Electronic Communications of the EASST
Volume 21 (2009)

Proceedings of the
3rd International Workshop on
Multi-Paradigm Modeling
(MPM 2009)

An Architectural Approach to the Design and Analysis of Cyber-Physical Systems

Akshay Rajhans, Shang-Wen Cheng, Bradley Schmerl, David Garlan, Bruce H. Krogh, Clarence Agbi and Ajinkya Bhave

10 pages

Guest Editors: T. Levendovszky, L. Lengyel, G. Karsai, C. Hardebolle
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

An Architectural Approach to the Design and Analysis of Cyber-Physical Systems

Akshay Rajhans[1], Shang-Wen Cheng[2], Bradley Schmerl[2], David Garlan[2], Bruce H. Krogh[1], Clarence Agbi[1] and Ajinkya Bhave[1]

[1] Dept. of Electrical and Computer Engineering
[2] School of Computer Science

Carnegie Mellon University
Pittsburgh, PA USA 15213-3890

Abstract: This paper presents an extension of existing software architecture tools to model physical systems, their interconnections, and the interactions between physical and cyber components. A new CPS architectural style is introduced to support the principled design and evaluation of alternative architectures for cyber-physical systems (CPSs). The implementation of the CPS architectural style in AcmeStudio includes behavioral annotations on components and connectors using either finite state processes (FSP) or linear hybrid automata (LHA) with plug-ins to perform behavior analysis using the Labeled Transition System Analyzer (LTSA) or Polyhedral Hybrid Automata Verifier (PHAVer), respectively. The CPS architectural style and analysis plug-ins are illustrated with an example.

Keywords: architecture description languages, cyber-physical systems, finite state processes, linear hybrid automata

1 Introduction

Today's models and methods for analysis and design of cyber-physical systems (CPS) are typically fragmented along lines defined by disparate mathematical formalisms and dissimilar methodologies in engineering and computer science. While separation of concerns is needed for tractability, such analytical approaches often impose an early separation between the cyber and physical features of the system design, making it difficult to assess the impacts and tradeoffs of alternatives that cut across the boundaries between these domains.

This paper presents extensions to software architectural descriptions to encompass the full range of elements that comprise cyber-physical systems. Our goal is to create an extensible framework within which a comprehensive set of design tools can be created. As first steps in this direction, this paper presents a new CPS architectural style and behavioral annotations and plug-ins for verification.

The following section reviews previous work on software architectures. Section 3 then presents a CPS architectural style as the basis for developing architectural constructs for specific application domains. Section 4 illustrates the application of the components and connectors in the CPS architectural style for a simple example. Section 5 describes how behavioral analysis can be carried out with the help of annotations and plugins. Section 6 illustrates behavior verification for the example introduced in Section 4. The concluding section summarizes the contributions of this paper and discusses the next steps in this research.

2 Previous Work

Over the past decade software architecture has emerged as one of the primary techniques for disciplined engineering of large-scale software systems. A software architecture typically models a system as a graph of components and connectors, in which the components represent principal computational elements of a system's run-time structure and the connectors represent the pathways of communication between components [SG96, PW02, BCK03]. These elements are annotated with properties that characterize their abstract behavior and facilitate reasoning about system-level design tradeoffs.

There has been considerable research and development in architecture description languages (ADLs) and tools to support their analysis and realization as code [MT00, DHT02, GS06]. Standardized notations such as UML 2.0 [RJB04], SysML [SysML] and AADL [AADL] provide modeling vocabularies of components, connectors and properties. Additionally, a number of researchers have investigated methods for modeling architectural behavior, for example as protocols characterized by process algebras or state machines [AAG95, MK06]. These notations are supported by tools that provide graphical editing and viewing, hierarchical development (in which components may be refined as more detailed architectures) [MR97], checking for component compatibility or substitutability [AG97], and evaluation of quality attributes such as performance, reliability, and security [GS06, RM04, MI04, SG98]. Software architectures also support reuse of design expertise and code infrastructure.

In many cases an architecture of a system fits within a common family, referred to as an architectural style [SG96]. An architectural style defines a set of component and connector types, together with constraints that prescribe how elements can be composed. Some ADLs allow one to define architectural styles, develop systems in that style, and provide tools for checking whether a system is compliant with a given style [AAG95, GAO94, GMW00, SG04].

