience conducted on the SRIS case study shows the feasibility and the benefit of this approach. It is clear that our applications may exhibit an imperative control structure and that Lustre is no longer suitable to express it.

This first step is very encouraging and further work is needed both on theoretical aspects and on tool aspects. In particular, it is important that this approach be implemented through the Scade tool.

Conclusion

Within the SYRF project, Schneider has provided a real case study. Three methods/tools resulting from SYRF have been experimented with on this case study.

- Formal verification of properties with XLESAR. We note that the development of this graphical interface corresponds to an industrial need. But the integration of the method is still difficult since it is very far from our traditional verification practices.
- Automatic testing with Lurette. Further to the SYRF results and to an internal project, Schneider Electric will pursue this work in order to extend and disseminate this tool in the industrial world. The continuation of the SYRF work will be carried out in collaboration with Verimag within a project funded by Region Rhone Alpes.
- Mode-Automata is a new interesting topic. Schneider encourages further work in that direction and recommends to implement this approach in the Scade tool.

SAAB Conclusions

We recall here the conclusions of deliverable 1 on SABB case studies.

Even though the climatic chamber was a small demo system it reflects much of the functionality included in real systems except for the size. Our interpretation is that the example has been useful for demonstrating both technical problems and new ideas.

Unfortunately the ECS has not been investigated so much due to the problem of transforming the ASA+ format to some format utilized by the project partners, except for Prover. This is of course a drawback since one of the main interests in industry is to investigate the ability of this "formal method approach" to analyze large systems since most industrial systems are large. Anyhow, a few analyses have been performed demonstrating the ability to successfully do automatic verification for this large system.

So, one lesson learned is that it is impossible in this type of project to utilize real systems for testing unless there is a "computer model" at hand from the industrial partner.

This is mainly because the time required to do the modeling, starting from a specification in natural language, is enormous compared to the total project budget.

The use of formal methods (FM) in practical applications have been evaluated a number of times during the last ten years, see for example [1] and [2]. From the experiences of participating in the SYRF project, doing some research for utilizing formal methods for reliability analysis and having used NP-Tools at Saab for occasional investigations, in our opinion, the future of "formal method" depends on three areas:

Tools: This area includes both the "engine", i.e. the algorithms being able to handle large industrial systems, the formats and the translations between formats and the interface between the user and the tool. Some of the reasons why industry have not yet started to use FM can be found in this area. One difficulty is that of translating between different formats which results in problems both when we want to use two or more design tools or when we want to be able to perform analysis, for example automatic verification, not included in the design tool. Another one is the doubt about being able to handle practically large systems.

Methodology: Both from our own experience and from references [1] and [2] it is clear that just giving the engineer access to a tool does not mean he will be able to perform a better and more effective work. Much work needs to be done before a methodology can be presented on how to use the different tools efficiently.

People: A third area is that a proper education is needed of the engineers who are going to use these tools and methodologies. One important circumstance to bear in mind is the pitfall of using a too complicated mathematical notation within formal methods. So, in the further development of tools it is desirable to "hide" as much as possible of the complicated mathematics within the tool and trying to have an engineering friendly interface, which would make it easier to educate larger groups of engineers.

The concluding Saab remark of the SYRF project is that it certainly has made contributions, specially to the "Tool" part, since both algorithms, formats and tool-interfaces have been further developed within the project.

Bibliography

[1] M. Hinchey and J. Bowen. Applications of Formal Methods. Prentice Hall, 1995.

[2] D. Craigen. An introduction to formal methods, Part III: In dustrial usage. ORA Ottawa Canada, 1992.

Work Package 2

Combination of Formalisms

Results

There are two interpretations of the title of the work package which have guided the research:

- a *loose integration* mixing existing languages, and
- a *tight integration* mixing different styles.

Loose integration

For the first theme, the existing languages ESTEREL, LUSTRE, and SIGNAL have been translated to the synchronous intermediate format DC. The approach has been very successful in that its power has been demonstrated in case studies: components of an application have been separately developed in different languages (and programming environments), and then linked on the level of DC.

This is also possible to integrate these languages at the level of DC+. The SIGNAL/DC+ environment can be used to integrate components developed in different formalisms and to generate a distributed implementation (LUSTRE and ESTEREL programs are integrated through their DC representation). This approach has been demonstrated on the SAAB case study.

For this theme, we had to develop a lot of translation tools, and a link editor at the level of DC.

Tight integration

For the second theme, we developed three kinds of approaches:

- the combination of (restricted versions of) the synchronous languages ESTEREL, ARGOS, and LUSTRE within the *Synchronie Workbench* (SWB),
- the extension of LUSTRE by *mode-automata*, and
- the combination of language constructs for sporadic and periodic signals within the synchronous object oriented language *sE*.

All these approaches are supported by programming environments and have been successfully applied in the case studies.

The results achieved comply with the project programme in that the original goals have been established in spite of some deviations of the original workplan. The original plan of a uniform framework has been modified to investigate the extension of LUSTRE by mode-automata and the semantics of the workbench separately. Both approaches have a well defined semantics for the combination of the respective formalisms (cf. papers Conference 17, 24, Report 6).

The three main goals have been:

- to establish a common semantical framework for the translation and integration of the different synchronous formalisms,
- to base an efficient translation scheme on this framework, to substantiate it within a workbench, and
- to check whether the results transfer to industrial strength compilers.

The corresponding work is summarised as follows, hinting at the deviations from the workplan.

Mode-Automata

Mode-automata have been driven by the industrial wish to talk about the *running modes* of a reactive system, in a data-flow program. The main motivations were: the readability of the program, the efficiency of the generated sequential code. It appeared that a mixed-language approach does not allow to fully benefit from the mode-structure of a system, especially when the final sequential code is concerned. This is the main reason why we proposed to introduce a new language feature in a data-flow language. It has been done for Lustre, but could be applied to other data-flow languages as well.

