In Proc. Intl. Workshop Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97), LNCS 1217: 112-129, 1997

Graphs in METAFrame: The Unifying Power of Polymorphism

M. von der Beeck, V. Braun, A. Claßen, A. Dannecker, C. Friedrich,
D. Koschützki, T. Margaria, F. Schreiber, B. Steffen

Lehrstuhl für Programmiersysteme
Universität Passau
D-94030 Passau (Germany)
steffen@fmi.uni-passau.de

Abstract. We present a highly polymorphic tool for the construction, synthesis, structuring, manipulation, investigation, and (symbolic) execution of graphs. The flexibility of this tool, which mainly arises as a consequence of combining complex graph labelings expressing the intended semantics with hierarchy and customized graphical node representations, is illustrated along a representative choice of application scenarios.

1 Introduction

Graphs are theoretically well-studied and widely used in practice to model e.g. dependence, hierarchy and computation. Particularly powerful are semantics-based versions of *labelled* graphs,[1] which adequately represent structures like flow graphs, transition systems, automata, partial orders, and Petri nets. Therefore a number of tools like AUTOGRAPH [15], the Concurrency Factory [2], SDT [18] and PEP [6] has been developed supporting these structures. However, except for a few cases, where a small set of similar structures is supported, they are limited to exactly one structure. In particular, as far as we know, the support of a (new) structure is always achieved by a specific coding, and not, as proposed below, via library instantiation within a generic scheme.

In this paper we present the graph component of METAFrame[2], which was developed as polymorphically as possible in order to cover a wide range of applications. Our approach, which combines complex labeling, e.g. for expressing semantic properties, with hierarchy and customized graphical node representation, turned out to be flexible enough to capture application areas like flow graph-based dataflow analysis [11], hierarchical specifications in SDL [17], library-based organization of software reuse [22], process modelling [27], and simulation and verification of distributed systems [28]. Particularly interesting is an industrial

[1] Label may be as simple as action names used in an automata representation or as complex as execution codes for large tools and procedures.

[2] METAFrame is a meta-level environment for the construction, analysis and verification of systems [26].

application which was carried out under severe real-time constraints: an environment for the development of Intelligent Network Services had to be completed and turned into an industrial product within 10 months (cf. Section 2). It was the described flexibility of METAFrame which allowed us to meet this deadline.

The advantage of our approach lies in its high degree of sharing. All the features based on our graph component, like 1) symbolic execution or prototype animation, 2) hypertext-based browsing for navigation aid and user's manual, 3) hierarchical structuring and macros, 4) formal verification via model checking, 5) abstraction and diagnostic features, 6) automatic synthesis, and 7) automatic layout can be used in *all* METAFrame applications (Fig.1).

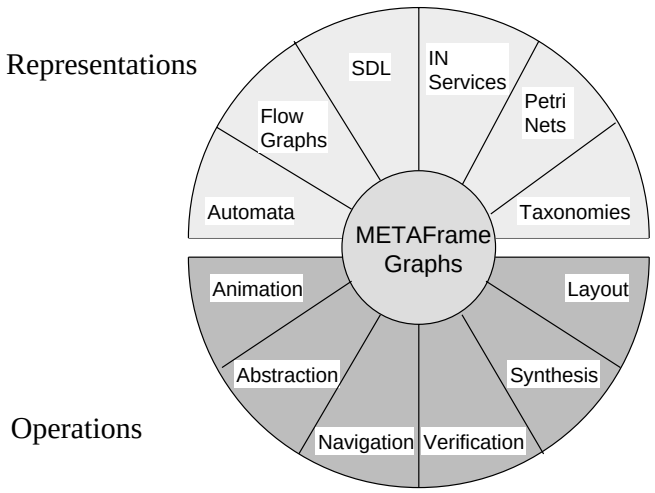

Fig. 1. Graphs, Syntax and Semantic in METAFrame

The power of our graphical component is centered around the concept of *complex, semantic labelling* discussed in Sect. 4. There we illustrate its power and flexibility along concrete examples taken from the previously mentioned application domains. To this aim, we first briefly introduce some of METAFrame's applications in the Sect. 2 and 3. In particular, the next section is entirely devoted to IN-METAFrame, our currently most comprehensive application, with the aim of showing how the different features mentioned above and depicted in Fig. 1 interplay and complement each other, and how METAFrame's graphical component plays a central supporting role.

The graphs shown in this paper are produced using the ffgraph library [5], developed within the METAFrame project, which uses the Tcl/Tk library [14]. They are automatically layouted with a specially tuned version of the algorithm proposed by Sugiyama et al. [29].

2 A Complex Application: Definition of Intelligent Network Services in IN-METAFrame

Intelligent Network Services are customized telephone services, like e.g., 1) 'Free-Phone', where the receiver of the call can be billed if some conditions are met, 2) 'Virtual Private Network', enabling groups of customers to define their own private nets within the public net, or 3) 'Info Lines', where a number of menus leads a caller to the most convenient connection for information and services [23].

In the following paragraphs we illustrate METAFrame's principal features (prototype animation, hypertext-based browsing, macros, formal verification, and diagnostic features) along a simple IN service in order to show how they are used in the context of industrial design practice.

The service shown in Fig. 4 is simple kind of Free-Phone (800-service), called 'Granny's Free-Phone' (Fig. 4). It is designed for people who would like to encourage friends and relatives to call them by paying their calls if they are done at a convenient time of the day.

2.1 Abstract Modelling

The service graph represents a high-level model of the actual service: nodes represent in fact *Service Independent Building Blocks* (SIBs), which are specific procedures for IN applications, and edges the conditional flow of control. The node/edge behaviours themselves come in several levels of abstraction, e.g., symbolic execution codes (Fig. 3) (c) or the actual C code, which will eventually run on the network machinery. All this information, which actually constitutes the label information of this modelling, is available to IN service designers via the SIB library (Section 2.3).