Software architecture has been applied effectively to numerous embedded and control systems. For instance, architecture description languages such as Meta-H and AADL have been used to model avionics systems, automotive control systems, and other applications [BV93, HuF07].

3 A CPS Architectural Style

Architectures typically represent systems at a higher level than simulation models, which represent the details of a particular system implementation. Although architectural modeling has been used in specific domains to incorporate physical elements as components, there is currently no way of treating cyber and physical elements equally in a general fashion. This is because the components and connectors in software architectural styles are inadequate for representing the types of physical components found in CPS and their interactions with cyber entities. This section presents extensions that allow both physical and cyber elements to be modeled together in a CPS architectural style.

We use the Acme ADL [GMW00], which has strong support for defining flexible architectural styles. In Acme, an architectural style is represented as a family of element types that follow certain rules and structure in terms of what kind of components and connectors can be present in the system and the manner in which they can be connected with each other. A general family can be refined into an application-specific family by adding additional elements and rules [DRRS99].

The challenge in defining an architectural style is to find a balance between specificity and generality. For the CPS domain we focus on embedded monitoring and control systems. Our goal is to create a family of general components and connectors that can serve as the foundation for application-specific styles in this broad domain. Towards this end we define three related families pertaining to the cyber domain, the physical domain, and their interconnection.

3.1 Cyber family

The cyber side of CPS is the traditional domain for ADLs. The following components and connectors provide support for standard real-time monitoring and control applications. The component types are:

- *Data Stores:* These components store data as an interface between the computational elements in the system. In simple systems, these could be just passive memory blocks. In complex systems there could be further details specifying what components can read and write to the data store components.
- *Computation:* Computation components operate on and update data posted in data store components. This includes components that perform filtering, state estimation, and control.
- *IO Interfaces:* These components perform the computations and timing functions required to sense and control the physical world. This would include, for example, smart sensor software that processes raw sensor data.

In addition to the computational aspects of the software, it is important to represent the communication elements in the system to reason about timing between software elements and how this affects the physical behavior of the complete system. We represent two major types of cyber connectors, a *call-return connector* representing one-to-one communication and a *publish-subscribe connector* representing one-to-many. Each of these types can be further specialized to represent particular communication protocols and network characteristics.

3.2 Physical family

There are several challenges in developing a suitable representation of the physical side of cyber-physical systems at the architectural level. Architectural models should not have all the details required for a full simulation of the physical dynamics. At the same time, the architectural components and connectors should correspond to intuitive notions of physical dynamics in the same way that cyber components and connectors correspond to elements of computational systems. To achieve this balance, we introduce components and connectors based on an energy view of physical systems. This provides a domain-independent perspective, including the ability to represent interactions between different physical domains and the possibility to specify system properties such as power flow and causality. This is similar to the perspectives taken in bond graphs [GaS96] and Lagrangian mechanics [JeS99], where power-conjugated variables (effort and flow) describe signal flows between sources, energy storage elements, and dissipative elements.

The physical component types are:

- *Sources:* A source component specifies source elements that deliver power to other components. These components have only output ports because they deliver effort or flow into the system.
- *Energy storage:* An energy storage component models dynamic elements or subsystems that store energy, such as components that have capacitive and inductive properties in electrical systems. Ports on these components allow power transfer to other subsystems.
- *Physical transducers:* Transducers represent power transfer or energy conversion between different types of physical domains. These components are particularly useful in modeling multi-domain systems with, for example, electromechanical devices that transform energy between the electrical and mechanical domains.

The physical connector types are:

- *Power flow:* Power flow connectors model bidirectional interactions between physical components, including dissipative interactions. These connectors lead to dynamic coupling between components.
- *Shared variable:* This connector indicates that the variables in two components are identical. There is no directionality associated with this connector.
- *Measurement:* Measurement connectors indicate variables that are determined by one physical component and used as an input in another physical component. Thus, these connectors are directional and correspond to connections in traditional block diagrams (e.g., signal lines in Simulink).

