Development of SDL Specifications in Focus

Ketil Stølen

Fakultät für Informatik, Technische Universität München
Postfach 20 24 20, D-80290 München

The objective of this paper is to explain how Focus can be used for the formal development and verification of SDL specifications. We first give a brief introduction to Focus. Then it is explained how Focus and SDL can be combined into one method by connecting them via an intermediate language called F-SDL. F-SDL characterizes a subset of Focus specifications whose elements structurally and semantically match SDL specifications to such a degree that an automatic translation is almost straightforward. Finally we sketch how the proposed approach can be used to develop an SDL specification of a sliding window protocol.

1. INTRODUCTION

Focus [1] is a framework for the formal specification and development of reactive systems. Focus offers specification formalisms and refinement calculi which allow reactive systems to be described and designed in a step-wise, modular style. Recently a restricted version of SDL has been embedded as a target language in Focus. We refer to [2] for a detailed description of this embedding. This paper concentrates on the more pragmatic/methodological issues:

- In what way does the Focus/SDL approach support system development? In particular, should Focus be used top-down or bottom-up?
- What is the relationship between Focus and SDL? In particular, how can system, block and process specifications be expressed in Focus?
- What are the advantages/disadvantages of the proposed approach? Does it scale?

After having given a brief informal introduction to Focus in Section 2, we attempt to answer the questions under the first bullet in Section 3. In Section 4 it is outlined how the proposed approach can be used to develop an SDL specification of a sliding window protocol. At each design level the questions under the second bullet are addressed. Finally, Section 5 gives a short summary and attempts to answer the remaining questions.

2. FOCUS

Depending on the techniques and logical concepts they employ, Focus specifications can be divided into a number of sub-styles:

- Trace Specifications [3]: specifies a system in terms of its allowed traces. This style is particularly useful for the specification of closed systems.

- Relational Specifications [4]: specifies an open system by explicitly characterizing the relationship between the complete communication histories of the external input channels and the complete communication histories of the external output channels. Note that both trace specifications and relational specifications can be written in purely property-oriented style — purely property-oriented in the sense that the specifications list the required properties without giving any algorithm for their realization.

- Assumption/Commitment Specifications [5]: relational specifications which are split into a pair of two requirements (A, C), where A characterizes the assumptions about the environment in which the specified system is supposed to run, and C characterizes how the specified system is committed to behave whenever its environment behaves in accordance with A.

- Algorithmic Specifications: specifies a system in an algorithmic style inspired from functional programming languages. This style is particularly suited in the later, implementation close phases of a development.

- Tabular Specifications: allows algorithmic specifications to be expressed in a tabular form.

These specification styles are all given a common denotational semantics based on streams and stream processing functions. As shown in Section 4.3, specifications can be composed into networks of specifications modeling systems consisting of several components. In this paper only the relational and the algorithmic styles are employed. However, the proposed approach can easily be combined with the other specification techniques mentioned above.

Focus [6], [7] offers powerful refinement principles which allow system specifications to be refined into concrete implementations in a step-wise, modular manner via a number of intermediate specifications. In this paper only the most basic of these principles is needed — namely the principle of behavioral refinement. A specification S_2 is a behavioral refinement of a specification S_1 iff any behavior allowed by S_2 is also a behavior allowed by S_1. Logically this means that the specification S_2 implies S_1.

3. METHODOLOGY

In [2] it is explained how Focus and a restricted version of SDL can be assigned a common denotational semantics based on streams and stream processing functions. Since Focus already has such a semantics it is enough to specify the different SDL constructs in Focus. Based on these Focus specifications of SDL constructs a language called F-SDL is defined. In fact F-SDL characterizes a set of algorithmic Focus specifications whose elements allow an automatic (almost) one-to-one translation into SDL. Note that, as indicated by Figure 1, F-SDL models only a subset of SDL, and F-SDL is only a

