# Evaluating an Embedded Software Reference Architecture – Industrial Experience Report –

Bas Graaf and Hylke van Dijk
*Delft University of Technology*
*The Netherlands*
{*b.s.graaf, h.w.vandijk*}*@ewi.tudelft.nl*

Arie van Deursen
*CWI and Delft University of Technology*
*The Netherlands*
*arie.van.deursen@cwi.nl*

## Abstract

*In this paper, we discuss experiences gained in evaluating the maintainability of a reference architecture in use at Océ, one of the world's leading printer and copier manufacturers. The evaluation is conducted using an approach based on SEI's Software Architecture Analysis Method (*SAAM*). The paper proposes a variant of* SAAM *that helps to reduce the organisational impact of conducting architectural evaluations. Second, we analyse the implications of evaluating reference architectures as opposed to single-product architectures. Furthermore, we share our experience in conducting the evaluation, draw lessons for practitioners, and propose new research topics.*

## 1. Introduction

In industry new products are rarely developed from scratch. Most products are based on previous generations of similar products. Therefore, the capability of reusing large parts of earlier development efforts when developing new products can increase the development efficiency of companies tremendously [11]. However, currently many companies have no structured approach for reuse, as a recent inventory conducted among several companies developing embedded software confirmed [10].

One way to arrive at such structured reuse, is to adopt architectural concepts, including product-line approaches, into the software development process. Architecture-based development increases the development efficiency and makes system more easy to maintain and evolve. It does so by increasing the conceptual integrity [12] of software systems and providing a common software infrastructure which makes it easier to understand systems and integrate new components. A product-line architecture extends these ideas beyond single product developments to a whole generation of products and thus enables the reuse of components in new product-line members.

At Océ, one of the world's leading printer and copier manufacturers, every couple of years, a new product generation, comprising a family of similar products, is launched. To make development and maintenance of such generations more effective and efficient Océ decided to define a reference architecture for part of its products, that establishes a common software infrastructure for different generations, thus facilitating reuse across generation boundaries.

Since this architecture will potentially have a high impact on all embedded software to be developed at Océ, the architecture team at Océ decided to conduct an evaluation of the quality of this reference architecture, using an approach based on SEI's Software Architecture Analysis Method (SAAM [3, 13]). In this paper we report on this evaluation.

The contributions of this paper are threefold. First, we propose a variant of SAAM that helps to reduce the organisational impact of conducting architecture evaluations. Second, we analyse the implications of evaluating reference architectures as opposed to product architectures. Last but not least, we share our experience in conducting an evaluation of a real-life reference architecture that is actually used in industry. The lessons drawn are useful for practitioners, and lead to new research questions in software architecture evaluations.

In order to protect Océ's interests, we cannot discuss Océ-sensitive details of the reference architecture. Instead, we will discuss a modified version, which we will call the *Reference Architecture for Copier Engines* (RACE). We believe that the architectural issues and the evaluation method are not materially affected by these changes.

This paper is organised as follows. In Section 2 we summarise the content and context of the RACE embedded software reference architecture. In Section 3, we describe why we selected SAAM to conduct the RACE evaluation and why Océ's situation required some modifications to it. In Section 4 we explain how the actual evaluation was carried out and how certain practical problems were solved. Then, in Section 5 we reflect on the evaluation and iden-

tify future work. We conclude with a discussion of related work and a summary of the paper's contributions.

## 2. An Overview of RACE

In this section we sketch the key elements of the RACE embedded software reference architecture.

### 2.1. Business drivers

RACE serves several purposes, of which the most important are:

**Knowledge base** The RACE documentation can be referred to in product specific documentation. It provides common terminology for software architects that is applicable to several products. The shared terminology together with the RACE-meetings enable architects to share experiences more efficiently.

**Starting point** The RACE documentation can be used by new projects as a starting point. This greatly reduces the effort required for designing an engine architecture for a new product.

**Reuse** As RACE describes the generic structure and behaviour of the engine software components (even the product-specific ones), it makes integrating existing 'RACE-compliant' software components easier, and thus increases the reuse potential of those components. This not only includes binary components, but also designs, requirements and other software artifacts.

In fact the three points above are all related to reuse (either knowledge, documentation, or other software products). Eventually these should make it possible to speed up the development of prototypes and new products significantly.

### 2.2. Reference Architecture

RACE is a reference architecture for the domain of Océ copiers. It defines the fundamental elements of the domain, relations between these elements, and properties of other, product-specific elements. As such RACE provides the key elements that will be part of any future copier developed by Océ.

RACE addresses the *engine* software for Océ document processing systems (copiers). The engine is the part of the system that handles either the scanning, or the printing of documents, as illustrated in Figure 1. The scanner engine only extracts an image from the original sheet, whereas the printer engine reproduces the image data on blank sheets. RACE describes an abstract engine that can potentially be used for any Océ copier.