3.3 Cyber-physical interface family

The cyber-physical interface family inherits all the elements from the cyber and physical families and adds new elements that bridge the gap between computational and physical systems. To model the interactions between the cyber and the physical worlds, we introduce two directed connector types, *P2C connector* (physical-to-cyber) and *C2P connector* (cyber-to-physical). Simple sensors can be modeled as P2C connectors and simple actuators can be modeled as C2P connectors.

For more complex interfaces between cyber and physical elements in a CPS, we define the *P2C transducer* and *C2P transducer* components, which have ports to cyber elements on one side and ports to physical elements on the other side. Devices are modeled as P2C/C2P transducers if they do more than a simple translation between cyber and physical domains, e.g., intelligent sensor nodes.

4 Example

To illustrate the architectural modeling using the CPS architectural style, consider a temperature control system for two zones (rooms) illustrated in Fig. 1. The zones have temperatures T_1 and T_2 respectively, and the temperature of the ambient environment is T_a. The zones retain heat with thermal capacities C_1 and C_2, respectively, and lose heat to the ambient environment with thermal resistivities R_1 and R_2 respectively. Zone 2 is heated through a shared wall with thermal resistivity R_w with Zone 1. The thermostat is physically located in Zone 1, so it can only read the temperature in Zone 1. The furnace can be powered on or off manually. When the furnace is on, the thermostat determines whether or not the furnace should heat the room and turns a blower that forces hot air into the room on or off accordingly. The goal is to maintain the measured temperature of Zone 1 close to the thermostat setpoint.

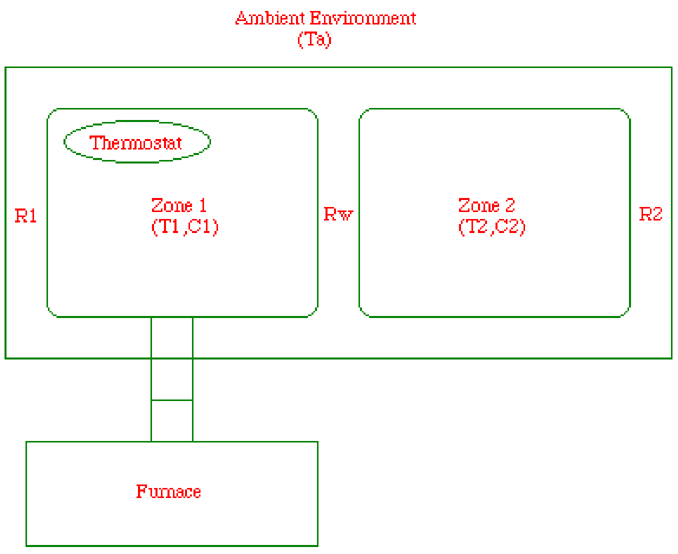

Figure 1: Illustration of thermal example.

Figure 2 shows the architecture of this system in AcmeStudio [SG04]. The right hand pane shows the cyber, physical and the cyber-physical interface families. Each zone is modeled as an energy storage component. The ambient temperature is modeled as a source component since it is independent of the room dynamics.

The top right corner of the canvas shows a bounding box that corresponds to the thermostat. The box contains a temperature sensor that senses the temperature of Zone 1 and broadcasts it on a communication channel. A temperature estimator then reads the temperature from the communication channel and updates the temperature state variable. The furnace estimator reads the temperature and keeps updating the furnace state. The temperature controller is the part of the software that keeps track of when the furnace estimator changes the furnace state and communicates these changes to the furnace as commands to turn the heating on or off.

The furnace comprises a furnace actuator component that receives commands sent by the