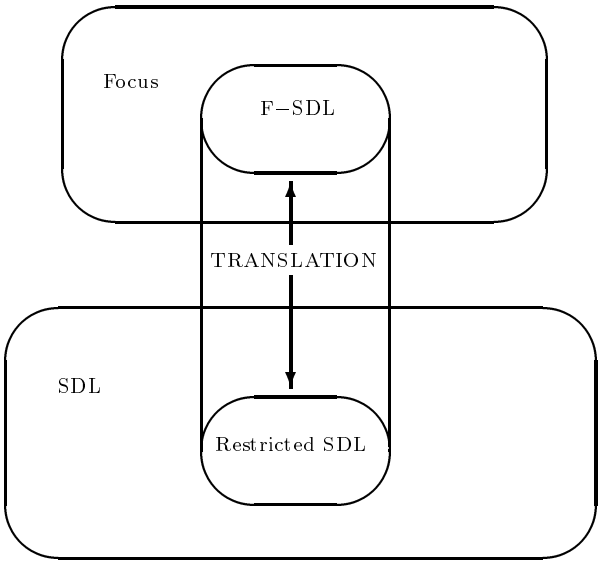

Figure 1. Relationship between Focus and SDL

subset of Focus. Thus, although the considered sub-language is sufficiently expressive to deal with non-trivial applications, many SDL facilities have been ignored. In fact so far, with the exception of SDL-92's statements for explicit nondeterminism, all the constructs considered in [2] are contained in what [8] calls Basic-SDL. Moreover, only some restricted aspects of the SDL facilities for timers and process creation are modeled. For more detailed information on F-SDL, see [2].

Since F-SDL specifications can be translated into SDL, and SDL specifications can be translated into F-SDL, there are two fundamentally different ways to use Focus in connection with system developments based on SDL:

- Bottom-Up Verification: First a specification is formulated in SDL. The SDL specification is automatically translated into F-SDL. The Focus-calculus is then used to prove that this specification satisfies the overall requirements stated in Focus. The overall requirements can be specified in any of the specification styles mentioned above. When this has been verified, an implementation is generated from the SDL specification using standard SDL tools and environments.

  Thus this approach allows already completed SDL specifications to be verified with respect to some requirements stated in Focus — requirements which can be specified in any of the specification styles mentioned earlier.

- Top-Down Verification: First the requirement specification is formulated in Focus using one of the already mentioned styles. The requirement specification is then in a step-wise fashion refined into an F-SDL specification, which is automatically translated into SDL. The resulting SDL specification is transformed into its final implementation using standard SDL tools and environments.

  Thus this approach allows verification during the development.