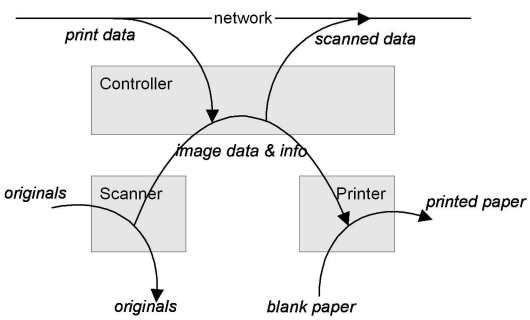


Figure 1. Main flows in a printer/copier.

The RACE reference architecture is typically used to derive a software architecture for engines incorporated in a specific series of Océ printers. From this software architecture, individual engines can be configured that are integrated in Océ's products. In this way RACE defines a (product) family of copier engines.

A classification of product families with respect to level of reuse is introduced by Deelstra et al. [4]. We use this classification and the accompanying terminology to position RACE. Four (ordered) levels are identified: 1) standardised infrastructure, 2) platform, 3) software product line, and 4) configurable product family. These levels denote to which extent the commonalities between related products in the product family are exploited. RACE is best positioned as a platform, since it provides RACE-compatible reusable components that are developed by a separate reuse group. RACE defines a standardised infrastructure by prescribing how components should interact and what functional components should look like. Additionally it offers a platform that realizes common functionality, such as error handling and scheduling. However, in order to qualify as a product-line architecture RACE must define the functional variability between different engines. As all business drivers of RACE are related to reuse, Océ is particularly interested in investigating whether it is possible and worthwhile to raise the current reuse level of RACE to that of a product line.

### 2.3. Structure

RACE is extensively documented using text illustrated with UML diagrams, in total over 500 pages. The documentation is structured according to the *Architecture MetaModel* (AMM) developed by ATOS Origin [6]. This architecture documentation method builds upon the Siemens four-views model [18] and Kruchten's 4+1 View

Table 1. Views used in in the RACE documentation.

| View | Static | Dynamic |
|---|---|---|
| Conceptual | System context, stakeholders, key requirements, external interfaces | Use cases, user visible states, configurations, variants |
| Logical | System components and dependencies, subsystem decomposition, persistent data, internal interfaces | Behaviour, component connection and disconnection, startup, key algorithms. |
| Physical | Files, directories, code, build rules | Threads, tasks, scheduling, interrupts. |

Model [15]. It is organised around three views: the *conceptual*, *logical*, and *physical* view. For each view, a static and a dynamic perspective is offered. This gives rise to six views, as illustrated in Table 1.

The RACE documentation includes one *overview* document of approximately 50 pages, and a dozen documents describing the architecture for specific *functionalities*, such as status control, software downloading, data persistence, and diagnostics. Each of these documents is organised according to AMM. The views are illustrated with diagrams expressed in UML-RT, a real-time extension of UML – which is widely used at Océ [8]. In particular, many use cases are elaborated in sequence diagrams.

### 2.4. Usage

Currently the use of RACE is voluntary. However, architects that want to use RACE for their project are supposed to first participate in the RACE meetings for some months to get the same shared understanding of the reference architecture as the other RACE-architects. This ensures that RACE is more than a pile of documents. These meetings are very important as they provide a communication platform which is essential for meeting the initial objectives of RACE (Sec. 2.1).

In agreement with the objectives of RACE, there is a logical link between the RACE group and the group that develops reusable software components for the engine software. In the current situation, only the reusable components refer to the RACE documentation, which means that the RACE documentation does not show what components can be used to implement the different elements of the architecture.

The actual usage of RACE leads to refinements of and additions to RACE. This interplay between usage and evolution of RACE is depicted in Figure 2. The horizontal line represents the evolution of RACE. Each pi represents a project in which an engine is developed for a series of Océ copiers based on RACE. Each project can join RACE for some time, contributing to its development and benefiting from modifications made to RACE. This is indicated by the oblique lines for projects p1, p2, and p4. After a while, such projects may decide to leave RACE, and continue on

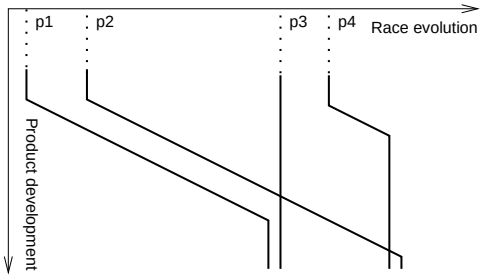


Figure 2. RACE and derived projects.

their own using a fixed version of RACE (the lines become vertical). Other projects (p3) may decide to use a fixed version of RACE right from the start, extracting just whatever is necessary from that version of RACE.

RACE came into existence based on the documentation and experience of several previous projects. In fact it was developed largely in parallel with one specific project. As such RACE can currently be understood as the common denominator of several product specific architectures.

As said RACE's use is voluntary and it is not yet known to all potential stakeholders. Therefore we can say that RACE is currently in an emerging phase. So besides confirmation that its reference architecture is suitable for its intended purpose, now and in the future, the RACE team could also raise the RACE awareness within Océ by doing this evaluation.