Mode-automata are now compiled into DC, thus allowing to mix mode-automata programs with Lustre, Signal or Esterel ones, in a loose integration framework.

From SWB to *sE*

Though quite a mature prototype, SWB was replaced by *sE*. The rationale is this: we have found a more generic semantic model (than the one originally used for SWB) based on operations on synchronous automata (cf. conference paper 24). In consequence, we have built a small generic semantical kernel implementing these operations. We have decided to use the object-oriented synchronous language *sE* as a testbed since experiments with SWB (within and outside of the project) showed that programmers are not keen to learn and use several languages at the same time. *sE*, in contrast, offers a smooth integration of paradigms; flows (= periodic signals) and discrete control (= sporadic signals) are special incarnations of the general notion of signal, but may be used in different syntactic contexts. Presence is specified either by emittance or by clocks. Values may only change if a signal is present, the respective constraints being either specified by the emittance statement or by equations. In spite of its compositional translation scheme, *sE* is very efficient in terms of space and time consumption of the generated code that, for middle size applications, has been down-loaded even to small micro controllers.

Transfer

Transfer of the results to industrial strength tools has been restricted to the mode-automata approach, which will probably be integrated to an industrial tool. The translation schemes originally conceived for the Swb have not been used for industrial tools, since they demand some knowledge about the internal workings of the industrial tools.

Here we have adopted the policy of a loose integration, which was not part of the original workplan, but the only feasible strategy.

Conclusions and Perspectives

Each of the two interpretations of "combination of formalisms" - mixing styles, and mixing languages - has its merits and its disadvantages.

Mixing languages

"Mixing languages" is useful when one has to combine components that have been developed in different contexts, with different languages and their associated programming environments. Developers can use their favourite language for components in big projects. Reuse of components is supported. However, the integration through Dc may be too "low level" enough to take advantage of the full power of the corresponding tools, mainly because the clocks are not explicitly present in Dc. Further, a component may rely on both, data flow and discrete control. If developing in a single language, one of the paradigms has to be recoded in the other, e.g. discrete control in data flow, which often leads to solutions that do not naturally reflect the problem structure and are more complex than necessary.

One important lesson learned from these language combinations from the Esterel point-of-view was the complementary role played by models or representation formats where control and data were either dissociated or merged together. This duality corresponds (very roughly) to the complementarity between Esterel (control) and Lustre/Signal (data-flow), or between control paths and data paths in hardware terminology. In particular the modeling of the SAAB climatic chamber example confirmed that Esterel was well-fit to implement the changes of control modes triggering operations on data, while Lustre was perfectly suited to turn back test conditions on data value thresholds into signal events to prompt these control changes. Separation or integration of control from data both have their intrinsic expressive and algorithmic qualities and limitations (FSM model-checking for control only, or with drastically abstracted data types at most; finer analysis and connections to simulators for integrated models). The opportunities for optimization, ressource allocation, or code distribution are also seen from a completely different angle in either case. All these topics deserve further research in our opinion. Being able to fully travel back and forth between these two viewpoints is already a big step for practical experimentation.

Mixing programming styles

Mixing of styles is appropriate when designing components from scratch. The design benefits in that different paradigms are represented by the respective formalisms with a well understood but smooth interface. Mixed style languages, however, have the disadvantage that the programmer has to master several formalisms at the same time. By experience, programmers rather stick to one, familiar language only. From this point of view, the Swb may be seen as a failure. Mixed-style languages should be simple to use. We cannot expect people to write programs using the full power of Lustre clocks together with the full power of Esterel reincarnation, and still understanding what happens. Mode-automata are a gentle way to extend the data flow paradigm with features of hierarchical automata. *sE* is slightly more ambitious in that it supports the full power of combination, but hides details by a uniform presentation and interpretation of signals. It will be seen if this generality will be restricted in future.

For the perspective, we expect that the results achieved for the loose combination, and for the mode-automata will be part of, or will be supported by industrial strength tools in future. The Swb is available as a mature prototype for experimentation. *sE* will be public domain software. This will be supported by writing a book on object-oriented embedded software design based on *sE* as a programming language. A compiler from mode-automata to Dc is under development, and will be available soon.

Work Package 3

Program Validation and Verification

Task 3.1: Symbolic abstraction of automata

Work in this task went in four major directions, namely *symbolic bisimulation* of synchronous reactive processes, efficient expression and verification of *simple generic properties*, *controller synthesis* based on transition system models, and *compositional optimization* of Boolean equation systems. The common feature of all these topics is that they refer to analysis functions defined at the interpretation level of automata (Mealy FSM to be more precise), while they attempt to provide effective algorithmic solutions avoidinf the explicit state space representation. This is done either by relying on symbolic BDD techniques or even by computing at higher-level of modeling, taking syntactic or structural informations into account.

Results

Symbolic bisimulation

During the first year methods were designed to compute bisimulation at the level of *reactive automata* (INRIA Meije) or *implicit labelled transition systems* (INRIA Ep-Atr). At this level the states may be treated explicitly or intensionally, but the behaviours are always considered symbolically (as predicate logic formulas on input leading to sets of outputs). A given transition is then a *meta*element, gathering in fact a set of possible behaviours, and the bisimulation match may require to compare sets of transitions altogether. We extendedthe classical algorithms for bisimulation-based model-checking to this framework. In the second year the tool Fc2symbmin (INRIA Meije) was fully implemented, tested and documented to follow this method using BDD representation of behaviour predicates [2], while Sigali (INRIA Ep-Atr) was extended to support the same method using polynomials as symbolic data structure [3].