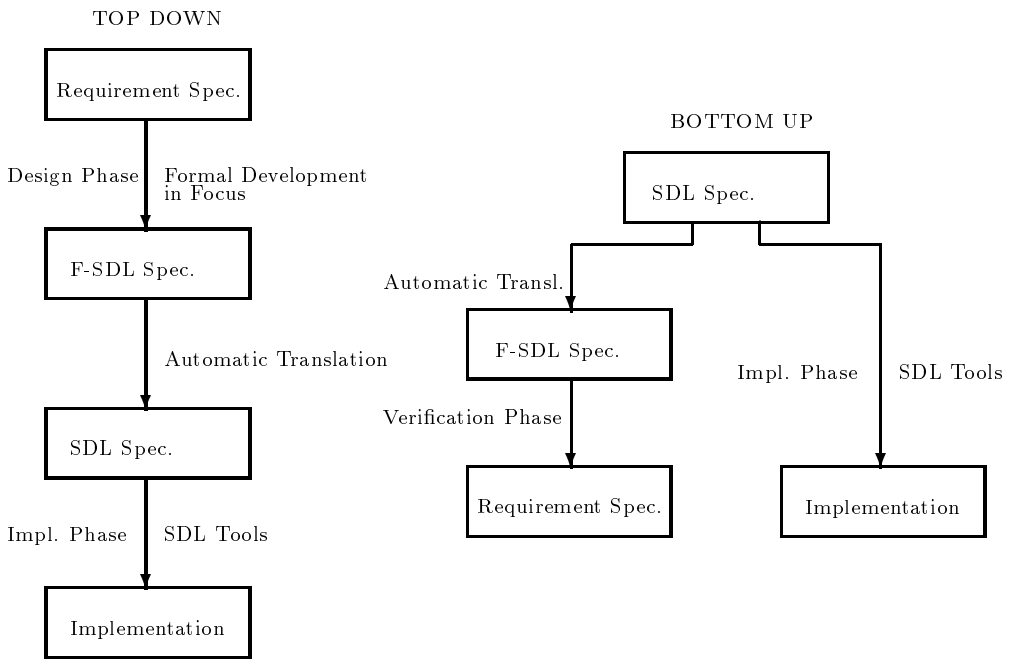

Figure 2. Bottom-Up versus Top-Down

The two alternatives are illustrated graphically in Figure 2. In the early days of program verification more than 25 years ago [9], [10], the emphasize was solely on bottom-up verification, i.e., on the verification of already completed programs. However, this approach turned out to be rather tedious, and during the 1970'ties the interest turned towards top-down verification, i.e., towards verification as a part of the development process. Today, if approaches for automatic verification, which can only be employed in very restricted areas, are ignored, the top-down alternative is dominating. Well-known examples of such methods are [11], [12], [13], [14].

It is important to realize that this change has taken place not only because the proofs become easier when conducted during the development process, but also because top-down verification allows bugs to be discovered and mended during the early phases of a development. Thus one has realized that formal verification is not only an alternative to testing — it can also be used to guide and shorten the development process. This top-down philosophy is the underlying thesis of Focus.

4. SLIDING WINDOW PROTOCOL

The objective of this section is to outline how an SDL specification of a sliding window protocol can be developed in Focus. First an overview of the whole development process is given. Then some of the specification/refinement steps are investigated in more detail.

We have chosen a sliding window protocol for two reasons. Firstly, sliding window protocols are well-known and relatively simple. Thus we do not have to spend much time on explaining how it works. Secondly, a sliding window protocol is nevertheless so complex that a formal development is worthwhile.

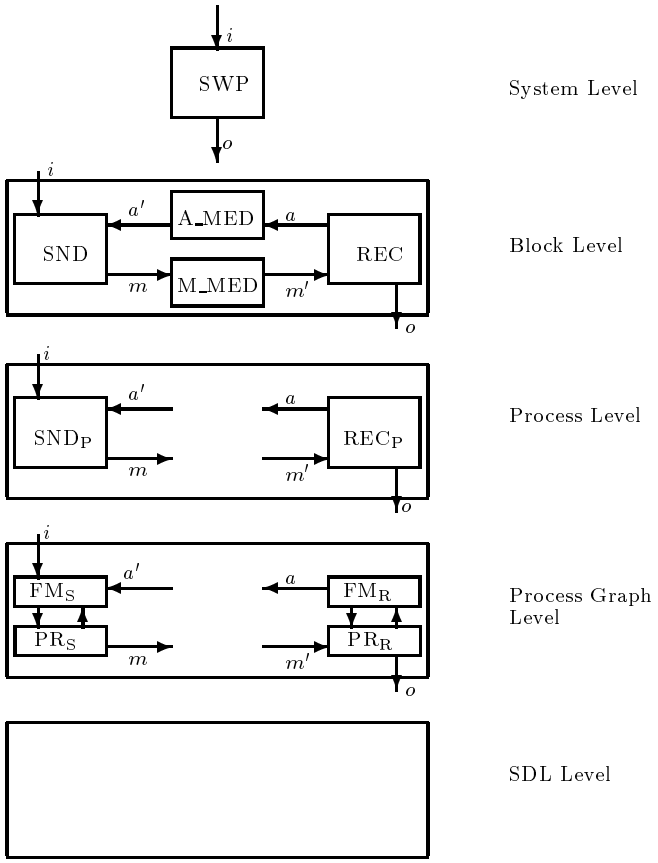

Figure 3. Overview of the Development

4.1. Outline of the Development Process

The sliding window protocol considered here can be summarized as follows. The sender and the receiver communicate via channels that are lossy in the sense that messages may disappear. However, the channels maintain the order of the messages sent, and messages cannot be duplicated. Each message is tagged with a sequence number. The sender is permitted to dispatch several messages with consecutive tags while awaiting their acknowledgments. The messages are said to fall within the sender's window. At the other end, the receiver maintains a receiver's window, which contains messages that have been received but which to this point in time cannot be output because some message with a lower sequence number is still to be received. It is assumed that both windows are of the same size. The receiver repeatedly acknowledges the last message it has output by sending the corresponding sequence number back to the sender.