2.2 Prototype Animation: Simulation of the Service

To get a better feeling for the behaviour of a service, or to control its reliability at the design level, designers usually employ *animation/symbolic execution* (also called red-line tracing). The symbolic run shown in Fig. 2 concentrates on a specific aspect of the runtime behaviour, namely user interaction: after a call initialization section, the subscriber specific Free-Phone number is traversed before the caller is prompted to enter a Personal Identification Number (PIN) and, depending on the time of the day, the call is either released (in the forbidden time windows) after an announcement, or it is routed to the desired destination number.

2.3 On-line Documentation: Hypertext Navigation Aid and User's Manual

An on-line documentation via hypertext offers at any stage of the service definition comfortable information about central aspects of this application (including,

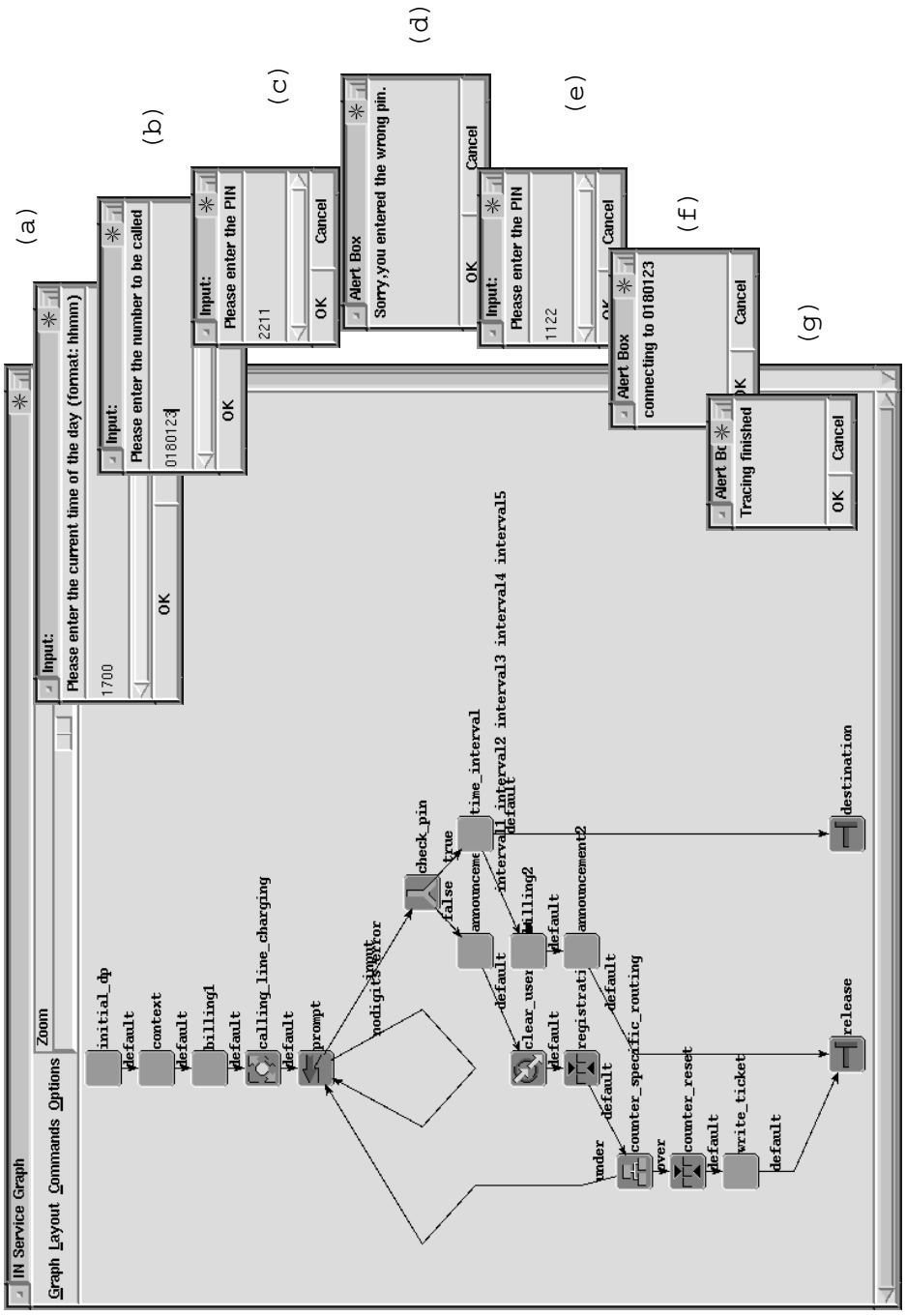

Fig. 2. Symbolic Execution via Red-Line Tracing

Fig. 3. Hypertext Descriptors, the Browser's History and SIB Taxonomy

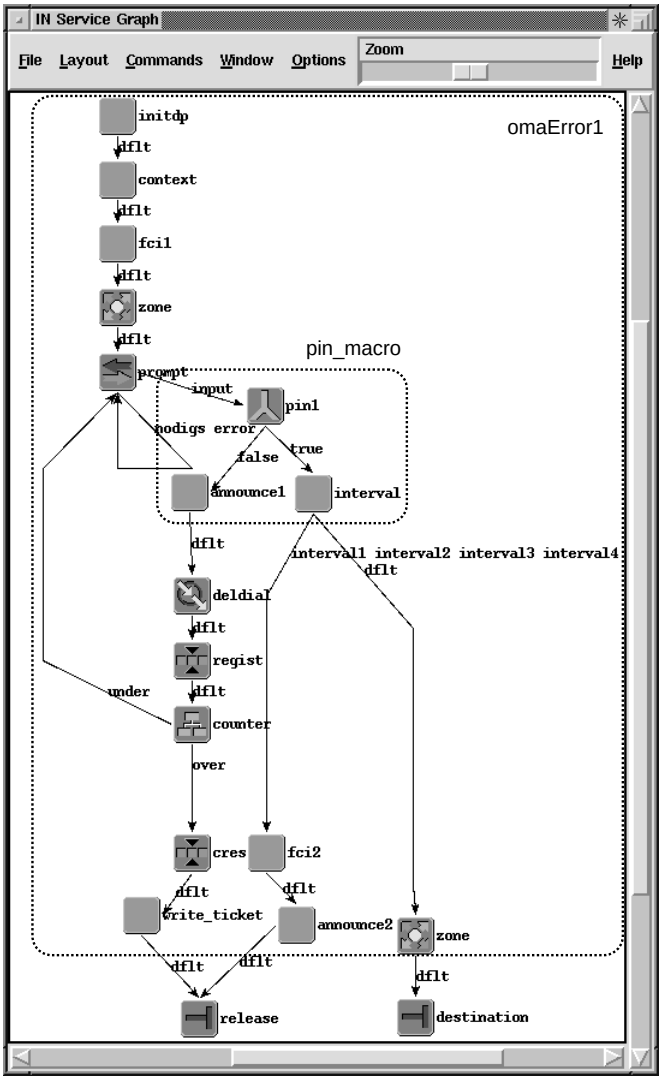

Fig. 4. The Granny's Free-Phone service.

e.g., SIBs, services, and constraints), but also about the METAFrame system, like information about the windows, menu entries (help), and the hypertext system itself. The hypertext system presents a hypercard like the one of Fig. 3 (c) for each SIB, whose many fields cover a variety of documentation aspects and are displayed in a user-specific customized way, depending e.g. on access permissions. All these fields may contain hyperlinks which allow users to browse through the hypertext system using it as an online User's Manual. Navigation functions (step back, consult the history of visited cards as in Fig. 3 (d), step forward in the

history), as well as other editing functions (open a new card, edit or delete the current card) are easily accessed by means of the corresponding menu entries. It is also possible to step from a single card to the taxonomy of SIBs, which provides a fine-grained classification of all available SIBs according to diverse criteria (Fig. 3 (a)). The hypertext browser helps navigating along the taxonomy too, showing ancestors, siblings, or as in Fig. 3 (b) descendants from any selected node.[3]