In the third year the model of *inplicit labelled transition systems* was revisited to implement in this data structure bisimulation-related notions such as *hiding* or *restrictions*, and state fusion according to equivalent classes [4].

Simple generic properties

During the second year research was conducted at INRIA Meije on efficient algorithmic for the detection of divergent loops in processes. The notion tightly linked with verification of liveness properties, specially when considering the use of observers for property reduction to simpler properties. In fact instead of computing exact loops we compute by simple forward fixpoint on states an overapproximation, with the important property that emptyness of this set is equivalent to emptyness of the original loop set. This provides a method for checking some simple liveness properties at the cost of safety properties (a fixpoint on sets of states reached through forward symbolic breadth-first search). The method is implemented as part of the Xeve model-checker [1]

Also in the second year work was conducted at VERIMAG (formerly INRIA Spectre) on verification of *stability* properties. A system is stable when its eventual further activity is prompted only by further inputs fed to it, not by inifinite self-activation. A static approximation of this property was characterized, with the correspoding static analysis detection method. This result was obtained in collaboration with researchers at EDF [6].

Controller synthesis

During the second year, extending into the third year, research was conducted at INRIA Ep-Atr on the (now classical) Controller synthesis problem. The issue here is to design a controller program to ve used as a constraint, operating in parallel to a plant, restricting the behaviours to remain inside a given specification of correct systems. In the context where both the plant to be controlled and the correctness objective ar eprovided as Signal programs, an algorithmic solution is provided in terms of *polynomial dynamic systems*, which provide a symbolic data structure where to conduct computations. These methods were included into the Sigali verification tool [5].

Compositional optimization

Work was conducted at INRIA Meije from the second year on in the study of *compositional optimization of Boolean equation systems*. The prospect here was to allow better compilation and analysis of very large Esterel programs. In general Esterel translates into a model of harware circuits expressed as Boolean equation systems (with boolean *latch* memory elements). The structural translation is often far from optimal, because all preemptive contextual situations need to be taken into account. General optimization tools from the hardware logic synthesis community, such as Sis from U.C. Berkeley, perform drastic reduction in size on BESs, but they require at times global algorithmic methods, including symbolic computation of the Reachable State Space, which often never converge on large circuits. Therefore modular compositional approaches are very welcome, as they allow to perform local reductions prior to the composition into larger systems. Nevertheless extra information must be preserved if proper linking of optimized subcomponents is meant to perserve the overall semantics. Then again, keeping this information naively may destroy the huge potential gain in size reduction. All our work was aimed at providing sufficient constraints and devising adequate algorithms to allow consistent modular optimization together with semantic preservation. In the third year the Esterel processor ScBlif, BlifSc and ScLink were implemented or extended to allow modular optimisation and analysis of Esterel programs [7]. The prototypes are now being tested at industrial sites by users.

Task 3.2: Use of the Stålmarck Method

This task concerned the use of Stålmarck's method [12] for formal verification of synchronous programs. The results of this task demonstrated that Stålmarck's method complements the existing methods for verification well, in that it is not sensitive to problem size, rather a proof-theoretic notion of *hardness*. One reseach direction within this task demonstrated ways of reducing hardness, by a careful relaxation of the sub formula principle, making a previously unfeasible problem category easy to prove [11]. However, much remains to be done before this kind of hardness-reducing trick becomes generally applicable.

On the application of Stålmarck's method to verification of synchronous programs, this resulted in the development of the Lucifer tool [10].

Results

The tool Lucifer Lucifer is a tool that translates a Lustre program into a model in propositional logic, and provides a number of ways in which verification of safety properties can be made. Safety properties are expressed using synchronous observers.

Lucifer is based on NP-Tools, which in turn is based on Prover, the Stålmarck method implementation. Lucifer offers verification based on induction proofs or model checking techniques. In inductive proofs the length of the induction base is the number of times the transition function is unrolled, given as a parameter to Lucifer. The model checking implemented in Lucifer offers complete reachability model checking, and can be done in different ways. The difference lies in the type of constraints that are added to the unrolled transition functions. Optimally, one should never iterate beyond the diameter of the system. The current type of constraint implemented does not acheive this, though we believe to have found a way to do this, by adding constraints using quantified boolean formulas (QBF). Here, we intend to make use of a technique which reduces QBF to propositional logic [13] available as a special case of a theorem prover generalising Stålmarck's method to many sorted first order logic.

In Lucifer, countermodels are provided when a program fails to satisfy the required property. Because the translation is into a hierarchical network of components, matching the structure of the Lustre source, these countermodels can easily be used to trace bugs in the graphical interface of NP-Tools.

Lucifer offers also a kind of validation support by using the mechanism for unrolling a transition function; by giving partial models as input in the graphical interface of NP-Tools and applying model generation (based on satisfiability testing), it can be established whether the partial models given can be extended to real models or not.

Conclusion and perspective

The potential for industrialisation of Lucifer seems good; already, Lucifer has been used in more than 5 industrial projects performed by Prover Technology, and some customers, like Aerospatiale and Xilinx Inc. [9], are already using Lucifer themselves. The planned industrialization of Lucifer includes integration of Lucifer with a CASE tool in the synchronous area. This would make the proof techniques in Lucifer accessible to the relevant engineers using synchronous programming.

Concerning the results on model checking using propositional formulas, we believe much remains to be done. Model checking started off in quite concrete terms, by collecting explicit states, followed by increased levels of abstraction — symbolic model checking using BDDs and later also model checking based on propositional formulas, e.g. bounded model checking. So the trend could be said to be towards more abstract techniques. Prover Technology's work in this area includes a specification of a generalisation of Stålmarck's method to the modal logic S4 [8], and an implementation of Stålmarck's method for propositional linear temporal logic. We would guess that future model checkers will coincide with temporal logic theorem provers, maybe specialized for proving e.g. safety properties.