The development consists of five main steps as indicated by Figure 3. First the protocol's overall input/output behavior is characterized. The resulting specification corresponds to a system specification in SDL. This overall specification is then decomposed into a sender SND, a receiver REC, and two medium components A_MED and M_MED. The mediums model the unreliable behavior of the two channels connecting the sender and the receiver. Both mediums are assumed to be friendly in the sense that if they repeatedly receive the same signal then this signal will eventually get through. The four

component specifications can be thought of as SDL specifications at the block level. The mediums are environment components and therefore not refined any further. The two block specifications SND and REC are both very simple — each has only one process, namely $\mathrm{SND_P}$ and $\mathrm{REC_P}$, respectively. Each of these two process specifications is decomposed into a fair merge component FM and a processing component PR. The fair merge component simulates the unbounded buffer of an SDL process. The processing component characterizes the SDL process graph. As explained in [2], this seems to be the best way to model SDL processes (given our restrictions) in Focus. At the end of this phase the abstract system specification has been refined into a system specification written solely in F-SDL syntax. This specification can then be automatically translated into SDL.

4.2. System Specification

In Focus the communication histories of channels are modeled by streams. A stream is a finite or infinite sequence of actions. Each action models a message sent along the actual channel. The protocol's overall behavior is that of an identity component with one input and one output channel. This is expressed by a relational specification:

$$\textsf{system}\ \mathrm{SWP}(i \in \mathrm{DT}^\omega \rhd o \in \mathrm{DT}^\omega) \equiv o = i\ \textsf{end}.$$

$\mathrm{DT} = \{dt(d) \mid d \in D\}$ is the set of data signals, where D is some nonempty set of data. i and o are streams representing the communication histories of the input and the output channel, respectively. The symbol $\rhd$ is used to distinguish the input streams from the output streams. DT^ω denotes the set of all streams over DT. SWP is the specification's name. The system's external behavior is characterized by the formula to the right of "$\equiv$". It states that the output history is equal to the input history.

There is a fundamental difference between this relational specification and a system specification in SDL. In SWP the system's behavior is stated explicitly independent of its later decomposition. In SDL the behavior of a system specification can only be determined implicitly based on its block and process specifications. Thus this system specification is not in F-SDL. However, via a number of intermediate steps, it will be refined into an F-SDL specification by fixing its internal structure in the style of SDL.

4.3. Decomposing the System into Blocks

The first step is to split the system specification into four component specifications at the block level: a sender, two mediums and a receiver. Such a network of specifications can be characterized as below:

$$\begin{array}{l}
\textsf{system}\ \mathrm{SWP}(i \in \mathrm{DT}^\omega \rhd o \in \mathrm{DT}^\omega) \equiv \\
\quad (m) = \mathrm{SND}(i, a'),\ (a') = \mathrm{A_MED}(a),\ (m') = \mathrm{M_MED}(m),\ (a, o) = \mathrm{REC}(m') \\
\textsf{where} \\
\quad \textsf{block}\ \mathrm{SND}(i \in \mathrm{DT}^\omega, a' \in \mathrm{A}^\omega \rhd m \in \mathrm{MG}^\omega) \equiv R_{\mathrm{SND}}\ \textsf{end}, \\
\quad \textsf{block}\ \mathrm{A_MED}(a \in \mathrm{A}^\omega \rhd a' \in \mathrm{A}^\omega) \equiv R_{\mathrm{A_MED}}\ \textsf{end}, \\
\quad \textsf{block}\ \mathrm{M_MED}(m \in \mathrm{MG}^\omega \rhd m' \in \mathrm{MG}^\omega) \equiv R_{\mathrm{M_MED}}\ \textsf{end}, \\
\quad \textsf{block}\ \mathrm{REC}(m' \in \mathrm{MG}^\omega \rhd a \in \mathrm{A}^\omega, o \in \mathrm{DT}^\omega) \equiv R_{\mathrm{REC}}\ \textsf{end} \\
\textsf{end}
\end{array}$$