2.4 Structuring: Hierarchical Service Definition

Hierarchical design of IN services is technically realized in form of a powerful macro mechanism which allows developers to define whole subservices as primitive entities. As macros have formally the same interfaces as single SIBs, this supports the reuse of already designed (sub-) services. The macro concept of IN-METAFrame goes hand in hand with its formal verification and abstract views features [24].

Macros may be nested, allowing the definition of truly hierarchical structures, and they can be folded and unfolded in any graph in order to support abstraction from and focussed investigation of details. Fig. 4 shows an unfolded nested macro within a service graph: the frames indicate the levels of the macros, the omaError1[4] macro containing itself a pin_macro.

2.5 Formal Verification in the Presence of Hierarchy

In order to guarantee successful execution of the designed services in the context of the complex, distributed IN architecture, certain frame conditions (constraints) must be met. To this aim we use IN-METAFrame's automatic verification wrt. formal specifications. While *local* correctness checks, guaranteeing e.g. the correct parameterization of single SIBs, are quite straightforward to implement, and are present also in other Service Definition environments, our environment is unique in offering automatic check of *global* correctness and consistency conditions of the service prototype, concerning the interplay between the SIBs of a service. This can be verified at any time in a push-button fashion [24], by means of the *model checker* for the modal mu-calculus described in [21].

In case of hierarchical structures, verification is always done relative to the fully *unfolded or decapsulated*[5] graph (see [1]). This design decision is consistent with the principle that macros are an (auxiliary) means to structure complex services, but do not have any semantic impact.

---

[3] Details about the hypertext system can be found under `http://brahms.fmi.uni-passau.de/ dannecke/hypertext/hypertext-mail.html`.

[4] This name indicates a design error within the macro, which will be discovered using formal verification in the next paragraph.

[5] **Decapsulation** zooms into the macro structure. Nested structures are decapsulated level-wise.

