Overcoming System Design Challenges

The Creation of a System Description Language

Ingmar Ögren
Tofs AB

It is difficult for system developers to create designs that represent the system in a manner that is understandable, detailed, and useful enough to everyone who must give input to the design. Yet, no matter how well this obstacle is dealt with, it is still difficult to find and fix design flaws in written system descriptions. This article describes Odel (Object Design Language), a readable, "executable" system description language that can be mastered by hardware and software developers in a couple of hours and that can easily be explained to end users. Not only does Odel increase understanding among all players but also combined hardware, software, and user descriptions written in Odel can be electronically analyzed and debugged to allow designers to spot incompleteness, flaws, and inconsistencies long before the design is implemented.

Systems engineering requires that developers work at the system level, which means they must understand the requirements of and interaction between hardware, software, and end users. End users, who usually have little experience with systems development, need a detailed understanding of how they will be required to jointly operate with software and hardware modules to complete their missions; without this understanding, they may not be able to provide the quality of input needed during the design process.

As a result, systems engineers' problems largely concern barriers to understanding, as they must design and describe complex systems in a way that can be readily understood by software designers, hardware designers, and end users. To create a system description that is correct and can be analyzed, it helps if the description is both understandable and reasonably formal.

Evolution of Odel

The Ada programming language was introduced to the Swedish defense community about 11 years ago. To popularize Ada, the Swedish government contracted with Swedish contractor Sypro to create a simple Ada-based pseudo language to introduce Ada's semantics and syntax. The language, named "Adel" (Ada-based Design Language), was designed with simplifications of Ada, including the following:

- Tasking was simplified into a "concur" construct where a set of concurrent procedures are listed between the new reserved words "concur" and "end concur."
- Type and interface management was informal.
- The control structures from Ada were retained with some minor simplifications.

To support the language, an analyzer named Adela (Adel Analyzer) was developed to help write correct Adel. The Adel language was tried in several projects with varying degrees of success. Some experiences were

- People with a programming background could learn Adel in roughly two hours.
- Adel could be used not only for software design but also for logic design of hardware and operator parts of systems.
- The informal management of interfaces between objects (packages) made it necessary to perform manual checks on system description consistency.
- End users could learn to understand Adel descriptions if the description was given line by line.
- In software engineering, Adel helped sort out problems before programming commenced.

Development of Odel

Experiences with Adel were encouraging, which led to the vision of a system design language based on Ada 95. Although the new language was to provide system descriptions that could be used for programs written in any language, Ada was a logical foundation because it gives developers the ability to split a large system into packages, each of which contain a manageable and understandable part of the system with clear interfaces.

The idea was to create a language that was still simple enough to be taught to people with a programming background in a couple of hours and that was explainable to end users. At the same time, the language needed to be sufficiently formal that hardware engineers, software developers, and end users could check for consistency problems among the descriptions.

The resulting language, Odel, was developed by the private company Romet as part of the "complex systems" program sponsored by the Swedish Authority for Industrial Development. Romet retained ownership of Odel but decided to make the language publicly available without cost. Although Odel is derived from Ada 95, some constructs are different, and you cannot use Ada 95 tools to analyze Odel descriptions.

Benefits of the Odel Language

Detailed system descriptions written in the Odel can be understood not only by users and developers but also by tools that support the Odel language. This

gives system designers the ability to "execute" the system description line by line—identifying many potential problems long before the system is implemented. Following are some of the tasks in which Odel can assist.

- Reveal incompleteness and inconsistencies in the understanding of a system that may be hidden in natural language design descriptions.
- Model a system so that you can estimate its feasibility.
- Use operator role descriptions as a base for writing operator manuals and for analyzing possible operator behavior, including erroneous behavior.
- Check a design for syntax and consistency errors and correct them.
- Find design contradictions.
- Find undefined states in a design.
- Investigate a system's behavior under load without implementation of the system, which requires timing information for actions and capacity measurement for supporting hardware objects. This is best done with simulation first.
- Find out how system behavior depends on the operator's actions within the relevant part of the operator's behavior space.
- Create a basis to prepare tests for a system implementation, based on design understanding, possible inputs, and expected outputs.
- Create Fault Trees and Failure Mode Effects Trees for analysis of critical systems based on actions in Odel descriptions.
- Create a basis for definition of simulations.
- Compare software and hardware design descriptions with Ada 95 [1] VHDL (VHSIC Hardware Description Language [2]), and C++ (or Java) code. (This is a long-term objective for Odel, as it requires experience before implementation in a tool.)
- Transform Ada 95 source code into a design language description (long-term objective, probably requiring Ada Semantic Interface Specification [3]).
- Create a formal basis for building software tools for work with Odel descriptions.
- Teach Odel to software developers in a couple of hours.
- Explain Odel descriptions to end users, with full understanding.