## 3. Evaluation Approach

### 3.1. Evaluation Goal

The initial question that triggered this work was "How good is RACE?" Additionally another important and related question was asked: "Does RACE have a reason to exist?" The RACE team mainly wanted to get confirmation that RACE is useful and that it is of good quality.

We first define what the terms 'quality' and 'good' mean in relation to RACE. As 'good' is always relative to particular requirements, the first step is to determine these requirements for RACE, which were unknown since their defini-

tion was neglected during development.

As RACE is intended to be used for several years and product generations, it is essential that RACE can support future changes to its environment and new product requirements. This is the main type of quality under consideration in the evaluation. Furthermore, in view of the fact that the objectives of RACE as presented in Section 2 are centred around reuse, the impact that future changes will have on the reuse potential offered by RACE is essential. In the rest of this paper we will use the term maintainability to refer to the type of quality required for a reference architecture described above.

Thus, the central question is: "How well is RACE prepared for the future?" As this future is not always known at the time of evaluation, the selected method must explicitly address anticipated extensions. Next to this technical view the evaluation also serves an organisational purpose: that of creating awareness.

### 3.2. Selection of Evaluation Method

A literature overview of architecture evaluation methods [7] was used to select an appropriate approach to answer the central question above. Besides addressing maintainability as we described it in Section 3.1, Océ further required the method to be lightweight and well-documented. The method must have a low organisational impact because, as RACE is still in an emerging phase, its evaluation must not affect other processes at Océ. Additionally the method must be executable without additional training. This requires that a clear procedure for doing an evaluation based on the selected method is available. These constraints imply the exclusion of many of the inventoried methods because these either focus on a different quality attribute or lack sufficient detail, e.g. many methods are defined in only one published article.

The most suited methods described in the inventory seem to be SAAM and the Architecture Tradeoff Analysis Method (ATAM [3, 14]), SAAM's successor. Both methods address maintainability and are extensively documented. Although ATAM is likely to produce more objective and accurate results, it also seems more difficult to apply for unexperienced assessors. The use of attribute-based architecture styles and their associated quality attribute characterisations for analysis of architectural decisions is not straightforward. Also the identification of sensitivity and tradeoff points and the generation of a utility tree requires more effort and experience. Due to Océ's requirements with respect to the need for training (no need) and organisational impact (low) of the method, SAAM was selected.

In a SAAM evaluation, scenarios are developed to assess a software architecture's support for maintainability. The scenarios are used to express the required type of maintainability and thus SAAM can also be used to evaluate the type of maintainability we described previously. The developed scenarios represent possible future changes to the software system. An important aspect of SAAM is that it involves all stakeholders of a software architecture in a joint evaluation session, which results in a better appreciation and a more widely shared understanding of the software architecture.

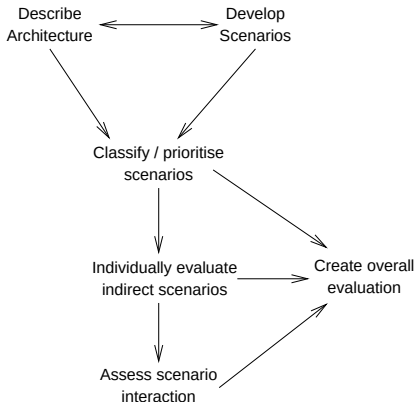


Figure 3. SAAM steps [3].

The different phases of SAAM are presented in Figure 3. A SAAM evaluation session starts with two iterative activities that take place in parallel: scenario development and description of the architecture. New scenarios can make it necessary to describe the architecture further, so that the architects can analyse them, while describing aspects of the architecture makes it necessary to think of possible scenarios addressing these aspects.

Next, the scenarios are prioritised and classified, and the so-called *indirect* scenarios (those that require changes to the architecture) are analysed for their impact. Furthermore, the scenario *interaction* is determined. Two scenarios interact when they require changes to the same architectural component. Information on scenario interaction is indicative of the quality of the decomposition.

Finally, the classification, prioritisation, analysis of the individual scenarios, and the scenario interaction are used to create an overall evaluation.

A SAAM evaluation session typically takes two days and involves an external evaluation team of 3 to 4 people. A session also involves system architects and other stakeholders. The type of stakeholders involved is very diverse: architects, developers, maintainers, integrators, managers, customers, end users, and so on.

### 3.3. Tailoring SAAM

SAAM has been selected as the method for evaluating RACE, yet it had to be tailored to Océ's situation. The current situation at Océ makes it necessary to modify SAAM

for two reasons: 1) organisational impact of SAAM and 2) level of abstraction of RACE.

The impact of gathering all potential RACE stakeholders (Table 2), was still considered too great. Two characteristics of a situation as found at Océ caused this.

