The JADE Project - Final Report

Wolfgang Mayer, Markus Stumptner, Dominik Wieland and Franz Wotawa
Inst. f. Informationssysteme (184/2),
Technische Universität Wien,
Favoritenstraße 9,
A-1040 Wien

December 21, 2001

Contents

1 Introduction **2**

2 Objectives of the JADE Project **3**

2.1 Original Project Objectives 3

2.2 Year 3 5

3 Progress Report **6**

3.1 Years 1 and 2 6

3.2 Year 3 16

4 Publications & Presentations **19**

1 Introduction

Project P12344-INF, the JADE project, started in June 1998 with a scheduled runtime of two years. Because of the project's success and a long list of resulting publications, in the summer of 2000 JADE was prolonged by another year and officially ended in September 2001. In general, the project has been very successful with the original research goals for the first two years and the additional objectives for year 3 being achieved or exceeded. This is documented by the following:

- We have produced an implementation of a model-based debugger prototype for the object-oriented language Java. This tool is the first of its kind in debugging and diagnosis research literature.
- The debugger prototype incorporates different formal models for modeling Java programs for debugging. Currently, the debugger incorporates five different models. Three of these are dependency-based models, and two are value-based.
- The work has found wide acceptance in the model-based research community as exhibited by the publications and presentations at the main workshops and conferences in the field. The list of publications resulting from this project includes multiple conference papers as well as papers at IJCAI 99 (the biennial conference generally acknowledged as the foremost in the field), ECAI 2000 (the primary European conference on AI), and KI 2001 (the main German/Austrian conference on AI). Further notable works, which at least in part are consequences of the JADE project, are a master's thesis, a Ph.D. thesis, and a professorial (habilitation) thesis on model-based debugging.
- JADE has followed the projected schedule during all stages in so far as concrete design and implementation problems have made this possible. All objectives set by the application document and the request for prolongation have either been fulfilled or turned out to be better met by alternative means.
- The main practical achievement of JADE is the prototype debugger. Currently, it is fully functional with regard to nearly all aspects of the Java programming language. It comes complete with a graphical user interface and a set of example Java programs.

This document is the final report of the JADE project. It is meant to present the progress made during the last three years in the field of model-based software debugging (MBSD) and highlights the major achievements of the JADE project. It is organized as follows: Section 2 summarizes the main goals of the JADE project. This includes both, the initial objectives for the first two years and the additional goals for the third year. Section 3 deals with the progress made in the course of the project. We show in detail the individual tasks performed and discuss how these activities have to be valued in respect to the given objectives. Finally, Section 4 lists all publications and talks which emerged as a direct consequence of the JADE project.

2 Objectives of the JADE Project

This section presents the goals of the JADE project. First, we list the original goals as specified in the project's application document. Second, we consider the additional objectives set by the request for prolongation for the third year of the project.

2.1 Original Project Objectives

The original JADE objectives as specified by the application document can be summarized as follows:

1. Development of a theory of model-based software debugging.
2. Description of Java semantics in terms of logical models usable for diagnosis.
3. Examination of the use of alternative models for scalable diagnosis and their flexible use in a multi-model diagnosis environment.
4. Development of an intelligent debugging environment for Java programs based on the theoretic results.
5. Collection of example programs and establishment of a set of benchmarks for diagnosis evaluation.

Theoretical Aspects of MBSD This part of the project aimed at the development of a theory of model-based software debugging, building on the results gained in our previous work [1, 19, 18, 17, 16]. It dealt with the examination of different approaches to the modeling of programming language constructs, in particular with regard to object-oriented languages. The results gained entered into the design of the actual diagnosis system for debugging Java code.

Applying MBSD principles to object-oriented (Java) programs The use of MBSD for Java required the description of Java semantics in terms of logical models usable for diagnosis. Depending on the characteristics of individual programs, different models had to be developed for use in different situations. This part of the project also dealt with any required extensions to the general diagnosis principles for dealing with special Java language constructs.

Use of alternative models for diagnosis A topic of high relevance for the practical application of diagnosis algorithms to programs was the choice of the correct representation. This part of the project examined the use of alternative models for diagnosis. For example, large programs may require a more abstract representation to make diagnosis feasible. A goal was also the definition and examination of multi-model diagnosis algorithms which can, e.g., switch between different levels of abstraction for a more efficient and flexible diagnosis process.

Development of an Intelligent Debugging Environment Assistant This was the most strongly implementation-oriented part of the project. It dealt with the development of an intelligent debugging environment for Java programs based on the theoretic results gained from other activities during JADE. This also required (for experimentation and instrumentation purposes) the implementation of a Java interpreter.

Work on this part of the project proceeded in parallel with the more theoretical stages to guarantee that the environment was ready for use when needed for experimenting with different diagnosis models and algorithms.

Accumulating a set of example programs Evaluation and testing of the diagnosis algorithms required the compilation of a set of example programs of different size, complexity, and application domain. This set of programs was used as the base of a set of benchmarks for the (performance- and quality-related) evaluation of different diagnosis approaches.

2.2 Year 3

In the first two years of the JADE project we produced a solid foundation on which further research in the domain of debugging could be based. This included the development of the dependency and value-based models for an important subset of Java, as well as first empirical results which showed improvements compared to traditional debugging. In addition, we identified further research directions which could not be handled during the first two years of the JADE project. One direction dealt with extensions of the models to handle the full Java semantics. This was necessary if the final goal was to provide a debugger that could directly be used in a production environment. Another point was coupling the debugger directly with standardized Java libraries, e.g., the JDK package, and with a Java run-time environment. Again, solving this problem was required for building a debugger that was to be used by programmers in a real-world environment. This latter capability would be a prerequisite for large-scale experimentation with the system, i.e., free use of the debugger prototype by programmers external to the project on actual applications. Basically these were incremental modifications based on the currently implemented prototype.