Task 3.3: Handling properties depending on numerical variables

This task was rather research-oriented. In fact it is the subject of Bertand Jeannet's thesis, which should be terminated during the year 2000.

The goal was to solve the problem of the size of the control structure which is used as a basis for *linear relation analysis*. We designed a technique for progressively refining an initial rough control structure, in function of the property we want to prove. This technique was implemented into an experimental prototype. The refinement is based on heuristics, which have been designed and tuned experimentally. The results are encouraging, although the technique does not yet work for really big examples.

It was not possible to apply the results of this task to SYRF case studies, because none of these case studies showed dynamic numerical properties. We are still looking for good examples for applying these techniques. Such examples are cases were the prominent cost comes from the size of the control structure: the dynamic property should depend on a relatively small numerical part of a big, control-dominated program.

Concerning perspectives, this approach could be applied to other, less expensive analyses, like interval analysis. The same explosion of the control structure occurs critically in the verification of timed automata, which is also a good application area.

Task 3.4: Deductive Proofs of Data-flow Programs

Results

This task aimed at providing methods and tools for applying deductive tools to synchronous programs that cannot be model-checked. As an example, the chosen synchronous language was Lustre and the deductive proof tool was PVS. In the course of the project, the Irisa team working on Signal joined the task and proposed another instantiation, based on Signal and the proof assistant tool Coq.

During the project, two approaches have been prototyped:

- A Signal-Coq one [17] based on co-induction and mu-calculus. It takes the form of of a Coq library axiomatizing the Signal constructs.

- A Lustre-PVS one [15], based on Kahn semantic and also mu-calculus. Here also this theory has been axiomatized but, unlike co-induction and Coq, the Kahn theory is not hardwired into the PVS proof tool. Thus, for efficiency purposes, an off-line tool unfolds a PVS strategy and generates obligation proofs that are left to PVS.

In addition, some experiments on Lustre programs translated to NP-Tools via the tool Lucifer were performed.

In the course of the project, several significant applications have been identified and partially experimented with the tools:

- An emergency braking system [14]
- The Steam Boiler example [16]
- The climatic chamber case study [18]

Conclusion and Perspectives

At the end of the project, we come up with two approaches: the Lustre-PVS one and the Signal-Coq one, which seem to operate quite satisfactorily. Unfortunately, we have not been able to make comparisons between them and this remains to be done, both theoretically and from the efficiency point of view.

Also, the question of scalability has to be better investigated. One possibility could be to study some refinement-based design method, adapted to the synchronous programming context. Another possibility could consist of investigating the cooperation between model checkers and theorem provers.

As with the NP-Tools proofs, while the tool is industrial-strength, the approach with proving closed loop properties based on Lustre programs was new. While there were childhood problems with the tool Lucifer, necessitating rewriting of parts of the program to be verified, the experiments indicate feasibility of further work in this direction.

Task 3.5: Use of automata technology and synchronous observers for automatic testing

Results

This task concerned a completely new research area, when the project started. It resulted in a tool, called Lurette, for testing synchronous programs, and in a technique for evaluating and improving test suites.

The tool Lurette: In the chosen approach [21, 20], the goal is to generate and run *realistic* test sequences for functional testing. The principles are as follows:

- The program under test is considered as a black-box, since testing is supposed to be particularly useful for programs which are too complex to be verified. We only assume that it behaves as a synchronous program, i.e., by a sequence of atomic reactions, and that we are able to run it, step by step. Moreover, a better test can be performed if we can store and retrieve the current state of the program, for running several reactions from the sane state.

- The test sequences are randomly selected, but submitted to constraints. Such constraints either express some knowledge about the program environment (*realism*) or allow the user to drive the test towards some *test objectives*. Constraints are safety properties involving inputs and outputs of the program. In Lurette they are expressed by synchronous observers, written in Lustre. We investigate also other, more imperative, formalisms to express such constraints.

- The test oracle, which decides whether a test "passes" or "fails", is also provided by an observer.

Given the executable code of the program, and the observers describing the environment and the oracle, Lurete is able to generate and run arbitrary long test sequences, satisfying the environment contraints, while checking that the oracle constraints are fulfilled.

Lurette has been successfully experimented on several examples, including the case study proposed by Schneider-Electric. Unfortunately, because of lack of time, the prototype was not complete enough to deal with the SAAB case study equipped with the specification of the environment designed in WP7.

Evaluating and improving Test Suites: A more recent research [19], concerns the use of an approximation of the reachable states of a program (for the time being, the approach is dedicated to Esterel), to evaluate and improve a set of test sequences. Basically, the approach consists in comparing the set of states that are "hit" by the test, with the set of reachable states.

Conclusion and Perspectives

There are very good perspectives for developping and industrializing the Lurette prototype. A project in that direction is under construction, involving Vérimag (formerly Inria-Spectre), Schneider-Electric, Vérilog, and maybe Aérospatiale. It is likely that, in a near future, the Scade environment will be provided with a testing tool derived from Lurette.

The key point is the specification of constraints, which, from a practical point of view, is a difficult task imposed to the user. On one hand, as said before, we investigate more convenient languages for that task. A less ambitious idea would be to let the programmer write simple constraints in the program (e.g., advanced type information, like intervals for numerical inputs), which would be extracted and used by the testing tool.

Bibliography

[1] A. Bouali Xeve, an Esterel verification environment, *Tool paper, Proceedings of the Tenth CAV Conference.* LNCS 1427, 1998.