First, the stakeholders of a software architecture typically include some of the key-persons in an organisation. For RACE this is especially true as it is the RACE group's ambition to make it a reference architecture that will impact development of many of Océ's copiers for years. The scope of a reference architecture is thus larger than that of a single-product architecture. Next to a group of *direct* stakeholders a large group of *indirect* stakeholders (Table 2) exist, which makes the complete group of people with an interest in a reference architecture much larger.

Second, because of the current status of RACE (emerging) and the resulting RACE awareness within Océ (low), many of its (potential) stakeholders are not prepared to spent too much effort on an evaluation or change their schedules.

The increased number of stakeholders and their willingness to participate made that it was not possible to find a date that suited all stakeholders and undesirable to take one or two full days of each stakeholders' time.

Next to an effect on the number of stakeholders the fact that we are studying a reference architecture also affects the level of abstraction, as a reference architecture is more abstract than a single-product architecture.

The characteristics of the situation as found at Océ that we discussed above have several implications for the evaluation. Below we will discuss how these issues lead to modifications to the typical SAAM process as described in [13].

Table 2. Potential RACE stakeholders.

| **Stakeholder** | **Interest** |
|---|---|
| Architects | RACE architects |
| Users | Product architects |
| Management | Sponsors and decision makers |
| Potential users | Product architects not in RACE |
| Reuse group | Provider of RACE components |
| Indirect | Stakeholders of RACE based products |

The proposed tailored version is a distributed implementation of SAAM (D-SAAM), that implements parts of the SAAM activities off-line, apart from the joint session. For instance, in preparation to the evaluation session, stakeholders are consulted individually. The joint D-SAAM session itself involves only participants fully aware of and well-informed on the reference architecture. The advantage of this approach is that the organisational impact is much smaller. Off-line consultation of individual stakeholders takes less time than a joint SAAM session. Additionally these consultations can be scheduled fitting the stakeholders' agenda's. Of course this approach increases the effort required by the assessors involved in these preparations. However, because we tried to minimise organisational impact, we aimed at reducing the required stakeholder effort.

An additional advantage is that smaller gatherings potentially induce less ambiguity, meaning a more efficient meeting. Therefore the actual D-SAAM evaluation session lasts half a day instead of the usual two days. This further decreases the organisational impact of the RACE evaluation.

## 4. Conducting the Evaluation

The evaluation consisted of roughly three phases. First, the joint D-SAAM session had to be prepared. Second, the D-SAAM evaluation session itself was executed. And finally an overall evaluation of RACE was created. Three RACE architects and two external observers participated in the joint session. One of the RACE architects played the role of evaluation leader and prepared, chaired, and evaluated the actual execution of D-SAAM. For each SAAM step in Figure 3, we explain below how it was included in the different phases of the D-SAAM assessment.

### 4.1. Preparation

In preparation to the execution of the joint D-SAAM session the available documentation (on RACE and SAAM) was distributed among the participants. The RACE documentation was especially useful for the external observers as it explains the architecture and the applied architectural mechanisms. The documents on SAAM were only used by the evaluation leader.

The step 'develop scenarios' was carried out in two steps. During the preparation phase, the evaluation leader consulted stakeholders off-line. This resulted in an initial set of high-level scenarios representing possible futures from a stakeholder's perspective. The set of stakeholders included the sponsor of RACE, members of the software reuse group, hardware experts and domain experts. Unfortunately, the marketing and maintenance groups were not consulted, which limited the view on the road maps for Océ copier machines. The scenarios were related to either existing products or foreseen products. Whether these products were also used to develop RACE, is irrelevant. The evaluation leader then added more detail to these scenarios according to a template for scenarios based on [1].

### 4.2. Scenarios

In total sixteen scenarios were developed off-line. The majority of the scenarios aimed at reducing material costs,

for example by sharing resources, using low-power designs, or offloading or re-mapping functionality. For example, one scenario aimed at moving functionality from the engine software to the main controller, another subsystem of a complete document processing system.

A second kind of scenarios was developed to reduce development costs. For instance, introduction of code generation for controllers of sensors and actuators based on mathematical models of those hardware devices. These scenarios were especially targeted at interactions which go beyond the domain level, such as communications with the mechatronics, testing, and manufacturing groups.

Finally, a minor source of scenarios involved an upgrade of the functionality, such as supporting colour prints and so-called wide-format printing systems (continuous printing roll systems). An example scenario is depicted in Table 3 in a format described by Bass et al. [1].

Table 3. An example scenario.

| | |
|---|---|
| *Stimulus* | Reduce power consumption by turning parts of the copier machine off during low-power mode |
| *Response* | Solve in engine specific projects |
| *Source* | Electronics department |
| *Environment* | Engine development time |
| *Stimulated artifact* | RACE documentation |
| *Response measure* | Reuse percentage remains on same level |

### 4.3. Execution

In the joint session each architect represented a product as user of RACE. Additionally all architects played the role of assessor. As some of the participants had no experience in SAAM evaluations and to explain the steps of the D-SAAM process, the session started with a brief introduction of the process. For the process observers also the role of RACE in the organisation of Océ was explained.

