

## Summary Project Plan

# Ideals

Idiom Design for Embedded Applications on Large Scale

### Authors:

Hans Onvlee, ASML  
Remco van Engelen, ASML  
Frans Beenker, Embedded Systems Institute

**Document status:** Approved

**Document release date:** June 6, 2003

### Distribution list:

Embedded Systems Institute Ideals project internet site

### Document History

Version	Status/ Date	Description of changes
01	<i>Proposal/ 050503</i>	Creation of the document on basis of approved Ideals project plan
02	<i>Approved / 060603</i>	Approved version.

---

**Table of contents**

<b>IDEALS MANAGEMENT SUMMARY .....</b>	<b>3</b>
<b>1. INTRODUCTION.....</b>	<b>5</b>
1.1. PROJECT OVERVIEW.....	5
1.2. GENERAL APPLICABILITY OF PROJECT RESULTS .....	9
1.3. PROJECT CONSORTIUM .....	10
<b>2. MULTI-PARTNER PROJECT CONTEXT .....</b>	<b>11</b>
2.1. PARTNER'S STRATEGY AND FIT WITH THE IDEALS PROJECT .....	11
2.2. A NEED FOR CO-OPERATION .....	12
<b>3. PROJECT SCOPE.....</b>	<b>14</b>
3.1. SOME ASML IDIOM EXAMPLES.....	14
3.2. INTENDED RESULTS.....	16
<b>4. TECHNOLOGICAL INNOVATION ASPECTS .....</b>	<b>17</b>
4.1. NEW TECHNOLOGY WITH RESPECT TO INTERNATIONAL STATE OF AFFAIRS.....	17
4.2. NEW APPLICATION OF EXISTING TECHNOLOGY .....	17
4.3. IMPROVEMENT OF EXISTING TECHNOLOGY.....	18
4.4. HOW CONSIDERABLE IS THE STEP TAKEN?.....	18
<b>5. PROJECT EXECUTION .....</b>	<b>20</b>
5.1. OVERALL PLAN WITH MAJOR MILESTONES AND DECISION MOMENTS .....	20
5.2. LINES OF ATTENTION: INTRODUCTION.....	20
5.3. LINES OF ATTENTION: DEFINITION AND RELATION TO THE PROJECT OBJECTIVES .....	21
5.4. LINE OF ATTENTION RA: REFACTORING AND ANALYSIS .....	22
5.5. LINE OF ATTENTION AC: ARCHITECTURE LEVEL CONCERNS .....	24
5.6. LINE OF ATTENTION CC: CROSS-CUTTING CONCERNS .....	25
5.7. KNOWLEDGE SHARING AND KNOWLEDGE TRANSFER.....	25

## IDEALS MANAGEMENT SUMMARY

In embedded systems different technologies -- such as electronics, software and mechanics -- come together. The trend is that the software share increases. There are two kinds of embedded software in an embedded system. An embedded system contains software components, just like it contains mechanical or optical components. But a more important role of the embedded software is that of integrating different components and subsystems. This *integrating software* is the software that "glues" the various parts together, and coordinates and controls them.

Decomposing a system into subsystems (or modules) is a well-known technique in software engineering to manage the complexity and evolution of software systems. Such decompositions allow "separations of concerns". They facilitate parallel work, team specialization, localized change, incremental design, use of standard components, and systematic validation. Taking into account that software, and in particular ASML software, is often developed in large teams, the problems of handling dependencies among the subsystems become time-consuming and error-prone.

Unfortunately, certain aspects of software systems are inherently difficult to decompose and isolate. This is in particular the case for embedded software in its role of integrating software. This software runs through all the different subsystems and components, which often leads to strong, mutual dependencies. A typical example of such an *aspect* in the embedded software in ASML wafer scanners is the computer code to support tracing and stepping. The tracing and stepping facilities help the software developer to debug the code, but they are also essential for adequate problem solving when an installed system fails. The aspect of tracing typically affects many different modules. Such aspects that affect different modules are called crosscutting aspects (or crosscutting concerns). It is in particular the presence of many crosscutting concerns that makes the design of embedded software so complex.

The aim of the project is to improve the handling of crosscutting concerns in embedded software systems. This should result in:

- reducing the design effort, compared to manually implementing the aspects,
- reducing the many errors introduced by manual implementation,
- smaller code size, because the aspects are not replicated over the code,
- improving possibilities for maintenance and change, by making changes in existing aspects simpler to realize,
- improving the efficiency of software designers, by allowing them to reason and design at the higher abstraction level of crosscutting concerns, instead of at the level of implementations.

In large embedded software systems, such as the machine control software developed at ASML in Veldhoven, many different crosscutting aspects occur. It is our belief that replacing the current manual techniques by a systematic use of Aspect-Oriented Programming (AOP) improves the design of embedded software, leading to more reliable embedded software, shorter design times, smaller code sizes, and smaller design teams. This is very important for the market of embedded systems, which is characterized by rapid change and innovation.

We use cases from the ASML system to research how the software design techniques related to such crosscutting aspects can be improved, with the following three objectives in mind:

1. Reduce the effort of re-applying the same software pattern in different places.
2. Improve the solutions for the aspects by designing them explicitly from a system abstraction view.
3. Enhance the effectiveness of the AOP technology by creating a software architecture in which aspects can be used in better ways and in a larger number of situations.

A range of partners carries out the research project. These partners are chosen in such a way that the requisite fields of expertise (embedded software, AOP, software engineering) are present. The project is carried out under the project management responsibility of the Embedded Systems Institute. The project participants are co-located at the Embedded Systems Institute and ASML.

The importance of this research, of course, goes beyond the ASML cases. It bears the promise of becoming one of the most important methods for developing embedded software. Special attention is, therefore, given to knowledge dissemination. The results can, for example, be of interest to subcontractors in the embedded software business. Well-isolated components that give rise to clear and manageable crosscutting concerns have the potential of becoming standard components that can be used for many different embedded systems. To this day there are hardly any standard software components for embedded systems. Software subcontractors that develop such components can thus migrate towards becoming component manufacturers.

## 1. INTRODUCTION

### About the Logo:

*Ballroom dances can be used to convey various emotions or images to spectators. Yet, the core of each dance is a number of standardized patterns that are typical for the dance type. The combination of these simple patterns in a complex and pleasing result is the artistic expression mechanism used by dancers. This project aims to achieve something similar in the domain of software design. We hope to leverage the power of simple patterns to allow a software designer more freedom to focus on his "real" job: the art of creating a functional and satisfying software system. (Graphics created by Jopie Schekkerman)*

Complexity is the prime obstacle to the successful incorporation of embedded software in products. If no appropriate measures are taken, design teams for embedded software will continue to expand and their designs will take increasingly longer to complete, if they can be successfully completed at all. Since its embedded systems are among the most complex in the world, ASML faces these software problems head-on.

### 1.1. Project overview

#### 1.1.1. Technical domain description

##### Multidisciplinary System Design and Development

System Engineers think and communicate in terms of different views on the system. Software architects and developers have to think in terms of both architectures and software implementations. There are two reasons why design and clarity can get lost in the transfer from architecture to software.

#### 1. Scattering of system behavior throughout the software architecture

Some system behavior can be captured in a *local* piece of software, e.g. a component, function, or object. Other behavior scatters through the software, missing the concept of locality; see Figure 1.1.

