User-Oriented Descriptions of Smalltalk Systems

Trygve M H Reenskaug
Central Institute for Industrial Research
Blindern, Oslo 3
Norway

For many people, the workings of a computer remain a mystery. Just exactly what the computer does and how it does it is locked within the code of a computer language. The computer and the user understand two completely different languages. It is well known that only a few systems are designed and written so that they can be understood by the user. More than twenty years of experience has shown that a bad system design can never be hidden from the user, even by a masterfully devised user interface. A quality system, therefore, must be based on sound design that can be described in terms with which the user is familiar.

The Smalltalk system has been designed to handle a great variety of problems and solutions. It, therefore, provides the greatest possible flexibility for writing any kind of system a programmer may desire. While this flexibility is essential for experimenting, there is the potential for disastrous results if restrictions are not put on the system structures that are available to the application programmer.

This article shows how the basic metaphors of Smalltalk can be used to describe complex systems. Since this magazine is not yet distributed in a form readable by Smalltalk, we have to restrict ourselves to traditional written documentation. (Let it be a challenge to Smalltalk experimenters to convert this presentation into a graphic and dynamic one.) The Smalltalk system user will most likely employ his system to organize the large amount of information that will be available to him.

More than twenty years of experience has shown us that a bad system design can never be hidden from the user, even by a masterfully devised user interface.

such as reference materials in the form of market information, news services, and weather forecasts. Some data, such as travel information and bank transactions, may flow both to and from the owner. Other information, such as personal notes or material that is not yet ready for distribution, can remain private.

An individual's total information needs are very large and complex. His Smalltalk system, therefore, is also likely to be large and complex. The challenge to the Smalltalk experimenter is to find ways to structure systems so the user will not only understand how to use them, but also get an intuitive feel for their inner workings. In this way, the user can really be the master and the systems his faithful slaves.

An important part of any system is the software that controls the user's interaction with the information. Mastering the software is crucial to handling the information. With Smalltalk, software is just a special kind of information and is treated as any other information within the total system. It is available to the user in the usual manner.

A traditional way of describing software is through written documentation. Smalltalk provides more dynamic interfaces through the use of two-dimensional graphics and animation on the computer screen. Devising such interfaces is probably the greatest challenge in personal computing today, and it provides a rich field of endeavor for the interested experimenter.

System Descriptions

We can describe any application system in three different ways: how it is used, its system structure, and its implementation:

- How it is used—This is the least satisfactory type of description. The user operates the system through rote command sequences such as: switch on the machine, type your password, hit button A, listen to your system saluting you by playing "Hail to the Chief." Since 80% of all user manuals for electronic data processing systems
are of this kind, we will not discuss them further here.

This level of understanding has been likened to walking around in a strange city following directions such as: “Go outside, turn right, walk straight ahead for four blocks, turn left . . . ” It is easy to get lost under such circumstances.

**System structure**—With this type of description, the user has an intuition about the kinds of building blocks that make up the system, how they behave, and how they interact to form the complete system. We show that the basic Smalltalk metaphors of objects and messages are well suited to function as building blocks. The metaphors are simple and easy to understand; yet they permit construction of immensely powerful systems.

---

**A basic system will have several thousand objects, and typical applications would contain many more.**

Any Smalltalk system contains a large number of objects. A basic system will have several thousand objects, and typical applications would contain many more. The common software engineering device of layering becomes essential in making the whole thing manageable. In the description of a layer, essential function on that level is highlighted and consequential detail is relegated to lower levels. There is one absolute requirement of these simplified descriptions appearing on the different layers: what is shown should be correct and complete as far as it goes. This means that the structure of the description has to be a pure tree structure: the function of each module has to be limited to that module with no hidden side effects upon the other modules.

This level of understanding corresponds to the user having a street map of the system. He knows the major landmarks and the most important streets. This gives the user the intuition about the total structure and permits him to find his way anywhere where. It is almost impossible to get totally lost under these circumstances.