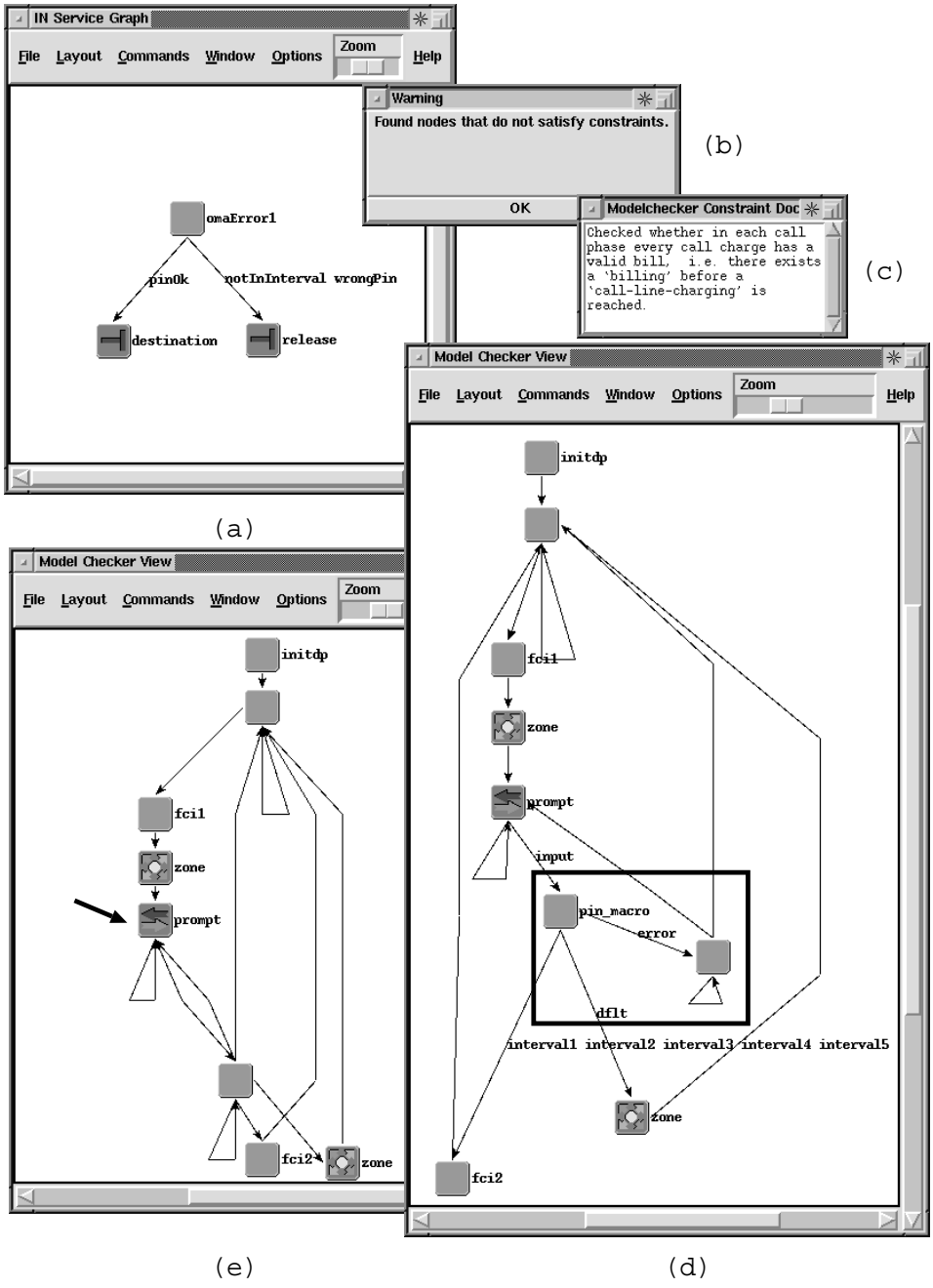

Fig. 5. The Model Checker Finds a Billing Error

Model checking the hierarchical version of Granny's Free-Phone service shown in Fig. 5 (a) results in the discovery of erroneous paths in the graph, which violate the constraint shown in the same figure in the top right window (c): On each segment (phase) of the call, a bill should be issued before the system can charge for the segment. The problematic part of the paths starts at the prompt node marked by the arrow (e), whose icon is framed in red on the screen.

2.6 Property-Oriented Abstraction and Views: Application to Error Diagnosis and Correction

A major application of property-oriented abstraction is the generation of *error views*, which are defined relative to the fully unfolded graph. They hide all aspects of the considered (service) graph that are unimportant for the location of the error, as shown in Fig. 5 (e) for the error detected by the model checker in the previous paragraph.

Error views are generated automatically whenever a model checking attempt fails. Although this error view looks identical to the one obtained when model checking the fully expanded version of the service shown in Fig. 4, there is an essential difference. This becomes apparent when applying e.g. the decapsulate source command, which is a view-specific operation that supports error correction, to the left edge into the zone SIB. This results in the graph shown in Fig. 5 (d): the extracted source node is the macro pin_macro, showing that the macro structure is preserved as far as possible also in error views, which allows *hierarchical error correction*. Details about error views and error correction can be found in [24].

3 Further Applications

In this section we sketch a number of further application scenarios, where all the features discussed in the previous section are also available, even though not all of them are always adequate.

3.1 Heterogeneous Visualization for the MOSEL Toolset

MOSEL [12] is a new toolset for the analysis and verification of Monadic Second order Logic designed as a system-level environment supporting automated reasoning in this logic. In its complete realization it will include a flexible set of decision procedures for several theories of the logic (e.g., finite and infinite strings, and trees) complemented by a variety of support components to provide input format translations, visualization, and interfaces to other logics and analysis, verification, and synthesis tools.

The bidirectional link of MOSEL's automata to METAFrame's ffgraphs library [5] allows us to read and generate automata which can be not only displayed, but also may stem from or be fed into other algorithms and tools of

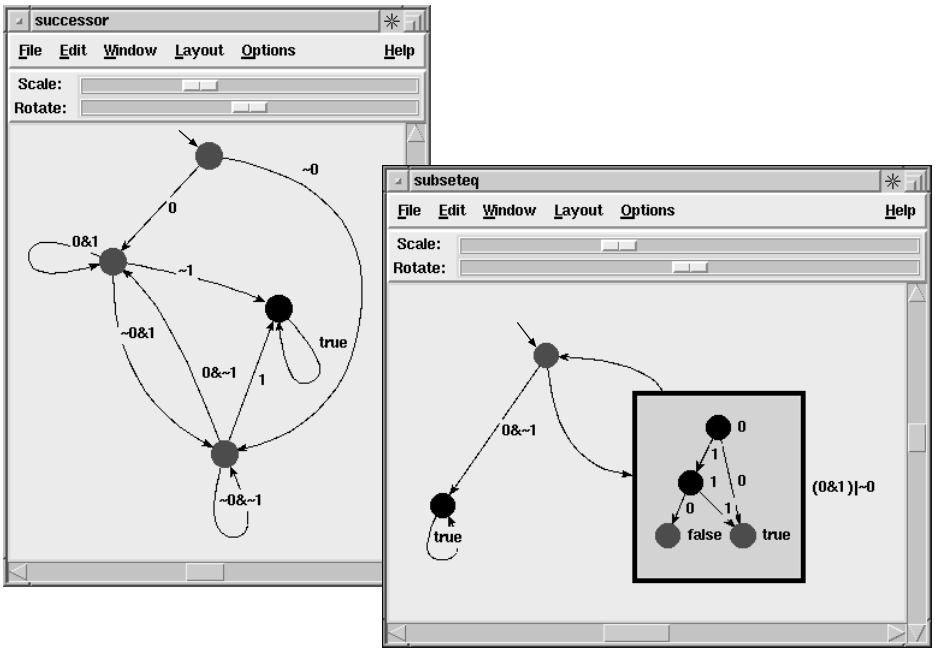

Fig. 6. Automata for Atomic Formulas

the METAFrame environment. This way graphs are not a pure visualization commodity, but an alternative import/export mechanism, crucial for the cooperation between heterogeneous tools. An important consequence is that MOSEL can be used as a model checker too, to check imported automata wrt. properties in the logic.[6]

Powerful graph manipulation features available in METAFrame like the window-in-window browser shown in Fig. 6 (used in the IN context to show macro expansion within a hierarchical service graph) are useful also when dealing with automata. Here, by clicking on an edge of the automaton, the representation of its label as BDD is locally shown in a separate, virtual, window which can be moved, edited, and steered through the menu bars and commands of the outer window. The graphical display of the automata is implemented as a method of the MOSEL automata class, and relies on the label mechanism of the ffgraphs, which allow in particular entire windows to be part of complex labels.