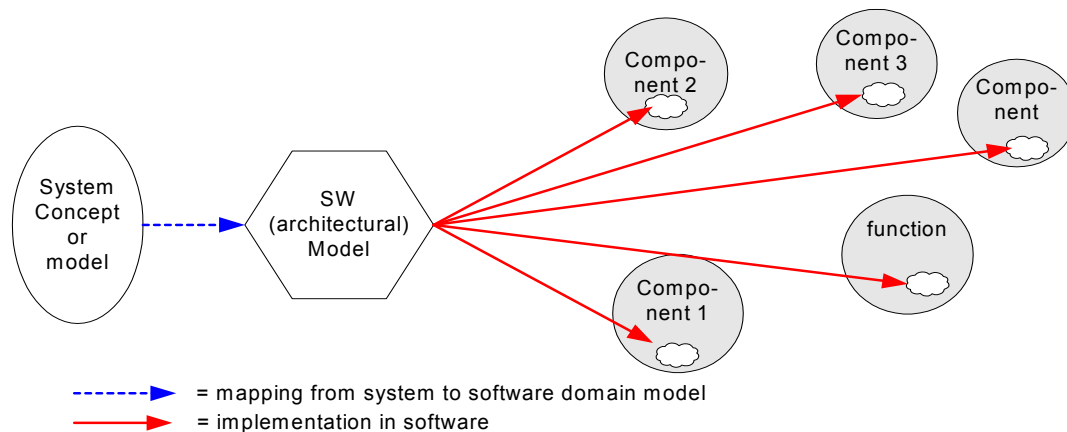


Figure 1.1 Scattering of system behavior throughout the software architecture

#### 2. Complex mapping from system level model to software implementation.

Some simple system-level models result in a large amount of code, which may be hard to implement consistently, or be affected by other system requirements. Schematically, this is shown in Figure 1.2.

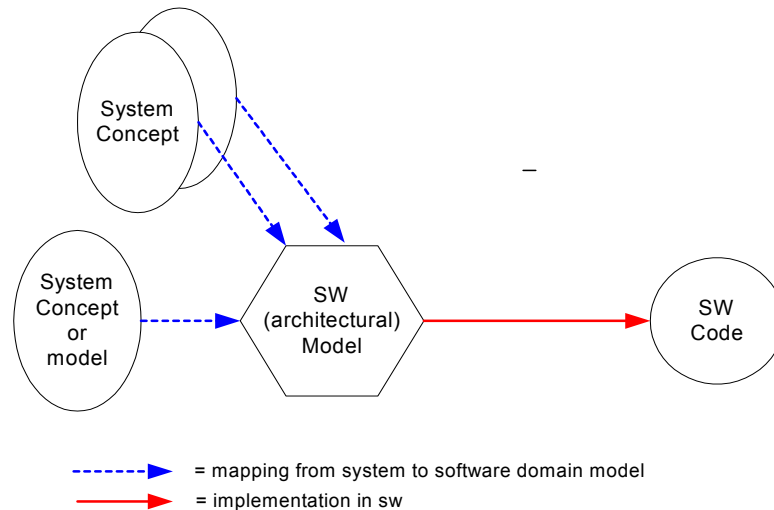


Figure 1.2. Complex mapping from a system level model to software implementation

Typical examples are *tracing*, *error handling*, *scheduling*, and *system initialization*. This results in complex relations between system behavior on one hand, and the implementation on the other. For instance, it is difficult to verify if the software actually implemented the specified behavior, and it is difficult to analyze the impact of system behavior/model modifications on the software architecture and implementation. It is clear that this type of problems have a negative impact on system quality, and therefore on the business of ASML.

What is needed therefore, is a generalization of *locality*, able to cope with the above-mentioned type of complex, scattered system behaviors.

### 3. Software System Complexity

In large systems, developers cannot efficiently understand the details of the entire system. They understand in detail the narrowly scoped portion of the system that they work on. However, without an understanding of the responsibilities and interdependencies of the substructures, this results in sub-optimal systems, and in late surprises that are not detected until the phases of system integration and test. In complex systems, like wafer scanners, the embedded machine control software serves as the glue to put the system functionality together. This implies that many of the above mentioned surprises are reflected as problems in this machine control software and that the (software) system complexity is a problem by itself. A number of metrics can be used to estimate code complexity:

#### □ Code size – ASML example

A rough analysis on the size and complexity of the ASML machine control software has been performed. This analysis shows a linear growth in time of the code size for the different machine platforms.

#### □ Code complexity – ASML example

The code complexity has been estimated by counting the number of code structures like *if*, *for*, *while/do*, *goto* etc, and the number of functions per KLOC. From this analysis, it can be concluded that the code complexity is constant over time and over the different platforms. If nothing changes, the number of developers needed over time will grow at least linear (with the code size), but probably even faster, to compensate for the additional communication overhead in larger teams.

#### □ Non-locality of system behavior – ASML example

Further analysis of the ASML Machine Control software shows, that significant portions of the software are dedicated to implementing *tracing* and *error handling*. A considerable percentage of the code is dedicated to these *non-local* system aspects. This shows, that the non-local system aspects have significant impact on the system software. It also shows that refactoring (i.e., restructuring the design of) such aspects will be a very expensive, time-consuming activity

What is needed are advanced design and implementation concepts that better support the implementation of *local* system behavior.

### 1.1.2. Application domain description

Modern, complex integrated circuits (ICs) consist of many interconnected pattern layers of different materials on top of a silicon base. These patterns are produced on silicon wafers by means of photolithography to create the pattern image. Different techniques, like etching, diffusion and ion-beam implanting are used to create the actual pattern. A wafer scanner performs the lithographic production step. A wafer scanner projects an image onto a thin layer of photosensitive "resist" on the wafer. Key parameters in this step are the minimum structure size (Critical Dimension) and the relative position accuracy (Overlay) of the different layers. Current state of the art Critical Dimension is in the order of 130nm, with roadmaps down to the range of 30 nm. In order to position the wafers with respect to the optical system, and to project an image on the wafer with extreme accuracy *and* high throughput, many different tasks have to be performed as efficiently as possible and with the shortest processing time possible. A typical system contains about 10 high performance computers, and a large number of other programmable devices and processors for dedicated functions. The Machine Control Software is responsible for coordinating these tasks, and consists of several millions of lines of code.

The ever-decreasing Critical Dimension results in increasing cost and complexity of lithographic tools. As a result, the competitiveness of these tools also depends strongly on the productivity, and therefore on the throughput and reliability of operation. The high level of innovation in the semiconductor business results in a high level of innovation in the lithographic tools. Finally, due to the high profits associated with the latest technologies, short time to market is crucial for the profitability.

From this it is clear, that methods and tools that help to keep the software complexity under control are crucial for the business success of ASML.

### 1.1.3. Summary of project objectives and intended results

The objectives of the Ideals project are twofold.

- **Refactoring: Reduction of complexity of embedded machine control software**  
To research and develop means and methods for *refactoring* a huge, existing code base, in order to come to more *local* implementations of complex, distributed system behavior. The results of this research allow reducing the *complexity*, *design efforts*, and *maintenance efforts* of existing software systems.
- **Design: Intuitive mapping of system models onto software architecture and implementation**  
To research and develop means and methods that support mapping of complex system behavior with as much *locality* as possible. The results of this research support *optimal mapping of complex, (new) system behavior onto software*. This includes research of means and methods for describing and designing (electronic) hardware interfaces to support optimal mappings.

