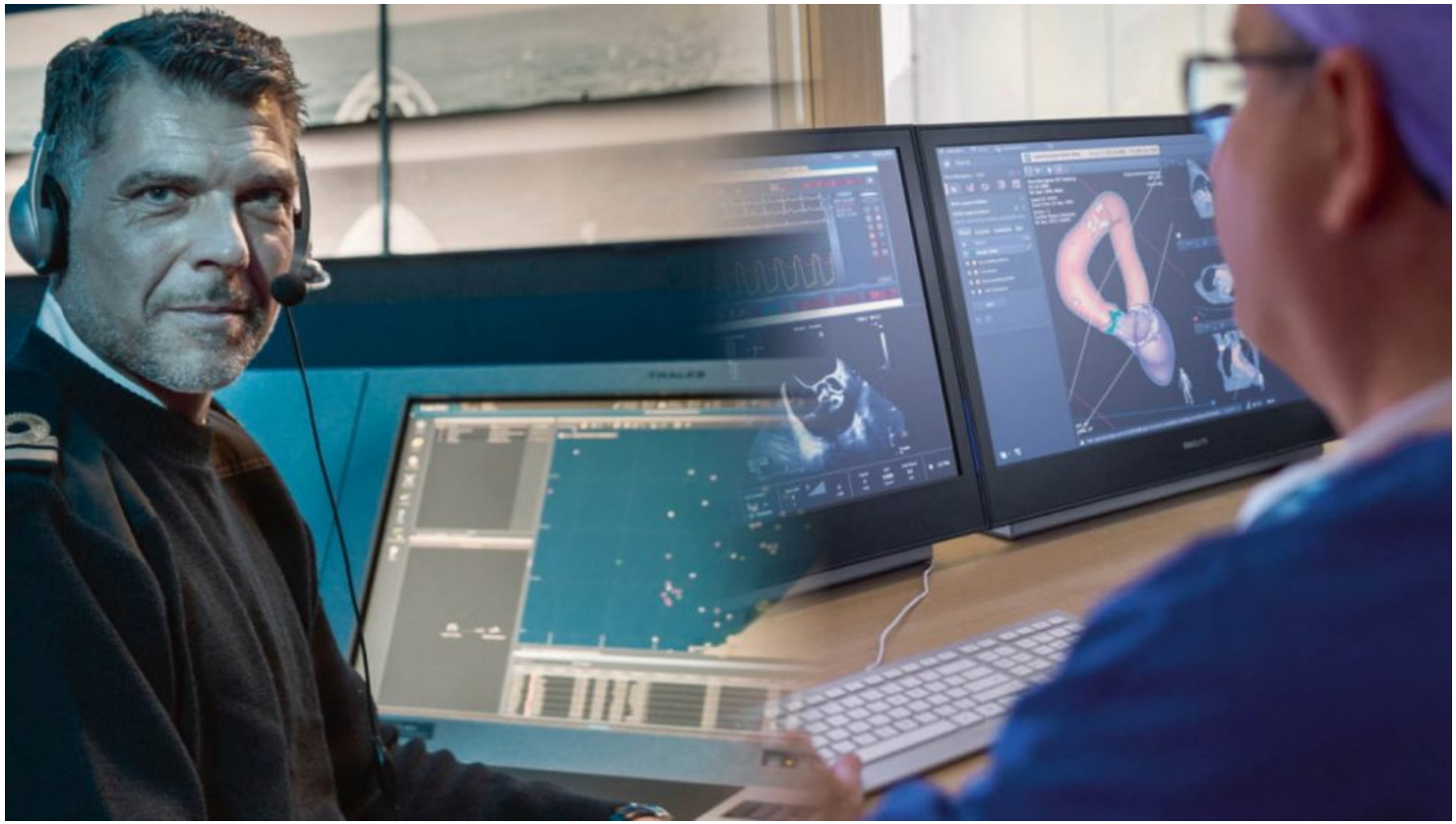


BACKGROUND

# Comma interfaces open the door to reliable high-tech systems

*Nieke Roos*

yesterday

Once a research project initiated by ESI (TNO) and Philips, the Comma framework is developing into a mature product for creating and managing software interfaces. Now, Thales is also looking to use it to streamline its software engineering, as are Thermo Fisher Scientific and Kulicke & Soffa. “Comma is the place where you express everything you want and from there, you generate everything you need, like documentation, monitoring, simulation, visualization and, as of recently, test cases.”