A third issue turned out to be a highly relevant research subject in its own right during the work on the project. This topic was the issue of multi-model diagnosis. One of the basic issues in model-based diagnosis is the development of good models and even in the subfield of software diagnosis the outcome of modeling can be quite different for different language paradigms, making different languages into research topics of their own. We became aware during the project that the interaction between different models of the same domain was also highly relevant, could significantly influence outcomes and had only seen limited exposure in the model-based literature, although its relevance had been recognized in the past.

3 Progress Report

In this section we present progress made in the course of the JADE project, with the project plans given for comparison purposes. First, we look at the original project plan for the first two years of the project. We then look at the plan for the project's third year and discuss to what extent the additional objectives could be fulfilled.

3.1 Years 1 and 2

Task T1 involved the theoretical investigation of Model-Based Software Debugging (MBSD).

- A compilation of an overview of the existing literature on model-based diagnosis and debugging

  In the course of the project several articles and books from the field of MBD and related research topics have been studied by the project members and used during all phases of the project. The file storing all scientific publications relevant for the project has reached about eight hundred entries by now. A comprehensive overview of existing techniques can be found in Wotawa's professorial thesis "Model-Based Debugging". Short analyses can be found in [11, 23]. An analysis and comparison of model-based diagnosis strategies and traditional approaches to software debugging is given in [26, 27]

- Developing a classification of programming errors, with input from the literature on programmer cognition: types and occurrences of errors, effects of different types of errors etc.

  This activity turned out to be of relatively little importance to the representations, because experimentation showed that the models we currently use worked a priori. Nevertheless, a classification of software faults, a sample fault taxonomy, and fault statistics are included in [23]. The clarification of the terminology of various software anomalies and an appropriate fault classification provided the basis for discussions about the performance of the interactive diagnosis and debugging tool implemented in the course of the JADE *project.*

- Development of a representation for causality in programs

The examination and formal definition of program dependencies was crucial to this research task, which resulted in the development of multiple dependency-based models in use in the current debugger prototype. Specifications of the process and the resulting system descriptions can be found in [8, 9, 6, 7, 14, 15, 23]. The knowledge gained from the dependency-based approaches provided a good basis for the creation of the more detailed value-based models. These models were defined and analyzed theoretically. They were implemented and integrated into the debugging environment in later steps of the project. Although these models significantly extend the pure representation of causality, they heavily rely on the underlying program dependencies. Descriptions of various value-based models and their specification can be found in [10, 11, 12].

- Development of a representation of imperative languages for diagnosis

  This topic provides foundations for the second class of program models, the value-based models. These model classes have been specified and analyzed theoretically. Various types of VBMs have also been implemented and incorporated into the debugger. These models are described in [10, 11, 12].

- Adaptation and development of MBD algorithms for MBSD

  This was a crucial task that was a prerequisite for the implementation of the current prototype debugger. Building on earlier research projects carried out at the Institute of Information Systems, Database and AI Group, data structures and algorithms had to be adapted to suit the needs of the JADE *project. In particular, this included the following activities: (1) adaptation of constraint networks and standard theorem provers to work on all used model classes, i.e., functional dependency models and value-based models, (2) adaptation of theorem provers and constraint propagators to handle multiple test-cases concurrently (see [21, 23]), and (3) modification of existing measurement selection process to optimize the debugging performance of the individual model classes (see [23]). This activity also included the improvement of existing MBD algorithms on a more theoretical basis (see [20, 22, 25]).*

Task T2 dealt with the application of MBSD principles specifically to object-oriented (Java) programs.

- Examination of representation of the semantics of Java constructs in a diagnosis context

  This activity included the modeling of object-oriented language structures in general and the representation of Java-specific features in particular. Tasks within this activity dealt with the modeling of classes and class instances, interfaces, inheritance, polymorphism, aliasing problems, object references, etc... Theoretical descriptions of these modeling strategies can be found in [8, 9, 6, 7, 14, 15, 23, 10, 11, 12].

- Examination of class boundaries as abstraction borders for diagnosis

  This task included an analysis of what parts of a Java system could be excluded from the modeling process or be treated differently in order to speed up the modeling process or make the resulting model better for diagnosis. In case of the dependency-based models this led to the specification of two approaches: (1) the creation of default models, which allow us to model an individual class without knowing its internal structure and (2) an approach which lets the user specify a priori dependencies of certain classes and methods. A description of these techniques can be found in [24, 23]. In case of the value-based models used in the debugging environment it was found that similar approaches are not possible. This leads to the conclusion that the more detailed and computationally expensive value-based models are restricted in their use to small parts of a Java system's source code, whereas the more abstract dependency-based models should be used on a higher level of debugging to narrow the search space of bug candidates and provide a starting point for the application of value-based debugging techniques.