As a comparison, Fig. 5 in [12] shows a similar display in its daVinci realization: there we have to resort to two distinct windows, disjointly representing information at two different abstraction levels (an automaton, whose edges are labelled by formulas, and a multi-root BDD with a root for each of the formulas). Unfortunately, daVinci does not show which of the root nodes corresponds to which formula, and the BDD root labels must be added manually to

[6] This was e.g. not possible for the Mona tool [9], due to its I/O format restrictions.

the automata edges outside daVinci. In contrast, working at the semantic level, METAFrame keeps the correspondence between representations at the different abstraction levels, and displays it either directly, by locally opening a view to a different abstraction level in a virtual window as shown in Fig. 6, or disjointly, by opening an autonomous window. In both cases the labels are automatically displayed appropriately.

3.2 Automatic Synthesis of Design Plans in CAD-METAFrame

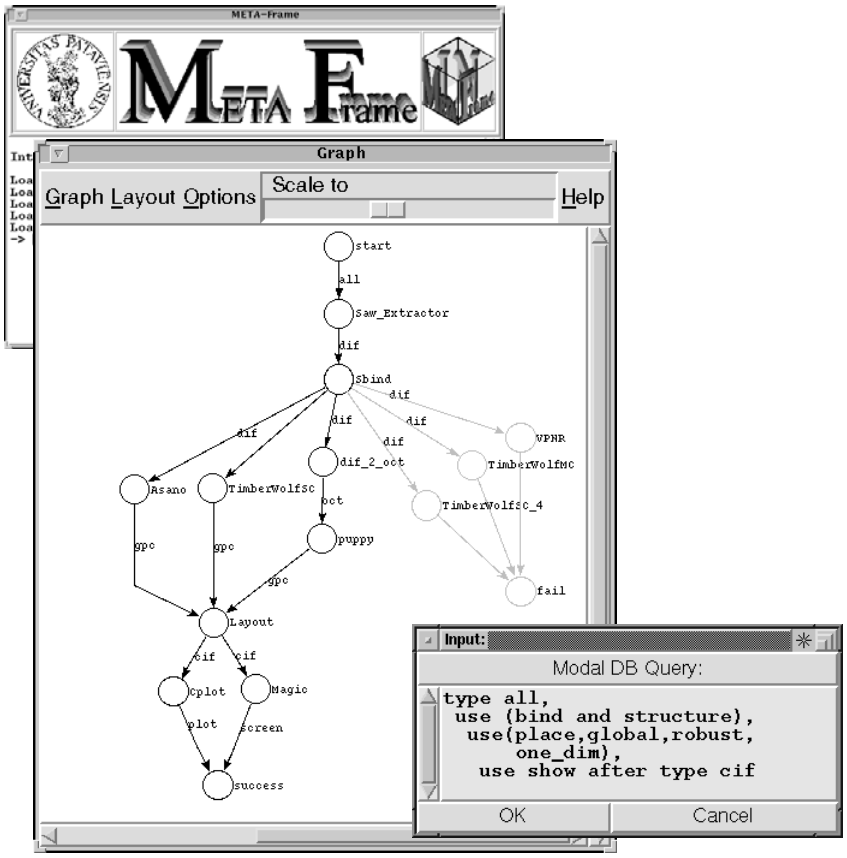

Fig. 7. Synthesis Result and Corresponding Constraints

The CAD-METAFrame [28] application is an environment supporting the flexible integration of CAD tools for VLSI and advanced workflow organization through the application-specific construction of design plans, which avoids the insurgence of unsuccessful design plans at design time. CAD-METAFrame can

be regarded as a global planner where logic specifications are used to direct the automatic construction of design plans according to *global properties* guaranteeing executability, tool compatibility, and other consistency conditions. During a planning phase a graph representing all complete, executable workplans is automatically synthesized on the basis of simple constraint-like specifications and a library of available tools (Fig. 7).

Proposed solutions can be investigated by means of the hypertext browser, which provides information about single tools and modules as well as about the type and module classification schemes (taxonomies) used for specification. Since each path in the graph corresponds to a target program, solutions are directly executable and can be run as soon as the corresponding path is selected. The user's interaction works exactly as shown in the previous section for the animation/simulation of IN services: input required during execution can either be interactively typed in a pop-up window or loaded from a file. Satisfactory plans can be made persistent through compilation into a new module that can be saved in the repository for later reuse.

Fig. 7 shows one of the possible visualizations of specification refinement: here we have strengthened the requirements of a query, and the new result is displayed as projection wrt. the previous solution space. In particular, the complete graph displays the solution space to the first query, whose dark portion still satisfies the stronger query. The grey portion is no longer valid.

3.3 Executable Hierarchical Specifications in SDL

SDL (*Specification and Description Language*) is a graphical specification language widely used in telecommunication applications. Standardized by ITU (International Telecommunication Union) in 1988 [17], it is designed to provide a visual language for the definition of protocols, or communication in distributed systems. Fig. 8 shows our hierarchical METAFrame representation of the Daemon Game (see [31], Chapt. 5). The upper window in the figure shows that the Game block consists of two process types, Monitor and Game. The lower window shows how the Game process type is actually refined by a finite state machine by local expansion of the icon into a whole (inner) window. Once this window is selected, its layout can be steered by means of the scroll bars and of the menus of the outer window. This possibility of local node/edge expansion within the same window is new in this application area: even state-of-the art commercial SDL tools like SDT3.0 [18] show each refinement in separate windows. Our tool supports also this way of expansion, but in our experience nested window expansions are often easier to comprehend.

