THE STATE OF THE ART IN CACSD A DECADE AFTER ECSTASY

Christopher P. Jobling * and **Thomas Varsamidis** **

* *Department of Electrical and Electronic Engineering, University of Wales Swansea, UK.* C.P.Jobling@Swansea.ac.uk
** *School of Informatics, University of Wales, Bangor, UK.*
thomas@sees.bangor.ac.uk

Abstract: In the mid 1980s, the UK Science and Engineering Research Council funded a number of projects to develop computer-aided control system design (CACSD) environments. The flagship project was ECSTASY (Environment for Control System Theory and SYnthesis), a database-mediated framework for hosting a number of popular control systems design tools. As the ECSTASY project neared its completion, some of the most urgently requested features appeared in rival commercial packages, and the short-falls in tool provision that ECSTASY was designed to address apparently went away. Using as a benchmark the capabilities of the current generation of the popular CACSD package MATLAB, the prescience of the original ECSTASY proposal will be demonstrated. A couple of areas that the environment excelled in, the use of a common information model and common user interface, will be discussed in the context of recent work done in the University of Wales and elsewhere. Finally, two topics of particular interest to the authors, the proposed standard reference model for computer-aided control engineering and an Object-Oriented Unified Information Model for CACSD will be revisited in the light of recent developments in internet computing. *Copyright © 2000 IFAC*

Keywords: Computer-aided control system design, computer-aided control engineering, design environments, object-oriented programming, framework reference model, internet computing.

1. INTRODUCTION

In 1985 the Control and Instrumentation (C&I) subcommittee of the UK's Science and Engineering Research Council (SERC) commissioned a survey of the then available commercial and well-developed academic software packages available for computer-aided control system design (CACSD) and found them wanting. A year later the C&I committee funded the development of a new CACSD package which was to address the perceived deficiencies. This package, which became known as ECSTASY (Environment for Control System Theory and SYnthesis), was a database-mediated framework for hosting a number of the most popular control systems design tools (Munro and Jobling, 1993). ECSTASY contained a number of the best features advocated by the CACSD literature: a common command language, graphical user interface, support for both symbolic and numerical tools, embedded simulation tools based around a graphical system editor, and extensive algorithms for advanced control which had rarely been used outside university research laboratories. The whole system was built upon a database to assure the consistency of the information in use at any time.

As it happened SIMULINK, a block diagram editor and simulation tool, was added to the MATLAB environment at about the same time as the ECSTASY project neared its completion. As this was its most visible feature, the need for ECSTASY was suddenly (apparently) satisfied by a commercial package, and the ECSTASY environment died in the research lab. In the intervening years, many of the novel features

offered by ECSTASY have found their way into the commercial CACSD packages. However, it is safe to say that the main feature, ECSTASY's ability to interact with a number of third-party tools, have yet to be bettered (at least in the commercial domain).

In this paper, the aims and goals of the ECSTASY project will be re-examined. Using as a benchmark the capabilities of the current generation of MATLAB tools, the prescience of the original proposal will be demonstrated. A couple of areas that the environment excelled in, the use of a common information model and common user interface, will be discussed in the context of recent work done in the University of Wales and elsewhere. Finally, a further two topics of particular interest to the authors, the proposed framework reference model for Computer-Aided Control Engineering (CACE) (Barker *et al.*, 1993) and the object-oriented unified information model (UIM) (Varsamidis *et al.*, 1996) will be revisited in the light of recent developments in internet computing and the world-wide web.

2. THE ECSTASY ENVIRONMENT

The architecture of ECSTASY is illustrated in Figure 1. It was implemented using a "fourth generation programming toolkit" which provided common database access, device-independent graphics and user interface facilities, on top of the UNIX operating system. The ambitious scope of the project included the integration of numerical tools (Matlab, EISPAC, LINPAC), control tools (Matlab-based Control-C and MathWorks' Control System Toolbox), simulation tools (ACSL, TSIM and Simnon), a number of useful control libraries (IDPAC, MODPAC, and SLICE) and the research results of a number of pioneering CACSD groups. This integration was made possible by means of the features identified as essential by the C&I subcommittee's survey:

- a common command language,
- control system data types,
- support for advanced (e.g. MIMO) analysis and design,
- database services,
- and the graphical input of system models.