Descriptions of operator's roles and hardware are special exceptions to the above, as they are not intended to be developed into code except for cases in which you want software to model hardware or operator behavior.

Overview of Odel

Overall Structure

Figures 1-3 show the overall structure of an Odel description. The central element in an Odel description is the *action*. The word "action" was chosen based on an idea from Professor Vitalis Sh Kaufman from Helsingfors, Finland, and it does not mean anything in the Ada programming language. The "action" represents Ada's procedures, functions, and tasks.

Figure 1 shows how an action contains interfaces and behavior and how it is supported by type definitions and by definitions of parameters, local variables, and messages.

Administratively, actions are grouped into the following objects (Figure 2).

- Ordinary objects – contain a number of actions to be kept together for design and review.
- Configuration Item object (CI object) – encompasses the amount of work suited for a small group during a limited time.
- Project object – encompasses the complete work within a development project.

Presuppositions for Odel

A number of presuppositions were formed as a basis for Odel:

- Any information system is constituted of one or more processes, which can be active in parallel (concurrently).
- Any process can be defined as an *action*, which has
  - An offered interface (action call).
  - A behavior, defined as a sequence of statements, that may include sending and receiving messages.
  - A required interface toward other actions (optional).
- Information system actions can be implemented as
  - operator actions.
  - software actions.
  - hardware actions.
- Each information system aims at fulfillment of one or more missions, with the relevant actions defined for each mission.
- Each mission is completed through one or more actions.

Figure 1. *Overall Odel structure.*

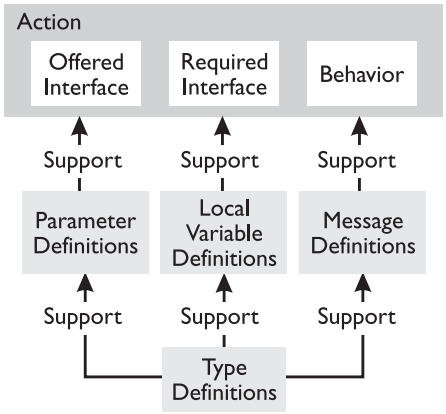


Figure 2. *Action and object hierarchy.*

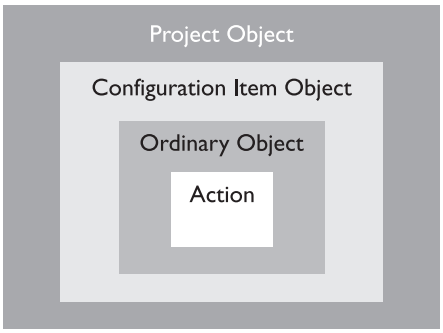
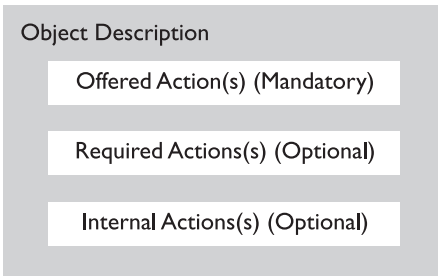


Figure 3. *Object description.*



- Ada 95 qualifies as a formal base for part of the definition of the Odel language.

The Object with Its Interfaces

An ordinary Odel object is basically a container of actions (Figure 3):

Offered action – an action that can be invoked by an action outside the current object. "Outside" then includes other objects in the current CI and any object in its environment. At least one offered action is mandatory, as each object must have an offered interface. Parameters are defined in connection with actions.

Required action(s) – an action that the current object requires from objects outside the current object to complete its own actions. The required action(s) are optional, as "bottom-level" objects will not have any required interface. However, this entry is mandatory for all objects that have support objects. The required action names are qualified in action calls in two forms:

- CI_name.Object_name.Action_name (*parameters*), for actions defined in an object in a foreign CI. Objects in foreign CIs are called "attached objects."
- Object_name.Action_name (*parameters*), for actions defined in another object in the same CI as the current object. Objects within the current CI are called "contained objects."