$\mathrm{MG} = \{mg(d,n) \mid d \in D \wedge n \in \mathsf{N}\}$ is the set of signals sent by the sender to the receiver. Each such signal consists of a data element and a sequence number. $\mathrm{A} = \{a(n) \mid n \in \mathsf{N}\}$ is the set of acknowledgments. Note the close correspondence between the four equations and the block level network pictured in Figure 3. R_{SND}, $R_{\mathrm{A_MED}}$, $R_{\mathrm{R_MED}}$ and R_{REC} are formulas characterizing the required behavior of the four blocks in the same way as $i = o$ characterizes the behavior of the previous SWP specification. Given such formulas it can be verified whether this new specification refines the earlier more abstract specification or not, i.e., that externally this network behaves as an identity component. For this purpose a simple deduction rule has been formulated [4]. This rule is closely related to the while-rule of Hoare-logic [10]. Because of the space constraints such a proof cannot be given here. However, we would like to mention that in order to carry out this proof the two mediums are required to satisfy a liveness constraint which says that if the same signal is sent infinitely many times from a certain point in time, then it will eventually get through.

4.4. Processing Component and its Translation into SDL

To show how an SDL process graph can be modeled in F-SDL we give the specification of the sender's processing component ($\mathrm{PR_S}$ in Figure 3).

$$\mathrm{PR_S}(b \in (\mathrm{DT} \cup \mathrm{A} \cup \mathrm{MG})^\omega \rhd m \in \mathrm{MG}^\omega, t \in \mathrm{MG}^\omega) \equiv$$

$$\exists x, n \in \mathsf{N} : \exists start \in \mathsf{N} \times \mathsf{N} \to (\mathrm{DT} \cup \mathrm{A} \cup \mathrm{MG})^\omega \to \mathrm{MG}^\omega \times \mathrm{MG}^\omega :$$

$$start(x,n)(b) = (m,t)$$

$$\mathsf{where}\ \forall x, n, r \in \mathsf{N} : p \in \mathrm{D} : in \in (\mathrm{DT} \cup \mathrm{A} \cup \mathrm{MG})^\omega :$$

$$start(x,n)(in) = next(0,1)(in) \tag{1}$$

$$\begin{aligned} &next(x,n)(dt(p) \,\&\, in) = \\ &\quad \mathsf{let}\ x = x+1\ \mathsf{in}\ mg(p,x) \,\&_2 \\ &\qquad \mathsf{if}\ x \le n+w\ \mathsf{then}\ mg(p,x) \,\&_1 next(x,n)(in) \\ &\qquad \mathsf{else}\ next(x,n)(in) \end{aligned} \tag{2}$$

$$\begin{aligned} &next(x,n)(mg(p,r) \,\&\, in) = \\ &\quad \mathsf{if}\ r < n\ \mathsf{then}\ next(x,n)(in)\ \mathsf{else}\ mg(p,r) \,\&_2 \\ &\qquad \mathsf{if}\ r > n+w\ \mathsf{then}\ next(x,n)(in)\ \mathsf{else}\ mg(p,r) \,\&_1 next(x,n)(in) \end{aligned} \tag{3}$$

$$\begin{aligned} &next(x,n)(a(r) \,\&\, in) = \\ &\quad \mathsf{if}\ r > n\ \mathsf{then}\ next(x,r)(in)\ \mathsf{else}\ next(x,n)(in) \end{aligned} \tag{4}$$

The input channel b receives signals from the fair merge component. Signals to the sender are sent along m, and t is used as a feedback channel.

The process graph is modeled by the "functional program" *start* whose recursive definition is given in the where clause. The existentially quantified variables n and x represent