“Our medical devices are growing bigger and bigger,” observes Daan van der Munnik, software manager at Philips Healthcare in Best. “We have to chop them up in smaller subsystems to keep their development manageable, but also for validation purposes. Up to a year ago, we validated a complete device in one go – a huge effort. By chopping it up in smaller subsystems, we can focus our validation efforts on the parts of the system we actually touch for a particular feature. We do need to show that when we put everything together, it still does what it’s supposed to do. Both the disassembling and reassembling call for good interface management.”

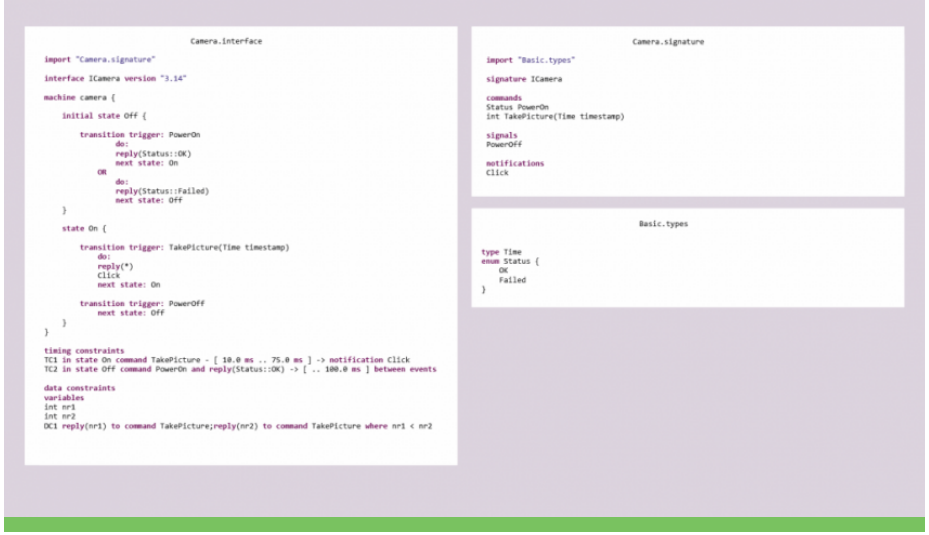
Subsystems are also increasingly being farmed out to subcontractors. “We’re really moving to a system-of-systems development, where we make some parts ourselves and some parts come from outside,” notes Van der Munnik. “For instance, in one of our image-guided therapy systems, we have three types of patient tables. One is developed by us, two are made by other companies. From a user perspective, however, they all have to appear to be an integral part of the system – the user experience, for instance when moving or tilting, has to be exactly the same. This means that, for our subcontractors, the interfaces need to be clearly defined, both on a low technical level and a high subsystem level.”

Last but certainly not least, good interface management is key for system evolvability. Van der Munnik: “Our medical devices have very long lifetimes. We need to ensure that, over their lifetime, they’re expandable and suitable for form/fit/function replacements.”

## Comma

Comma (Component Modeling and Analysis) is an ecosystem supporting model-based component engineering. It’s a combination of domain-specific languages (DSLs) in which the interface between a server and its clients can be specified by three main ingredients: the interface signature, the allowed client-server interactions and the time and data constraints. The interface signature consists of groups of commands, signals and asynchronous notifications. Commands are synchronous: the caller is blocked until a reply is received, whereas signals are asynchronous: they do not block the caller and do not require a reply. State machines are used to describe the allowed client-server interactions, such as the allowed order of client calls and the allowed notifications from the server in any state. Finally, Comma enables the definition of constraints such as the allowed

response time, notification periodicity and data relationships between the parameters of subsequent calls.