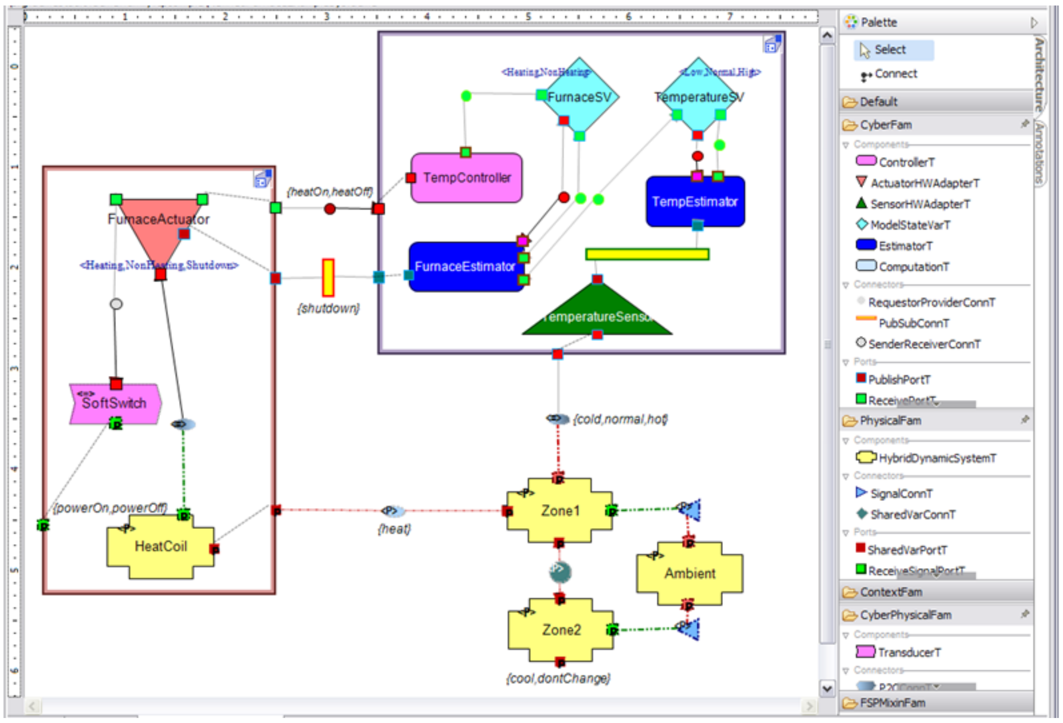

Figure 2: Architectural modeling of a temperature control system using cyber-physical family.

temperature controller in the thermostat and accordingly turns the corresponding heating element (heat coil) on or off. Finally, a transducer element called SoftSwitch captures the manual on/off signal and communicates it to the furnace actuator software element to turn the heating element on or off.

Zone 1, Zone 2 are modeled as energy storage components and the ambient environment as a source. The connectors between ambient and the two zones are directional input-output connectors capturing the one-way coupling, while the connector between the two zones is a bidirectional shared-variable connector that captures the bidirectional coupling.

5 Behavioral Annotations

The architectural elements describe only the structural information about a system. To be able to do meaningful formal analysis on the system behavior, one must annotate the architecture with behavioral information. In Acme ADL, the behavioral annotation can be implemented via *properties* to capture the behavioral information. We have implemented the annotations for two types of behavioral modeling frameworks - Finite State Processes [MK06] and Linear Hybrid Automata [Hen96].

We have developed plug-ins as extension points of AcmeStudio, which will display only the relevant information pertaining to the element or system selected by the user. We have also built into the plugins the ability to generate analyzable text files from these properties. The plug-ins

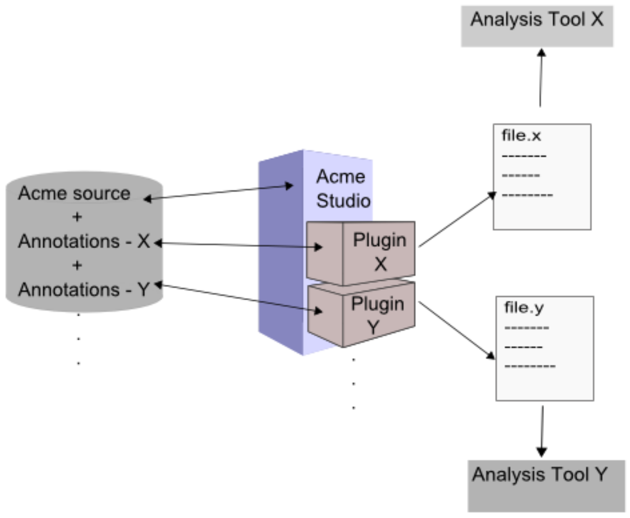

Figure 3: Behavioral analysis using AcmeStudio plugins

traverse the architecture, gather the relevant information distributed throughout the structure and generate a text file that is analyzed by the relevant tool. Figure 3 shows a schematic of this analysis flow. In our case, we have developed one plug-in for FSP that generates a file that can be analyzed by the Labelled Transition System Analyser [MK06] and another plug-in for LHA that generates a file that can be analyzed by Polyhedral Hybrid Automaton Verifyer (PHAVer) [Fre05].