#### Idiom

In order to describe this research, the term *Idiom* is introduced. An idiom is a commonly accepted solution for commonly faced problems within a given context, like a problem domain. Examples of commonly used idiom solutions are *Functions*, *Objects*, *Macros*, *Patterns*, and *Templates*. A relatively new idiom solution is *Aspects (oriented programming)*<sup>1</sup>. An *Aspect* is a piece of software code or system functionality *that appears in many places in the code* and is not intrinsically a part of any class.

The objective of this project is to research on *Idiom Systems* and confrontation of the results to real, complex, cases in the ASML machine control context. The research consists of means and methods to design idioms, both bottom-up and top-down, as well as methods and means to generate idiom code in target languages. The following explains the areas of attention in more detail.

#### □ Bottom up Idiom Design

The objective of bottom up idiom design is to search for idiom patterns by using an existing, complex, code base, and to design a concise notation for this idiom in terms of *functions*, *objects*, *macros*, *patterns*, *templates* etc. Which form will be chosen, may differ for the different identified idioms. Topics to investigate cover issues like:

- The nature of the constructs and aspects of the embedded software domain.
- How complete is a pattern captured by its idiom?

---

<sup>1</sup> See website <http://aosd.net>

- How well is the idiom capable to capture different *dialects* and *variations* of the pattern?
- Estimate and measure the impact of pattern introduction on software system complexity reduction (KLOC, control structures etc).
- Estimate and measure the improvement of software clarity.
- In how far does this method support or limit *refactoring* (existing) idiom?

#### Expected results

The research should result in *proofs of concepts* of the following.

- Classification of the typical embedded software idioms.
- Detection tools to identify patterns and close matches (“dialects”).
- Impact analysis on the software, in terms of (reduced) complexity, testability, maintainability etc.
- Means, and methods that support the introduction of refactoring, and suggestions on a *way of working* to facilitate the introduction.

Since this work is based on existing ASML code, giving guidelines on what to select for idiomization, a fast design phase and early results may be expected. Concrete idiom patterns that are suitable for this type of analysis are e.g. *tracing* and *error handling*.

#### □ Idiom Expansion / Generation

Although many idiom expansion methods already exist, there are no ‘COTS’ (Commercial Off The Shelf) techniques for e.g. *Error Handling* or *Tracing*. Some well-known examples are *functional decomposition*, *Macros*, *Templates*, *patterns*, *reflection*, *aspect oriented programming*, and *code generation*. Each programming language implements its own idioms, which implies that the selection of programming languages influences the availability of idioms to the code developer. *Domain-specific languages* can be designed that implement idioms that are relevant for a particular domain. Research is needed on generator theory (syntax, semantics, completeness, tool correctness, and idiom composition). The bottom-up derived idioms (see: *Bottom up Idiom Design* above) are used as cases.

#### Expected results

The research focuses on methods, tools, and processes for expanding idiom models in a software context.

Results are expected in the following areas.

- Expansion tools that can mix different aspects and models.
- Migration tools to perform and validate migration.
- Means to expand and weave idioms into an existing code base.
- Impact analysis on the software, in terms of reduced complexity, testability, maintainability etc.
- Means and methods that support the introduction of idiom expansion, and suggestions on a *way of working* to facilitate the introduction.

Means and methods could be designs for Domain-Specific Languages and well-founded arguments for introducing different programming languages, based on the match between required idioms in embedded software systems, and the idioms implemented in the languages. Currently a lot of software is implemented in C. Experiments to introduce interfaces to languages like Matlab and Python have been initiated but lack such a foundation in a complex analysis of required idioms.

#### □ Top down Idiom Design

The aim of top-down idiom design is the development of models, methods, and tools that support idiom design. The design is based on an analysis of the functional and non-functional requirements, related domains, and underlying system characteristics. The models developed capture the essential features of the system behavior and structure under investigation in a coherent manner. This includes designing a notation that is expressive and unambiguous. This approach has a more conceptual character than the bottom-up design, and a firm foundation in the semantics of the aspects studied. This leads to balanced and expressive idioms. Relevant research topics are the following.

- Identify the relevant classes of system behavior/idiom that are eligible to this approach.
- Formulate the functional and non-functional problems and analyze the appropriate solution techniques.
- Define methods and tools to support the design of expressive models of idioms.

- Research on evolution of system behavior requirements, and their impact on models of idioms.
- Estimate and/or measure the improvement in software quality factors such as expressiveness, evolvability, clarity, and testability.

#### **Expected results**

- Identification and classification of a set of crucial idioms.
- A clear description of the idioms studied, and an explicit understanding of their impact on software.
- Expressive and clear description (models) of syntax and semantics of idioms.
- Description of the necessary operators on idiom models, for example to support composition of evolution of idioms.
- Examples and methods of evolution of software and software models through changes on the system models.
- Insight in necessary restrictions on and the form of system models, software models and/or (electronic) hardware interfaces to allow efficient and clear mappings between them, leading to design goals for such models.
- Techniques to verify the desired quality factors of idiom models and operators, such as correctness and evolvability, and to analyze and manage the impact of idioms on software.
- Examples of application of idioms outside of the software domain.

In case the idiom specification models, techniques and tools prove to be successful, they can and will be further developed by ASML to efficiently introduce new system functionality, or to efficiently redesign / evolve existing functionality.

#### **□ Confrontation of results with ASML cases as proof of concept**

The methods and tools developed in this research project are applied to realistic cases based on current ASML product design problems. The objective of these case studies is to serve as “proof of concept”. In those cases where experiences of our project show improvements compared to the current practices and designs, the results can be fed back into ASML product development, and impact new designs. It is explicitly stated that the results of our study and experiments are in the area of “proof of concept”, and are by no means to the level of product quality. Achieving product feasible results out of the project results will take substantial additional effort within ASML, which is outside the scope of this research project.

#### **Expected results**

Our work is based on realistic system level idioms that are relevant for ASML. The final selection of relevant cases partly depends on the progress made in the different research areas. Several relevant cases are the following.

- Error Handling and Tracing.
- Scheduling, (now implemented in different layers of the software, in multiple components).
- Data Handling (currently under investigation, no system wide implementation available yet).
- Driver Initialization, (a common driver model/concept is missing).
- Hardware Software Interfacing / Hardware Abstraction Layer (how to handle large variation in electronic designs).
- Motion Control, (different performance, cost price requirements, but much commonality).

## **1.2. General applicability of project results**

The results of this project are general and can be used by companies that develop embedded systems. This is because, in a large category of embedded systems, broadly scoped properties such as error handling, performance and real-time behavior severely constrain quality attributes of software systems. These attributes include evolvability, cost-efficiency, reliability, reusability, scalability, dependability and adaptability. By adopting aspect-oriented, refactoring, and software architecture techniques, this project addresses broadly scoped properties by providing means for their systematic identification, separation, representation and composition throughout the software life cycle. In fact, MIT Technology Review labeled aspect-orientation one of the ten most promising technologies in

2000 <sup>2</sup>. Its potential benefits include improved ability to reason about the problem domain and corresponding solution; reduction in application code size, development costs and maintenance time; improved code reuse; architectural and design level reuse by separating non-functional concerns from key business domain logic; improved ability to engineer product-lines; application adaptation in response to context information and better modeling methods across the lifecycle.

### 1.3. Project consortium

In this project, a competent consortium works together; see Table 1.1.

Partner	Department
ASML	Sw engineering, system engineering
Twente University (UT)	Faculty of Computer Science
CWI	Software Engineering (SEN-1)
Eindhoven University of Technology (TU/e)	Faculty of mathematics and computing science
Embedded Systems Institute	

Table 1.1. Project consortium.