The step 'describe the architecture' was largely omitted during the D-SAAM session, since the D-SAAM session only involved people that are well-informed with respect to RACE, and extensive documentation was already available.

The second part of the step 'develop scenarios' was done during the D-SAAM session. This involved only architects of products on which RACE was based. However, since soliciting for extra scenarios gave no results, the scenarios gathered and elaborated by the evaluation leader were used.

Scenarios were classified, prioritised, and evaluated as in SAAM, during the session itself. The scenarios were classified and evaluated one by one, bypassing prioritisation (Fig. 3). Figure 4 gives an impression of the final result of the SAAM session. Scenarios were classified in directly and indirectly supported scenarios.

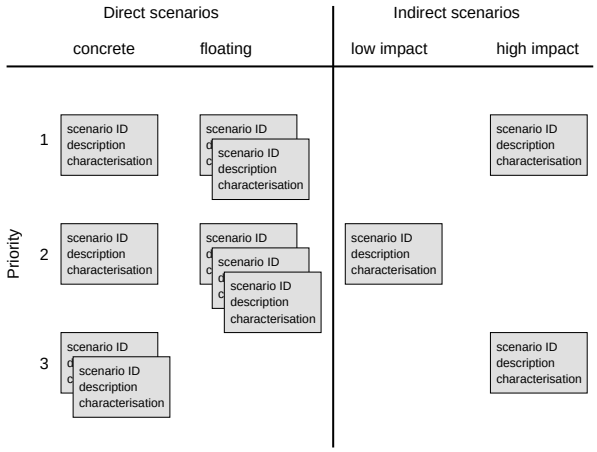


Figure 4. SAAM results.

In general, first the impact of a scenario on a specific product was evaluated, and then its impact on RACE. Classification and evaluation required a different attitude because we were evaluating a reference architecture instead of a product architecture. The difficulty lied in the fact that while scenarios are concrete, representing future functionality, or the quality of actual products, the reference architecture is abstract. The question:"What is the impact on the *reference* architecture?" needed to be answered consistently for all scenarios. Therefore we defined two types of direct scenarios:

1. Scenarios that are supported by the reference architecture as is and for which it provides concrete guidelines on how to realize them in product instantiations, and
2. Scenarios that can be realized by systems based on the reference architecture, but for which it does not (yet) provide detailed information on how to realize them (floating).

The class of floating scenarios calls for a cookbook with recipes that describe specific solutions for specific variation points in the reference architecture. Cookbook recipes describe how RACE can be used to realize a specific (floating) scenario. For example, it might be necessary to describe what kind of components need to be defined or how some of the already defined components should cooperate to implement the desired behaviour. This information can be included in RACE in a separate document without affecting the existing RACE documentation. By realizing scenarios this way the scope of reuse for RACE is extended and RACE's classification moves from platform towards product line. An example of such a cookbook recipe was the

description of how to realize sharing of hardware resources within a RACE engine. The recipes were just new RACE documents. In fact, some of the existing RACE documents already were such recipes, such as the document describing how function component should look like, without actually defining concrete function components. These documents had a different nature than the other RACE documents that describe specific RACE components, like a scheduler. Observe from Figure 4 that most of the scenarios fall in the category of directly supported scenarios.

The indirect scenarios were, as usual in SAAM, partitioned in two subsets: a subset with low impact and a subset with high impact. Overall this assessment session did not discover many design flaws. The architects spent most of their time on the single high impact, high priority scenario (resource sharing).

The indirect scenario interaction was considered very briefly as only a few indirect scenarios were discovered. It was concluded that those did not interact.

### 4.4. Overall Evaluation

This final stage of the assessment involved the overall evaluation, which resulted in a set of strong points and weak points. The set of strong points includes the aforementioned use of RACE and its flexibility; most of the evaluated scenarios are directly supported.

The set of weak points includes a design flaw that prevents seamless support of parallel and distributed computing, which was required for implementing the scenarios that aimed at sharing hardware resources and reusing engine parts amongst different engines in a single copier. Additionally RACE seemed incomplete as it missed several cookbook recipes. For instance, recipes for sharing hardware resources are currently not included in RACE. Another weakness was that variation points were explicit in the RACE documentation. Related to this issue is a missing structure for documenting a RACE instantiation, i.e. a engine generation, with respect to the RACE documentation. It was not clear how conformance to and deviations from RACE should be specified by projects that develop a RACE instantiation. However, this is important for the maintainability of the reference architecture.

## 5. Discussion

Below we both discuss the implications of evaluating a reference architecture and using a distributed SAAM approach and we indicate where these lead to suggestions for future work and research questions.

### 5.1. Reference architecture

**Reuse level** In Section 2.2 we positioned RACE as a platform. Furthermore the business drivers for RACE were all related to reuse (Sec. 2.1). Therefore the positioning raised two questions: is it correct for the current situation, and for the future? If correct, the current reuse positioning of RACE as a platform should be supported by links between the documentation of RACE and the documentation of instantiated products. In view of the reuse positioning of RACE, we expect a considerable reduction in the effort of documenting a product instantiation compared to a single-product architecture approach. A prerequisite for this conjecture is that there must be a systematic way of documenting product instantiations with respect to RACE. It is unclear whether such a systematic documentation process exists.