**Implementation**—Descriptions at this level of understanding explain what the user must do to get each object to work as designed. A method is similar to a subroutine in other languages; it prescribes the actions to be taken by an object when it receives a message. On all layers but the lowest, the behavior of an object is fairly complex, and we can think of it as composed of a number of sub-objects that are used to implement it. The purpose of the method is to enlist the aid of the sub-objects to implement the desired behavior. The user thus finds that the typical object is structured in much the same manner as his total system, and it consists of a number of sub-objects that send messages to each other. The description tool is recursive in that the same tool is used on all levels. This recursion description is probably the most powerful feature of Smalltalk. Once the user masters the few very general concepts, he can learn more and more about his system by simply using these concepts to dig deeper and deeper into the system layers. In addition, the user can modify and expand the system on any level by collecting new components out of the building blocks provided by the next level below it.

The user at this level now has an intuition of the overall layout of the city. He also has sub-maps of all the details and he knows how to read them. Depending on his personality, he may use these maps only when absolutely necessary, or he may use them to explore unknown territory. In contrast to the tourist, the Smalltalk user can even make modifications and new extensions to the city. The tools are there. The user decides if, when, and how he wants to use them.

**Example of a System Description**

*The problem:* Consider a small manufacturing company that has two...
departments: sales and production. The responsibility of the sales department is to find customers for any product the company can make, to contact the production department to find out when the product can be delivered, and to sign a contract with the customer. The responsibility of the production department is to manufacture each product as cheaply as possible at a specified level of quality and to have it finished on the promised date. When the production department has manufactured the product, it is dispatched to the customer through the sales department.

The system: A natural way to map this into a Smalltalk system would be to represent each department as an object. The function of the Sales object would be to keep track of the state of each sale in the following sequence:

1. Fill in and send proposals
2. Reserve the necessary resources in production for the product
3. Send contracts and packing notes to the customer

The function of the Production object would be:

1. Keep track of commitments
2. Schedule the manufacture of products
3. Help keep the product quality
4. Control the manufacturing process to get the products completed on time

It also seems reasonable to include a third kind of object in our system: Customer objects. The purpose of these objects would be to act as a receptacle for the messages being passed from the company to the customer and from the customer to the company. The various objects with a set of reasonable communication channels is shown in figure 1.

The overall processing of an order: The Smalltalk system would be programmed to reflect everything of importance that takes place during the processing of an order and to support its user on every step. The process that takes place inside the Smalltalk system would, therefore, closely resemble the actual processing of an order. Let us assume the following real-life process, which is depicted in figure 2. A customer submits an intention to buy, a request for offer, to the company. The sales department books resources from the production department and returns an offer with the cost and delivery date to the customer. The customer answers with a purchase order. This is transcribed and passed from sales to production as a requisition. The product is manufactured in production, and a ready-note is sent to sales, which arranges for transport and sends packing notes to the customer.

In the Smalltalk system, the Sales object would help the user of the system in corresponding with the customer, in keeping track of progress, and in sending the required forms to the production department. The Production object would help the user in the planning and control of the manufacturing process.

In order to highlight the principles, we have made this a very simple system. The reader will have no difficulty in expanding it, for example, by adding an object for the accounting department that takes care of bill-
Figure 2: The processing of an order. The Smalltalk system supports this processing through interaction with its owner in real-time.

An Implementation Description

Let us inspect the Production object of figure 1 and see how it processes the message `bookProductionFacilities:after:`. When this message is received by the Production object, it consults its message dictionary to find the corresponding method. If the products were simple and the workshop small, the object could contain the current production plan directly and the method could go something like that shown in listing 1.

One of the instance variables of the Production object is the table `productDuration` which contains the time it takes to manufacture various products. Looking at this table, we find the duration for a product. In this simple example, there is only one resource, and we find the first available time slot for the product by sending self the message `findFirstFreePeriod:after:`. This corresponds to calling a local subroutine in other systems. We then reserve the resource for our product in that period. (These two steps could have been combined into one, but the separation gives us more flexibility in varying the planning algorithm if we wish to do so later.)
New: SS16K/IEEE RAM

It's everything you need in a 16K static RAM board — at the lowest price you've ever seen. The SS16K/IEEE comes with all the high performance features listed below. And unlike obsolete-design RAM's (without bank select) you can add-on our universal software bank-selector system anytime, now just $20.00. This makes the SS16K/EEE capable of addressing 2,048 different banks. You can add memory beyond the 64K limit. You can expand to a multi-terminal system.