Internal actions – are only invoked from other actions within the current object.

The textual form for an object is

object current_object_name is
required actions : list of qualified names of actions
offered action : list of names of actions in current object with visibility = offered
internal actions : list of names of actions in current object with visibility = internal list of action descriptions
end current_object_name

The Odel Definition

Because Odel is based on Ada 95, it was defined through copying and referencing the Ada 95 Language Reference Manual [1]. The main part of the language is merely a subset of Ada 95. However, some extensions were needed to describe parallel processes. An example from the Odel definition is shown for the "Send" and "Receive" statements:

Send Statements

A send statement assigns a value to a message and makes it globally readable.

send_statement ::= send message_name (expression)

The message denoted message_name must be declared, and the expression must be of the type of the message.

Example:

In action header:

Messages: notification : string

In Odel description:

send notification ("OK")

Receive Statements

A receive statement retrieves a value from a message sent by another (or possibly the same) action and assigns it to a variable, an out parameter or an in_out parameter.

receive_statement ::=
receive message_name (variable | parameter)

The message denoted message_name must be declared. The type of the variable or parameter must be the type of the message. For example:

receive notification (answer).

In real-time systems, you sometimes need to wait for a new message and then receive the new message. This can be expressed with a comment. Another, more formal possibility is to construct a loop and compare received messages until a new message is found. This possibility is, however, not normally recommended, as the loop may become obscure and unnecessarily prescribe a detailed programming solution.

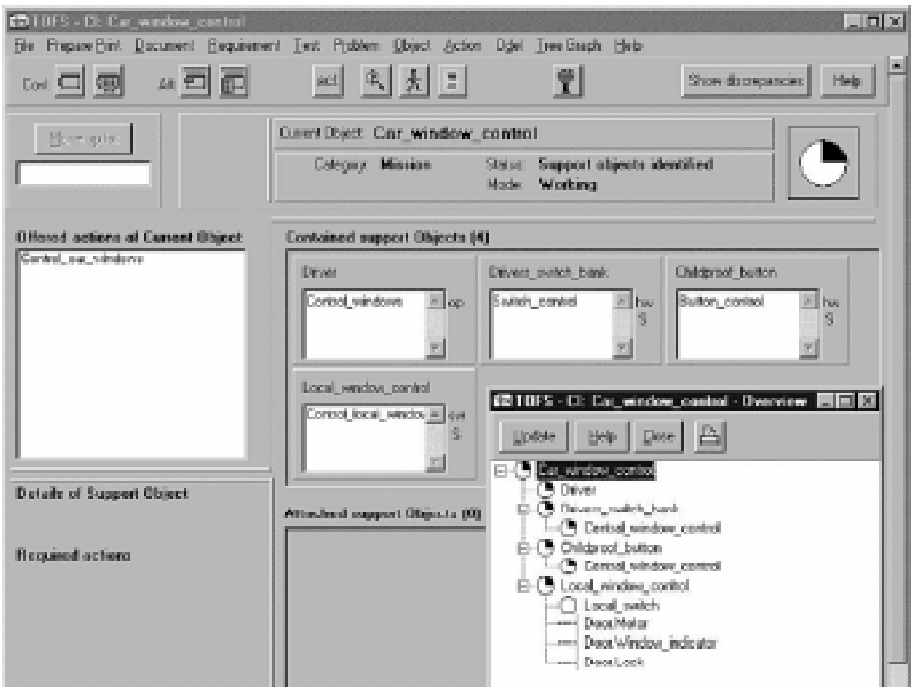
Tofs, the Odel-Based Tool

The Odel language was used as a foundation to create a system engineering tool, Tofs (TOol For Systems). Figure 4 shows Tofs' main screen with a "tree graph" window. This main screen allows you to work with the system structure and allows you to reach different parts of the Tofs toolkit.

To work with Odel, you select an action and open an editing window. After you have written some Odel, you can invoke the analyzer, which will then analyze the Odel description and then show any error messages by highlighting the lines where errors were found.