> **Research question** *Can we define and deploy a systematic process for documenting product architectures with respect to a reference architecture.*

In order to find out if product instances are documented with respect to their reference architecture in a systematic way reuse metrics are required [17]. As an example of such a reuse metric, consider two indicative figures: the relative size and a normalised cohesion factor. The size factor calculates the lines of documentation of a product instantiation relative to the size of the RACE documentation. The cohesion factor takes the number of references from the documentation of a concrete product to the RACE documentation that handle variation points, normalised with the total number of references from the product instantiation documentation to the RACE documentation. A low relative size and high cohesion factor indicate a high reuse factor and thus a systematic approach for reusing RACE in product instantiations.

> **Future work** *Define a metric to position a reference architecture with respect to scope of reuse?*

With respect to the future reuse positioning of RACE we would expect, looking at RACE's reuse-oriented business drivers, that Océ aims to increase the reuse scope of RACE. This objective is supported by the identification of various direct floating scenarios, which will be implemented by the RACE team in a cookbook (Fig. 4). This implies that Océ indeed foresees that the reuse positioning of RACE is raised from platform to software product line in the near future.

**Updates** RACE's maintainability was the central quality aspect in the evaluation. One aspect of maintainability is

the possibility to *update* the reference architecture with developments that take place in a product instantiation: during the oblique lines in Figure 2. In order to successfully implement a proposed update to RACE two issues need to be considered: conformance and permissiveness.

Conformance is the extent to which the product architecture and reference architecture match. One must specify the update in agreement with the existing reference architecture. This is necessary, for example, to prevent specifying updates to components that do not exist at all in the reference architecture. The architecture of a product may undergo small changes during its development. Consequently, there may be a discrepancy between the product architecture and the reference architecture. The discrepancy may obstruct the transfer of architectural fragments, e.g., a cookbook recipe, from the reference architecture to the product architecture. But it may also obstruct the update of the reference architecture itself.

To detect these architectural discrepancies and suggest possible repairs, one could check the conformance by first using reverse engineering techniques to raise the level of abstraction of concrete RACE-based product architectures and then compare the result with RACE [5].

> **Future work** *Develop a technique to measure to conformance of a product architecture with respect to the reference architecture on which it is based in order to assess the possibility to transfer fragments from product architecture to reference architecture?*

The bare fact that a product has an architecture that conforms with the reference architecture does not ensure by itself that a proposed update will be successful. The reference architecture also has to be permissive with respect to the update. The reference architecture must provide the flexibility to incorporate the proposed update. An update might violate some of the design decisions taken earlier; whether this is so is in practice generally hard to assess. One reason for this is that design decisions are not completely documented. Most times only the structural effect of a design decision is documented. Documenting other aspects of design decisions, such as their rationale and effect with respect to (non)-functional requirements is often neglected.

> **Research question** *How can we document design decisions explicitly and how can we then use them to assess an architecture's permissiveness with respect to a proposed update?*

**Use of reference architectures** Another point of discussion is RACE's use. Besides its technical use as a starting point for product specific software architectures, the reference architecture served according to its objectives as a discussion platform for the software architects of different products. In that sense RACE indeed is an efficient way to exchange experiences among product teams.

Another use of RACE appeared during discussions in the D-SAAM evaluation. RACE acts as a stable platform for negotiations amongst different domains: the mechatronics, manufacturing, and software reuse groups at Océ. By introducing a generic and more stable architecture for the engine software of Océ copiers the RACE group tries to prevent that software is automatically considered to be the means to solve problems during engine development. As such defining an embedded software reference architecture helps creating a better balance between the different disciplines involved in engine development. This is a typical problem in the embedded software domain as was also observed in the inventory described in [10].

In the evaluation we conducted, the usage of the reference architecture was not addressed explicitly. Considering the specific use of *reference* architectures described above, it seems useful to do so, especially in the case of embedded systems.

> **Research question** *How can we include the usage of a reference architecture in an evaluation?*

### 5.2. D-SAAM

The status and awareness of RACE in Océ had their impact on the evaluation. As we think this situation at Océ is not unique it is discussed here together with some issues with respect to the scope of the distributed approach.

**Status and awareness** The current low status of RACE and the low awareness of how projects can potentially benefit from it, made that only a small number of stakeholders was involved in the joint D-SAAM evaluation session. As discussed above this was not a real problem for the evaluation. However, the low status and awareness do limit the potential of RACE.

The D-SAAM process was intended to answer a number of questions derived from the initial one: 'How good is RACE?' These questions not only refer to the (technical) maintainability of RACE, but to its current and future use and status in the organisation as well, i.e.:

- Is it possible to raise the scope of reuse of RACE from platform to product line?
- Can RACE support future developments of existing currently compatible products and can it facilitate a reference architecture for near future products?