- Modeling of the standard Java class library for MBSD

  The goal of this task was to compile the complete set of Java standard libraries into an internal parse tree representation and to automatically convert all system classes into logical models. The parsing of all standard libraries did not pose serious problems, neither theoretically nor during implementation phases. Moreover, this task proved to be an excellent test for the functionality and correctness of the JADE *parser*

module. However, the modeling of all system classes turned out to be a bit more challenging. The theoretical solution to this problem is to model all system classes off-line and then load the models into the compiler, either before a debugging session or on demand. Currently, this can be done in most cases. Problems only arise in the context of specific nonsequential program structures, which at the moment cannot be handled by the modeling algorithms or do no yet score satisfying results (e.g., exceptions, jump instructions, etc...). If dependency-based models are used, the application of default models (see above) is a possible solution to deal with system classes containing such program structures. However, improving the modeling algorithm and extending it to yet uncovered language structures is still an ongoing task within the JADE *project.*

Task T3 involved the use of alternative models for diagnosis.

- Evaluation of implementation results gained in the initial stages of task **I1**

  This activity included tests with the initial model types, i.e., the DFDM and VBM. A description of the DFDM and results obtained from experiments with the DFDM can be found in [8, 9, 6, 7, 14, 23]. A description of the original VBM together with empirical results can be found in [10, 11, 12]. [15] compares the performances of both model classes and analyzes, which model should be preferred under which circumstances. It also includes empirical results obtained from tests with the current JADE *test suite* [1]*. However, it has shown that the two analyzed models have weaknesses under certain circumstances. Therefore, the creation and testing of additional model types (ETFDM, SFDM, LF-VBM) was added to the agenda of the* JADE *project.*

- Examination of the effects of extending models by explicit representation for functional dependencies in programs.

  Building on the original FDM, i.e., the DFDM, additional dependency-based models have been developed and implemented. This activity mainly focused on the effects of the model extensions on theoretical properties and the diagnosis and debugging performance of the

[1] The current JADE test suite can be downloaded from the project web site (http://www.dbai.tuwien.ac.at/proj/Jade/javaPrograms/index.html)

new model types. The ETFDM can be seen as an extension of the DFDM, which makes use of a concrete execution trace and thus incorporates additional (run-time) information. It can be interpreted as a first step towards a dynamic model (like the VBM) and thus its implementation revealed interesting details valuable for the design and implementation of the various VBMs. The SFDM is a simplified version of either the DFDM or ETFDM. Their properties and advantages are described in [23]. Another task within this activity was the explicit representation of functional dependencies for certain source code structures (e.g., defined by the use of annotations). These dependencies could then be compared with actually computed dependencies in order to determine possible fault locations. A theoretical description of this approach in the context of the VHDL is given in [24]. The application of this technique to the debugging of Java programs seems promising.

- Development of additional fault models and estimates on the occurrences of different fault models.

  Fault models have been introduced and evaluated in the context of value-based models and individual system components with complex behavior, such as loop statements. However, in general the behavior of statements and expressions proved to be too complex to explicitly model all possible fault scenarios. This is why the focus has been shifted from the use of fault models to the creation of different (special-purpose) models, which allow us to diagnose and debug Java methods on different levels of abstraction and in respect to particular language features. Examples of additional model types are the LF-VBM and the SFDM.

Task I1 dealt with the development of an Intelligent Debugging Environment Assistant, basically the knowledge-based component of the JADE Debugger.

- Implementation of the inference and generic MBD class library

  Building on an existing implementation of the basic MBD algorithms and generic MBD classes taken from earlier projects, all extensions of data structures and modifications of algorithms have been finished in early stages of the project. In particular, this includes the derivation of classes used to create system descriptions of Java programs, the adaptation of standard theorem provers and constraint propagators to

the individual needs of the JADE *project, and the modification of measurement selection algorithms to work with all used model types. Due to current results from diagnosis and debugging experiments with the prototype debugger, the measurement selection algorithms have been improved in later stages of the project. Moreover, the adaptation of the used theorem provers and constraint propagators to the concurrent application of multiple test-cases has been an additional activity in year three of the project.*

- Java representation specific classes

  This activity included the creation of classes needed to represent all possible Java language features in an internal parse tree. Works within this activity were done at the very beginning of the project and concluded in year 2, when the initial prototype implementation was extended so that it could handle the full range of Java language features as defined in [2]. GUI classes for the visualization of parts of the parse tree were created in parallel.

- Java compiler classes

  This task included the creation of a Java parser, which transformed a valid Java program into an internal parse tree representation. The first version of the JADE *parser was finished in early stages of the project in order to allow us to start the modeling and debugging activities as soon as possible. Similar to the extension of the parse tree classes, version 2 of the* JADE *parser was completed in year 2 of the project and could handle the full syntax and static semantics of the Java programming language. The parser has been tested on various test programs and the full set of Java system libraries. Version 2 of the parser has been in use for almost two years and works reliably on all input programs. Small modifications to the parser were carried out in year three. The current version also accepts the input of an assertion language (see [13]), which is embedded in Java comments.*

- Testing compiler on example programs from **I2**

  The Java parser and all classes and data structures needed for an internal parse tree representation have been tested on the complete JADE *test suite. Further tests have been carried out with the complete set of Java standard libraries. As mentioned above, the current version of the* JADE *parser works reliably on all possible inputs.*

- MBSD classes

  This activity included the implementation of all data structures and algorithms needed to automatically transform the internal parse tree representation into a model for diagnosis. In particular, the following model types have been implemented:

  ETFDM: *A dependency-based model, which makes use of a concrete execution trace. By using run-time information all dependencies, as they arise during program execution, can unambiguously be determined and added to the model. These models represent a single program run. For a detailed description of the modeling of ETFDMs see [23].*

  DFDM: *A dependency-based model, which only makes use of static (compile-time) information, such as the Java source code and the programming language semantics. Due to a lack of run-time information the resulting models are less detailed and in certain cases represent a higher level of abstraction. However, they model a given Java system in all possible run-time scenarios. For a detailed description of the modeling of DFDMs see [23].*