<sup>2</sup> [http://trese.cs.utwente.nl/news/mit\\_report.htm](http://trese.cs.utwente.nl/news/mit_report.htm)

## 2. MULTI-PARTNER PROJECT CONTEXT

### 2.1. Partner's strategy and fit with the Ideals project

Per project participant, a description of their strategy is given. We argue why the Ideals project fits with each participant's strategic direction.

#### **ASML**

ASML has a very strong system engineering and architectural focus in all disciplines, and has shown to be able to integrate complex technology from a wide range of disciplines. Software does play a major role, still increasing in importance, but it is not the intention of ASML to become *the* leading company in developing (embedded) software technology. It is the intention to apply state of the art software technologies. If necessary, new (software) technologies will be developed in participation with research partners. The role of main industrial partner, responsible for defining relevant research goals and supplying realistic, complex cases, in a research setting as described in this proposal, is therefore a good fit to this strategy.

#### **Twente University, Department of Computer Science, Trese Group**

One of the strategic research programs of the Software Engineering chair of the University of Twente is Aspect-Oriented Software Development. The chair is among the pioneers in aspect-oriented programming (AOP) with the introduction of *composition filters*. Currently, it is also accepted as one of the three to four recognized mainstream AOP approaches in the AOSD community. A clear indication of this is the fact that the first international conference on aspect-oriented software development has been held in April 2002, in Enschede, The Netherlands.

The Software Engineering Chair has defined the following objectives in the area of AOSD.

1. Managing complexity by identifying the important *crosscutting concerns* (or aspects), and by independently specifying and implementing these concerns.
2. Defining a set of mechanisms for composing concerns into meaningful models.
  - The composition mechanisms should at least satisfy the following two objectives.
    - Support reuse and system evolution through extensibility and adaptability.
    - Support both compile-time and run-time adaptability.

Certain concerns cannot be directly represented by the abstractions of the current programming languages; the implementation of these concerns are generally scattered to the multiple abstractions of the software model. In such a case we say that a concern *crosscuts* the abstractions of the model. *Crosscutting concerns* impede software composition.

Clearly, these objectives of the Software Engineering chair perfectly match the objectives of the Ideals project. The software engineer chair would like to keep its leading position within the AOSD community, and the Ideals project provides an excellent opportunity in this direction.

#### **CWI**

The mission of CWI is

- To perform frontier research in mathematics and computer science
- To transfer new knowledge in these fields to society in general and trade and industry in particular

CWI is funded for 70 percent by NWO, the National Organization for Scientific Research. The remaining 30 percent is obtained through national and international programs and contract research commissioned by industry. An important means for technology transfer is collaboration with industry and the establishment of spin-off companies (on average one per year).

The Ideals project will be conducted within the Software Engineering cluster (SEN), in particular by the Software Renovation Research Group part of the SEN-1 theme. This group aims at developing methods, tools, and techniques that help to make software systems sufficiently flexible. The group has published widely in strong scientific journals and conference proceedings, covering such areas as reverse engineering, program analysis, software evolution, and domain-specific languages. The group's leading position in the field of reengineering is furthermore illustrated by the fact that CWI

delivers program chairs for key conferences in the field, such as the IEEE Working Conference on Reverse Engineering and the European Conference on Software Maintenance and Reengineering.

The Ideals project helps to realize the following strategic goals.

- Widening the scope of current research to embedded systems and aspect oriented programming.
- Studying very large-scale software evolution in an industrial context.
- Setting up or strengthening existing collaborations with academic research groups and software industry.

### **Eindhoven University of Technology, Faculty of mathematics and computing science, SAN group**

The core task of a system architect is to find the balance between different requirements. Lacking objective methods to compare alternatives, these choices are often based on the intuition of the architect. Such intuitive estimates are often not reliable and can lead to high costs for repairing the consequences of such incorrect choices later on in the systems development cycle.

The SAN group of the TUE focuses on the development of methods and techniques for making objective comparisons between different architecture alternatives. These methods and techniques usually require the development of quantitative terms for expressing extra-functional properties of a system (such as testability and maintainability).

The motivation for the TUE to participate in this project lies in the opportunity to do empirical research on state of the art complex systems (providing the opportunity to validate research methods in practice) and building relationships with industrial and academic partners

### ***Embedded Systems Institute***

The Embedded Systems Institute has the ambition to improve and transfer the knowledge and skills of analyzing, specifying, designing, and implementing complex embedded systems. Furthermore, it strives for the position of an expertise center, where professionals from both universities and industry can meet and debate, where lectures, courses and demos are given, where an information center on embedded systems can be found.

The knowledge improvement ambition is realized in co-operative research projects where interested parties together work on methods, processes, tools, etc. for embedded systems. The result of each project is transferred to interested parties for various purposes such as education and application. Results are made available in terms of course material, technical reports, scientific publications, patent licenses, tools, educated and experienced personnel, etc.

To realize its ambition, the Embedded Systems Institute positions itself in between the knowledge institutes and industry, and claims the management responsibility of all processes necessary to make the cooperation work.

The project on idiom design is one of the first projects in a series of embedded systems projects. To a maximum extent, the project organization utilizes a cross-fertilization profit of the co-existence of multiple projects.

## **2.2. A need for co-operation**

The Netherlands has a long and impressive history in embedded systems. Where most of the software industry has disappeared to the US, we still have a strong position in embedded systems. Companies such as Philips Electronics, ASML, Océ, and many others, have a worldwide strong position. As pointed out in many documents, this situation needs careful attention<sup>3 4</sup>. To safeguard and strengthen our position, co-operation is required between all parties involved. This co-operation strategy also fits with the industrial strategy of strategic partnership.

The current project aims at reengineering embedded software architectures by taking advantage of modern developments. To achieve these aims, expertise of different parties must be combined:

- Reengineering expertise is offered by CWI

<sup>3</sup> M. Porter, *Innovation lecture 2001*, Ministry of Economical Affairs

<sup>4</sup> *Concurreren met ICT-competencies*, Beleidsnota, Ministerie van Economische Zaken, Ministerie van OCenW, april 2000.

- Embedded software expertise is offered by ASML and ESI
- Software architecture expertise is offered by TU/e
- Aspect-orientation expertise is offered by UT

All parties participating in this project have developed experience in developing methods that improve the productivity of software engineering, but from different viewpoints. This project is an excellent opportunity to bring this complementary experience and knowledge together and to create synergy, based on concrete, industrial, embedded software problems.

### 3. PROJECT SCOPE

#### 3.1. Some ASML idiom examples

In this section we provide some background information on the software design of the machine control software of ASML, and give some examples of relevant idiom cases within the ASML context. This list of examples is by no means complete, nor need all these examples be used as cases within the project.

The ASML software architecture consists of a large number of parallel processes, distributed over a number of computers. Each process implements a specific subset of services, decomposed along the functional axis. An ASML-developed, generic, communication layer provides transparent access to the services of all processes over all platforms. Client-Server communication can be synchronous and asynchronous, or a call-back mechanism can be used to allow servers to notify clients of events. All user interaction and system logging (error reporting, file management) takes place on the host.