Positive answers to these questions will raise the status of RACE in the organisation, which incidently will lead to increased involvement of more stakeholders for future evaluations.

A frequently quoted side effect of the general gathering in a SAAM meeting is the increased awareness of architectural details. In this case the goal is more modest, creating awareness of RACE is considered already an asset. The consulting phase prior to the D-SAAM session actually created the aimed awareness of RACE.

**Scope of evaluation** The main concern of scenario-based evaluation methods is whether the coverage and scope is broad enough to be conclusive about the findings of the evaluation. SAAM overcomes this by organising a general two-day gathering, which is moderated by experienced assessors. In D-SAAM we had to take alternative measures.

In view of the two questions above and the development of RACE the number of *direct* stakeholders is limited (Table 2), although many *indirect* stakeholders can be identified. These two groups of stakeholders seem to have different interests in RACE.

Raising the scope of reuse of RACE directly concerns the architects of compatible products as users and architects of RACE. It implies that RACE not only should identify variation points but also explicitly give alternatives. The cookbook of the previous section provides these.

Scenarios that describe future development of existing and foreseen products are the concern of the stakeholders of those products. The development of these scenarios is the responsibility of these stakeholders, which are not necessarily also direct stakeholders of RACE. On the other hand the resulting scenarios are input to D-SAAM session, thus indirectly they are.

One measure we took to include indirect stakeholders in the evaluation was to split the process of developing the set of scenarios in two phases: an off-line phase with the indirect stakeholders, and a D-SAAM phase with the direct stakeholders. The scenarios provided by the indirect stakeholders were product specific. Evaluating the impact on the *reference* architecture was not their concern, but that of the *direct* stakeholders. Furthermore the direct stakeholders are the only ones capable of doing so. Therefore because only the indirect stakeholders were excluded from the joint session, we think the scope of the D-SAAM session was not affected by the lack of stakeholder interaction during evaluation.

However, this also prevented indirect stakeholders to interfere or interact during scenario prioritisation. During the D-SAAM session, the architects concentrated on the most likely scenarios, from the perspective of an architect. Although scenarios were prioritised with respect to their impact, there was no clear rationale for this ranking. Hence D-SAAM's scope was still at risk due to the possibility of a wrong scenario prioritisation.

In order to validate D-SAAM's scope we recommend to organise indirect stakeholder involvement after the joint session as well. During this feedback phase stakeholders might be consulted, for instance in small sessions or individual interviews, in the same way as we did in preparation to the session. This time the indirect stakeholders can comment on the prioritisation of the scenarios and verify whether the evaluation covered all relevant aspects of the architecture. This preserves the small impact on the organisation offered by D-SAAM. During the feedback phase, indirect stakeholders may conclude that some likely scenarios have not been evaluated thoroughly enough. Thus the feedback phase may yield newly developed scenarios, emphasising certain concerns. This new set of scenarios has to be evaluated in a new D-SAAM session. An additional advantage of the proposed wrap-up session with indirect stakeholders is that it will raise the awareness of RACE within Océ, as discussed above, even further.

> **Future work** *Extend* D-SAAM *with an off-line feedback phase after the joint session for indirect stakeholders.*

**Use of Documentation** During the assessment we were somewhat surprised that the actual RACE documentation was not used at all during the session. This means that the architecture assessed is the one that is in the team's heads, and not the *documented* architecture. The corresponding risk is that the team may have different architectures in their minds, that the documented architecture is inadequate, and that architects not participating may have different perspectives. Thus we have:

> **Research question** *How can we involve the architecture as documented explicitly in the assessment process?*

Solution directions will require explicit, analysable representations of both the architecture and the scenarios used in the assessment. An interesting research topic is whether text analysis techniques can be used to analyse the relationship between these two representations.

## 6. Related Work

An overview of SAAM and ATAM, as well as references to many other methods for evaluating software architectures can be found in the book by Clements et al. [3].

Galagher discusses the application of ATAM to a reference architecture [9]. Unfortunately, he hardly discusses any issues specific to the evaluation of reference architectures (such as the different role of scenarios). The reference

architecture is more or less evaluated as a normal software architecture with specific business drivers.

Since the boundary between product line architectures and reference architectures is not always distinct (Sec. 2), another area of relevant related work is the field of product line evaluation. Lutz and Gannod discuss the architectural analysis of a product line architecture [16]. The authors present a three-phased approach consisting of architecture recovery, scenario-based assessment, and model checking of safety-critical behaviour. Here a software architecture needed to be recovered from an existing product, which is then evaluated in order to see whether this type of product is amenable to a product-line-development approach.

Of particular interest are evaluation methods focusing on *maintainability*. The *architecture-level modifiability analysis* (ALMA) method from Bengtsson et al. integrates a number of different scenario-based approaches for assessing architecture maintainability [2].