  SFDM: *Another dependency-based model, which is based on either the ETFDM or the DFDM. It can automatically be derived from either of the two models. It involves a higher level of abstraction by removing the distinction between locations and references. It is easier to handle, but in some cases results in a longer debugging process (see [23]).*

  VBM: *A value-based model, which makes use of concrete evaluation values and the full programming language semantics. It computes possible fault scenarios not only by making use of the underlying program dependencies, but also by actually propagating concrete run-time values from the model's inputs to its outputs and (in some cases) from the model's outputs to its inputs. In general, the VBM yields better results as far as both, diagnosis and debugging, is concerned. As a drawback, its computation is much more expensive and currently restricted to small and medium-size programs. See [11] for a comprehensive description of VBMs.*

  LF-VBM: *A second value-based model, which is based on the unfolded source code for a particular program run. This techniques allows us to detect faults in loop statements provided that the number*

of iterations used during the program unfolding has been correct. This model type makes use of stronger fault models of if statements, which are now the most important source code structures (see [12]).

- Testing and evaluation of diagnosis

  In order to test and evaluate the performance of the JADE *debugging environment we distinguished two different performance measures: the system's diagnosis performance and its debugging performance. The diagnosis performance is given by the percentage of source code, which can be excluded from the search space of potential bug candidates in a single diagnosis step. The debugging performance, on the other hand, determines the number of user interactions needed to exactly locate a given source code fault in a Java method. The latter index can then be compared with the amount of user interaction needed by traditional debugging tools. We have tested the diagnosis performance of the prototype debugger using all created models. In general, the results are quite promising and reveal that large parts of the source code can be eliminated from the debugging scope in a single diagnosis step. However, diagnosis alone cannot exactly locate single faults in most cases and thus further debugging actions are required. Detailed results of conducted tests can be found in [23, 12]. In a second step the debugging performance has been evaluated. Results of these test series can be found in [14, 15, 23].*

- User interface development (implementation+testing)

  This activity included the design and implementation of the main user interface of the JADE *prototype debugging environment. The following modules constitute the main graphical interface between the user and the* JADE *system:*

  - *The main debugger interface triggers the parser, modeling algorithms, diagnosis procedures and interactive debugging process. It has been designed to support the user during all steps of a debugging project in an optimal fashion.*
  - *Inspector windows for parse tree nodes. This allows us to inspect the internal state of the analyzed Java program.*
  - *Visualization tools used to present the different diagnosis models generated from the code.*

- *Dialogs to control the debugging process. These provide means to specify observations and perform hierarchical debugging.*
- *Fault report windows visualize the results of a diagnosis or debugging session.*
- *Code instrumentation dialogs are needed to control the creation of execution traces. These traces are needed during the creation of ETFDMs and during individual debugging sessions.*
- *Database controller interfaces allow the user to remove individual methods from the debugging scope. Default models are used for such methods.*
- *Observation controller interfaces allow us to provide observations a priori and thus simulate the behavior of a potential user. These GUIs are used during test phases.*
- *Assertion handling interfaces enable the user to specify, inspect and use assertions during debugging sessions.*

All described GUIs have been implemented and integrated in the JADE *debugging environment. They have been tested and improved in the course of experiments carried out with the* JADE *test suite and the different model types. An empirical evaluation of the user-friendliness of the used GUIs could further improve the debugging performance of the tool in future versions.*

- Global system test

  After the individual JADE *components and modeling algorithms had been tested individually, the complete* JADE *test suite was tested on the integrated debugging environment. Results of tests of the diagnosis and debugging performance in the context of ETFDMs, DFDMs, and SFDMs are presented in [14, 15, 23]. Results obtained from the use of VBMs and LF-VBMs can be found in [11, 12]. [14, 15] includes empirical results comparing the performance of dependency-based and value-based models.*

- User manuals

  The main user interface windows of the JADE *debugging environment were designed to be as intuitive and self-explanatory as possible. Therefore, the use of the debugging tool is possible without long learning times, even for unexperienced software developers. Nevertheless,*

short descriptions of the interactive debugging environment exist and can be found in various papers (e.g., [6, 7]) and [23]. Furthermore, there exist specifications of various specific aspects of the JADE *project, such as a description of used test-cases (see [4]) and a specification of the used assertion language (see [13]). However, the creation of a complete user manual and the compilation of all existing specifications into a single document is still ongoing work.*

- Implementation of alternate models developed in **T3**.

  This included the design, implementation and testings of the ETFDM, SFDM, and LF-VBM. A short description of these model types is given above.

Task I2 dealt with the compilation of a set of example programs

- Definition of a catalog of examples

  There exists a set of Java methods, which have been used for the evaluation and comparison of the performance of the individual model types. This test suite can be downloaded from the JADE *project site* [2]*. A second set of test programs has been designed especially for tests with FDMs. This test suite is described in [4]. A third set of test programs has been developed especially for tests with VBMs and LF-VBMs. This test suite can also be downloaded from the project web page.*

- Establishing an ordering of implementation tasks according to complexity and relevance of the examples

  Roughly, the actual sequence of implementation tasks can be summarized as follows:

  1. *Implementation of simple programs including different source code structures and syntactical aspects of Java. These programs were used to test the first version of the parser. The second version of the parser, which works with the complete syntax of the Java programming language (see [2]) was tested on various standard libraries, which reduced the amount of implementation tasks during that period.*