FEATURES OF SS16K/IEEE:
- Low-power 2114's
- All inputs and outputs meet the proposed IEEE standards for the 5100 bus.
- 4.0 MHz operation.
- Schmitt trigger buffer on all signals for maximum noise reduction.
- Addressable on 16K boundaries, 0-64K, dip switch selectable.
- Phantom option, dip switch selectable.
- PWR/MWRITE option, dip switch selectable.
- Glass epoxy PC board with gold-plated contacts and double-sided solder mask.
- Fully socketed.
- Four separate regulators, for maximum stability.

WITH BANK SELECT OPTION (now just $20.00) YOU ADD THIS:
- Software bank selector featuring a universal decoder works with Cromenco, Alpha Micro, Netronics, most other systems, or your design.
- On-board dip switches: Bank Select Enable, Reset Enable, Reset Disable, Port Address, Port Data.
- LED Indicato to display status.

10-DAY MONEY-BACK TRIAL: Try a fully wired and tested board for 10 days — then either keep it, return it for fit, or simply return it in working condition.

TO ORDER CALL TOLL FREE 800-243-7428
From Connecticut or for Assistance: (203) 354-8975

Please send the items checked below:
- SS16K/IEEE without bank select: $175.95
- SS16K/IEEE with bank select: $195.95
- SS16K/IEEE bank select option: $20.00

Plus $2 postage & Insurance ($5.00 Canada). Connecticut residents add sales tax.

Total enclosed: $_________

Personal Check □ Money Order or Cashier’s Check □ Visa □ Master Card (Bank No. ______)
Acct. No. ______ Exp. Date ______
Signature __________ Name __________
Address __________ City __________ State __________ Zip __________

Listing 1: Smalltalk method for the message bookProductionFacilities:after:

bookProductionFacilities: productType after: earliestStartTime
"Reserves production facilities for a new product of given type as soon as possible after the specified earliest starting time. Returns the planned completion time for the product."
<table>
<thead>
<tr>
<th>duration plannedStartTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration — productDuration at: productType.</td>
</tr>
<tr>
<td>plannedStartTime — self findFreePeriod: duration after: earliestStartTime.</td>
</tr>
<tr>
<td>self reservePeriod: duration from: plannedStartTime.</td>
</tr>
<tr>
<td>1 (plannedStartTime + duration)</td>
</tr>
</tbody>
</table>

Listing 2: Alternate Smalltalk method for the message bookProductionFacilities:after:

bookProductionFacilities: productType after: earliestStartTime
"Reserves production facilities for a new product of given type as soon as possible after the specified earliest starting time. Returns the planned completion time for the product."
<table>
<thead>
<tr>
<th>productIdentification</th>
</tr>
</thead>
<tbody>
<tr>
<td>productIdentification ← jobManager defineProduct: productType.</td>
</tr>
<tr>
<td>jobManager schedule: productIdentification after: earliestStartTime.</td>
</tr>
<tr>
<td>1 (jobManager plannedCompletionTime: productIdentification).</td>
</tr>
</tbody>
</table>

Figure 3: The internals of the Production object.

The planned completion time is returned to the sender, in this case the Sales object.

Lower-Level System Description

If the user wants more advanced aids for production control, the Production object would call upon the services of a subsystem of interconnected objects. A possible subsystem is shown in figure 3.

The entrance to the internals of the Production objects is through a Production Manager object; it is connected to a Job Manager object and a Resource Manager object.

The manufacturing of a product is split into a number of jobs. The available production facilities (people and machines) are split into a number of resources. Each job is to be performed by a single resource. A natural way to map this into a Smalltalk system is to represent each job by a Job object and each resource by a Resource object.

In this scheme, each Job object ensures that the job is performed by its
resource within the available time. Similarly, each Resource object ensures that its resource is used in an efficient manner, that there is sufficient time available for preventive maintenance, and that there are no unacceptable overloads. The method in the Production object that handles the bookProductionFacilitiesAfterMessage could now be written as shown in listing 2.

One of the instance variables of the Production Manager object is a pointer to the Job Manager object. By using that pointer as a communication channel, the Production Manager object passes most of the work on to the Job Manager object. First, the Job Manager is asked to define the new product. The Job Manager creates the Job objects (see figure 3), links them to the proper Resource objects, and returns an identification that is to be used for future references to the product. The Job Manager is then asked to schedule the product for manufacturing as soon as possible after the given date. Finally, the Job Manager is asked when the product will be completed, and this value is returned to the outside world (in this case, the Sales object). The planning process in the Production subsystem that is shown in figure 4 is controlled by this method.