Unfortunately, the ECSTASY environment was ahead of its time in that it pushed the computing envelope too far beyond the state-of-the-art. The software infrastructure used for its development proved to be unstable, the platform (Sun and DEC-VAX workstations) too expensive, and the embedded tools subjected to incompatible evolution outside the control of the ECSTASY project team. When the C&I subcommittee re-evaluated the marketplace in 1991 they saw that several key features of the environment (block diagram input and advanced control analysis and synthesis tools) had been incorporated in the mainstream

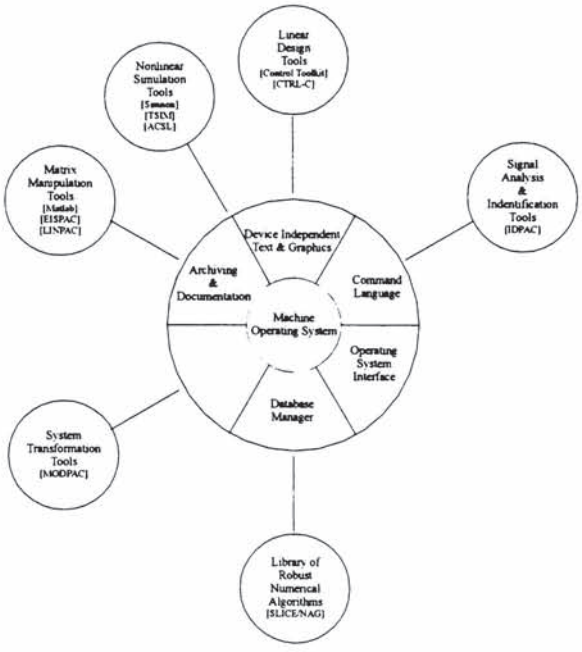

Fig. 1. The ECSTASY Environment

commercial tools. Funding stopped and ECSTASY was not developed beyond its β-testing phase.

However, it is still worth looking back at the goals for ECSTASY to see how far they have been satisfied by the current crop of commercial CACSD packages. We believe that some aspects are still to be satisfactorily addressed and that there is room, with emerging technologies, for further improvement.

3. THE USER'S VIEW OF CACSD

Rimvall (1987) gives a list of requirements for a CACSD environment which are largely still valid today.

- *Software packages must support the same entities used by human specialists in the field.*
- *The basic commands of an interactive environment must be fast yet flexible.*
- *CACSD packages should support an algorithmic interface.*
- *The transition from basic use to advanced use must be gradual.*
- *The system must be transparent.*
- *Small and large systems should be equally treated by the user interface.*
- *The system must be able to communicate with the outside world.*

The following discussion, adapted from Rimvall and Jobling (1997), compares these requirements with the provisions of ECSTASY (Munro and Jobling, 1993) and a modern CACSD environment (Matlab's Control System Toolbox (Mathworks Inc., 1999*a*)).

3.1 *Support for control data-types*

In 1985, many of the commercial tools that were available for linear control systems analysis and design had been developed from Cleve Moler's "public domain" Matlab (Moler, 1980). The only data structure provided in this programme was the complex matrix, therefore the data structures needed by control engineers (e.g. rational polynomials, state-space models, time and frequency responses) had to be synthesized using either tuples of matrices or by packed data structures.

A tuple representation uses two or more matrices to define a logical structure. Three examples are Matlab's state space model, which is defined by four matrices, A, B, C and D, its transfer function model, defined by two row vectors *num* and *den*, and its frequency response defined by a three vectors, frequency vector, magnitude and phase. All parts of the tuple must be present in any context that expects to use the logical whole: e.g. `[num,den] = ss2tf(A,B,C,D);`, `[mag,phase]=bode(num,den,w);`. If any component of the tuple is accidentally changed the whole becomes invalid. A packed structure packs multi-dimensional data into a single matrix: E.g.G = ss(A,B,C,D) This enables a single named variable to represent a control object which may be used as `y=step(G,t);`. However, there is no real indication that G is anything other than a large matrix and it can easily be misused of accidentally destroyed.