[2] http://www.dbai.tuwien.ac.at/proj/Jade/javaPrograms/index.html

2. *Implementation of simple and medium-size programs, which were designed to test the dependency-based modeling algorithms and the resulting FDMs.*
3. *In parallel, we developed a second set of programs designed to test the value-based modeling algorithms and resulting VBMs.*
4. *Finally, the current* JADE *test suite has been compiled. All programs in this test suite were used to evaluate the diagnosis and debugging performance of the* JADE *debugger and compare the results scored by different underlying model types.*

- Implementation of simple programs

  This activity included two tasks: (1) The design and implementation of very simple and small Java programs, which could be used during implementation and debugging to demonstrate specific features of the Java programming language. These programs had to be created on demand depending on concrete implementation problems. Some of these programs were used in our publications in order to present specific properties of the modeling routines (e.g., see[11, 23]). (2) The design and implementation of a simple test suite for the evaluation of the JADE *debugging environment. This has been accomplished as stated above.*

- Implementation of complex example programs

  More complex test suites have been developed to test and improve FDMs and VBMs. These test suites can be found in [3] or be downloaded from the JADE *project web page.*

3.2 Year 3

Multiple models: Using multiple models required both the development of different logical models for programs, and a mechanism for reasoning about these models, with the aim of selecting the most appropriate model in a given situation. The final goal of this part of the project was the development of different models, their implementation using the available debugging framework, formalizing the model selection process, and its implementation. In order to reach this goal the following subtasks were required:

- **M1** Extending the value-based model to form a general model from which sub-models can be easily derived.

  In the course of the third year several VBM types have been implemented and tested. Although the original VBM could not directly be used as the base for all these models, the VBM provided a lot of theoretical know-how and reusable software modules. It could thus be regarded as a general model, which was indispensable during the design and implementation of other VBMs, such as the LF-VBM.

- **M2** Deriving two different models from the general value-based model handling (a) structural bugs and (b) functional bugs.

  Empirical results showed that the distinction into functional and structural bugs proved to be very useful. This distinction not only served as an appropriate fault classification but also as the basis for individual special-purpose models. In our understanding functional bugs are all faults, which result in the computation of incorrect values, but leave the structure of the analyzed program unaltered. As a consequence, the functional dependencies of the incorrect program are the same as those of the program's correct version. The opposite is true for structural bugs, which indeed modify a programs dependency structure. In the course of the JADE *project two value-based models have been developed, which both handle the full range of functional bugs and some classes of structural bugs. A more comprehensive approach to handling structural faults in VHDL programs, which in future could easily be applied to the debugging of Java programs, is given in [26].*

- **M3** Formalize the model selection process. Identify important parameters allowing to select the most probable model.

  Evaluation of the various implemented models has revealed strengths and weaknesses of the individual model types depending on the structure and complexity of the debugged Java program. FDMs have shown to be most effective in combination with longer programs and a rather simple object structure. In particular, the DFDM has problems with programs including a lot of selection statements and loops due to a complete lack of run-time information used during the modeling process. In such cases the use of the ETFDM, which makes use of concrete evaluation traces, yields significantly better results. Even better performance is achieved with VBMs, which use concrete run-time values and

the full semantics of the analyzed program. The use of VBMs generally results in a lower number of bug candidates, but poses significantly higher computational requirements to the system. In case of programs making extensive use of loops, the LF-VBM is the best solution as an underlying program model.

- **M4** Implement the models and model selection using our debugging framework developed during the first 2 years of JADE.

  Different model types have been implemented and tested on various Java programs. Currently, three dependency-based models (ETFDM, DFDM, SFDM) and two value-based models (VBM, LF-VBM) are in use in the JADE *prototype debugging environment. Empirical tests have shown that a sequential use of different models is superior to the parallel use of different model types. This means that a debugging session should first use FDMs, which can be constructed and applied very quickly, but are not as detailed as VBMs. In such a scenario FDMs should be used to limit the initial bug candidate search space or to find faults at statement level. Only then, the much more detailed, but significantly slower VBMs should be used for further debugging actions, e.g., finding faults at expression level. The parallel use of FDMs and VBMs did not turn out to be a good strategy in the context of the current prototype implementation. Such an approach could be followed in the future, if less complex VBMs and a faster implementation exist.*

Empirical analysis: The empirical analysis was done for showing improvements in debugging when using multiple models, comparing the effectiveness of the use of different models for different programs and error types, and for identifying open problems. We actually expected the detection of particular "hard cases" for the current models would lead to the development of new models of programs and hence foster research in this area. In summary, the following subtasks were planned in the empirical analysis category:

- **E1** Creating more test-cases, including the development of large Java applications/programs for this purpose. A detailed test plan had to be worked out. This plan had to include the type of the model, the test cases to be used, and the expected output.

In the third year of JADE *additional test programs have been developed. The new test suite included: (1) Java programs, which were designed to evaluate the performance of VBMs and LF-VBMs, the latter of which were only developed in year three. (2) A general test suite, which was used to compare the diagnosis and debugging performances of the different model types (see [14, 15]). Note that in practice it turned out to be more difficult than anticipated to create large Java programs which did not heavily depend on I/O or user interface code. Therefore, the majority of the existing test programs are medium-size applications, which demonstrate certain aspects of the Java programming language.*

- **E2** Analyzing each model on its own using the developed test-cases. This analysis should show that the models are specialized for a particular kind of bugs and fail to deliver good answers for other bugs.

  This has been accomplished by performing several test session. The results of tests with various FDMs can be found in [23]. An evaluation of the performance of VBMs is presented in [11, 12]. See also [14, 15] for a comparison of the two approaches (FDM vs. VBM).