6 Behavioral Verification

Consider the example from Section 4. The goal is to maintain the Zone 1 temperature close to the thermostat setpoint. The furnace is a memoryless passive element that receives commands from thermostat and the local manual power on/off switch. Thus, the furnace can be in one of the three states - poweredOff, poweredOn (but not heating) and Heating. Whenever it is locally powered on, it goes by default into the poweredOn state. Suppose the thermostat only sends commands to the furnace turn the heat on (off) and doesn't cross-check to see whether the furnace actually responds. If the thermostat issues a command to the furnace to turn the heat on while the furnace is locally powered off, the furnace misses the command. If the furnace is powered back on, it goes by default into the poweredOn state and won't turn the heat on until it receives the next heat on command from the thermostat. But the thermostat won't issue another command because it has already issued a heat-on command and is waiting for the temperature to increase to the point that a heat-off command needs to be issued.

We want our analysis to catch this error, and would also like to propose a remedy. We start from the more basic, discrete-only FSP analysis. This can be done by posing the question as a liveness check – if the thermostat issues a command to start heat, will the temperature of the room eventually rise? Liveness properties can be easily analyzed in LTSA, which not only catches this bug, but also gives a counterexample-trace indicating where the liveness requirement fails. We immediately catch the error that the thermostat is unaware of the furnace state and one proposed remedy would be to have the furnace report its state to the thermostat. This corrected architecture is shown in Fig. 2 with an added shutdown notification connector between the furnace and the thermostat.

This solution might be difficult to implement if the furnace is a dumb device that can only receive commands and cannot transmit acknowledgments. Fortunately, there is another solution, which is to have the thermostat check to see if the room temperature has actually risen to the expected level after a specified timeout interval. If the temperature has not risen, the heat-on command is resent. The success of this strategy depends on the choice of the timeout period with respect to the rate of change of room temperature while heating and cooling. Neither the real time clock nor the rates of change of temperature can be represented in the purely discrete FSP model, however. So we turn to LHA, where both of these behaviors can be represented. With LHA analysis done in PHAVer, we can find out the good range of the timeout period given the min and max ranges of rates of change of temperature while heating and cooling.

7 Discussion

This paper introduces tools for the modeling and analysis of cyber-physical systems at the architectural level. A CPS architectural style is proposed along with tools for annotating CPS architectures with behavioral descriptions modeled as finite state processes and linear hybrid automata. The CPS architectural style presented in this paper provides a unifying framework to develop new methods for optimizing designs with respect to performance measures that characterize important features of the system behaviors. We are currently developing the capability to offer different views of a CPS architecture, where each view can correspond to a particular type of mathematical abstraction and different performance measures can be associated with different views. The collection of views and associated performance measures leads to a network of design optimization problems, which are interconnected through shared variables and constraints. We are also developing methods for formulating and solving these interconnected design problems as a basis for a comprehensive approach to the analysis and design of cyber-physical systems.

8 Acknowledgments

This work was supported in part by National Science Foundation (NSF) under grant no. CNS0834701 and by Air Force Office of Scientific Research (AFOSR) under contract no. FA9550-06-1-0312.

Bibliography

[AADL] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The Architecture Analysis and Design Language (AADL): An Introduction. Technical Report CMU/SEI-2006-TN-011, Software Engineering Institute, Carnegie Mellon University, February 2006.

[AAG95] G. Abowd, R. Allen and D. Garlan, Formalizing Style to Understand Descriptions of Software Architecture, ACM Transactions on Software Engineering and Methodology, Vol. 4(4):319-364, October 1995.

[AG97] R. Allen and D. Garlan, A Formal Basis for Architectural Connection, ACM Transactions on Software Engineering & Methodology (TOSEM), July 1997.

[BCK03] Bass, L., Clements, P., and Kazman, R. Software Architecture in Practice, Second Edition. Addison-Wesley, 2003.