**Definition of New Objects**

The first task of the Job Manager object is to define the new object. It receives message definition when this is to be done. The corresponding method could be something like that shown in listing 3. We are referencing two instance variables of the Job Manager object in this method: productDescriptions and productionManager. productDescriptions is an ordered collection with one member for each product type. Each of these members contains a sequence of small objects with the class, duration, and resource type for each of the jobs that go into the manufacture of such a product. productionManager contains a pointer to the Production Manager object. The result of the product creation is put into a third instance variable, the productDic-
MMFSORTH VERSION 2.0:
MORE FOR YOUR RADIO SHACK
TRS-80 MODEL I OR MODEL III!

★ MORE SPEED
10-20 times faster than Level II BASIC.

★ MORE ROM
Very compact compiled code plus VIRTUAL
MEMORY makes your RAM 10x larger. Variable
number of block buffers, 32-character unique
names use only 4 bytes in header.

★ MORE INSTRUCTIONS
Add YOUR commands to its 70-STANDARD plus
instruction set. Far more complete than most Fortes: single &
double precision, arrays, string-handling, clock,
more.

★ MORE EASE
Excellent full-screen Editor, structured &
modular programming
Word search utility
NOTES/ASCII editor
Optimized for your TRES-20 with keyboard
repeats, upper/lower case display driver, full
ASCII, single & double-width graphics, etc.

★ MORE POWER
Forth operating system
Interpreter and compiler
ROM Assembler
(L200 Assembler also available)
Interacts 35-60 times faster than
Model IV system you can read, write & run Model I
and II.
VIRTUAL 20 for video display and printer, disk and tape
(10 Megabyte hard disk available).

THE PROFESSIONAL FORTH
FOR TRS-80
(Over 1,500 systems in use)

Price:
MMFSORTH Disk System V2.0 requires 1 disk drive &
16K RAM, 32K for Model III)...
$129.95* 
MMFSORTH Cassette System V2.0 (requires Level I
BASIC & 16K RAM)...
$59.95*

AND MMFSORTH GIVES IT
PROFESSIONAL SUPPORT

Source code included
MMFSORTH Newsletter
Many utility programs included
MMFSORTH User Groups

Interactive upgrades to latest version
Programmer’s staff can provide advice, modifications
and custom programs to fit YOUR needs.

MMFSORTH UTILITIES DISKETTE includes:
FLOATING POINT MATH (2 BASIC ROM routines plus
Complex numbers, Rectangle Polar coordinate conversions,
Degree mode, more, plus a full Forteside 263 ASSEMB-
LE, plus a powerful CrossthREADER to list
Forth words by block and line. All on one cassette
requires MMFSORTH V2.0, 1 disk drive & 32K RAM.

FORTHCOM communications package provides RS-232
driver and terminal mode capable of FORTH BIOS
and host mode to operate a remote TRS-80 requires
MMFSORTH V2.0, 1 disk drive & 32K RAM.

THE DATAMANAGER V1.2, a very sophisticated data
base management system operates by nonpro-
grahmics requires MMFSORTH V2.0, 1 disk drive & 32K
RAM. 

MMFSORTH GAMES DISKETTE, realtime games &
DOS games without diskette includes BREAKTHROUGH
and ARISTOCRAT. Requires MMFSORTH
V2.0, 1 disk drive & 32K RAM.

Other MMFSORTH products under development

FORTH BOOKS AVAILABLE

MICROFORTH PRIMER - comes with MMFSORTH-
Primer...
$15.00*
USING FORTH - more detailed and advanced than above...
$20.00*
INVITATION TO FORTH - detailed beginner book
on FORTH...
$17.00*

THREADED INTERPRETIVE LANGUAGES - advanced,
excellent analysis of MMFSORTH line-
guages...
$15.00*

PROGRAM DESIGN & CONSTRUCTION - intro. to struc-
tured program, good for FORTH...
$9.00*

FORTH: THE STANDARD MANUAL, official reference
to 70-STANDARD words, etc...
$10.00*

CALFORTH PORTMANUE: a good on both internal
structure, etc...
$15.00*

FORTH SPECIAL ISSUE, BYTE Magazine (Aug., 1980),
we stock this collector's item for FORTH users and
beginners...
$4.00*

ORDERING INFORMATION: Software prices include
manuals and require a signed copy of single system, single
license. SPECIFY Model I or Model III. Add
$2.00 S&H plus $1.00 per additional book. Mass. orders
add 5% tax. Foreign orders add 20%. UPS COD, A.C.O.D.
& MG accepted; no unpaid purchase orders, please.