- **E3** Analyzing the model selection process. This analysis should show how the use of multiple models improves the overall debugging process.

  As already described, it turned out that the parallel use of FDMs and VBMs is not practical with the current prototype implementation. At any given source code level (e.g., modules, methods, statements, or expressions) the decision seems trivial, since VBMs are still too slow to be used in combination with large applications. We therefore propose a two-stage debugging process, which uses FDMs to initially limit the search space of bug candidates. The remaining suspect code can then be further debugged using the more detailed VBMs (see [23] for a discussion about the combination of multiple models).

4 Publications & Presentations

Academic Theses

1. Wolfgang Mayer. *Modellbasierte Diagnose von Java-Programmen, Entwurf und Implementierung eines wertbasierten Modells.* Master's

thesis (Diplomarbeit), TU Wien, Institut für Informationssysteme, Abteilung für Datenbanken und Artificial Intelligence, TU Wien, 2000.

2. Dominik Wieland *Model-Based Debugging of Java Programs Using Dependencies*. PhD thesis, TU Wien, Institut für Informationssysteme, Abteilung für Datenbanken und Artificial Intelligence, TU Wien, 2001.

Journals (refereed)

1. Cristinel Mateis. Quantitative Disjunctive Logic Programming: Semantics and Computation In *AI Communications* (13/4), IOS Press, 2000.

2. Markus Stumptner, and Franz Wotawa. Diagnosing Tree-Structured Systems. *Artificial Intelligence*, 127(1):1–29, 2001.[3]

3. Franz Wotawa. On the relationship between model-based debugging and program slicing. *Artificial Intelligence*, 135(1–2):125–143, 2002.

Conferences and Workshops (refereed)

1. Cristinel Mateis, Markus Stumptner, and Franz Wotawa. A Value-Based Diagnosis Model for Java Programs. In *Proceedings on the Eleventh International Workshop on Principles of Diagnosis*, Morelia, Mexico, June 2000.

2. Cristinel Mateis, Markus Stumptner, Dominik Wieland, and Franz Wotawa. JADE - A Step towards an Intelligent Debugger. In *Proceedings on the Eleventh International Workshop on Principles of Diagnosis*, Morelia, Mexico, June 2000.

3. Cristinel Mateis, Markus Stumptner, Dominik Wieland, and Franz Wotawa. Model-Based Debugging of Java Programs. In *Proceedings of the Fourth International Workshop on Automatic Debugging (AADEBUG-00)*, Munich, Germany, 2000.

[3]Parts of this paper were published in preliminary form in the *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1997, Nagoya, Japan.

4. Cristinel Mateis, Markus Stumptner, and Franz Wotawa. Locating bugs in Java programs – first results of the Java Diagnosis Experiments (Jade) project. In *Proceedings of the International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, New Orleans, 2000. Springer-Verlag.

5. Cristinel Mateis, Markus Stumptner, and Franz Wotawa. Modeling Java programs for Debugging. In *Proceedings of the European Conference on Artificial Intelligence (ECAI 2000)*, Berlin, Germany, 2000.

6. Cristinel Mateis and Markus Stumptner and Dominik Wieland and Franz Wotawa. JADE - AI Support for Debugging Java Programs. In *Proceedings of the 12th International Conference on Tools with Artificial Intelligence*, Canada, Nov. 2000. Also appears in [5].

7. Cristinel Mateis. Extending disjunctive logic programming by T-norms. In *Proceedings of the International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR)*, 1999.

8. Cristinel Mateis, Markus Stumptner, and Franz Wotawa. Debugging of Java programs using a model-based approach. In *Proc. DX'99 Workshop*, Loch Awe, Scotland, 1999.

9. Markus Stumptner and Dominik Wieland and Franz Wotawa. Analysing models for software debugging. In *Proceedings of the Twelfth International Workshop on Principles of Diagnosis*, Sansicario, Italy, 2001.

10. Markus Stumptner and Dominik Wieland and Franz Wotawa. Comparing Two Models for Software Debugging. In *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI)*, Vienna, Austria, 2001.

11. Markus Stumptner and Franz Wotawa. Debugging Functional Programs. In *Proceedings* 16^{th} *International Joint Conf. on Artificial Intelligence (IJCAI 99)*, Stockholm, Sweden, August 1999.

12. Markus Stumptner and Franz Wotawa. Jade – Java diagnosis experiments – status and outlook. In *IJCAI '99 Workshop on Qualitative and Model Based Reasoning for Complex Systems and their Control*, Stockholm, Sweden, 1999.

13. Franz Wotawa. Using Multiple Models for Debugging VHDL Designs. In *Proceedings of the International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expertsystems*, Budapest, Hungary, 2001. Springer-Verlag.

14. Franz Wotawa On the Relationship between Model-based Debugging and Program Mutation. In *Proceedings of the Twelfth International Workshop on Principles of Diagnosis*, Sansicario, Italy, 2001.

Technical Reports

1. Cristinel Mateis. Jade – Java Test Cases. Technical Report DBAI-TR, Technische Universität Wien, Institut für Informationssysteme 184/2, Favoritenstrasse 9, A-1040 Wien, Austria, 1999.

2. Cristinel Mateis. Jade – Subsets of Java considered for Debugging. Technical Report DBAI-TR, Technische Universität Wien, Institut für Informationssysteme 184/2, Favoritenstrasse 9, A-1040 Wien, Austria, 1999.