##### Idiom example 1: Error handling

The default error administration mechanism within ASML is to use the return value of a function for the error code. The return of the global constant 'OK' signifies successful completion of a request. Each process defines a series of local constants 'CCXA\_<error\_description>' for all the different errors it can report. If an error is returned, none of the output parameters can be used, and the called function is supposed to have cleaned up all (temporary) resources used until the point where the error was detected (i.e. closed open files, de-allocated memory, handled asynchronous function requests etc.). This model is used for both inter-process requests, as well as local function calls within a process. A dedicated process logs the occurrence of an error into an event log. The logged information states the error code and a string describing the error (and things like date and time of the error, and filename, line number and version of the software that detected the error).

Most functions only do error administration, and if at some step in their execution they detect an error, or receive an error from a called request, they skip the remainder of their steps and return to the caller with an error. Functions can add information to an error (describing context information), by creating a new error and linking it to the 'root' error. In this way a list of error messages can be constructed, starting at the root error, and ending in an error describing which global action failed. The correct error linking and supplying sufficient information with each error is critical for quick problem solving at our customers. The two actions (skipping remainder of a function on error, and linking errors to other errors) are a very typical coding pattern that occurs very frequent in the ASML. To show an example, we use the following simple function:

```
void f(int a, int *b)
{
    g(a);
    h(b);
}
```

Adding the error handling idiom to this function leads to:

```
int f(int a, int *b)
{
    int r = OK;    // current error status parameter

    r = g(a);     // error returned as return value of all functions

    if (r != OK)  // if not OK, link to a new error and state context info
    {
        LOG(CCXA_error, r, ("Calling g failed with value %d", a));
        r = CCXA_error;
    }

    if (r == OK)  // only do next step if no error so far
    {
        r = h(b);
    }

    return r;    // return error status to caller
}
```

The error handling coding pattern can be combined with the functional control flow of a function (e.g. 'if ((r == OK) && some condition)' or 'for (i=0; (r == OK) && (i <= N); i++)') to make it less obtrusive, but it still constitutes a large part of the control flow complexity. Combining can also be incorrect, particularly in if-then-else constructions.

Some function also attempt actual error recovery, e.g. by retrying a hardware action or cleaning action queues and restarting action sequences. These recovery actions are usually very specific for each function and do not follow typical coding patterns.

*Goals of using idiom design and generation tooling:*

- Reduce the time-consuming effort to create the error administration code.
- Improve the quality of the error linking, which is critical for successful error diagnostics, but currently can fail due to coding errors.
- Improve the information available for debugging purposes, by adding more information (e.g. values of parameters and global variables, recent request history) to the logging.
- More use of error recovery through standard recovery patterns.

**Idiom example 2: Tracing**

Tracing is a method to log the software activity in a distributed parallel software system. It writes each software event (typically the start and end of a function) into one global file, allowing software engineers to examine the call sequence and data flow during operation of the system. Each event is time-stamped, and linked to the process in which it occurs, and from which it originates. This allows the reconstruction of call sequences, even in a parallel system. A separate process collects all events from all processes. This process can (run-time) be programmed to filter trace events from specific process and/or specific detail level and log only those to a trace file. In this way a software engineer can examine the operation of only part of the software system.

The value of tracing depends highly on the amount and accuracy of the information traced with each event. For each function start, we would like to trace the value of all input parameters, and for each function end the value of all output parameters and of the return code. Software engineers are free to add additional tracing at crucial points in a function when deemed useful. An example, using the same function as in the previous section, looks like this:

```
int f(int a, int *b)
{
    int r = OK;

    THXA_trace("f", "> (a=%d)", a);

    ... // function body omitted here

    trace("f", "< (b=%d) = %s", *b, r == OK ? "OK" : "ERROR");

    return r;
}
```

*Goals of using idiom design and generation tooling:*

- Reduce the time-consuming effort, leading to more accurate and/or more complete trace events (e.g. more parameters logged).
- Enhance the usefulness of tracing, e.g. by creating means to input trace events into the system for testing purposes, or by adding global variable state to the tracing.

**Idiom example 3: Context Mechanism for Data Collection**

The trace mechanism as described above can be used to log events as they happen in the call sequence of the software. However, the call sequence has a strong 'machine control' orientation, which does not always correspond to the view of the user of data. For examples, process engineers using the system often think in terms of the control structure of the production process, i.e. batches of wafers undergoing a number of steps before being delivered back to the fab.

ASML is at this moment working on a design for the software that can export data in the structure of the process engineers view. This means that machine control actions must be labeled with a context in the process engineers view, enabling the information resulting from a machine control action to be

interpreted correctly. It is very important that the implementation of this design adds minimal extra complexity to the machine control software, since this will affect a large portion of the software.

*Goals of using idiom design and generation tooling:*

- Comprehensive design of an idiom to capture the requirements of the context mechanism concept.
- Support for the implementation of the context mechanism mixed with the existing machine control software.

**Other idiom examples**

- Scheduling
- Calibration methods
- Diagnostics
- Synchronization
- Initialization
- Configuration Management
- Testing
- Factory Integration
- Closed Loop Controlling

### **3.2. Intended results**

#### **3.2.1. Background knowledge**

The background knowledge of each project partner is described in the technical annex of the co-operation agreement.

#### **3.2.2. Global description of foreground knowledge**

The foreground knowledge of the project participants is described in the technical annex of the co-operation agreement.

## 4. TECHNOLOGICAL INNOVATION ASPECTS

### 4.1. New technology with respect to international state of affairs

The current state of the art is work on separation of concerns in general, with a strong focus on object-oriented languages (such as Java). The aspect models that have been studied so far have been inspired by the domain of administrative or application software systems. While staying within the same overall paradigm, this project provides a springboard for innovation in the following categories:

#### **New aspect models**

The domain of large scale, embedded software systems has unique aspects to investigate. Creating models for these typical aspects requires new ways of modeling techniques and tools. Large-scale application of such models requires better composition techniques, and also a higher level of verification tools to support their introduction.

#### **Procedural Language as context**

New techniques for generating code from aspects models for *procedural*, non-Object Oriented programming (OO) languages are developed. The main programming language within ASML is “C”, which falls into this category. Some attempts in this direction are undertaken, for instance at the University of Columbia, Canada <sup>5</sup>. Experiments are being undertaken where Aspect Oriented Techniques developed in the context of Object Oriented languages (Java, AspectJ <sup>6</sup>), are applied as AspectC on the “C” language <sup>7</sup>. These results, however, are still immature, using manual aspect weaving technologies, and limited domains.

#### **Identification and mining**

New techniques for automatic identification and mining of aspects and/or idioms in procedural languages are developed <sup>8</sup>. Some initial work on mining aspects in Java code is being conducted, but this is still in the very early stages.

#### **Architectural design**

The project aims to develop techniques that support making of architectural decisions through objective means. This implies that models are developed that predict, typically extra-functional, system properties based on architecture level system design.

#### **Aspect-based architectural description**

‘Aspect-based’ architectural description, i.e. some means of modelling a system as a whole including the places where ‘aspects’ fit in, is new.

#### **Architecture and migration**

Assessing the impact of idiom on an architectural level, and devising methods to optimize their effectiveness is a new domain for the architecture evaluation research. The tools developed in this domain so far must be extended, and applied to a large-scale system. Furthermore, automated methods are developed for the migration of an existing, large, software system to a new architecture.

### 4.2. New application of existing technology

#### **New application domain: Large Embedded Systems**

The application domain under investigation concerns large embedded software systems, which implies that embedded systems specific issues must be addressed. Examples are concerns related to hardware driver (architecture), real time behavior, and limited system resources.