Send SASE for FREE MMFSORTH information.
Great deals sought.

Get MMFSORTH products from your
dealers or

MILLER MICROCOMPUTER
SERVICES (88)
81 Lake Shore Road, Natick, MA 01760
(617) 653-8180

Figure 4: A simple planning algorithm implemented in a Smalltalk system.

Listing 3: Smalltalk method for the message define.Product:

**defineProduct: productType**

"To create a new product of given type. The corresponding Job objects are created and linked to their resource objects."

[productIdentification jobDescriptions jobList resourceObject]

**productIdentification = self nextProductIdentification.**

**jobDescriptions = productDescriptions at: productType.**

**jobList = jobDescriptions collect: [:description |**

**job = (description class) new.**

**job duration: (description duration).**

**resourceObject = productManager getResource: (description resourceType).**

**job resource: resourceObject.**

**productDictionary at: productIdentification put: jobList.**

**1 productIdentification.**

ictionary. In this dictionary, each key is a product identification; the corresponding entry is the sequence of job objects for that product.

The first line of code gets a new, unique identification for the new product. Next, the list of job descriptions is retrieved from the productSpecification collection. We then build the sequence of Job objects by going systematically through the job descriptions. For each description, we create a new Job object of the given class, feed it its duration, and let it
**SOFTWARE DEVELOPMENT TOOLS FOR INDUSTRY**

**CP/M CROSS-ASSEMBLERS**
Fast, comprehensive cross-assemblers to run under CP/M. Extensive pseudo-ops include full listing control, nested conditionals, mnemonic synonyms, and inclusion of external source files. Generate object files, assembly listing, and symbol table from source code for nine popular microprocessor families.

XASM05 .... 6805
XASM09 .... 6809
XASM18 .... 1802
XASM48 .... 8048
XASM51 .... 8051
XASM65 .... 6502
XASM88 .... 6800/6801
XASM88F .... 68/3870
XASM400 .... COP400
Assembler ... $200.00 each
Manual only ... $25.00

**8048 DEVELOPMENT PACKAGE**
Now you can use the 8048 family of single-chip microcomputers without buying expensive equipment. Develop 8048 software with the XASM48 cross-assembler. Then plug our EPR-48 board into your S-100 system to program the 8748 EPROM version. (Similar packages for 8051 and TMS9900E coming soon.)

8048 Development Package ... $574.00
EPR-48 alone ... $449.00

**PROM EMULATOR BOARD**
Debug dedicated systems quickly. Our PSB-100 PROM Emulator is an S-100 board with up to 8K of RAM. Cable with 24-pin plug replaces PROM(s) in your target system for instant program testing.

PSB-100 PROM Emulator ... $445.00 w/ 2K RAM

---

**Figure 5:** Sub-objects in the Job Manager actually create the new Job objects.

![Diagram](image1)

**Figure 6:** All objects contain a pointer to a Class object that contains their message dictionary and methods.

![Diagram](image2)

**Figure 7:** The superclass-subclass chains of pointers. The user does not meet them unless he wants to become a real Smalltalk expert.

![Diagram](image3)

connect itself to its Resource object. From figure 3, we see that there is no direct connection between the Job Manager object and the resources. We therefore have to go via the Production Manager object to get the pointer to the Resource object that we give to the new Job object.

We finally insert the new list of jobs into the productDictionary in the Production Manager object and return the product identification.

The Job Manager is built so that Job objects may belong to several different classes. The different Job objects created would all understand the
same message protocols, but they would differ in their implementation. For example, a job might be: "wait for 24 hours while a resin glue is curing. This does not need any resources, and the planning of such a job would be very simple—wait 24 hours. Another kind of job, such as pouring concrete, should not span a weekend, since joining old and new concrete could give weak spots in the product.