Camera.interface

```
import "Camera.signature"

interface ICamera version "3.14"

machine camera {

    initial state Off {

        transition trigger: PowerOn
            do:
            reply(Status::OK)
            next state: On
        OR
            do:
            reply(Status::Failed)
            next state: Off
    }

    state On {

        transition trigger: TakePicture(Time timestamp)
            do:
            reply(*)
            Click
            next state: On

        transition trigger: PowerOff
            next state: Off
    }
}

timing constraints
TC1 in state On command TakePicture - [ 10.0 ms .. 75.0 ms ] -> notification Click
TC2 in state Off command PowerOn and reply(Status::OK) -> [ .. 100.0 ms ] between events

data constraints
variables
int nr1
int nr2
DC1 reply(nr1) to command TakePicture;reply(nr2) to command TakePicture where nr1 < nr2
```

Camera.signature

```
import "Basic.types"

signature ICamera

commands
Status PowerOn
int TakePicture(Time timestamp)

signals
PowerOff

notifications
Click
```

Basic.types

```
type Time
enum Status {
    OK
    Failed
}
```

*An example Comma model.*

## Evolving interfaces

Fellow high-tech company Thales faces similar challenges. “Traditionally, we developed, built and qualified our combat management and radar systems, delivered them to the customer and mostly touched them to replace obsolete components – to avoid unnecessary risks, functional changes were rather limited and implemented at long intervals,” explains Pepijn Noltes, software architect at the Hengelo-based company. The last ten years, however, the operational scene is changing more rapidly at extended operational lifecycles, with customers increasingly

demanding new features. Thales is adapting to this need by looking for ways to implement software updates more frequently, including incremental enhancements.



But that's easier said than done. "For complex software-centric systems like ours, it's very expensive to change something, integrate and test it – especially so in the military domain that we're in, where you may have to do live firing trials to really validate the system," says Noltes. "Also, even the tiniest update may cause an avalanche of changes. It then boils down to the question: how well can you revise part of your system without touching the rest?"

Noltes has learned that to be able to continuously update a complex system, you need to keep the changes local and to do that, you need to focus on the interfacing. "We tend to touch the

interfaces as little as possible because they're expensive to change. Bigger problems that you can't work around will eventually get fixed, but small issues will remain, as a result of which the code quality will slowly deteriorate. We're now looking at evolving interfaces to facilitate the need for change."

## Single source of truth

Enter Comma (Component Modeling and Analysis), an ecosystem supporting model-based component engineering. "It started about six years ago as a research project between ESI and Philips," recalls Jozef Hooman, senior scientist at ESI, the high-tech embedded systems joint innovation center of the Netherlands Organization for Applied Scientific Research (TNO). "We began using domain-specific languages for all kinds of purposes, generating code, analysis tools and much more. While doing this, we noticed that a lot of issues Philips had with its software were due to interface problems and, gradually, the insight came to us that these DSLs were especially useful for describing the interfaces. So, in small steps, we moved from general-purpose languages to a domain-specific language, Comma, which we reused for many different interfaces."

Although a research project, the development of Comma wasn't driven by research considerations, Hooman points out. "We really

*Thales is looking to use Comma to make the interfaces in its software-centric systems evolvable. Credit: Thales*

looked at what the engineers at Philips needed and adapted the language accordingly. We started with a state machine describing the interface protocol, ie the interaction between client and

server. Based on user feedback, we modified it to make it more user friendly and include things like timing and data constraints. The patient table, for instance, is very sensitive to both timing and data – when the controlling joystick stops, the table should stop too within a certain amount of time and without moving too much."

Step by step, Comma developed into what it is now: "the single source of truth," as Hooman calls it. "This DSL is the place where you express everything you want and from there, you generate everything you need, like documentation, monitoring, simulation, visualization and, as of recently, test cases. Monitoring, especially, is very important. You can use that to see if your implementation satisfies the specification by running the system, collecting traces and check whether the execution conforms to the interface. If an interface changes, you can re-generate everything, and if your developers, or your third-party suppliers for that matter, introduce a software update, you can check that for conformity – all with the push of a button, continuously, as an integral part of your test process."