[2] M. Bourdellès. The steam boiler controller problem in Esterel and its verification by means of symbolic analysis. Research report Nr. 3285, Inria, October 1997.

[3] O. Kouchnarenko and S. Pinchinat. Intensional Approachs for Symbolic Methods. Electronic Notes in Theoretical Computer Science, vol.18, 1998.

[4] S. Pinchinat, H. Marchand, M. Le Borgne. Symbolic Abstractions of Automata and their application to the Supervisory Control Problem. Research report Irisa, No 1279, November 1999.

[5] H. Marchand, M. Le Borgne. The Supervisory Control Problem of Discrete Event Systems using polynomial Methods. Research report Irisa, No1271, October 1999.

[6] J.-F. Héry, J.-C. Laleuf, N. Halbwachs, F. Lagnier. Stability of Discrete Sampled Systems Research Report, Vérimag, 1998.

[7] A. Bouali, L. Henry-Gréard, R. de Simone. Compositional Optimization and Analysis of Esterel programs. *Draft internal report, to be submitted*, 1999.

[8] Per Larsson. Ett effektivt bevissystem för S4. Master's thesis, The Department of Philosophy, Stockholm University, 1996.

[9] Carl-Johan Lillieroth and Satnam Singh. Formal verification of fpga cores. *Nordic Journal of Computing*, 1999.

[10] Magnus Ljung. Formal modelling and automatic verification of Lustre programs using NP-Tools. Master's thesis, Royal Institute of Technology. Department of Teleinformatics, 1999.

[11] Mary Sheeran and Arne Borälv. How to prove properties of recursively defiend circuits using Stålmarck's method. In *Proceedings of Workshop on Formal Methods for Hardware and Hardware-like systems, Marstrand*, June 1998.

[12] Mary Sheeran and Gunnar Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. *Formal Methods in System Design*, 16:1, January 1999.

[13] Gunnar Stålmarck. Stalmarck's method with extensions to Quantified Boolean Formulas. In *Proceedings of the Eleventh International Conference on Computer Aided Verification*, Lecture Notes in Computer Science. Springer Verlag, 1999.

[14] S. Bensalem, P. Caspi, C. Dumas, and C. Parent-Vigouroux. A methodology for proving control programs with Lustre and PVS. In *Dependable Computing for Critical Applications, DCCA-7, San Jose.* IEEE Computer Society, January 1999.

[15] C. Dumas. A PVS proof obligation generator for Lustre programs. *Unpublished draft*, 1999.

[16] M. Kerboeuf, D. Nowak, J.-P. Talpin. The steam-boiler problem in SIGNAL-COQ. Research Report No. 3773. INRIA, October 1999.

[17] D. Nowak, J.-R. Beauvais, J.-P. Talpin. Co-Inductive Axiomatization of a Synchronous Language. In *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics*, number 1479 in LNCS, pages 387–399 Springer Verlag, October 1998. http://www.irisa.fr/prive/nowak/publis/tphols98.ps.gz.

[18] S. Nadjm-Tehrani, O. Åkerlund. Combining theorem proving and continuous models in synchronous design. In *Proceedings of the World Congress on Formal Methods*, Toulouse, France, September 1999, Volume II, LNCS 1709, pages 1384-1399, (C)

[19] L. Arditi, A. Bouali, H. Boufaied, G. Clavé, M. Hadj-Chaib, R. de Simone. Using Esterel and Formal Methods to Increase the Confidence in the Functional Validation of a Commercial DSP. *FMICS'99 Workshop* , Trento, June 1999.

[20] N. Halbwachs and P. Raymond. Validation of synchronous reactive systems: from formal verification to automatic testing. In *ASIAN'99, Asian Computing Science Conference*, Phuket (Thailand), December 1999.

[21] P. Raymond, D. Weber, X. Nicollin, and N. Halbwachs. Automatic testing of reactive systems. In *19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

Work Package 4/5

Integrating synchrony and asynchrony

Deviationism

The final report deviates in many respect from the objectives and even from the structure of the proposal. In the proposal, the following was promised :

1. Have a theory of desynchronization relying of the only notion of endochrony.
2. Based on this, develop a methodology for synchronous program distribution, including component integration.
3. Develop a library of communication elements, written in Signal, which could be used to model a certain class of asynchronous communication media, for use to model the system {deployed application + communication layer}.
4. Implement this and have an application example.

These items were spread among different tasks for WP 4 and 5. Very early in the project we informed the project evaluation committee that we would delay item 3, due to the lack of a clearly selected target application domain. WP 4 and 5 were subsequently merged. In replacement of the above promised items, the following results have been obtained :

1. During the second year of the project, we discovered the more important notion of isochrony, and recognized the importance of its compositional and local nature.
2. This year, we found a systematic theory supporting the specific data structures, called clock hierarchies, used in the Signal compiler. These data structures are instrumental in handling endo/isochrony efficiently. We feel we now have a fairly complete picture of what synchronous program desynchronization means.
3. We now understand that this corpus allows us to address dynamic creation of synchronous instances communicating over an asynchronous medium, therefore opening the perspective of deeply using synchronous technology for reactive systems object oriented design.
4. Although not all ideas are implemented, a large part of them is, both in the academic Signal Inria tool (for both Signal and the DC_+ format), and in the Sildex tool by TNI.

Then, in the original proposal, we promised to develop asynchronous program resynchronisation. In some sense, the above theory provides some kind of answer, at least when the asynchronous program has been derived from a synchronous one. But we did not provide a deep and satisfactory answer for the re-synchronization of asynchronous programs in the general case [1]. Instead, we came up with the idea of timed models for reactive systems using the MaxPlus algebra. Only draft results have been obtained so far, but we are convinced this is a good route to follow. These results are reported in task 4/5.3.3.