3.4 Dataflow Analysis

The DFA&OPT-METAFrame toolkit of [11] supports compiler construction by *automatically* generating efficient dataflow analysis algorithms from concise specifications given in a modal logic. A high level programming language allows to

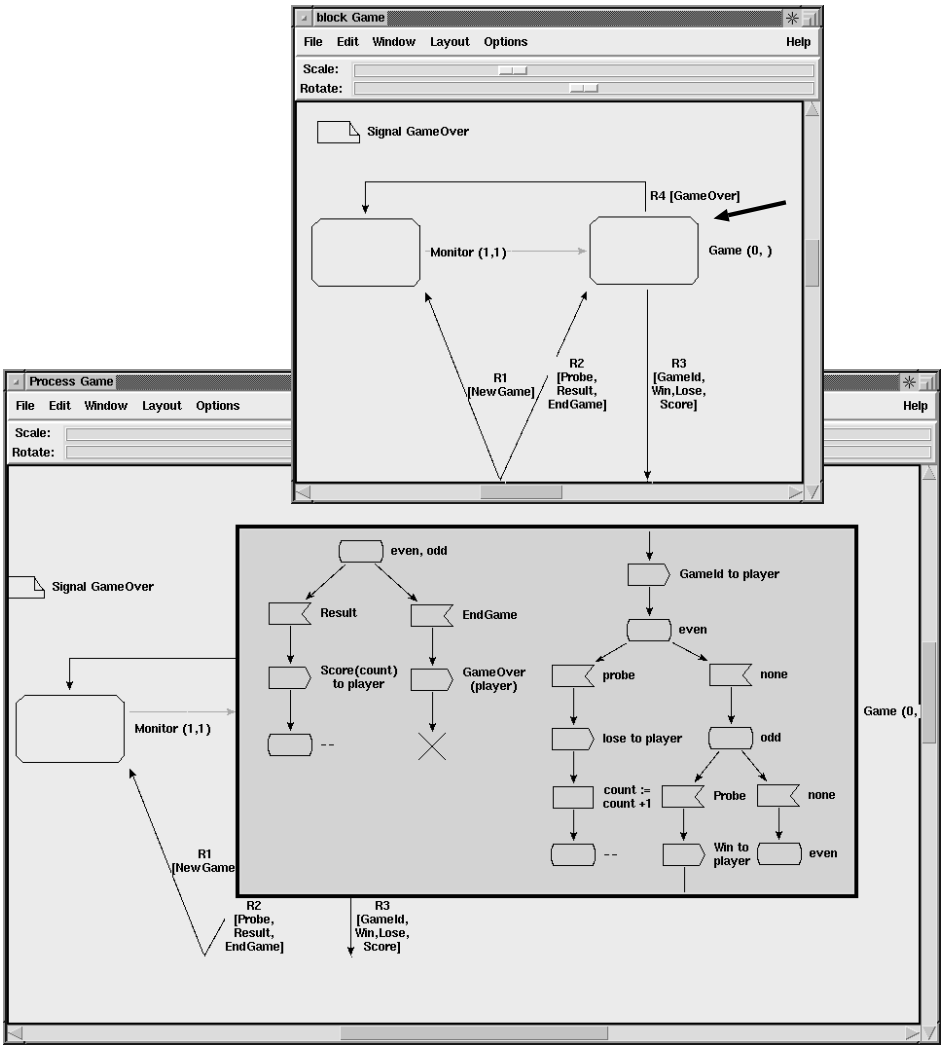

Fig. 8. Snapshots of the Daemon Game Specification in SDL

combine the results of different analyses into *optimizing* program transformations. It serves as the connecting link for combining program analysis and optimization, such that the toolkit supports the complete process of the optimizer construction.

The screen shot of Fig. 9 illustrates the use of the toolkit: the results of the DFA-algorithms, which are automatically generated from the specifications shown in the middle left window, are displayed in the right window, which shows the argument program in an automatically generated and layouted transition system-like representation. The states represent program points, and the tran-

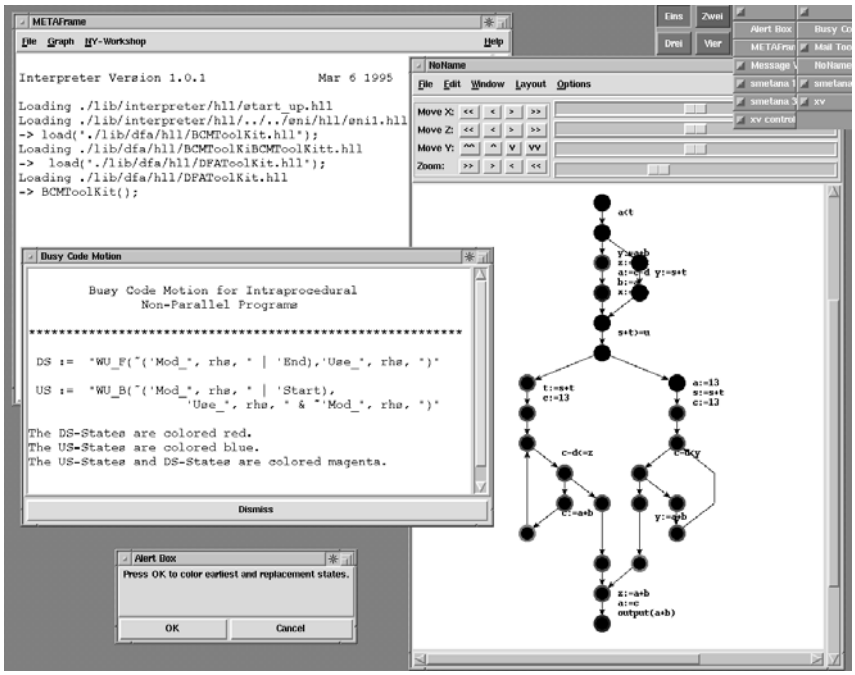

Fig. 9. Dataflow Analysis Scenario

sitions the control flow and the basic blocks of the underlying procedure. Nodes enjoying the property checked by the dataflow analysis are highlighted.