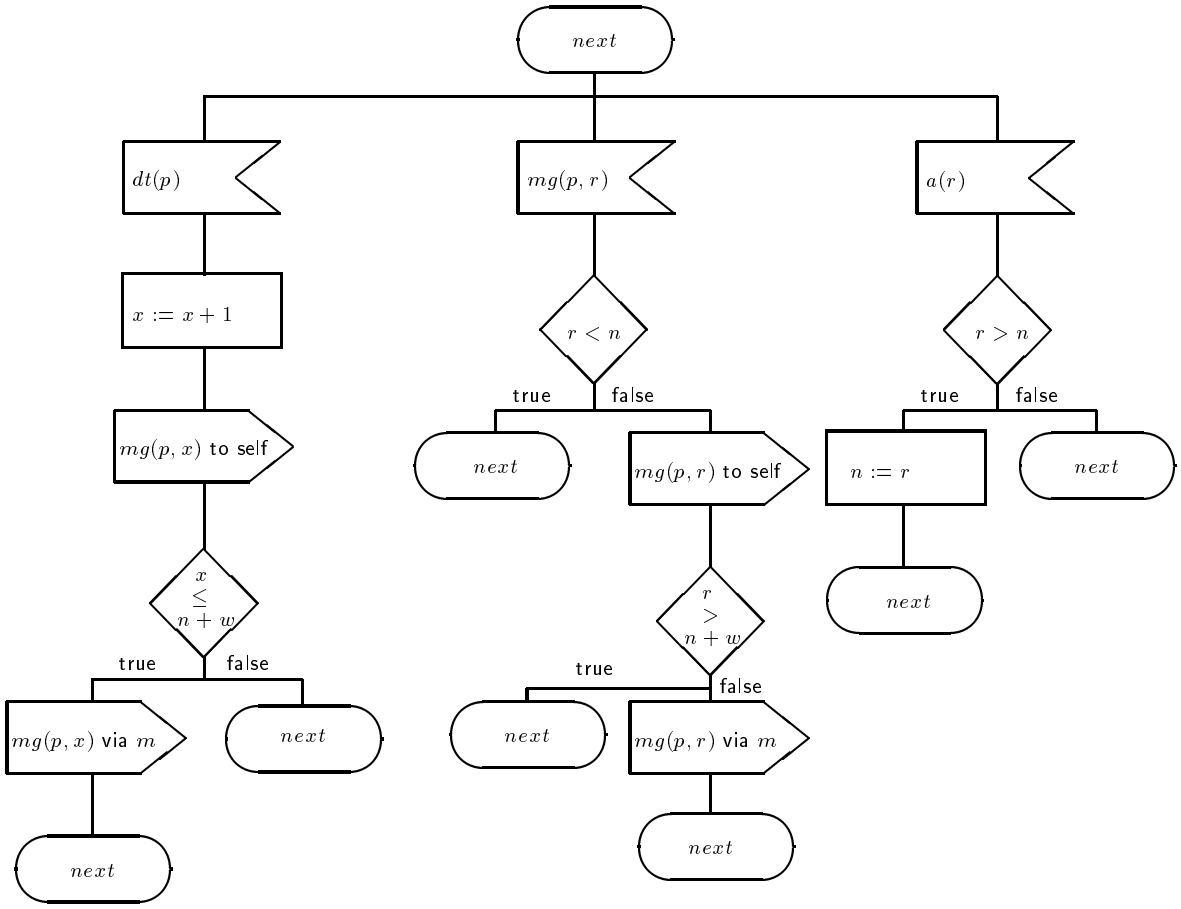

Figure 4. State Transition of the Sender

the internal state. The existential quantifier is used to model that their initial values are not fixed. Thus it can be thought of as a local variable declaration which assigns some arbitrary, type-correct values to the declared variables. However, because of (1) this is of no importance for this example.

w is a constant representing the maximal size of the sender's window.

The where clause has four equations. Each equation models a state-transition in the SDL process. The first equation (1) models the initial transition. Its task is to initialize the two state variables. & is the append operator on streams, i.e., it takes an action and a stream and returns a stream. $\&_j$ is used to to say that its left-hand side argument is output along the j'th output channel. What the equations (2), (3) and (4) represent in SDL is pictured in Figure 4. Note the almost one-to-one correspondence.

5. CONCLUSIONS

This paper has outlined how Focus can be used for top-down development of SDL specifications. First a requirement specification is formulated in Focus. This requirement specification is then refined into an F-SDL specification via a number of intermediate

specifications. The F-SDL specification can then be automatically translated into SDL. If the correctness of each refinement step has been properly verified, the resulting SDL specification is correct with respect to the overall requirement specification stated in Focus.

In some sense Focus and SDL specialize in different areas. Because of its very formal nature, Focus has its strength in the area of formal refinement and verification. SDL, due to its graphical notation and many constructs for the structuring of specifications, is well-suited for the formulation of large and complicated specifications as found in industry. It is therefore very tempting to try to combine Focus and SDL into one framework inheriting the strength of both.