Using tuples or packed structures in CACSD is complex and error prone so ECSTASY hid the complexity by use of the command language tools and database support: objects where created in the database using an analogous command to that used above for the packed data structure, but behind the scenes the tuples were actually stored in a database record, extracted and passed to the embedded tools when needed. As a comparison of the ECSTASY approach with traditional CACSD tools see Figures 2 and 3. These illustrate the common tasks of model transformation (state-space to transfer function in this case), combining a compensator with plant and producing open-loop bode diagram and closed-loop step response. The ECSTASY method is clearly superior but more importantly, because real high-level records were being used, the ECSTASY approach was *type safe*. In addition to simplifying the definition and use of linear SISO system model objects, ECSTASY also provided MIMO objects like transfer function matrices and multi-dimensional time and frequency responses. In traditional MATLAB, the support for such data structures, which requires sophisticated packing and unpacking algorithms are even more cumbersome to use than SISO objects. ECSTASY, with its database support, took them in its stride and was therefore able to include many sophisticated MIMO analysis and design tools not previously available outside of research laboratories.

Today, a major redesign of MATLAB has finally provided multi-dimensional arrays, records and objects as built-in types and has removed the restrictions on data-types. The Linear Time-Invariant (LTI) data object provided by the modern Control System Toolbox (Mathworks Inc., 1999*a*) is able to handle SISO and MIMO data objects with arguably even more elegance than the ECSTASY system (see fig. 4). This is achieved by use of object-oriented programming techniques: the components of an LTI model are safely encapsulated in the object; operator overloading allows constructs like `D*G` and `Go/(1 + Go)` to be expressed; and polymorphism enables a single function name, e.g. *bode*, to be used with any LTI system representation. In addition, features like automatic graphing make the interaction even more natural. Nonetheless, an important feature of ECSTASY was its built-in data persistence; the *automatic* storage of data objects to ensure their existence beyond the running of an interactive session. This is still not provided by the current crop of CACSD tools (except through the manual method of loading and saving the current workspace).

3.2 *Command languages*

The basic commands of the MATLAB of 1985 were fast and flexible and provided an algorithmic (i.e programmable) interface. Indeed, it was MATLAB that inspired Rimvall to list these required-features for CACSD packages in his 1987 article (Rimvall, 1987). However, MATLAB still suffers from several deficiencies that ECSTASY's user interface tools were designed to overcome.

The user interface tools used to develop ECSTASY had several features still not found in many modern systems. By using a command language generator based on a formal grammar it was possible to provide several interaction styles from the same core language definition. For example, as well as command driven interface, it was possible to provide a question and answer driven form of interaction (for novice users) and a graphical menu and forms-based dialogue for occasions, e.g. interaction with graphical input and output, when that was more appropriate. Thus the ECSTASY dialogue manager provided support for a gradual development from beginning to advanced use as well as the ability to support the ready integration of tools.

The approach taken by most commercial tool makers is one of "embrace and extend" whereby useful, (often) competitive, technologies are embedded within an environment in preference to providing bridges. The best modern analogy to ECSTASY's philosophy of integration is provided by the scripting languages (Visual Basic, Perl, TCL and Python) whose importance in modern computing is discussed by Ousterhout (Ousterhout, 1998). Such scripting languages are likely to become more important as the "glue" that al-

```
% Define plant: state-space model ...
G_A = [0 1; -1 -2]; G_b = [0; 1]; G_c = [1, 0];
% ... transformed to transfer function
[G_num,G_den] = tf2ss(G_A,G_b,G_c,[]);
% Define compensator
D_num = 10*[1, 0.5]; D_den=[1 5];
% Open-loop system model: D connected in series with G
[Go_num,Go_den]=series(D_num,D_den,G_num,G_den);
% Open-loop frequency response calculations ...
w=logspace(-1:2); % logarithmic frequency range 0.1 to 100 rad/s
[mag,phase]=bode(Go_num,Go_den,w); % magnitude and phase
% ... and plots: magnitude (in dB) plotted above phase
subplot(211),semilogx(w,20.*log10(mag));
subplot(212),semilogx(w,phase);
% Closed-loop step response: unity gain feedback model ...
[Gc_num,Gc_den]=cloop(Go_num,Go_den,-1);
% ... step response calculations and plots
t = 0:0.1:100;
y = step(Gc_num,Gc_den);
plot(t,y);
```