The commands which control the analysis and optimization process are executed by an interpreter running in the upper left window.

4 Executable, Polymorphic Labels

Spanning this variety of application fields, keeping for each its characteristic look and feel is not only a matter of loading the correct set of icons to represent nodes and edges, but it involves dealing with the heterogeneous semantics of each application in an adequate and flexible way. This means, the *interpretation* of the graphical objects is *application* dependent, and, even subtler, it depends on the *abstraction* level of the model represented as graph as well as on the particular *mode* in which a METAFrame application is running.

Being linked to the (operational) meaning of nodes and edges, this means that the labels are the decoupling concept between look and feel, between representation and execution of a model. In the remainder of this section, we re-examine the presented application examples under this point of view.

1. **Application Dependency:** Although the representation of workflows in the CAD design environment looks like an automaton (cf. Fig. 7 and 6),

the execution of each node and transition corresponds to the activation of a complex tool and to the generation of large amount of data stored in one or more files, which is different from just simulating a path in a finite state machine. In the IN application we have an even more complex execution, since each node corresponds to a procedure capsulating a series of operations in a highly distributed system, and each branching condition may involve consulting remote databases in real-time (e.g., to check whether a credit card number is currently valid or blocked).

2. **Abstraction Level Dependency:** Dealing with node/edge refinement, the meaning of a graphical object can vary from one abstraction level to the other. In Fig. 6 we see that at the upper level the automata generated by MOSEL are labelled with a boolean expression, while clicking on any edge reveals in a separate window its canonical representation as a Binary Decision Diagram, as internally stored by MOSEL. Similar observations hold for the treatment of hierarchy in the SDL application: while at the system level an edge represents an asynchronous communication channel, at the block level in Fig. 8 the same edge representation means synchronous communication and at the process level (concerning finite automata modelling) an edge represents a single transition from a source to a target state.
3. **Application Mode:** The meaning of execution on an object may be bound also to configuration parameters, which allow the definition of running modes inside an application of METAFrame. We already saw in Fig. 2 that in the prototyping and demo modes an IN service can be run through red-line tracing, an animation facility based on attaching pseudo-code to each SIB and branch. This way, a service operator or a designer can effectively validate and demonstrate selected aspects (here, the user interaction) of the service under definition without having to actually deploy it in the net. Changing the running mode amounts to associating a different execution code to (some or all) objects, which can be done easily at any time. If the actual running code is available and activated, METAFrame can turn into a complete simulation and testing environment.

5 Related Work

The decision to design our graph component completely anew instead of using one of the available tools was due to our requirement for a flexible treatment of semantic graphs: rather than being interested in graphs and graph properties as such, we wanted to provide a maximum support for the integration of new application scenarios in our graph-based modelling environment. Even now, three years later, there is no other tool satisfying our needs: most available graph- or graphics-based tools and components offer either a purely algorithmic or purely graphical support, and the most similar tool, PEP, which provides a quite liberal semantic graph scenario, is not designed for flexible modification and extension.

Well known in the algorithmic category is the (now also commercially available) LEDA [13] object-oriented graph library. It offers a collection of highly

efficient primitives and algorithms to be used as components in a graphic environment, but contains hardly any visualization support, and no support for binding data or functionality to the graphs.

On the graphical/visualization side, generic tools like daVinci [3], GraphEd[10] or AT&T's commercial GraphViz [8] do not support application-oriented functionality, and DG [4], Lens and GROOVE [19], VCG [16], as well as the commercial Graph Layout Toolkit and Network Layout Assistant by Tom Sawyer Software focus on a specific application area, like display of dataflow graphs, visual support for constructing object-oriented programs or computer networks, without any explicit intent for extension. – Interestingly, even restricted to the purely graphical aspect, a comparison [20] of the VCG, EDGE, GraphViz, daVinci, ffgraphs, Graphlet[7] tools and libraries, favoured ffgraphs. It was only the better and guaranteed support of a commercial tool, which eventually led the authors to choose GraphViz for their purposes.

Closer in their intent to our aims are AUTOGRAPH [15], VTView[8] [30], and the PEP system [6], which support the semantic treatment of concurrent systems, but nevertheless fail the desired flexibility and heterogeneity. Providing editors, compilers, simulators, specific algorithms for checking a given set of abstract properties, and model checkers wrt. elementary temporal logic, for a number of Petri Net-based languages, the PEP system, which essentially arose in parallel with METAFrame, comes closest to our desires. However, even there the admittedly quite general scenario is 'hardwired', and there is no explicit support for extensions. In fact, whereas modifying the scope of the PEP system requires the modification of the underlying program, the same can be achieved dynamically in METAFrame simply by means of a library modification. Thus PEP may be seen as a very powerful possible instance of METAFrame, but not vice versa.

6 Conclusion

We have presented a highly polymorphic tool for the construction, synthesis, structuring, manipulation, investigation, and (symbolic) execution of graphs. The flexibility of this tool, which mainly arises as a consequence of combining complex graph labellings expressing the intended semantics with hierarchy and customized graphical node representations, has been illustrated along a choice of representative applications. In our experience the uniformity of our approach to cover all these different formats is particularly advantageous in large projects, where several formats appear, as it allows a high degree of sharing, while minimizing the required interfaces. We strongly profited from these features in our Telecommunication application sketched in Section 2.

[7] A system based on the LEDA library and GraphEd experience [7].
[8] VTView is the graphical editor of the Concurrency Factory [2].