As is the case with Job objects, we often find that several objects share the same message protocols and process the messages with the same methods. Their only difference is that they appear in different places in the total system and that their instance variables point to different objects (their states are different). Such objects are created by the same class object and are said to belong to the same class.

It would be very inefficient if each object of a class stored a replica of the message dictionary and all methods, and it would be extremely tiresome if we actually had to program each object by itself. We, therefore, use the concept of layering to let each and every object enlist the services of its class object in order to decode an incoming message and to select the proper method to process it. This mechanism is illustrated in figure 6. As in so many other parts of Smalltalk, we find a recursive argument.

Many classes are very similar; they differ only in the handling of a few messages. The different kinds of Job objects are a case in point. It seems reasonable to let a class object enlist the services of a superclass object whenever it is called upon to execute methods it shares with other classes.

The Smalltalk user should be able to "open up" the application object on the screen to see its component parts and to find out how they work together.

---

A Program to Recover "Crashed" Discettes AUTOMATICALLY!

Maybe it was a lightning storm, static from the rug, or just too late at night to be working. Whatever the cause, when a discette "crashes" and valuable data or programs are destroyed, the loss is enormous, both in time and money.

DISK DOCTOR is a program which automatically recovers bad discettes. Best of all DISK DOCTOR does not require any knowledge of CP/M file structure! If you can operate CP/M, then you can use DISK DOCTOR. The entire system is menu driven with key information displayed.

DISK DOCTOR is comprised of five "wards", each capable of performing a specific discette recovery operation.

- **Ward A**: Verifies discettes and looks out bad sectors without touching the good files that remain.
- **Ward B**: Copies whatever can be read out from a "crashed" file and places it into a good file under user control.
- **Ward C**: Copies discettes without stopping for bad sectors. Bad sectors are replaced by spaces.
- **Ward D**: "Un-erases" files. That is, Ward D will recover accidentally erased disk files.
- **Ward E**: Displays directory of recoverable erased files.

DISK DOCTOR will pay for itself the first time it is used.

Best of all, DISK DOCTOR operates almost complete automatically. The small amount of user interaction is explained in the manual as well as prompted by DISK DOCTOR.

Requires: 48K CP/M, two drives needed for complete operation.

| DISK DOCTOR | $100.00 |
| Manual alone | $ 10.00 |
| CP/M Format: 8" soft sectored, 5" Northstar, 5" Micropolis Mod II, Vector MZ, Superbrain DD/QD, Apple II |

---

SuperSoft
First in Software Technology
Many classes will then share the same superclass; we get a tree-shaped class structure as shown in Figure 7. Note that the purpose of this structure is convenience in programming and efficiency in implementation; it belongs on the lowest levels of the system hierarchy and is not part of the structure of the application system.

**Future Experiments**

When personal computing becomes sufficiently entertaining and interesting to become a widespread tool, the new user of a Smalltalk system is likely to begin by using its ready-made application systems for writing and illustrating documents, for designing aircraft wings, for doing homework, for searching through old court decisions, for composing music, or whatever. After a while, he may become curious as to how his system works. He should then be able to "open up" the application object on the screen to see its component parts and to find out how they work together. He could, for example, see something like Figure 1 together with his usual user interface. By exercising the application commands, the computing process could be illustrated on the system diagram. Using Smalltalk to document itself in this manner should make it possible to make some novel and extremely powerful system description tools.

The next thing the user might want to do is to build new systems similar to the one he has been using. A kit of graphical building blocks would let the user compose a new system by editing the system diagram on the screen. While the Trip system (as described in reference 2) is not a proper kit, it could be a good source of ideas to the experimenter on building such systems.

Finally, the expert user would want to make his own kits. Even here, it is important that he sees only what he needs and that all unimportant details are suppressed. Since what is important in one context might be unimportant in another, and vice versa, the concepts of filters (see reference 1) will be an essential ingredient for the experimenter when he develops tools for these expert users.

Much experimenting needs to be done before we learn how to make systems that are self-documenting on any level and that provide a smooth and stumble-free transition from one level to the next. It is hoped that the availability of Smalltalk will lead to great activity in this field, to the benefit of all future computer users.

---

**References**