3. Cristinel Mateis, Markus Stumptner, and Franz Wotawa. A Functional Dependency Model for Java Programs. Technical Report DBAI-TR, Technische Universität Wien, Institut für Informationssysteme 184/2, Favoritenstrasse 9, A-1040 Wien, Austria, 1999.

4. Wolfgang Mayer Evaluation of Value-Based Models for Java Debugging Technical Report DBAI-TR, Technische Universität Wien, Institut für Informationssysteme 184/2, Favoritenstrasse 9, A-1040 Wien, Austria, 2001.

5. Wolfgang Mayer and Dominik Wieland Assertions in model-based debugging Technical Report DBAI-TR, Technische Universität Wien, Institut für Informationssysteme 184/2, Favoritenstrasse 9, A-1040 Wien, Austria, 2001.

Talks

1. Cristinel Mateis: JADE - AI Support for Debugging Java Programs. *Proceedings of the Third International Workshop on Intelligent Software Engineering (WISE).* Limerick, Ireland, 2000.

2. Markus Stumptner: Model-based diagnosis of hardware designs. University of South Australia, Adelaide, April 1999.

3. Markus Stumptner: Debugging of Java Programs Using A Model-Based Approach. *DX99 – 10th Workshop on Principles of Diagnosis.* Loch Awe, Scotland, June 1999.

4. Markus Stumptner: Debugging Functional Programs. *16th International Joint Conference on Artificial Intelligence (IJCAI '99).* Stockholm, Sweden, July 1999.

5. Markus Stumptner: JADE – AI support for debugging Java programs. *2000 IEEE International Conference on Tools with AI.* Vancouver, Canada, November 2000.

6. Markus Stumptner: A value-based model for Java programs. TU Helsinki, Finland, December 2000.

7. Markus Stumptner: JADE – AI support for debugging Java programs. TU Munich, Germany, December 2000.

8. Markus Stumptner: Comparing Two Models for Software Debugging. *24th German - Joint German/Austrian Conference on Artificial Intelligence (KI'01).* Vienna, Austria, September 2001.

9. Dominik Wieland: Model-Based Debugging of Java Programs. *Fourth International Workshop on Automatic Debugging (AADEBUG-00).* Munich, Germany, 2000.

10. Franz Wotawa: Model-based debugging of functional programs. Poster Presentation. *Ninth International Workshop on Principles of Diagnosis (DX-98).* Cape Cod, USA, 1998.

11. Franz Wotawa: MBD Research Activities at Vienna University of Technology. *ECAI-98 Workshop W5 'Model-based systems and qualitative reasoning'.* Brighton, England, 1998.

12. Franz Wotawa: Jade – Java Diagnosis Experiments – Status and Outlook. *IJCAI–99 Workshop on Qualitative and Model Based Reasoning for Complex Systems and their Control.* Stockholm, Sweden, 1999.

13. Franz Wotawa: Detecting and Locating Faults in Hardware Designs. *AAAI 99 Workshop on Intelligent Software Engineering.* Orlando, Florida, 1999.

14. Franz Wotawa: *Debugging synthesizeable VHDL Programs.* Tenth International Workshop on Principles of Diagnosis (DX-99). *Loch Awe, Scotland, 1999*

15. Franz Wotawa: New Directions in Debugging Hardware Designs. International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. *Cairo, Egypt, 1999.*

16. Franz Wotawa: Model-based Diagnosis – Model-based Debugging. Masarik University, Brno, Czech Republic, 2000

17. Franz Wotawa: Model-based Debugging of Hardware Designs. Knowledge Systems Laboratory (KSL) Stanford University, California, USA, 2000.

18. Franz Wotawa: Model-based Debugging of Hardware Designs. Nasa Ames Research Center, California, USA, 2000.

19. Franz Wotawa: Locating Bugs in Java Programs – First Results of the Java Diagnosis Experiments (Jade) Project. International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. *New Orleans, USA, 2000*

20. Franz Wotawa: A Value-Based Diagnosis Model for Java Programs. Eleventh International Workshop on Principles of Diagnosis (DX-00). *Morelia, Mexico, 2000.*

21. Franz Wotawa: AD2L- A Programming Language for Model-Based Systems (Preliminary Report). Poster Presentation. Eleventh International Workshop on Principles of Diagnosis (DX-00). *Morelia, Mexico, 2000.*

22. Franz Wotawa: An environment and language for industrial use of model-based diagnosis. ECAI 2000 Workshop on Knowledge-Based Engineering. *Berlin, Germany, 2000.*

23. Franz Wotawa: Modeling Java Programs for Diagnosis. European Conference on Artificial Intelligence (ECAI). *Berlin, Germany, August 2000.*

24. Franz Wotawa: Analysing models for software debugging. Twelfth International Workshop on Principles of Diagnosis. *Sansicario, Italy, 2001.*

25. *Franz Wotawa: On the Relationship between Model-based Debugging and Program Mutation. Poster Presentation.* Twelfth International Workshop on Principles of Diagnosis. *Sansicario, Italy, 2001.*

26. *Franz Wotawa: Automatic Software Debugging – A Model-based Approach. Habilitation lecture. Technical University Vienna, March 2001.*

27. *Franz Wotawa: Model-based Debugging. Technical University Graz, May 2001.*

28. *Franz Wotawa: Using Multiple Models for Debugging VHDL Designs.* International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expertsystems. *Budapest, Hungary, 2001.*

References

[1] Gerhard Friedrich, Markus Stumptner, and Franz Wotawa. Model-based diagnosis of hardware designs. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, Budapest, August 1996.