Fig. 2. The definition of components and analysis procedures in a tuple-based matrix environment (MATLAB Control Systems Toolbox *circa 1985).*

```
% Define plant
ss G; % state space model ...
% ... and component definitions
G_a = [0 1; -1 -2]; G_b = [0; 1]; G_c = [1, 0];
tf G = G; % transformation to transfer function
% Define compensator
tf D;
D_num = 10*[1, 0.5]; D_den=[1 5];
% series connection of D and G
tf Go = D | G;
% Open-loop system bode response
w=logspace(-1:2); % frequency range
fr Go;
bodeplot(w,Go_fr);
% Closed-loop step system -- unity gain feedback
tf Gc = Go<-[1];
% closed-loop step response
t = 0:0.1;100;
step(t,Gc);
```

Fig. 3. The same interaction in ECSTASY

```
% Define plant
G = ss([0 1; -1 -2],[0; 1],[1, 0]);
G = tf(G); % transform to transfer function
% Define compensator
D = tf(10*[1, 0.5],[1 5]);
% Open-loop system bode response
Go = D*G; % series connection [overloaded "*" operator]
bode(Go); % frequency range and graphics setup automatically!
% Closed-loop system with unity gain feedback!
Gc = Go/(1 + Go);
step(Gc);
```

Fig. 4. The interaction in today's Matlab control system toolbox

lows components to be combined in the rapidly developing world of distributed computing. If one accepts Ousterhout's thesis, a better long-term strategy for tool makers would be to "specialize and integrate": that is make the specialized tools open to external extension and use scripting languages to provide the integration hinted at in ECSTASY. This would avoid awkward contrived marriages like symbolic algebra embedded in a numerical package, or *vice versa*, that can make packages unwieldy and difficult to use.

3.3 *Graphical tools*

Graphical input of system models, at least in block diagram form, is now so common place in CACSD that it is hard to recall that this was quite a novelty in the late 1980s. The ECSTASY block diagram editor and simulator font-end was provided by "BlockEdit", a tool adapted, with some simplification, from a research prototype called CES (Barker *et al.*, 1988). The emergence of a similar, but arguably better implemented facility, for MATLAB called SIMULINK was probably the single most significant event to foreshadow the demise of ECSTASY.

SIMULINK is intuitive to use, highly interactive and responsive. It is available on PC, Macintosh as well as UNIX implementations of MATLAB. ECSTASY's BlockEdit by contrast was slow, due to its top-down approach (today the trend is to integrate, bottom-up, by combining components), and restricted to expensive workstation platforms. Since ECSTASY's demise, SIMULINK has developed into a highly proficient tool. It provides a documented model representation language, can be transformed into real-time code for implementation and has good links with the rest of the MATLAB CACSD environment (Mathworks Inc., 1999*b*).

A reservation, equally applicable to ECSTASY and the ODE-based simulation tools ECSTASY attempted to integrate, is that the block diagram representation does not lend itself to easy integration with other modeling methodologies that become important in multidisciplinary applications of control such as mechatronics. Developments in object-based modeling (Cellier *et al.*, 1995) and the development of supporting technologies such as the Modelica modeling language (Anonymous, 1999), and algorithm support for DAEs (Otter and Cellier, 1995) have been largely ignored by the mainstream CACSD tool suppliers.

3.4 *Documentation services*

This facet of ECSTASY's specification was, in truth, the one to receive the least attention. The facilities envisioned were presumably similar to those provided by Matlab and Mathematica's *note book* interfaces wherein an interactive session is recorded as commands and results embedded in a word processed document. These notebooks provide a mechanism that is similar to the ideas of "literate programming" for software (Knuth, 1992), in which a commentary on the purposes of a particular design or analysis approach is provided along with the actual code needed to execute the design.

This form of self-documenting (or docucentric) interface works well when the interaction with a tool takes place in the medium of a command language. It is much less satisfactory when the interaction through the medium of a graphical user interface is required (for example the interactive setting up of a nonlinear system simulation through a tool like SIMULINK). An approach to such interactions that has become popular in the context of desktop operating systems is to use some form of "object linking and embedding". In this paradigm, an image of say a block diagram that appears in a document actually represents a simulation "applet". When the image is selected, the simulation tool becomes active, a set of simulation menus and tools appear in the context of the word processor, and a simulation may be performed. The results of which may be embedded on the document as an active graph. The state of the embedded object and any associated experimental results is recorded in the document and saved when the document is saved. This approach has been popularized in the Microsoft windows environment [1]. An attempt to make the same features available on any platform through OpenDoc (Feiler and Meadow, 1996) was largely unsuccessful, but recent work using Java and XML reported by Halepota *et al.* (1998) shows promise.