Tasks 4/5.1 and 4/5.2

Results

Here we summarize the results about what synchronous program desynchronisation means, and how it can be performed.

- We believe we have theoretically solved out the problem of desynchronisation of synchronous designs. We have investigated the issue of when is desynchronisation revertible, and how to make it revertible whenever wished. We have investigated the issue of preserving semantics, when broadcast synchronous communication is replaced by asynchronous communication using a network of point-to-point fifo channels. Corresponding properties of *endochrony* and *isochrony* have been discovered to guarantee that desynchronisation of a program is revertible, and that asynchronous communication still preserves semantics. As these properties can be checked on the synchronous specification, they can be synthesized too and this provides us with a technique for a systematic generation of schedulers and protocols for *correct asynchronous deployment of synchronous designs.*

  Valid deployment architectures for our theory include POSIX, synchronous and asynchronous models of object communications, and we have shown that the technique is indeed extendable to time-triggered type of busses such as preferred in control.

  Also, suitable data structures have been studied to implement this theory, namely the *Hierarchic Normal Forms* (HNF). HNF capture the nesting of clocks or state transitions, they immediately reveal endochrony. We have developed an incremental algorithm which computes the HNF of the composition of two designs from the HNF of their components, this algorithm can also be used to handle isochrony.

- This theory is, in large part, implemented in the SIGNAL compiler under the name of *clock calculus.* This applies to the DC_+ format as well. To handle desynchronisation in the SIGNAL compiler, endochrony is heavily used, also in replacement for the more recent notion of isochrony. Therefore, the more recent notion of isochrony is not yet implemented in its full power. The commercial SILDEX tool for SIGNAL marketed by TNI, also implements a restricted version of this theory. The restriction is that only executable programs can be distributed, not partial designs or architecture (something which is also possible in our theory).

- As for the perspectives, they are numerous. Firstly, enhancing both SIGNAL and SILDEX to handle architectures will make a deeper use of the reported results, both

[1] May I say that this is however still an interesting topic, for instance in the framework of mapping an high-level Message Sequence Chart to a finite state machine.

enhancements are planed. Second, we currently start an application to UML: we propose a new formalism, called BDL. BDL has dual synchronous/asynchronous semantics and features to help for behavioural inheritance (by adding more constraints and/or more behaviours), it supports a (restricted form of) dynamic instanciation with a clean semantics. It supports a synchronous design of the prototypes and asynchronous dynamic instanciation. It is intended to serve as a semantic backbone for both UML statecharts and sequence diagrams.

Conclusions and new perspectives

What is implemented? Industrial transfer. Distributed code generation for SIGNAL, the DC_+ common format, and the SILDEX tool by TNI is implemented using simplified versions of endochrony and isochrony. In particular only the current state is considered, not the previous state. Therefore nested states are not captured today, only clock hierarchies are captured — probably the most important, however.

Based on this, the following methodology for system design can be followed:

1. Specify the functional architecture. Resulting specification can be complete or alternatively partial (partial designs are currently not handled by SILDEX, however). Predefined components can be imported.

2. Deploy the functional architecture on an abstraction of the distributed execution architecture. To this end, additional schedulers and protocols are automatically generated to enforce endo/isochrony at wish. This makes the components to become robust against asynchrony of the communication medium.

3. Generate code for each properly enhanced component, by using calls to communication resources from the communication layer.

4. To analyse the whole system, model the communication medium using multiclock, nondeterministic synchronous systems. This emulates asynchrony at will, provided that only bounded memory is used by the communication channels — in particular unbounded fifos are not handled. Then the whole model {components + communication media} can be given to analysis tools, for property checking or timing analysis.

In talks given at Femsys'99 (Munich) and FM'99 (Toulouse), a Statecharts example was discussed.

Dynamicity and mobility. Thanks to the compositional and local nature of isochrony, a certain degree of dynamicity becomes easily implementable, based on the synchronous programming paradigm. This may be regarded as a small revolution, as it has always been said that dynamic instanciation and synchronous progrsamming are mutually exclusive concepts. Let us sketch how one can proceed (formalizing this is the subject of ongoing work).

1. Pick an endo/isochronous design, e.g., a GALS architecture.

2. Regard its components as *object classes* rather than programs.

3. Instanciating an object consists in renaming some prototypes by creating fresh names. Pick two classes Φ and Ψ with common set of variables W. Joint instanciation of these would consist in renamings $\Phi[V/V_{\text{new}}, \Psi[U/U_{\text{new}}]$ in such a way that $V \cap U = W$ is consistently mapped into $V_{\text{new}} \cap U_{\text{new}}$. As isochrony is compositional and local, it is preserved by such an instanciation.

4. Therefore an off-line preparation of the classes for endo/isochrony guarantees a correct on-line and asynchronous instanciation. It remains to resourcing to the available mechanisms for dynamic instanciation offered by existing object oriented languages for the execution (e.g., C_{++}, Java, Eiffel,...).

In the same vein, we are currently investigating mobility, by relating our approach to the Join Calculus of G. Gonthier [5].

UML. From the previous results and remarks, it is natural to investigate the use of our techniques to establish a behavioural semantic backbone for UML, targeted to reactive systems [2]. To this end we have proposed a new formalism called BDL (Behavioural Description Language) [3, 6]. As compared to existing synchronous languages, BDL possesses the following features:

- It is targeted to architectures and partial designs. It aims at providing behavioural mechanisms of inheritance (a recognized difficult subject). To this end it is equipped with two composition operators: the parallel composition $\|$ (to specialize behaviours by adding constraints), and the nondeterministic choice $\vee$ (to enrich a given behaviour by adding new behaviours).
- It has a dual synchronous/asynchronous semantic.
- Statecharts as well as Sequence Diagrams and Message Sequence Charts can be translated into BDL, thus offering a blending of these formalisms at a semantic level.
- A visual syntax is under study, as a mild variation of Statecharts.