## Forming an ecosystem

At Philips, Comma is now firmly embedded in the company's software engineering practice. Van der Munnik: "We use the DSL to write the interface specs and generate documentation and

code. As part of our continuous integration pipeline, we check interface conformance against the Comma specs when executing our automated test scenarios. We've created a maturity matrix, which sets off our interfaces against these development stages, and we're now raising the bar for all of them. Thanks to the unambiguous definition of interfaces and the subsequent automatic validation, Comma brings us a huge amount of business value as we find interface issues early, well before integration."

Two years ago, Thales started the Dynamics project to research dynamic system updates in collaboration with ESI. "We're looking into evolvable interfaces and so-called adapters to keep the old and the new working together," clarifies Noltes. "So when you introduce a client with an updated interface, you also generate an adapter that connects it to your existing server and provably ensures that nothing breaks down. ESI did a small technology survey on interface specifications and Comma came out as the solution that best fits our needs. Although Dynamics is still ongoing research, Comma is already useable out of the box and we're busy to include it in our component development framework. By doing more at design time, we hope to eliminate much of the risk in projects."

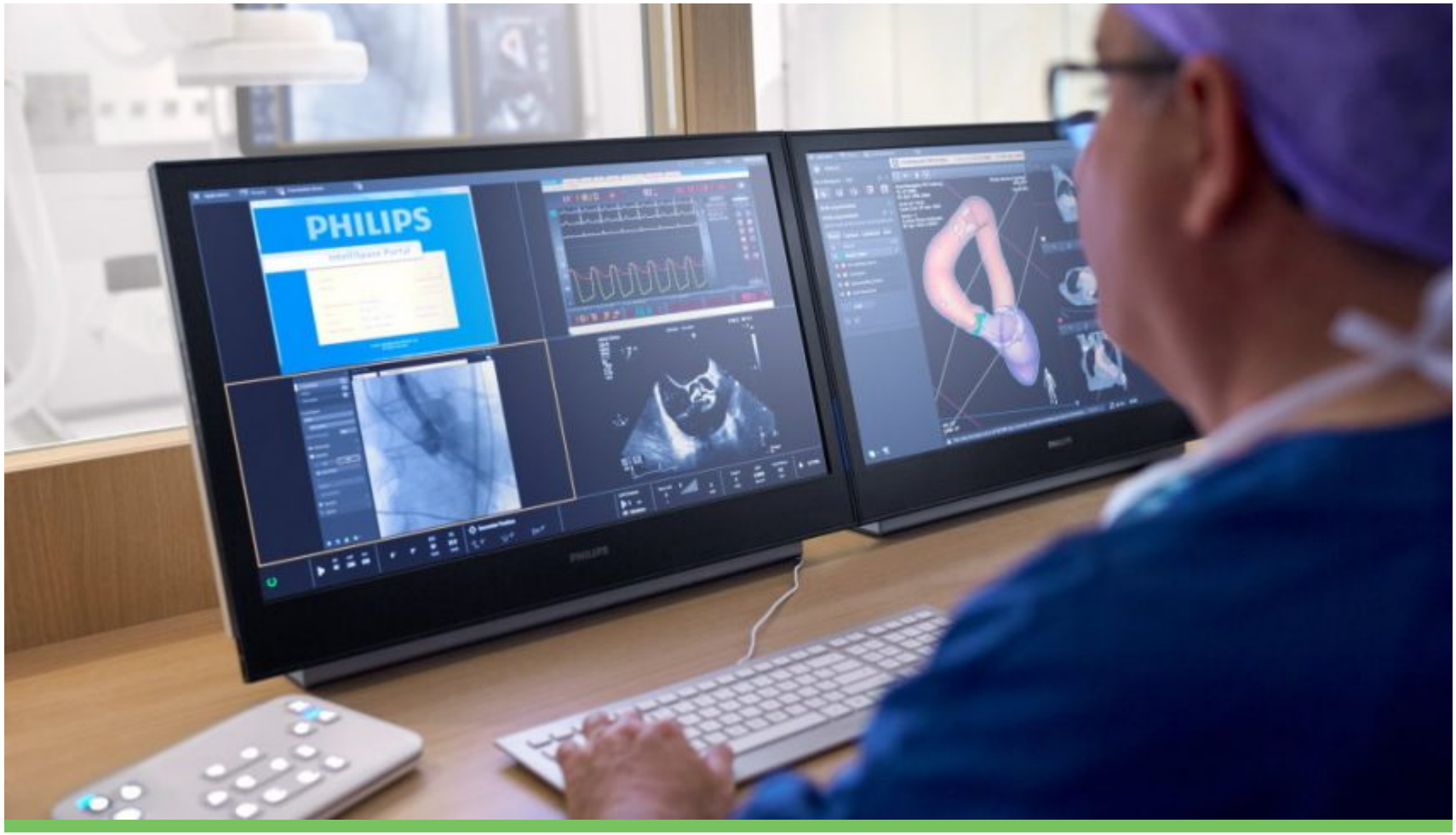
Slowly but surely, Comma is conquering the Dutch high tech. "We're also working with Thermo Fisher Scientific in Eindhoven,