## 7. Conclusion

In this paper we reported the evaluation of an embedded software reference architecture using a tailored SAAM-based approach. The objective of the assessment was to assess the maintainability of the reference architecture. Maintainability involved two aspects, raising the domain of reuse from a platform to a product line and facilitating anticipated extensions of derived products and future products.

The evaluation of RACE was based on a distributed SAAM (D-SAAM) method, which involves three phases: an preparation phase in which indirect stakeholders are consulted individually to collect scenarios, a joint evaluation session with only architects and observers, and an evaluation phase.

Assessing a reference architecture is different from assessing a product architecture. In an ordinary SAAM session, evaluated scenarios are categorised in directly and indirectly supported scenarios. Because RACE is a reference architecture, the set of directly supported scenarios was subdivided into those with evidence of being supported by RACE and those without evidence. The latter class typically consist of scenarios for which solutions are available in one of the products, but these have not been documented yet. In the D-SAAM session we defined a cookbook to cover these scenarios.

The experience provided valuable insights for industry (Océ) as well as for academia. In retrospect we argued that D-SAAM is a suitable approach for the given situation, assessing the maintainability of a maturing reference architecture. Both the coverage of D-SAAM and the quality of its session conclusions are tenable. Note that reference and product-line architectures enable efficient reuse, a key business driver in many organisations. The concepts on which this type of architectures are based are maturing. Therefore it is expected that more and more companies will adopt a product-line approach, possible involving reference architectures.

Océ gained insight into the positioning and status of RACE in their organisation; its current position and its future position. Océ also gained confidence in the maintainability of RACE.

We gained insight into the process of assessing a reference architecture. For instance, scenarios are typically evaluated based on a product instance and the result are abstracted into the reference architecture. This evokes all kinds of questions related to topics such as conformance checking and documenting design decisions, as discussed in Section 5.

## Acknowledgement

The research in this paper has been sponsored in part by the ITEA MOOSE project. We would like to thank Océ for their hospitality and cooperation, in particular Peter Aarts and Lou Dohmen.

## References

[1] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 2003.

[2] PerOlof Bengtsson, Nico Lassing, Jan Bosch, and Hans van Vliet. Architecture-level modifiability analysis (ALMA). *Journal of Systems and Software*, 69(1–2):129–147, jan 2004.

[3] Paul Clements, Rick Kazman, and Mark Klein. *Evaluating Software Architectures*. Addison-Wesley, 2002.

[4] Sybren Deelstra, Marco Sinnema, and Jan Bosch. Product derivation in software product families: a case study. Accepted for publication in The Journal of Systems and Software, 2004.

[5] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva. Symphony: View-driven software architecture reconstruction. In *Proceedings IEEE/IFIP Working Conference on Software Architecture (WICSA'04)*. IEEE Computer Society Press, 2004.

[6] Y. van Dinther, W. Schijfs, F. van den Berk, and K. Rijnierse. Architectural modeling: Introducing the Architecture Meta-Model. In *Landelijk Architectuur Congres*, Utrecht, The Netherlands, 2001. SERC.

[7] L. Dobrica and E. Niemelä. A survey on software architecture analysis methods. *IEEE Transactions on software Engineering*, 28(7):638–653, 2002.

[8] L. A. J. Dohmen and L. J Somers. Experiences and lessons learned using UML-RT to develop embedded printer software. In M. Oivo and S. Komi-Sirviö, editors, *Proceedings of PROFES 2002*, LNCS 2559, pages 475–484. Springer-Verlag, 2003.

[9] Brian P. Gallagher. Using the architecture tradeoff analysis method to evaluate a reference architecture: A case study. Technical Report CMU/SEI-2000-TN-007, Carnegie Mellon University, Software Engineering Institute, June 2000.

[10] Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: state of the practice. *IEEE Software*, 20(6):61–69, November–December 2003.

[11] Ivar Jacobson, Martin Griss, and Patrick Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. Addison-Wesley, 1997.

[12] Frederick P. Brooks Jr. *The Mythical Man-Month*. Addison-Wesley, anniversary edition, 1995.

[13] Rick Kazman, Gregory Abowd, Len Bass, and Paul Clements. Scenario-based analysis of software architecture. *IEEE Software*, 13(6):47–55, November 1996.

[14] Rick Kazman, Mark Klein, Mario Barbacci, Tom Longstaff, Howard Lipson, and Jeromy Carriere. The architecture tradeoff analysis method. In *Proceedings of the Fourth IEEE International Conference on Engineering of Complex Computer Systems*, pages 68–78. IEEE Computer Society, Augustus 1998.

[15] Philippe B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, November 1995.

[16] Robyn R. Lutz and Gerald C. Gannod. Analysis of a software product line architecture: an experience report. *The Journal of Systems and Software*, 66(3):253–267, June 2003.

[17] J. S. Poulin. *Measuring Software Reuse: Principles, Practices, and Economic Models*. Addison-Wesley, 1997.

[18] D. Soni, R. L. Nord, and C. Hofmeister. Software architecture in industrial applications. In *Proceedings 17th International Conference on Software Engineering (ICSE'95)*. ACM Press, 1995.