---

<sup>5</sup> <http://www.cs.ubc.ca/labs/spl/projects/aspectc.html>

<sup>6</sup> <http://aspectj.org/servlets/AJSite>

<sup>7</sup> Y. Coady, G. Kiczales, M. Feeley, and G. Smolyn, *Using AspectC to Improve the Modularity of Path-Specific Customization in Operating System Code*, Proceedings of the 9<sup>th</sup> ACM SIGSOFT Int. Symposium on the FSE, 2001.

<sup>8</sup> J. Hannemann, G. Kiczales, *Overcoming the Prevalent Decomposition in Legacy Code*, Proceedings of the ICSE workshop on Advanced Separation of Concerns, May 2001, Toronto.

**Software Exploration**

A common approach to support reverse engineering and software maintenance tasks is to offer tools that assist an engineer in software exploration. This typically involves the generation of high-level system representations from source code, as well as appropriate means for visualization and navigation.

Software exploration for the purpose of aspect identification is a completely new application of software exploration methods and tools and poses interesting new challenges and requirements<sup>9</sup>.

**Combination of methods**

The ambition of this project is to apply different methods in combination, in particular of domain-specific languages, aspect-orientation techniques, and reverse engineering.

**4.3. Improvement of existing technology****Software architecture**

Software *Architecture* is a relatively young topic. It is our ambition to study the interaction between *architecture* and modeling, or refactoring aspects and concerns. This interaction holds both ways: modifying the architecture influences the implementation of concepts and idioms, and if software is refactored, this has an impact on the architecture.

**Identification and mining**

It may be the case that interesting results in aspect mining can be achieved by using existing work on design pattern recognition and program plan recognition as a starting point. This work, however, is mostly in a research phase, and its application in industry has been limited. Further analysis of the methods is foreseen in the plan, in order to determine how they can be used to strengthen the Ideals research.

**Program Transformation**

To support automated aspect migration, existing program transformation technology must be extended in order to deal with complicating factors such as code elimination, preprocessor directives, and support for retaining the correspondence between target code and aspect specification. An overview of existing approaches to reengineering and program transformation can be obtained from the Reengineering Wiki<sup>10</sup>.

**Aspect Oriented Software Development**

Aspect orientation was originally conceived and worked out for object-oriented software systems, with a strong focus on programming language issues. The OO domain inspired the aspects that have been catalogued and investigated. This project provides an excellent opportunity to extend this technology to the new domain of embedded systems and test our concepts on the large scale involved here. We expect that the resulting insights and the new domain-specific aspects increase our general knowledge and understanding of aspect orientation as well as our library of available models and techniques for the specification, implementation, and composition of aspects.

**4.4. How considerable is the step taken?**

Considerable innovations are required in the following areas:

1. The *new domain* (embedded systems) results in new concerns, like constraints in time and system resources.
2. Although work is being done in the area of procedural languages, the results are still premature. (For example: AspectC).

---

<sup>9</sup> Moonen, *Exploring software system*, PhD thesis CWI and University of Amsterdam, 2002.

<sup>10</sup> A. van Deursen, E. Visser, *The Reengineering Wiki. Proceedings 6th European Conference on Software Maintenance and Reengineering (CSMR 2002)*, pp 217-220, IEEE Computer Society, 2002.

3. Aspect identification and mining involves new forms of sophisticated program analysis. Initial results in design pattern recognition offer a potential starting point: a major gap to aspect recognition remains to be bridged.
4. Current (research) experiments in this area based on 10 – 100 K lines of code, consider only a few aspects. It is expected that 10 – 20 relevant aspects will be identified in the ASML code base..
5. The role of architecture in software increases, if the size of the project / product / code increases. Also, software architecture as research topic is relatively new. Given the ambitions of the project, and the size of the ASML embedded software, the role of architecture is very important. In projects of this size architecture must be made explicit; while in smaller and limited cases, intuition will suffice.

## 5. PROJECT EXECUTION

### 5.1. Overall plan with major milestones and decision moments

#### Phases and reviews

The project master planning contains several phases, with major review milestones in between. The major review meetings have the purpose of discussing the status of the project, including the achievements obtained and budget spent. Further, insight is given in next phases, budgets required, and end-date of the project. In particular, this meeting decides and agrees unanimously upon updates of the technical annex of the co-operation agreement. Such updated versions of the technical annex specify ownership and usage of the project results of the next project phase. The result of a review meeting is one of the following.

- Continue the project.
- Continue the project with modified budget
- Continue the project with modified requirements
- Close the project completely.

#### Increments

Three consecutive increments are performed, each closed with a review. Each increment has the phases of exploration per Line of Attention, application of the results on an ASML case, and consolidation of all the results for further use. During the application and consolidation phases, also the detailed definition of the following increment is provided and the content of the technical annex is updated (and decided upon during the review). In the current version of the project plan, the increments are detailed as far as possible. It is anticipated that, due to the complex nature of this project, the detailed definition of the increments needs a regular update.

#### Coupling to ASML development projects

The ASML industrial application cases are taken either from an existing product or, in the course of the project, from actual development projects. In this manner, it is ensured that the project is fed with real-life complex industrial examples. It also provides a possibility for early knowledge transfer from the project to ASML and other interested parties. Again, it is explicitly stated that this project will not perform pre-competitive development for ASML. Although, the Ideals project is of considerable size for a scientific research project, it is not at all comparable in size and complexity to actual ASML development and engineering projects. The intermediate knowledge transfer from the Ideals project to ASML is concerns knowledge, insights, and structuring mechanisms.

Per Line of Attention, the activities are described in the following sections. The detailed relations between the contributions of the various partners will be topic of discussion of the first project phase. It is also in this phase, that the first technical annex with a detailed description of the foreground information will become available.

### 5.2. Lines of Attention: Introduction

In recent years, the phenomenon of "crosscutting concerns" has become an active area of research in software engineering, most notably in such topics as adaptive, subject-oriented, aspect-oriented, and generative programming, as well as in the use of composition filters. We use the term "aspect" to cover such a crosscutting concern.

The Ideals project aims at providing explicit aspect support for the purpose of refactoring (improving the current system) and design (when adding new functionality).

Throughout the project, the C code base of ASML is available for the following research purposes:

1. Identification of issues related to crosscutting concerns as occurring in industrial embedded software systems.
2. Validation of research results.

A crosscutting concern can arise for several reasons, which are described below.

❑ **Design related concerns: Refactoring and Analysis**

This category consists of crosscutting concerns that arise due to bad design. In those cases, the design can be *analyzed* and *refactored* in order to turn the crosscutting concern into a well-isolated module.

When a language without explicit aspects is used (and most languages to date have no explicit aspect support) coding conventions need to be used to circumvent the limitations of the programming language. In such cases, a typical "idiom" is used to code a particular aspect. Here, an idiom refers to a commonly accepted solution for commonly faced problems in a given context. In the Ideals project, we look for methods and techniques that make it possible to turn such informal idioms into explicit aspects. The purpose of such explicit aspects is a significant simplification of the C code base: by compiling the aspect model and by weaving the generated code through the C system, the complicating crosscutting behavior can be isolated.

❑ **System Decomposition related concerns: Architecture-level Concerns**

A second category includes those aspects that result from a given system decomposition or *architecture*. An example is updating a window whenever one of the elements of the window (represented by various objects) changes. It may be possible to avoid such concerns by choosing a different decomposition. In many cases, however, there will be no single decomposition that fits well with two conflicting concerns.

In this project we look into alternative architectures and their impact on the extent of the remaining crosscutting concerns, and we look into the relation between architectures and the design complexity of the concerns. The purpose is to find architectures that lead to a manageable set of crosscutting concerns.