Task 4/5.3: Timing Analysis

We have proposed a systematic modelling approach for the timing evaluation of systems which can be modelled by partial order automata. Partial order automata are a convenient abstraction for synchronous reactive systems, as discussed elsewhere in this project. In references [1, 2] we have introduced our model, based on so-called Max_+ algebraic approach. This approach handles tensor products of (usual) automata and so-called maxplus automata, which are timed models revealing the worst case execution times for partial order automata. Alternatively, such models are represented by *timed diagrams,* i.e., diagrams with pairs {signal,state} as vertices, and branches labelled by action words and time delay symbols.

This approach can be used to model High level Message Sequence Charts (HMSC) equipped with timing information. Walso described [4] how this model can be used to

[2] This ongoing work is the central subject of the ongoing REUTEL project, funded by Alcatel.

perform nonstandard analyses such as *maximum throughput*. Such type of analysis is not encompassed by usual formal methods for timed automata models, as these are unable to perform asymptotic analyses for the long time behaviour of the system.

As a perspective, we plan to build a tool implementing this approach, by connecting an HMSC editor to Max_+ tools developed at Inria Rocquencourt by S. Gaubert et al.

4/5.4: Automatic verification on mixed A/S systems

Large systems very often cannot be fully described naturally as completely synchronous reactive models, but rather as strongly tied synchronous sites communicating asynchronously in a larger network. We are thus here considering asynchronous networks of synchronous processes, a different issue from the will to distribute synchronous systems on asynchronous architectures. Very oftenthe synchronous part consists of a true user program application, while the asynchronous network is physically provided, and needs only to be modeled for purposes of analysis.

We were here concerned with issues of analysis and verification of system correctness, in the vein of model-checking. An important elementary step in symbolic model-cheking is the application of the transition relation to a symbolic set of states in order to obtain their immediate successors. BDD representation techniques can help keep a compact form for sets of states, burt they are not sufficient to help cope with a full symbolic representation of the transition relation itself for large programs. Therefore several techniques have been developped to *partition* the transition relation, and apply it "by chunks", using relatively small descriptions with an identical effect when combined properly. In synchronous formalisms the partition is performed according to single state elements (boolean registers) of the next-state description. In asynchronous formalisms teh partition is done according to visible global *events*.

In the first year we defined a method for combining the two types of partition methods in a two-level approach. One important point is that communication actions and messages have to be represented and extracted from the local synchronous behaviours. We then studied the possible BDD representation of usual structures used in asynchronous networks (FIFO queues mostly), following earlier work by Godefroid and Long. The originality of our approach has to be found in the combination os such representation technques with the partitioning methods involved in the synchronous approach [7]. As an exercice we also studies the abilities of synchronous languages to actually model such data structures as synchronous processes.

We developped a prototype software to implement these methods. Unfortunately due to lack of manpower we were not able to conduct true benchmarking or extensive tests of this software.

Bibliography

[1] A. Benveniste, C. Jard, S. Gaubert, "Algebraic techniques for timed systems". In 9th International Conference on Concurrency Theory, Concur'98, Nice, France, sep. 1998.

[2] A. Benveniste, S. Gaubert, C. Jard, "Monotone rational series and max-plus algebraic models of real-timesystems". Wodes'98, August 1998.

[3] C. Jard, J-M. Jézéquel, A. Le Guennec, and B. Caillaud. Protocol engineering using uml. Draft paper, submitted, October 1999.

[4] Pierre Le Maigat and Loic Hélouët, "Max_+, a new apporach for time in Message Sequence Charts", draft, 1999.

[5] J-P. Talpin and A. Benveniste. Pre-order transition systems for calculi of mobile processes. Draft paper, submitted, October 1999.

[6] J-P. Talpin, A. Benveniste, and P. Le Guernic. Asynchronous deployment of synchronous pre-order transition systems. Technical report, Inria Research Report, October 1999. in preparation.

[7] A. Bouali, R. de Simone A Symbolic Representation of Asynchronous Networks of Synchronous Processes. *Proceedings of the Tenth International Workshop on Logic Synthesis (IWLS'98)*, Lake Tahoe, 1998.

Work Package 6

Connection with Hardware/Software Codesign

Task 6.1. Translation into VHDL-Verilog

Approach

The translation from synchronous reactive formalisms into so-called *Hardware Description Languages* might seem a simple matter, provided the proximity of their semantic models and the general acceptance of global clocks for the triggering of instant reactions. In practice the connection is far from obvious, due to a (large) number of subtle differences. In our opinion synchronous formalisms, with their clear-cut semnatics, are highly superior in modeling power and predictability to the classical HDL, in which interpretation and admissibility for synthesis very often depend on the individual tool used, and varies with the context of development. But HDL are a cornerstone when it comes to hardware CAD synthesis, if only as entry point to synthesis tool suites.

Two different attempts at translation were conducted as part of Syrf. In the first two years people at INRIA Ep-Atr concentrated on the issue of relating the notion of clock in Signal with the actual notion of clock in VHDL. The translation then consisted of a rewriting inside the signal framework to bring the two notions together, followed by the actual syntactic translation from Signal to VHDL [1]. In the last two years people at INRIA Meije concentrated on producing Verilog or VHDL code from "flat" DC code. Then the code produced depends on the single clock, and it occurred then that the DC code suffered some deficiencies concerning interpretation of the new values of variables on slower clocks (should they be reset to an absent value or refreshed to the old one while "waiting for the next ticj of their clock"?). Alternative styles of translations were designed, and experimentation on connections with various synthesis tools was started to provide a better understanding on intrinsic efficiency and correctness in each case [2].

