AutoFocus - A Tool for Distributed Systems Specification*

Franz Huber, Bernhard Schätz, Alexander Schmidt, Katharina Spies
Institut für Informatik, Technische Universität München
D-80333 München, Germany
e-mail: (huberf|schaetz|schmiale|spiesk)@informatik.tu-muenchen.de

Abstract. We describe the concept of AutoFocus, a tool for the specification of distributed systems. AutoFocus is based on the formal development method Focus and uses graphical description formalisms embedded into its semantical framework, thus offering well-accepted notations while retaining the ability for exact consistency checks of a system under development. The tool uses a client/server architecture, with a central repository and distributed client applications in a computer network. The paper at hand focuses on the architectural and implementation-related issues of AutoFocus.

1 Introduction

In this paper, we describe AutoFocus, a tool intended to be used for specifying distributed systems on a formal basis. This basis is provided by the semantical framework of the Focus method [BDD+93], [BFG+94]. With Focus being a mathematically founded method for specifying distributed systems, AutoFocus adds graphical description formalisms to this framework that are well-known and well-accepted in industrial systems development, thus offering advantages of both approaches.

The following section 2 briefly outlines the basis of AutoFocus, the graphical description formalisms, whereas section 3, the main part of this paper, focuses on the implementation-related aspects of the tool.

2 Description Techniques

2.1 System Structure Diagrams – SSDs

System structure diagrams describe the static aspects of a distributed system by a network of interconnected components, exchanging data over channels. Each component has a unique identifier and a set of input and output channels

* This work was carried out within the Subproject A6 of the "Sonderforschungsbereich 342" and the Project SysLab, sponsored by the German Research Community (DFG) under the Leibniz program and by Siemens-Nixdorf

attached to it. Each channel is characterized by a channel identifier and a data type describing the set of messages that may be sent on it. Thus system structure diagrams provide both the topological view of a distributed system and the signature (syntactic interface) of each individual component.

Graphically, system structure diagrams are represented as graphs, where rectangular vertices symbolize components and arrow-shaped edges stand for channels.

2.2 Datatype Definitions

The types of the data processed by a distributed system are defined in a textual notation. We use the basic types and data type constructors from the functional programming language Gofer [Jon93] for this purpose. Data defined in this way may be referenced e.g. in SSDs and STDs.

2.3 State Transition Diagrams – STDs

State transition diagrams, which are extended finite automata similar to the concepts introduced in [GKRB], are used to describe the dynamic aspects, i.e. the behaviour, of a distributed system and of its components. Each system component can be associated with an automaton. Each transition has a set of annotations: a pre- and a postcondition, encoded as predicates over the data state of the system, which are satisfied before and after the transition, and a set of input and output patterns describing the messages that are read from or written to the input and output channels of the corresponding component.

Graphically, automata are represented as graphs with labeled ovals as states and arrows as transitions.

2.4 Extended Event Traces – EETs

Extended event traces are used to describe exemplary system runs from a component-based view. We use a notation similar to ITU-standardized message sequence charts (MSCs) with core concepts taken from MSC'96 [Int96].

2.5 The Concept of Hierarchy

A common property shared by all of AutoFocus' graphical description formalisms is the concept of hierarchy. Both structure diagrams and state transition diagrams – which are essentially graphs – as well as extended event traces allow hierarchical refinement: In a structure diagram, a system component may be viewed as a conceptual unit comprising a network of sub-components. In the same way, a state in a state transition diagram can be refined by another STD which details this state. In EETs, we allow so-called "boxes" as an abbreviating notation for parts of system runs specified in other EETs.

3 The AutoFocus Tool

3.1 Architectural Overview

The development of large systems requires a developer team to work on a project simultaneously. Therefore, the AutoFocus tool is designed as a distributed environment with a common repository containing all project-related documents and multiple clients connected over a network. The core of an AutoFocus client is the project browser, displaying all projects, their documents, and versions available in the repository. By selection of a document, the project browser requests the document from the repository and opens a window using the appropriate editor.

3.2 Project and Version Management

In AutoFocus, a project consists of several documents stored in the repository. In order to reuse documents, a document may be referenced by more than one project. The complete version history of every project and document is stored in the repository. In this approach, versions of a project are collections of specific document versions. The user may choose whether a changed document should be stored by default, incrementing the version number, or if it should be saved under an explicitly given version number.

The multiuser environment also requires the coordination of read/write access. The repository provides the usual strategy of one write and multiple read accesses to a document.

3.3 Graphical Editors

Currently, AutoFocus provides three different editors for the graphical description techniques mentioned above. All of these editors use an identical user interface concept with mouse-based user interactions to facilitate fast editing of SSDs, EETs and automata. Copy-and-paste functions simplify the process of drawing the diagrams. Future releases will allow to choose the appearance of the graphic items.

Hierarchical diagrams, which are a core concept of the AutoFocus system model, are fully supported by the editors. For example, a state of an automaton may contain another automaton describing the behaviour of the state. An automaton could also represent the behaviour of a component in a SSD document. By selecting such an element, an appropriate editor window is opened.

Finally, the editors provide PostScript output of the diagrams as exchange format for documentation purposes.

3.4 Implementation

A first version of AutoFocus with support for all the basic graphical notations described above is being implemented in a term project in software engineering

by students at TU München during the 1996 summer semester. The object-oriented Java environment is used as implementation platform. The repository is implemented as a multi-threaded Java application, using the UNIX revision control system RCS for version management and access control. A further replacement of RCS by a real database system is intended and would be simplified by the object-oriented design of AutoFocus. The clients are implemented in Java and are thus executable on any platform supporting the Java runtime environment. Communication between the clients and the repository is based on the TCP/IP networking protocol and the Java socket library.

This first version will be available by August 1996. It represents a starting point for the further development of AutoFocus.

4 Conclusion and Further Activities

The current version is planned as the core of a complete toolset for the development of distributed systems. In the next step, project-wide consistency checks between documents will be implemented (e.g. definedness of data types in SSDs and STDs, compatibility of hierarchical documents). Further steps will include code generation from executable graphical specifications or code frames for more general ones. Finally, special emphasis will be put on the methodological support for the developer; for instance schematic consistent refinement rules for some description formalisms will be supplied. Alternatively, proof obligations for general refinement relations might be generated to be used with the *Isabelle* [Pau94] theorem prover.

References

[BDD+93] Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas Gritzner, and Rainer Weber. The design of distributed systems - an introduction to FOCUS. TUM-I 9202-2, Technische Universität München, 1993.

[BFG+94] Manfred Broy, Max Fuchs, Thomas Gritzner, Bernhard Schätz, Katharina Spies, and Ketil Stølen. Summary of case studies in FOCUS - a design method for distributed systems. TUM-I 9423, Technische Universität München, 1994.

[GKRB] R. Grosu, C. Klein, B. Rumpe, and M. Broy. State Transition Diagrams. Syslab project, internal report, to be published.

[Int96] International Telecommunication Union, Geneva. *Message Sequence Charts*, 1996. ITU-T Recommendation Z.120.

[Jon93] M. P. Jones. *An Introduction to Gofer*, August 1993.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer-Verlag, 1994.

This article was processed using the LaTeX macro package with LLNCS style