[BV93] P. Binns and S. Vestal. Formal real-time architecture specification and analysis. 10th IEEE Workshop on Real-Time Operating Systems and Software, May 1993.

[DHT02] Dashofy, E., van der Hoek, A., and Taylor, R.N., An Infrastructure for the Rapid Development of XML-based Architecture Description Languages, In Proceedings of the 24th International Conference on Software Engineering (ICSE2002), Orlando, Florida.

[DRRS99] Dvorak D, Rasmussen R., Reeves G., Sacks A., Software Architecture Themes in JPL's Mission Data System, AIAA Space Technology Conference and Expo, Albuquerque, NM, 1999.

[Fre05] G. Frehse. PHAVer: Algorithmic Verification of Hybrid Systems past HyTech. Proceedings of the Fifth International Workshop on Hybrid Systems: Computation and Control (HSCC), Lecture Notes in Computer Science 3414, Springer-Verlag, 2005, pp. 258-273.

[GAO94] Garlan, D., Allen, R.J., and Ockerbloom, J. Exploiting Style in Architectural Design. Proc. Symposium on the Foundations of Software Engineering, New Orleans, LA, 1994.

[GMW00] D. Garlan, R. T. Monroe, and D. Wile, Acme: Architectural Description of Component-Based Systems. In Gary T. Leavens and Murali Sitaraman editors, Foundations of Component-Based Systems, Pages 47-68, Cambridge University Press, 2000.

[GaS96] Peter Gawthorp, Bond Graphs and Dynamic Systems, Prentice Hall, 1996.

[GS06] David Garlan and Bradley Schmerl, Architecture-driven Modelling and Analysis,Proceedings of the 11th Australian Workshop on Safety Related Programmable Systems (SCS'06), Vol. 69 of Conferences in Research and Practice in Information Technology, Melbourne, Australia, 2006

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In Proc. 11th Annual IEEE Symposium on Logic in Computer Science, LICS'96, New Brunswick, New Jersey, 27-30 July 1996, pages 278-292. IEEE Computer Society Press, 1996.

[Hoa85] C.A.R. Hoare. Communicating Sequential Processes. Prentice Hall, 1985

[HuF07] J. Hudak and P. Feiler, Developing AADL Models for Control Systems: A Practitioner's Guide, Technical Report CMU/SEI-2007-TR-014, Software Engineering Institute, 2007.

[JeS99] D. Jeltsema and J.M.A. Scherpen, Multidomain modeling of nonlinear networks and systems, Control Sytems Magazine, Aug. 2009.

[MI04] Antinisca Di Marco and Paola Inverardi. Compositional Generation of Software Architecture Performance QN Models. 4th Working IEEE/IFIP Conference on Software Architecture (WICSA'04), 2004.

[MK06] J. Magee and Jeff Kramer. Concurrency: State Models and Java Programming, Second Edition. Wiley 2006.

[MR97] Moriconi, M. and Reimenschneider, R.A. Introduction to SADL 1.0: A Language for Specifying Software Architecture Hierarchies. Technical Report SRI-CSL-97-01, SRI International, 1997.

[MT00] Medvidovic N., and Taylor R.N., A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering 26(1), pp. 70-93, 2000.

[PW02] Perry, D.E., and Wolf, A.L. Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes, 17(4):40-52, 1992.

[RJB04] The Unified Modeling Language Reference Manual, Second Edition. James Rumbaugh, Ivar Jacobson, Grady Booch. Addison Wesley 2004.

[RM04] Roshanak Roshandel and Nenad Medvidovic. Toward Architecture-Based Reliability Estimation. In Proceedings of Twin Workshops on Architecting Dependable Systems (WADS 2004), Edinburgh, UK, May 25, 2004 and Florence, Italy, June 30, 2004.

[SG96] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996.

[SG98] B. Spitznagel and D. Garlan. Architecture-Based Performance Analysis, In Proceedings of the Tenth International Conference on Software Engineering and Knowledge Engineering (SEKE'98). June, 1998.

[SG04] B. Schmerl and D. Garlan, AcmeStudio: Supporting Style-Centered Architecture Development, In Proceedings of the 26th International Conference on Software Engineering, Edinburgh, Scotland, 23-28 May 2004.

[SysML] SysML Home Page.
`http://www.sysml.org/`