[2] James Gosling, Bill Joy, Guy L. Steele, and Gilad Bracha. *The Java Language Specification, Second Edition*. Addison-Wesley, 2000.

[3] Cristinel Mateis. Extending disjunctive logic programming by t-norms. In *Proceedings of the International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR)*, 1999.

[4] Cristinel Mateis. Jade – Java Test Cases. Technical Report DBAI-TR-99/xx, Technische Universität Wien, Institut für Informationssysteme 184/2, Paniglgasse 16, A-1040 Wien, Austria, 1999.

[5] Cristinel Mateis, Markus Stumptner, Dominik Wieland, and Franz Wotawa. AI Support for Debugging Java Programs. In *3rd Workshop on Intelligent SW Eng.*, Limerick, Ireland, June 2000.

[6] Cristinel Mateis, Markus Stumptner, Dominik Wieland, and Franz Wotawa. JADE - A Step towards an Intelligent Debugger. In *Proc. DX'00 Workshop*, Morelia, Mexico, June 2000.

[7] Cristinel Mateis, Markus Stumptner, Dominik Wieland, and Franz Wotawa. Model-Based Debugging of Java Programs. In *Proceedings of the 4th International Workshop on Automated and Algorithmic Debugging, AADEBUG '00*, Munich, Germany, 2000.

[8] Cristinel Mateis, Markus Stumptner, and Franz Wotawa. Debugging of Java programs using a model-based approach. In *Proceedings of the Tenth International Workshop on Principles of Diagnosis*, Loch Awe, Scotland, 1999.

[9] Cristinel Mateis, Markus Stumptner, and Franz Wotawa. A Functional Dependency Model for Java Programs. Technical Report DBAI-TR-99/xx, Technische Universität Wien, Institut für Informationssysteme 184/2, Paniglgasse 16, A-1040 Wien, Austria, 1999.

[10] Cristinel Mateis, Markus Stumptner, and Franz Wotawa. A Value-Based Diagnosis Model for Java Programs. In *Proceedings of the*

Eleventh International Workshop on Principles of Diagnosis, Morelia, Mexico, June 2000.

[11] Wolfgang Mayer. Modellbasierte Diagnose von Java-Programmen, Entwurf und Implementierung eines wertbasierten Modells. Master's thesis, Institut für Informationssysteme, Abteilung für Datenbanken und Artificial Intelligence, TU Wien, 2000. (only available in German).

[12] Wolfgang Mayer. Evaluation of value-based models for java debugging. Technical report, Technische Universität Wien, Institut für Informationssysteme 184/2, Paniglgasse 16, A-1040 Wien, Austria, 2001.

[13] Wolfgang Mayer and Dominik Wieland. Assertions in model-based debugging. Technical report, Technische Universität Wien, Institut für Informationssysteme 184/2, Paniglgasse 16, A-1040 Wien, Austria, 2001.

[14] Markus Stumptner, Dominik Wieland, and Franz Wotawa. Analysing Models for Software Debugging. In *Proceedings of the Twelfth International Workshop on Principles of Diagnosis*, Sansicario, Italy, 2001.

[15] Markus Stumptner, Dominik Wieland, and Franz Wotawa. Comparing Two Models for Software Debugging. In *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI)*, Vienna, Austria, 2001.

[16] Markus Stumptner and Franz Wotawa. A model-based approach to software debugging. In *Proceedings of the Seventh International Workshop on Principles of Diagnosis*, Val Morin, Canada, 1996.

[17] Markus Stumptner and Franz Wotawa. Model-based diagnosis of hardware description languages. In *Proc. CESA'96 IMACS Multiconf.*, Lille, 1996.

[18] Markus Stumptner and Franz Wotawa. Model-based program debugging and repair. In *Proceedings IEA/AIE*, Fukuoka, 1996.

[19] Markus Stumptner and Franz Wotawa. A model-based tool for finding faults in hardware designs. In *Proceedings Artificial Intelligence in Design*, Stanford, 1996.

[20] Markus Stumptner and Franz Wotawa. Diagnosing tree-structured systems. In *Proceedings of the Eighth International Workshop on Princi-*

ples of Diagnosis, Le Mont-Saint-Michel, France, 1997. Also appeared in IJCAI-97.

[21] Markus Stumptner and Franz Wotawa. Debugging Functional Programs. In *Proceedings* 16^{th} *International Joint Conf. on Artificial Intelligence*, pages 1074–1079, Stockholm, Sweden, August 1999.

[22] Markus Stumptner and Franz Wotawa. Diagnosing tree-structured systems. *Artificial Intelligence*, 127(1):1–29, 2001.

[23] Dominik Wieland. *Model-Based Debugging of Java Programs Using Dependencies*. PhD thesis, Vienna University of Technology, Computer Science Department, Institute of Information Systems (184), Database and Artificial Intelligence Group (184/2), November 2001.

[24] Franz Wotawa. Debugging VHDL Designs using Model-Based Reasoning. *Artificial Intelligence in Engineering*, 14(4):331–351, 2000.

[25] Franz Wotawa. A Variant of Reiter's Hitting-Set Algorithm. *Information Processing Letters*, 79(1):45–51, 2000.

[26] Franz Wotawa. On the Relationship between Model-based Debugging and Programm Mutation. In *Proceedings of the Twelfth International Workshop on Principles of Diagnosis*, Sansicario, Italy, 2001.

[27] Franz Wotawa. On the Relationship between Model-Based Debugging and Program Slicing. *Artificial Intelligence*, 135(1–2):124–143, 2002.