3.5 *Data repository services*

The final aspect of ECSTASY that is largely still unsupported by modern tools are the data repository services. In ECSTASY these were limited to the storage and retrieval of single control system entities. Relationships between a model and its analysis or simulation results were not recorded, but research on these aspects (Varsamidis *et al.*, 1999*b*), (Varsamidis *et al.*, 1999*a*), have shown that these are potentially very valuable. Some work on the information repository services (Varsamidis *et al.*, 1997) and the use of advanced database architecture (Barker *et al.*, 1996) have illustrated the potential for a properly embedded data repository in CACSD. To the authors' knowledge, only ANDECS (Grübel, 1992) provides anything like these facilities in a form that is usable for serious design work.

---

[1] although a prototype appeared in the Xerox Star in the early 1980s

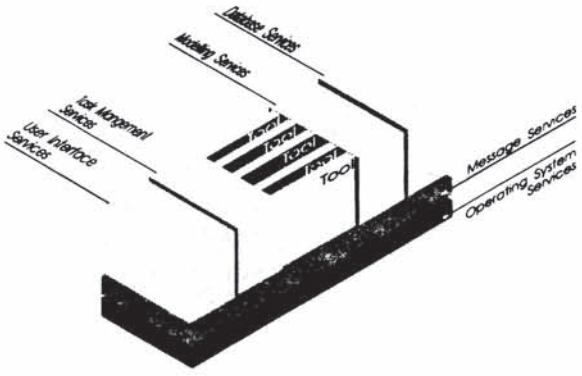

Fig. 5. The Swansea reference model for CACE also known as the "Toaster Model"

4. THE IMPORTANCE OF INTEGRATION SERVICES

ECSTASYs approach to providing a uniform environment for control engineering was a top-down one, in the sense that ECSTASY services were wrapped around other existing third-party tools. This way, the complexity of the overall design process would be reduced for the control engineer by giving a CACSD-oriented *feel* to the interaction (as figures 2 and 3 demonstrate).

The inclusion of third-party tools in ECSTASY required tailor-made adjustments to the ECSTASY software so that these tools would work under the direction of the ECSTASY environment. As more tools were added or as the existing tools evolved, more adjustments were needed; the potential problem of this is that, in order to keep the design process complexity low for the user, the software complexity increases substantially.

A suggested solution to this problem is the replacement of adjustments to third-party software with a set of generic environment services. The innovation in this suggestion lies not only in the types of services themselves, but also in the way that they are to be implemented: the tools should be made aware of the existence of the services and modify their data output so that the data can be fed to the service routines, be processed and returned to the tools. The approach expects that the data format supported by the services is a universal one. This, in turn, requires the existence of an information model which defines the structure and the relationship between data, so that all tools and services can refer to this common format. The Swansea "Toaster Model" (Figure 5 (Barker *et al.*, 1993)) defines one such framework of tool-independent framework services for the CACE area. This can be further supported by the Unified Information Model (Varsamidis *et al.*, 1996) which is a model targeted at a neutral definition of all CACE-related information. Details of how these features may be combined to help the user are given in Varsamidis *et al.* (1999*b*).

5. FUTURE TRENDS

The recent growth of internet-based languages and techniques is targeted at the creation of platforms that will make access to information possible by a varied range of software tools. In this respect, network-hosted platforms aim to provide a result *identical* to that of ECSTASY, through an architecture similar to the one we have just described. Since network technology is application-independent, it is possible (or even necessary) to make use of it in the area of CACSD support tool integration.

Most of the current network-related advances involve the symbiotic use of five technology components:

- the web-browser, which serves as a universal provider of user interface services,
- some form of scripting language to provide task management services,
- a network aware and platform neutral application programming language to provide the tools
- a universal data definition language to provide the modeling services,
- and relational database technology, which provides database services.

As a platform-neutral and network aware object-oriented application programming language, Java (Arnold and Gosling, 1998) seems the ideal candidate for building the infrastructure (if not some of the tools) and XML (Bradley, 1999), an information description language, seems the ideal candidate for building the modeling services. The combination of the two provides universal services (through Java) and data (through XML), in an approach parallel to that of the combination of the "Toaster Model" and the Unified Information Model.