It could actually be the case that different kinds of translation are needed to preserve the original synchronous semantics form DC when translating in (synthesizable subsets of) HDLs for different tools with different functionalities in mind (simulation or low-level netlist synthesis). This gives substance to our previous claim, in that synchronous languages differ from current HDL in that they respect their formal semantics all along the compilation line, while providing a high-level style of specification (including so-called "behavioural" or RTL level).

Bibliography

[1] A.A. Kountouris, C. Wolinski. A method for the generation of HDL code at the RTL level from high-level format specification language . *Proceedings of MWSCAS'97*, IEEE Computer Society Press, Sacramento, Aug. 1997

[2] A. Ressouche Compiling synchronous reactive languages into Verilog/VHDL. *Internal Report, Syrf deliverable annex*, 1999.

Work Package 7

Integration of Analog and Discrete Synchronous Design

In work package 7 we have considered modelling, transformation, and analysis techniques for properties which arise from interaction of the plant and the controller.

Modelling

The work on modelling was used to resolve three issues.

Basic requirements on a hybrid model

What are the basic requirements on a hybrid model which can accomodate both a non-trivial plant and a collection of controllers, where each controller can individually be seen as a synchronous program controlling some aspect of the plant? Time deterministic hybrid transition systems (TD-HTS) were proposed as a formalism for modelling such distributed controllers together with a plant.

Abstraction of time by steps

Can time be abstracted away when reasoning about such hybrid systems? As we know, the notion of computation step can be conveniently used to get an abstract notion of time in a synchronous controller. The question was whether models of a plant which have this characteristic can be systematically obtained and input to the verification process. It was fairly obvious that any systematic abstractions of a plant should be based on a continuous model derived from physical assumptions. Thus, a first step applied to the case study was the derivation of physical models.

Untimed models

We considered the "inductively inferred valid logical models" of Antsaklis and Lemmon as abstractions of the continuous model. It turned out that this framework did not fit our intention of using the synchronous technology.

First, any plant which includes some disturbances (unmodelled inputs) results in a non-deterministic I/O automaton and the synchronous programming environments use compilers with detect and reject nondeterminism! Secondly, a plant state space partitioned in finite number of regions (the significant regions for a particular reachability question) is only meaningful when composed with a controller which is geared towards changing control signals when crossing the borders of these regions. That is, the plant will only accept changes in its input (coming from the controller) at particular state transitions. In the main case study of work package 7, the controller code offered for analysis could change the controller voltage even when the plant was within a region significant for the reachability property.

Discrete-time models

On the other hand, under reasonable assumptions one can sometimes get piece-wise linear models of a plant. In our case, these assumptions were formulated in the presented case study – giving rise to a model of a plant as a mode-automaton, where a Lustre program is associated with every mode. The Lustre programs specify difference equations derived from the continuous model, assuming that the controller is a sampled synchronous controller.

Abstractions as hybrid automata

Which steps enable a systematic translation of a plant model from TD-HTS to linear hybrid automata (LHA), and how can such models be used in formal analysis? LHA provide the possibility of symbolic model checking provided the computation of the fixed point for the set of reachable states in a particular application terminates. As such they are a valuable tool in analysis. Infact, one result in work package 7 strengthens this characteristic by proposing a conservative approximation which guarantees termination. However, in defining transformations from general hybrid models to LHA there are some obstacles to overcome.

Transformations

Engineering models for system components are developed in a modular manner. This means that the state in a plant is typically defined in terms of control signals and disturbances as inputs. LHA are defined as (closed) systems which interact with the environment using shared events. Our case-study driven research provides guidelines about reasonable approximations on inputs which have a physical nature. However, transforming a plant model to LHA imposes such demands on the control signal that may or may not hold in a given controller submitted for verification.

On the other hand, the SYRF project has been successful in developing constructs and tools which support the systematic translation from general hybrid models to Lustre programs under certain restrictions. This transformation is performed on a modular model of the plant as mode-automaton where each mode is associated with a set of difference equations – compiling the model to flat Lustre. Thus, closed loop model of a system becomes just a union of two Lustre programs.

Analysis

Our verification efforts in the project were guided by the needs in two case studies. We have used deductive proofs in NP-Tools as well as simulation runs in Matlab and MatrixX.

Deductive proofs on the closed loop model

The SYRF project has addressed the use of theorem provers in connection with synchronous programs. In our part of the project we applied verification not only for the controller properties, but also for verifiying a closed loop model. These models were derived using a Lustre to NP-Tools translator (Lucifer) as well as a Statemate to NP-Tools translator. For safety properties which were decomposable along a plant/controller axis the proofs in NP-Tools were satisfactory and fast. For bounded response properties some anomalies remained unresolved. The tool Lucifer used for this purpose was found to have certain childhood weaknesses but its automatic unfoldings extend the range of applicability of NP-Tools to a larger class of problems.

Simulation models for a plant

In what ways can a physical model be used in the process of verification of a synchrnonous controller? We have already seen the use of such models for derivation of discrete-time models. In that context, we have used the plant simulations to obtain a longest possible sampling interval for the controller. This information is utilised in the closed loop models which incorporate the discrete-time models of the plant. These closed loop models, therefore, need not be unfolded more than necessary in inductive proofs in a theorem prover. An unplanned side-effect of these simulation models was the use in the test case generation tool developed in the testing work package.

We have also explored the uses of (Simulink) simulation models for the purpose of co-simulation with generated C-code from a (Signal) synchronous controller. The co-simulation environment was used to illustrate some undesired properties in a siphon pump case study.