❑ **Inherent cross cutting concerns: Cross-cutting Concerns Modeling**

Finally, an important category comprises the inherent crosscutting concerns. These are concerns that, whatever modularization is chosen, remain hard to isolate. Typical examples include tracing, error handling, and synchronization. These concerns are already visible at a systems abstraction level. The project aims to handle these concerns through an explicit modeling process, from a systems abstraction level to a software level, isolating the mapping of different concerns from each other. In the second part, we investigate how such different concerns can be composed at the software level.

The purpose is to develop clear designs of this type of concerns that are well maintainable and communicable.

### 5.3. Lines of Attention: definition and relation to the project objectives

The Lines of Attention are organized according to the types defined in the previous section. A summary of the expected Line of Attention results is given in Table 5.1. These results are identical to the expected results as indicated in Chapter 1, but now grouped and detailed per project attention area. The work packages for each line of attention are detailed in the following three sections.

	<b><i>Line of Attention RA: Refactoring and Analysis</i></b>
<b>RA1</b>	Concepts, methods, tools, and techniques to explore the presence of particular idioms and crosscutting concerns in a given software system
<b>RA2</b>	Techniques and tools that aid in the transformation of an embedded software system to an explicitly aspect-oriented software system.
<b>RA3</b>	Tools and techniques to integrate code generated from aspect models with an existing code base, taking care of issues such as maintainability, testability, and source-level debugging.
<b>RA4</b>	A systematic method to migrate an existing system to an aspect-oriented system.
	<b><i>Line of Attention AC: Architecture-Level Concerns</i></b>
<b>AC1</b>	Methods to understand the relationship between requirements, crosscutting concerns, idioms, and aspects in a given software architecture.
<b>AC2</b>	Methods, tools, and techniques to recover aspect-related architectural views from a software system.
<b>AC3</b>	Overview and consistent description of idioms in the embedded systems context.
<b>AC4</b>	Architecture alternatives evaluated for their ability to make use of idioms and related techniques.
	<b><i>Line of Attention CC: Cross-Cutting Concerns</i></b>

<b>CC1</b>	A modeling method and notation for describing crosscutting concerns in embedded systems
<b>CC2</b>	Methods, tools, and techniques to build systems and to generate code, given the concern-modeling notation.

Table 5.1. Expected Line of Attention results.

In the following sections, the Lines of Attention are detailed. Each Line of Attention consists of several work packages. The different work packages are grouped according to the expected results mentioned above. The relations between the activities, and their interdependencies are not shown in this document.

Every Line of Attention has case studies to confront the theoretical results with ASML cases. Wherever possible these case studies are coordinated over the 3 Lines of Attention. Every Line of Attention is finished with a work package to consolidate the results of the work. This results in thesis papers where appropriate. These work packages are not mentioned separately.

#### 5.4. Line of Attention RA: Refactoring and Analysis

Contribution by <sup>11</sup>

**CWI**

TU/e

UT

ASML

Esi

##### Summary

In this package we explore the following.

- New methods and techniques that are needed for the identification and analysis of crosscutting concerns in C code.
- The generation of C code from aspect representations.
- The transformation of existing C code such that explicit aspects replace implicit idioms.

Our main activities are the following.

##### □ **RA1.A: Infrastructure and Orientation**

In order to be able to conduct analysis experiments with ASML C code, we need a basic tool infrastructure for C code analysis. This includes means for parsing in the presence of a preprocessor, control and data flow analysis, dealing with dialects as well as with company-specific coding standards. Existing tools are used as much as possible.

The purpose of this activity is to set up such an infrastructure, and to conduct a first simple analysis to a selected ASML subsystem.

##### □ **RA1.B: Idiom Reconstruction**

Once we have techniques to obtain a broad perspective on crosscutting concerns in a C system, we can refine these to obtain a detailed view on specific concerns such as error handling, tracing, and scheduling. A list of idioms used at ASML for dealing with particular crosscutting concerns is compiled. Such an idiom description includes all characteristics that identify the idiom.

##### □ **RA1.C: Clone detection**

A promising technique to identify prospective crosscutting concerns automatically is "clone detection": the automatic detection of code that is copied almost literally. Traditional clone detection research aims at finding copy-paste code, in order to replace it by a more structured solution such as a new procedure and calls to that procedure for every occurrence of the same clone.

##### □ **RA1.D: Dynamic Analysis**

In addition to conducting static analysis for the purpose of idiom identification, it is possible to perform a run-time analysis using instrumented code. This has the advantage of resulting in very precise information of actually executed idiom code.

<sup>11</sup> The percentage-wise contribution is based on an allocation of planned personnel, based on the defined budgets. The main contributor is indicated in bold.

Dynamic analysis, however, is not without problems: the code must be instrumented, relevant test cases to exercise the idioms must be identified and executed, and the potentially very large traces must be filtered for idiom-specific code.

❑ **RA2.A: Aspect Representation**

To facilitate code generation, a formal aspect representation is required containing all necessary details to generate aspect code and to weave it into the target system. The design of these models is conducted.

❑ **RA2.B: Idiom Elimination**

One of the goals of the Ideals project is to simplify large C code bases by making aspects explicit. This includes the need to remove the implicit idioms from the code base, replacing them by separate aspect representations. We study how to remove idioms from the C code base safely. This requires a highly accurate idiom analysis, identifying only code that can certainly be eliminated

❑ **RA3.A: Infrastructure**

In order to be able to conduct code generation experiments producing C code, we need a basic infrastructure. A significant part of this infrastructure can be shared with the program analysis infrastructure covered by RA1.A. Specific code generation infrastructure issues include support for preprocessing, pretty printing, term rewriting, and program transformation. This activity includes determining selection criteria, tool experiments, and tool integration.

❑ **RA3.B: Pretty Printing**

Program transformation consists of a series of steps including:

1. Parsing the original sources, yielding an abstract syntax tree (AST);
2. Semantic analysis, enriching the AST with information on, for example, data and control flow;
3. Tree rewriting, turning the source AST into a target AST, exhibiting the required properties;
4. Pretty printing, turning the resulting AST into text.

In traditional compilers, the layout of the generated code is of little concern. In the context of migrating existing code to aspect-oriented code, however, the transformed code is maintained by humans, giving rise to different requirements for pretty printing.

❑ **RA3.C: Origin Tracking**

For the purposes of error handling, test coverage analysis, and source level debugging, it is essential that a proper relationship between the resulting target code and the original source code can be reconstructed. Establishing this link is referred to as origin tracking.

❑ **RA4.A: Aspect Migration**

Once the technology for idiom elimination and aspect introduction is in place, a systematic migration strategy is required to ensure a manageable, correct, and repeatable aspect migration process. Such a process includes an analysis of the software architecture, in order to determine an optimal order for component migration.

❑ **RA4.B: Development of migration strategies to new architectures**

Develop alternative paths to migrate the existing architecture to new architecture. Feasibility of the migration is a boundary condition during the design of the target architecture. This activity is based on actual ASML cases.

❑ **RA4.C: Implementation Model of Superimposition**

This activity aims to implement the superimposition specifications defined in AM, where aspect-code is woven at the predefined places within the software architecture.

❑ **RA4.D: Optimization model of Superimposition**

Since superimposition is a complex process, it requires further work in detecting inconsistencies in multiple aspect superimpositions and in optimizing the performance of the aspects weaved. This involves defining semantic checking algorithms on multiple parse trees, defining rewrite rules for redundancy elimination and code optimization, and defining a code generator for weaving aspects in the software architecture.