Implementing ECSTASY services in Java would immediately result in an operating system-independent system. The advantages of such a system are obvious: greater flexibility in supporting CACSD tools, easier access to the services, and removal of the need for OS-specific adjustments. At the same time, Java is ideal for the support of distributed services, which can free CACSD design from constraints on locality and availability.

Object-orientation is an integral element of Java; this ties in well with the ECSTASY concept of specialized handling of the various types of data (for graphical, documentation, interaction or other purposes). Java classes can easily implement data models such as the Unified Information Model providing, along with the recording of raw information, support for additional overloaded methods, i.e. methods sharing the same name that perform different tasks depending on the type of data they are applied to. Control data types can eventually be surrounded by a set of *methods* that the environment services may invoke, which would not simply return the data information, but also the

correct process for handling the data under the specific circumstances.

For the above platform, a modeling language such as XML provides the link between the data services and the actual data. Although control engineering data is the immediate target of environments such as ECSTASY, little progress has been achieved in the definition of a common set of data for CACE (or more precisely, data and its structures, which together constitute an information model). The lack of such definition is especially important as it undermines all efforts for further standardization of services, mechanisms and processes. Unlike CACE, other areas of engineering (e.g. petro-chemical, automotive, etc.) have been prompt to develop a common model, thus allowing the further development of the area on a firm basis.

A generic object-oriented for control engineering data model such as the Unified Information Model describes the core properties of this data. However, depending on the specific stage of the CACSD design, the same data may be handled in a totally different manner. As already explained, Java is able to provide an implementation supporting these different handlers, but it is a standard such as XML which describes the different forms data may assume depending on its context. This latter element of context-handling is the contribution of XML to data implementation. Applications reading XML data see not only pure data values, but also instructions on how to deal with them.

6. CONCLUSIONS

In the past half decade, developments in distributed computing, fostered by Berners-Lee's development of the world-wide web, have completely changed the traditional view of a computing architecture. No longer is a computer user restricted to using a node-locked application package on a single, isolated computer workstation. Perhaps even without being aware of it, she is more than likely to be using a combination of components (browser, word processor, internet access) linked together by scripting languages (e.g. visual basic), able to seamlessly communicate across networks and share expensive resources like databases.

In a distributed architecture it is perhaps more appropriate to view a matrix algebra package, computer algebra or simulation tool as a service provider. Integration can be provided by combining scripting languages, infrastructure languages like Java and modeling services provided by XML. Data persistence can be provided by a database with data interchange being facilitated by the combination of Java and XML. User interface services can be provided by browser/editors.

Given that many of these facilities exist or are in an advanced state of development, a modern view of an CACSD environment, lets call it ECSTASY 2000, might be as shown in Figure 6. This differs only slightly from the original architecture. The main difference is that the infrastructure components are open and readily available. It does however require a repositioning of the major tools to allow them to work cooperatively when the "network is the computer".

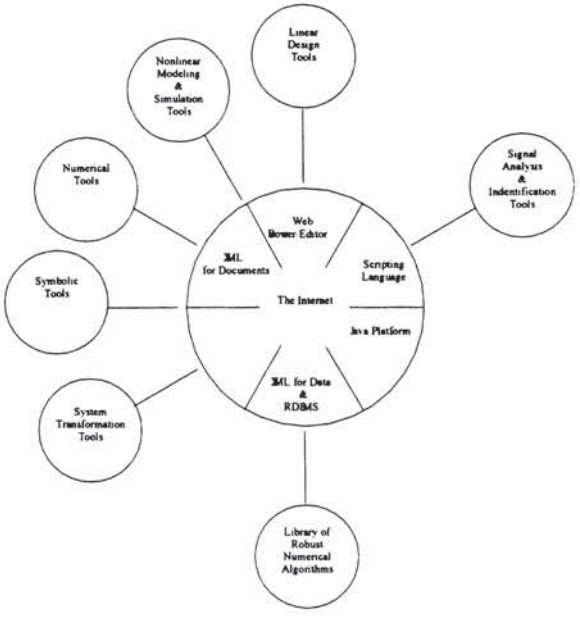

Fig. 6. ECSTASY 2000: a view of a modern integrated CACSD environment.