Focus allows the use of formal techniques for the validation, verification and development of SDL specifications. In particular Focus is well-suited for top-down development. In other words for development of the type described in Section 4. As we have outlined a nontrivial SDL specification can be refined from a requirement specification asserting that the overall network behaves as an identity component — a specification whose correctness is obvious!! More complicated protocols and algorithms can of course be developed accordingly.

Another advantage of Focus is the ease with which environment components are handled. The two components A_MED and M_MED were both specified and used to prove that the sender and the receiver communicate in the required manner, i.e., that the network consisting of SND, REC, A_MED and M_MED has the overall behavior of an identity component. As soon as this proof obligation is discharged the remaining refinement of SND and REC can be carried out in isolation (locally). Assumptions about a component's environment can also easily be stated explicitly in the component specifications by using the so-called assumption/commitment format [5].

SDL is a specification language. This means that the readability of SDL specifications often is of crucial importance. For this reason, F-SDL has been designed in such a way that there is a straightforward mapping into SDL. This means that the user has full control of the syntactic structure of the SDL specification he is developing. From the Focus user's point of view the embedding of SDL as a target language in Focus means that he gets access to the many tools and environments already designed for SDL.

Although the considered sub-language is sufficiently expressive to deal with non-trivial applications, many SDL facilities have been ignored. However, it seems to be relatively easy to extend F-SDL to handle a much richer part of SDL, including the full generality of the SDL timer constructs, procedures (not remote call), and services. On the other hand the treatment of the more OO-related facilities of SDL, including the full generality of the constructs for process creation, is difficult if at all possible in the context of Focus.

6. ACKNOWLEDGMENTS

The research reported in [2], on which this paper builds, was conducted as a collaboration between the research groups of Professor Joachim Fischer, Humbolt Universität, Berlin and Professor Manfred Broy, TU München. This cooperation was supported by the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen". I am indebted to several colleagues both in Berlin and Munich,

in particular Eckhardt Holz and Max Fuchs.

REFERENCES

1. M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner and R. Weber, The Design of Distributed Systems — An Introduction to Focus (Revised Version), Report SFB 342/2/92 A, Technische Universität München, (1993).
2. E. Holz and K. Stølen, An Attempt to Embed a Restricted Version of SDL as a Target Language in Focus, Proc. Forte'94, (extended version available as Report SFB 342/11/94 A, Technische Universität München), (1994).
3. R. Weber, Eine Methodik für die formale Anforderungspezifikation verteilter Systeme, Report 342/13/92 A, Technische Universität München, (1992).
4. M. Broy, and K. Stølen, Specification and Refinement of Finite Dataflow Networks — a Relational Approach, Proc. FTRTFT'94, Lecture Notes in Computer Science 863, pages 247-267, (1994).
5. K. Stølen, F. Dederichs and R. Weber, Assumption/Commitment Rules for Networks of Asynchronously Communicating Agents, Report SFB 342/2/93 A, Technische Universität München, (to appear in Formal Aspects of Computing), (1993).
6. M. Broy, Compositional Refinement of Interactive Systems, Report 89, Digital, SRC, Palo Alto, (1992).
7. M. Broy, (Inter-) Action Refinement: The Easy Way, Proc. Program Design Calculi, Summerschool, Marktoberdorf, pages 121-158, Springer, (1993).
8. F. Belina, D. Hogrefe and A. Sarma, SDL with Applications from Protocol Specification, Prentice Hall, (1991).
9. R. W. Floyd, Assigning Meaning to Programs, Proc. Symposium in Applied Mathematics, pages 19-32, (1967).
10. C. A. R. Hoare, An Axiomatic Basis for Computer Programming, Communications of the ACM, 12, pages 576-583, (1969).
11. C. B. Jones, Systematic Software Development Using VDM, Second Edition, Prentice-Hall, (1990).
12. C. Morgan, Programming from Specifications, Prentice-Hall, (1990).
13. K. M. Chandy and J. Misra, Parallel Program Design, A Foundation, Addison-Wesley, (1988).
14. L. Lamport, The Temporal Logic of Actions, Report 79, Digital, SRC, Palo Alto, (1991).