## 5.5. Line of Attention AC: Architecture level Concerns

*Contribution by:*  
**TU/e**  
**CWI**  
**UT**  
**ASML**  
**Esi**

### Summary

The goal of this LoA is to develop models of the system architecture that enable the analysis of properties of the new system before it is actually built or changes to an existing system are made. This involves developing models for separate concerns and methods for analyzing the relation between these concerns in the system as a whole. Our main activities are the following.

#### □ **AC1.A: Analysis current architecture**

Find out what idiom is used in the architecture and requirements of the system. The aim is to this find out what idiom is used in the architecture and requirement level of ASML systems. These activities are primarily based on existing documentation and interviews with architects.

#### □ **AC1.B: Idiom Classification**

Establish and classify idiom used throughout the system. Determine which concepts belong or do not belong to the idiom and classify identified idioms with respect to the different types of concerns.

#### □ **AC2.B: Architecture Reconstruction**

Research on reverse engineering has resulted in an approach in which "software exploration" plays a key role. This has resulted in an architecture reconstruction method consisting of the following iterative steps: Concept selection, Data gathering, Abstraction and Presentation. Presentation typically includes a combination of text and diagrams, connected via hyperlinks (HTML).

#### □ **AC3.A: Remodularization**

Crosscutting concerns cross module boundaries. Therefore, identifying crosscutting concerns is strongly related to an analysis of the adequacy of modularization decisions. A significant amount of research deals with cluster analysis or concept analysis in order to group programs into modules based on common program characteristics. The use of these techniques for the purpose of aspect identification is unexplored territory.

#### □ **AC3.B: Validate architectural refactoring on ASML cases**

Establish artifact for validation of techniques and select a (set of) concern(s) to be addressed for refactoring by adapting the architecture. Only isolated concerns are investigated. The current architecture is modeled, and the rules, principles and guidelines of the existing architecture are identified. Alternatives for the existing architecture are generated.

#### □ **AC4.A: Study the interaction between concerns**

Develop methods that support decisions between alternative architecture approaches. A set of concerns is selected and a set of models for these concerns is designed, taking the boundary conditions and constraints of the global ASML architecture into account. The trade-offs between different concerns are made explicit.

#### □ **AC4.B: Idiom Integration**

Most idioms are used together in the system. Therefore, the independent identification and specification of idioms (for crosscutting concerns) is not enough. The primary objective of this activity is to address the possible interaction and/or interference among multiple idioms. The impact of system evolution is investigated: e.g. the ability to add, modify or remove idioms.

#### □ **AC4.C: Specification of Crosscutting Relations**

This work package aims to specify the crosscutting relations among idioms using an appropriate model. Requirements on crosscutting relations are specified and a case study on applicability of the specification techniques for crosscutting relations is performed, based on the ASML architecture.

## 5.6. Line of Attention CC: Cross-Cutting Concerns

*Contribution by:*  
**UT**  
**CWI**  
**TU/e**  
**ASML**  
**Esi**

### Summary

The goal of this Line of Attention is to define models for expressing (crosscutting) idioms. The relevant aspects (crosscutting concerns) are identified and analyzed. These aspects are studied one by one to define a model that is expressive, coherent, canonical, and extensible. These models are then used to specify idioms that are both generic and precise. To implement the resulting aspect models, we study ASML requirements, the state-of-the-art technology, and the related literature. Tools and techniques are developed to generate code for the static and dynamic parts of aspect models.

#### ❑ **CC1.A: Aspect Analysis**

The ASML software architecture is studied, as well as the role of idioms for representing crosscutting concerns in the current architecture.

#### ❑ **CC1.B: Exploring Idiom: basic crosscutting concerns**

We explore a category of basic crosscutting concerns that deals with the collection and dissemination of information. We analyze some concrete idioms to find concise, expressive and composable ways to model them. This should allow us to map from system architecture to a software model in a well-defined and extensible manner. The applicability of the identified idioms is verified on the ASML software.

#### ❑ **CC2.A: Study on aspect implementation techniques**

We gain a comparative understanding of various aspect implementation techniques as used in current aspect languages. To this aim, various studies and experiments are carried out on aspect-oriented languages, tools, compilers and interpreters.

#### ❑ **CC2.B: Analysis of static characteristics of aspects**

The goal is to develop the basic techniques for parsing the models that represent aspects and to represent the static part of the model in such a way that it can be checked, optimized, and efficient code can be generated. The code generation algorithm consists of a language independent and dependent part (for example for the C language).

#### ❑ **CC2.C: Analysis of Dynamic Characteristics of Aspects**

The focus is on dynamic characteristics of aspect models to define the run-time characteristics and to provide the necessary features and utilities to support the desired run-time characteristics.

#### ❑ **CC2.D: Implementation Model for Specific Idioms**

The aim is to enhance the static and dynamic analysis techniques by considering the specific language constructs necessary to implement the aspect models. The semantics of the language keywords and primitive types necessary for defining the idioms are defined.

## 5.7. Knowledge sharing and knowledge transfer

The Ideals project, in which we tackle research problems through close interaction with ASML, creates ideal opportunities to share intermediate results with ASML and others. Feedback from ASML in the form of methods and tools used, and experiences obtained, continuously improves the project. The incremental project phases form the basis for continuous research on case studies. This approach thus creates many opportunities for knowledge sharing and transfer. However, sharing and transfer do not happen automatically, they require careful attention.

Because of the importance of knowledge sharing and transfer, the Embedded Systems Institute's knowledge manager actively participates in the Ideals project

The following sharing and transfer activities are organized for the Ideals project.

- ❑ Once a week technical presentations and discussions. The PhD supervisors, and the LoA content responsible people often attend these technical discussions, organized by the project content support team. These communications are not only directed towards project content, but also towards knowledge sharing and transfer. The latter seems to be a side effect. However, we optimize this side effect by organizing the discussion in such a way that lively discussions take place with a variety of experience and expertise present. People involved should challenge others to present and discuss opinions and ideas.
- ❑ Visiting project partners. Getting acquainted with the partner background and experiences.
- ❑ Interactive project workshops. Discuss ASML case studies and reflect ASML experiences with project activities and results.
- ❑ Interactive sessions with ASML employees about both generic and application-specific results. These workshops help to embed the project results in the ASML organization, and they stimulate active involvement of the ASML organization.
- ❑ All intermediate and final results of the project are visible via intranet in a project-specific workspace. Results can be shared, documents can be written by teams, and all results are available at one location.
- ❑ Sharing of ideas via reflection with experts in the field from other universities and industrial companies. This is done via personal contacts, by visiting conferences and workshops, and by organizing thematic workshops.
- ❑ Publishing results in written form via conference proceedings, journals, books, and PhD theses.

Gradually, the project provides tangible results that should be of interest to others. The proving of the results to real-life cases is done in the application phase of each increment. During these phases, a detailed look is taken which results are candidates for further transfer. Depending on the actual result, the transfer activities are directed towards improvement of education at universities, providing course material, and explicit transfer within ASML. As much as possible, the material is made available to other interested parties outside the project. The following activities are performed.

- ❑ Partner specific transfer and sharing activities.
- ❑ Providing course material and organizing courses.
- ❑ Extend and improve university and HBO curricula.
- ❑ Project symposiums to present results of the project, complemented by presentations by external experts on the same topic or on related topics.
- ❑ Web-portals to provide Internet access to relevant knowledge.