7. REFERENCES

Anonymous (1999). *Modelica – A Unified Object-Oriented Language for Physical Systems Modeling: Language Specification*. version 1.3 ed.. Modelica Design Group.

Arnold, Ken and James Gosling (1998). *The Java Programming Language*. second ed.. Addison-Wesley. Reading, MA, USA.

Barker, H. A., M. Chen, P. Townsend and I. T. Harvey (1988). CES — A workstation environment for computer-aided design in control systems. In: *Preprints of the 4th IFAC Symposium on Computer Aided Design in Control Systems — CADCS'88*. Beijing, PRC. pp. 248–251.

Barker, H. A., M. Chen, P. W. Grant, C. P. Jobling and P. Townsend (1993). Open architecture for computer-aided control engineering. *IEEE Control Systems* pp. 17–27.

Barker, H. A., P. W. Grant, I. T. Harvey, C. P. Jobling and P. Townsend (1996). An approach to project management in computer-aided control engineering. *Control Engineering Practice* **4**, 441–454.

Bradley, Neil (1999). *The XML companion*. second ed.

Cellier, F. E., H. Elmqvist and M. Otter (1995). Modeling from physical principles. In: *The Control Handbook* (W. S. Levine, Ed.). pp. 99–108. CRC Press. Boca Raton, FL. PostScript available as http://www.ece.arizona.edu/ cellier/levine_2.ps.

Feiler, Jesse and Anthony Meadow (1996). *Essential OpenDoc*. Addison-Wesley.

Grübel, G. (1992). AnDeCS: a concurrent control engineering project. In: *Proceedings IEEE Control Systems Society Symposium on CACSD*. Napa, CA, USA. pp. 37–45.

Halepota, Al Sadiq A. M. M., Philip .W. Grant and Christopher P. Jobling (1998). Design is a document. In: *UKACC Int. Conference Control '98*. Swansea, UK. pp. 1055–1059.

Knuth, Donald E. (1992). *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information. Stanford, CA, USA.

Mathworks Inc., The (1999*a*). *Control Systems Toolbox for Use with* MATLAB*: User's Guide*. version 4.2 ed.. The Mathworks Inc.. 24, Prime Park Way, Natick MA, USA.

Mathworks Inc., The (1999*b*). *Simulink Dynamic Systems Simulation for* MATLAB*: User's Guide*. version 3 ed.. The Mathworks Inc.. 24, Prime Park Way, Natick MA, USA.

Moler, C. (1980). MATLAB — user's guide. Technical report. Department of Computer Science, University of New Mexico. Alberquerque, USA.

Munro, N. and C. P. Jobling (1993). ECSTASY — an infrastructure for CACSD. In: *CAD for Control Systems* (D. Linkens, Ed.). pp. 449–468. Marcel Dekker. New York.

Otter, M. and F. E. Cellier (1995). Software for modeling and simulating control systems. In: *The Control Handbook* (W. S. Levine, Ed.). pp. 415–428. CRC Press. Boca Raton, FL. PostScript available as http://www.ece.arizona.edu/ cellier/levine.ps.

Ousterhout, John K. (1998). Scripting: Higher-level programming for the 21st century. *IEEE Computer* **31**(3), 23–30.

Rimvall, C. M. (1987). CACSD software and man-machine interfaces of modern control environments. *Transactions of the Institute of Measurment and Control*.

Rimvall, C. Magnus and Christopher P. Jobling (1997). *Computer-Aided Control Systems Design*. Chap. 112. 2nd ed.. CRC Press.

Varsamidis, T., C. P. Jobling and Sian Hope (1997). Towards a "repository service '" for computer-aided control engineering. In: *Preprints IFAC Symposium Computer Aided Control Systems Design (CACSD '97)*. Gent, Belgium.

Varsamidis, T., S. Hope and C. P. Jobling (1999*a*). Implementation issues of a UIM-based CACE integration environment. In: *Proc. IEEE Symp. Computer-Aided Control Engineering*. Hawaii, USA.

Varsamidis, T., S. Hope and C. P. Jobling (1999*b*). Use of a prototype CACE integration framework based on the unified information model. In: *Proc. IEEE Symp. Computer-Aided Control Engineering*. Hawaii, USA.

Varsamidis, Thomas, Siân Hope and Christopher P. Jobling (1996). An object-oriented information model for computer-aided control engineering. *Control Engineering Practice*.