for instance," illustrates ESI's Hooman. "For some critical interfaces, a model has been made and a monitor has been built into the nightly smoke tests, which automatically checks the log files. In the morning, they can see what properties have failed. And Kulicke & Soffa, also from Eindhoven, is looking into making a generator for its middleware layer." Senior research fellow Benny Akesson, ESI's liaison to the Dynamics project, adds: "It's interesting to see this ecosystem starting to form."

## Backward compatible

According to Akesson, there are basically three ways of using Comma. "This monitoring facility has already been there for years. If you have an interface and all the traces you run through the monitor are compliant, you know you're in a good place. When you update your interface, you can automatically generate a new monitor and feed it the same traces to see whether they still work. As you don't have to do a complete impact analysis of the change manually, that saves you time. The problem with this is that you're now no better than your traces. You need to have traces that are representative of all the desired system behavior."

Together with its industrial partners, ESI is working on a solution based on so-called Petri nets, a formal method using state-transition models to study concurrent and distributed systems. "By




*Thanks to Comma, it's easier for Philips to enhance its medical systems. Credit: Philips*

generating Petri nets for an interface, you can see the possible state transitions that can occur in the protocol," explains Akesson. "You can then produce tests that cover those possible transitions and thus systematically explore the state space." Philips is now using Petri nets to do exactly that: to create test cases from the Comma specifications.

A third approach is to play by a slightly more restricted playbook, continues Akesson. “This is what we’re doing in the Dynamics project with Thales. By not using certain constructions in Comma, and using Petri nets in a different way, it’s possible to build tooling that can statically tell you whether your new interface is backward compatible and if not, automatically generate an adapter – if one exists. We’re now lifting this from a proof-of-concept command-line tool into the Eclipse-based Comma environment, providing immediate developer feedback on why a change is or is not backward compatible and whether an adapter can be generated.”

## Open source

This static checking is high up on Philips’ wish list as well, divulges Van der Munnik. “The main benefit for us at the moment is still the dynamic conformance checking while running the test cases, but maybe some of that can also be done statically. Furthermore, we want to extend the Comma framework with the ability to create smart stubs and simulators for clients and servers. And we’re looking into reverse-engineering interfaces by automatically constructing Comma models from execution traces – but this is still more in the research phase.” At Thales, Noltes is hoping to get Comma out of that research phase and into the modeling practice.

“As part of our work with Philips and Thermo Fisher Scientific, we’re extending Comma with the concept of components, ie objects with multiple interfaces,” states Hooman. “These interfaces are often inter-related, which means that if you do an action on one, the state of another changes as well. We’re developing a component that lets you express the relations and possibly the timing constraints between the interfaces. We’re also looking into testing multiple interfaces.”

To further the spread, the partners are working on open-sourcing the framework. Hooman: “We’re defining a kind of Comma core in the form of an Eclipse plugin, which others can extend, for instance with their own generators.” Van der Munnik underlines the importance of this development: “It adds to the maturity of Comma. What started as a research project is now a product that can actually be used by developers, in terms of UI, speed, ease of installation and so on. By making it open source, we’re hoping that others will contribute back into Comma, thereby extending and improving the framework even further.”

## Related