References

1. M. von der Beeck, B. Steffen, T. Margaria: "*A Formal Requirements Engineering Method and an Environment for Specification, Synthesis, and Verification*", Proc. of SEE '97, 8th IEEE Conference on Software Engineering Environments, Cottbus (Germany) 8-9 April 1997.
2. R. Cleaveland, P. Lewis, S. Smolka, O. Sokolski: "*The Concurrency Factory: A Development Environment for Concurrent Sysytems,*" Proc. CAV'96 - Juli-Aug. 1996, New Brunswick, NJ, USA, LNCS 1102, pp.398-401, Springer Verlag.
3. daVinci: the tool is available via ftp at site `ftp://ftp.uni-bremen.de/pub/graphics/daVinci`
4. Information on DG is available at `http://www.cse.ogi.edu:80/Sparse/dg.html`
5. C. Friedrich: "*The* ffgraph *Library*", Techn. Rep. MIP95-20, Univ. of Passau (D), December 1995, http://www.fmi.uni-passau.de/ friedric/ffgraph/main.shtml
6. B. Grahlmann, E. Best: "*PEP – More than a Petri Net Tool*", Proc. TACAS'96, Passau (D), March '96, LNCS 1055, pp. 397-401, Springer Verlag (see also `http://www.informatik.uni-hildesheim.de/ pep/HomePage.html`).
7. Information on Graphlet is available at `http://www.uni-passau.de/Graphlet/`
8. Information on GraphViz is available at `http://www.research.att.com/sw/tools/graphviz/`.
9. J. Henriksen, J. Jensen, M. Jørgensen N. Klarlund, R. Paige, T. Rauhe, A. Sandholm: "*Mona: Monadic second-order logic in practice,*" Proc. TACAS'95, Århus (DK), May 1995, LNCS 1019, Springer V., pp. 89-110.
10. Michael Himsolt. *GraphEd User Manual.* Universität Passau, 1990, see also `http://www.uni-passau.de/ himsolt/GraphEd/`
11. M. Klein, J. Knoop, D. Koschützki, B. Steffen: "*DFA&OPT-*METAFrame*: A Tool Kit for Program Analysis and Optimization*", Proc. TACAS'96, Passau (D), March '96, LNCS 1055, pp. 422-426, Springer Verlag.
12. P. Kelb, T. Margaria, M. Mendler, C. Gsottberger: "MOSEL*: A Flexible Toolset for Monadic Second-Order Logic,*" Proc. TACAS'97, Univ. of Twente, Enschede (NL), April 1997, this volume.
13. S. Näher: "*LEDA user manual (version 3.0),*" Technical report, Max-Plank-Institut für Informatik, Saarbrücken, 1994,
see also `http://www.mpi-sb.mpg.de/LEDA/leda.html`
14. John K. Ousterhout: "*Tcl and the Tk Toolkit*", Addison–Wesley, April 1994.
15. V. Roy, R. de Simone: "*AUTO and autograph,*" Proc. CAV'90, New Brunswick NJ (USA), June 1990, AMS-DIMACS.
16. Georg Sander: "*Graph layout through the VCG tool,*" Technical Report A03/94, Universität des Saarlandes, Saarbrücken, D, October 1994.
17. "*Specification and Description Language*", Recommendation ITU Z.100, ITU, Geneva.
18. *SDT 3.0*, SDL Design Tool, Telelogic AB, Malmö (S),
`http://www.telelogic.se/`
19. Information on both Lens and GROOVE are available at the SoftViz site, see `http://www.cc.gatech.edu/gvu/softviz/SoftViz.html`
20. S. Spagnolo, A. Parker, K. Chan, M. Mahemoff: "*Graphing Package Report for Group V*", Dept. of CS, Univ. of Melbourne (AUS), Oct. 1996, available at `http://munkora.cs.mu.oz.au/440/html_versions_of_documents/visualiser/v_graphrep/v_graphrep.html`

21. B. Steffen, A. Claßen, M. Klein, J. Knoop. T. Margaria: "*The Fixpoint Analysis Machine*", (*invited paper*) to CONCUR'95, Pittsburgh (USA), August 1995, LNCS 962, Springer Verlag.
22. B. Steffen, B. Freitag, A. Claßen, T. Margaria, U. Zukowski: "*An Approach to Intelligent Software Library Management*", Proc. 4th Int. Conf. on Database Systems for Advanced Applications (DASFAA '95), Nat. Univ. of Singapore, April 10-13, 1995.
23. B. Steffen, T. Margaria, V. Braun, M. Reitenspieß: "*An Environment for the Creation of Intelligent Network Services*", invited contribution to the book "The Advanced Intelligent Network: A Comprehensive Report", Int. Engineering Consortium Chicago (USA), Dec. 1995, pp. 287-300. – also reprinted in the *Annual Review of Communications*, IEC, 1996.
24. B. Steffen, T. Margaria, V. Braun, A. Claßen, H. Wendler: "*Hierarchical Service Definition*", Appears in the *Annual Review of Communications*, IEC - Int. Engineering Consortium, Chicago (USA), 1997.
25. B. Steffen, T. Margaria, A. Claßen, V. Braun: "*Incremental Formalization: a Key to Industrial Success*", In "SOFTWARE: Concepts and Tools", Vol.17, N.2, pp. 78-91, Springer Verlag, July 1996.
26. B. Steffen, T. Margaria, A. Claßen, V. Braun: "*The* METAFRAME *'95 Environment*", Proc. CAV'96 - Juli-Aug. 1996, New Brunswick, NJ, USA, LNCS 1102, pp.450-453, Springer Verlag.
27. B. Steffen, T. Margaria, M. von der Beeck: "*Automatic Synthesis of Linear Process Models from Temporal Constraints: An Incremental Approach*, Proc. AAS'97, ACM/SIGPLAN Int. Workshop on Automated Analysis of Software, Paris (F), 14. Jan. 1997 (affiliated to POPL'97), pp. 127-141.
28. B. Steffen, T. Margaria, A. Claßen: "*Heterogeneous Analysis and Verification for Distributed Systems*", "SOFTWARE: Concepts and Tools", N. 17, pp. 13-25, March 1996, Springer Verlag.
29. K. Sugiyama, S. Tagawa, M. Toda: "*Methods for Visual Understanding of Hierarchical System Structures*", IEEE Transactions on Systems, Man, & Cybernetics, Vol.11, N.2, Feb. 1981, pp.109–125.
30. V. Trehan: "*VTView: A graphical editor for hierarchical networks of finite-state processes,*" Master's thesis, Dept. of Computer Science, North Carolina State University, Dec. 1992.
31. K. Turner: "*Using Formal Description Techniques*", J. Wiley, 1993.