Figure 5 shows the Odel editing window, with a message from the analyzer. Tofs further includes an "execu-

Figure 4. *Tofs main screen with automobile example.*



tor" (not shown in the figures), which is basically an Odel-level debugger, which shows concurrent actions in multiple windows in parallel.

Use of Odel and Tofs

With Odel and Tofs, the basic semantics and parts of the syntax of Ada 95 can be applied not only to software but also to complex concurrent systems where operators jointly operate with software and hardware modules to complete missions. Odel and Tofs are intended to support a system engineer in tasks such as

- Analyzing a system in its environment, with clarification of how the system communicates with and represents its environment.
- Designing a system as a set of objects that depend on each other through interfaces.
- Analyzing requirements and distributing fulfillment requirements to objects with listing of requirements with tracing.
- Managing problems that surface during system development.
- Analyzing a design concerning syntactic correctness and consistency.
- Reviewing a system, using the Odel "executor."
- Documenting a system, according to relevant standards, such as the EIA/IEEE 12207.
- Reengineering code (manually) into an Odel system description.
- Analyzing a system's dependability, using fault tree analysis and failure mode affects analysis.

Although Odel descriptions look similar to Ada code, they are not code but are a detailed and formal description of a system, including its software parts. To go from an Odel description to code, you can copy the Odel text into a source code editor and mark the text as comments. The Odel description will then be a detailed specification for the code to be written (in Ada or some other language).

Automatic generation of code has not been attempted because I believe that programming is professional work, which is best done by a competent programmer who understands what is required.

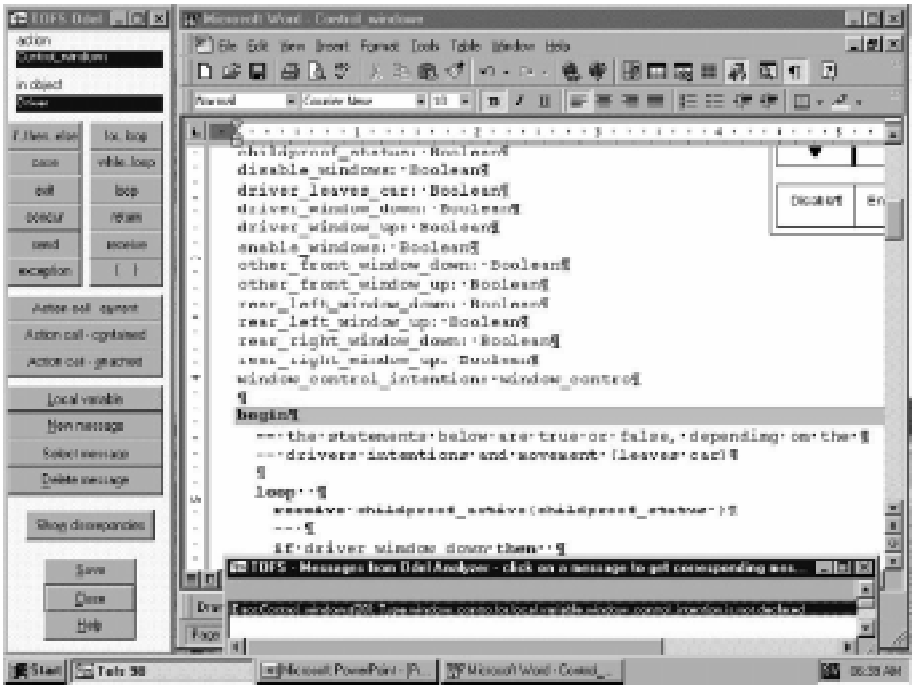


Figure 5. Odel editing window with error message.

Ownership, Availability, and Experience

The Tofs toolkit was developed by the Tofs AB company. The toolkit is available in two versions:

- A free version, with full functionality but limited capacity, intended for systems engineering education and for evaluation. It can be downloaded from http://www.toolforsystems.com
- A commercial version.

Courses in systems engineering with Tofs are held regularly in Sweden. Course length is three days. Some courses have also been held in the United States, and courses can be arranged on request by Abelia in Fairfax, Va. [4]. Some educational information also is available for download from http://www.toolforsystems.com

As the work with Ada-based systems engineering has been going on for more than 10 years, some experience has been gathered from various projects such as automobile industrial systems, military ship modeling (submarine), development of an Army radio communication system, structuring of a tactical aircraft simulator, and others. Following are some experiences from these applications.

- Some end users readily accept and understand the formality of Odel without additional explanation from system developers, while the reaction of others is "I do not read code."
- Using the Odel language without an automatic analyzer requires much tedious work for the reviewers. Tofs eliminates much of this type of work.
- Odel is useful when you want to document existing software, for example, for reengineering from FORTRAN into Ada.
- Inclusion of "mission objects" in the system structure helps to keep the priorities right and to understand how different parts of the system contribute to completion of missions.
- Inclusion of operator roles as objects in the system structure helps in optimizing human-machine interfaces.

Conclusion

Odel has created the ability to structure complex systems as a set of ob-

jects, connected through their offered and required interfaces.

Experience from multiple projects shows that work with complex systems should be assisted by software tools, because the complexity results in more information than can be managed manually. Some 10 years of experience of Ada-based systems engineering has been used to implement the Tofs toolkit. The result is a comparatively compact tool to assist systems engineering tasks from requirements management to documentation.

The main objective for Odel was to create a language that qualifies as formal while being understood by both end users and developers. This objective seems to have been reached, but more experience is needed to attain the full potential of the language. ◆

About the Author

Ingmar Ögren is president of Tofs AB in Sweden. His systems design experience began at the Royal University of Technology, Electronics Division, where he participated in pioneering military aircraft simulator systems. He later was active in the development of data transmission elements for air defense systems and the management of a multi-industry, multiprocessor airborne computer system. As a private company owner, he worked on civilian systems and helped introduce Ada to the Swedish defense community and helped use Ada to analyze, design, and document systems in general. This work led to his development of the O4S™ method used in large systems such as submarines and aircraft. He and his wife Anna used their O4S experience as a foundation for Tofs.

E-mail: iog@toolforsystems.com
Internet: http://www.toolforsystems.com

References

1. Ada 95 Reference Manual, ISO/IEC 8652, 1995.
2. IEEE Standard VHDL Language Reference Manual, ANSI/IEEE STD 1076-1993.
3. Information about ASIS is available from the URL http://www.ci.pwr.wroc.pl/cgi-bin/plcon/winHiso/www.acm.org/sigada/wg/asiswg.
4. Abelia is at http://www.abelia.com.

Mark Your Calendars Now for the Eleventh Annual Software Technology Conference

May 2–6, 1999, Salt Lake City, Utah

It's not too early to start making plans to attend STC '99. Judging from past conference attendance, the sooner you act the better. Contact us for the latest information on conference registration, exhibits, housing, and activities in the host city. If you have never attended—or if you want to know how this year's event will be different—we'll give you an update on speakers, networking opportunities, general sessions, tutorials, presentation tracks, and exhibitors.

The United States Air Force, Army, Navy, Marine Corps, and the Defense Information Systems Agency have again joined forces to co-sponsor STC '99, the premier Software Technology Conference in the Department of Defense. Utah State University Extension is the conference non-federal co-sponsor. We anticipate over 3,500 participants from the services, other government agencies, contractors, industry, and academia.

Registration for exhibit space opened Aug. 3, 1998. Reservations for exhibit space are processed on a first-come, first-served basis, by either mail or fax. Many exhibitors have already made reservations, and booth space is filling fast. New exhibitors will find background information on the Software Technology Conference, its history, and attendance statistics helpful in planning for the conference. This information, along with an updated exhibit hall layout, including assignments and organizations registered to date, will be maintained on the Internet. You can access this information at http://www.stc-online.org or from our STC '99 fax-on-demand line, 435-797-2358, item 303.

It is recommended that you make reservations for your hotel guest room as soon as possible. For your convenience, a Housing Reservation Form has been included in the center insert of this issue of *CROSSTALK*. In addition to the housing form, a listing of the potential hotels is available at our Web site and from our fax-on-demand line, 435-797-2358, item 200. Although this year more government- rate rooms have been made available in the immediate vicinity of the Convention Center, they will fill up quickly, so book early.

For more information, please visit our Web site at http://www.stc-online.org, call the management team at 801-777-7411, DSN 777-7411, or send an E-mail to Dana Dovenbarger or Lynne Wade at wadel@software.hill.af.mil. We look forward to seeing you next May.