5th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence and Informatics
January 25-26, 2007 ▒ Poprad, Slovakia

Open Design Architecture and Artificial Intelligence Agent Application in Information Systems Practice

Miroslav Beličák

Department of Computer and Informatics, Technical University of Košice
Letná 9, 051 20 Košice, Slovak Republic, miroslav.belicak@tuke.sk

Abstract: This paper describes new type of information systems architecture as well as current essential problems of information systems (IS). Article further introduces the opportunity of utilizing artificial intelligence agents at practise of run-time environment for initiated new architecture type. Proposed open design architecture should provide better facility for modification of IS, because it prescribes tight coupling of IS design with executable binary elements of IS. Thus, user can easily change the structure of IS without necessity of IS execution suspension or disconnection of users. To achieve these goals, it is possible to utilize software agents and artificial intelligence agents.

Keywords: information system architecture, design, artificial intelligence agent, software agent, information system, run-time environment, user, designer

1 Introduction

A design and implementation of enterprise systems can require sizeable time and finance means on customer and supplier side. Information system architecture (ISA) plays very important role in software engineering process [4]. A lot of information systems (IS) properties (such as scalability, security, robustness, stability, etc.) depend on ISA. Next sections will describe a new information system architecture type called ODA (Open Design Architecture) as possible way of simplifying IS design and maintenance. The main idea of this architecture lies in separating application functionality from presentation logic such as graphics user interface and in coupling IS design together with IS - as will be shown below together with advantages of such approach. Together with ODA architecture description we outline possible utilization of software agents and artificial intelligence (AI) agents in practise of information systems proposed for ODA run-time platform.

2 Current State in Information System Architecture Area

Presently three main information system architectures are in IT world. First is Service-Oriented Architecture (SOA) [13, 14, 15, 16], which is the key paradigm, that is used nowadays. It has significantly influenced organizational and technological development [12]. Next is Model Driven Architecture (MDA). OMG's Model Driven Architecture (MDA) provides an open, vendor-neutral approach to the challenge of business and technology change [17]. The last is Component-Based Architecture (CBA). Component-Based Architecture is also foundation for distributed component architectures [4] and includes mechanisms and techniques for developing coarse (but still reusable) bussiness/software implementation units. In next sections we will briefly describe these three main architecture trends.

2.1 Service-Oriented Architecture

Service oriented architecture [8, 9, 10] (*SOA*) – combines the ability of remote functions and objects (named services) calls with tools for dynamic service discovery, emphasis is on interoperability [11]. In service oriented architectures, applications are regarded as services accessible on network. IT infrastructure is maintained on abstract level, which means that functionality is presented in the shape of loosely coupled services, that can be made accessible to users independent from lower level technology supporting the services. SOA as abstract paradigm traditionally represented basic distributed architecture without references to implementation [10]. Several technologies exist at present, that are able of SOA's practical realization: CORBA (*Common Object Request Broker Architecture*), Microsoft DCOM (*Distributed Component Object Model*) and web services – all of them provide required functionality. Architecture of SOA applications differs from traditional software architectures, where architecture is mostly static in nature. Architecture of SOA application is dynamic, what means (except other consequences), that application can by composed in time of program execution using existing services. SOA thus provided new way of creating software architectures, where architecture is determined in time of application execution and can be dynamically changed for fulfillment of new software requirements [8].

Main components of SOA together with their roles are depicted on Figure 1.

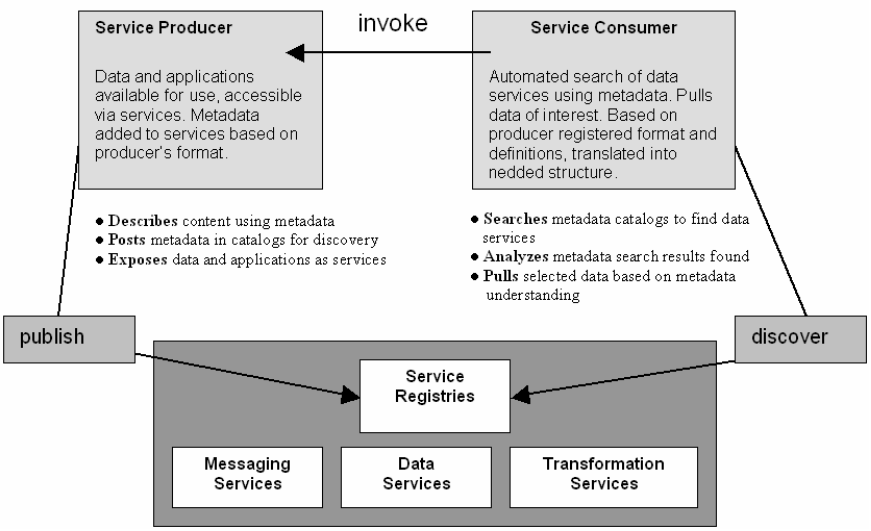

Figure 1
Main components of Service-Oriented Architecture

2.2 Model-Driven Architecture

Model driven architecture is based on OMG (Object Management Group) standards and it isolates business and application logic from technological platform [5, 12]. Core idea of MDA [17, 18, 19, 20] is using of UML language for specification of static interfaces and also dynamic behavior of components in platform-independent models (so-called PIM-models, *Platform Independent Models*). Furthermore, MDA defines rules, according to which can be PIM-models mapped to multitude of platform-specific models (PSM - *Platform Specific Model*) [17]. MDA is approach which supports development of the system by using process aimed at models and generative process of development. MDA improves quality of complex software systems based on creation of high-level models of the system and automatic generation of systems architecture from models [18].

OMG's MDA accentuates modeling in software development. One of the principal goals of MDA is support for platform independent application development [19].

Substantial part of endeavor and price of distributed applications development consists in modification of application according to requirements for support of particular middleware-platform [20]. Considerable part of work leading to MDA (or more generally MDE - *Model Driven Engineering*) was devoted to this topic. It appears, that ultimate idea leading to these efforts is definition of the model, which captures whole business logic of target application, but which is not bound

to specific middle-ware and is also not supporting concrete technology [20]. This model is called PIM (*Platform Independent Model*) and it defines set of transformations, automatically converting PIM to one or more models scaled to chosen middle-ware. Result of such transformation is called PSM (*Platform Specific Model).* On top of these models MDA presents so-called (*Computation Independent Model* – CIM). This model describes system in context of its environment (e.g. Business domain) and shows, what is expected from system's execution (without describing details of how is system constructed). The principle of Model-Driven Architecture is shown in the Figure 2.

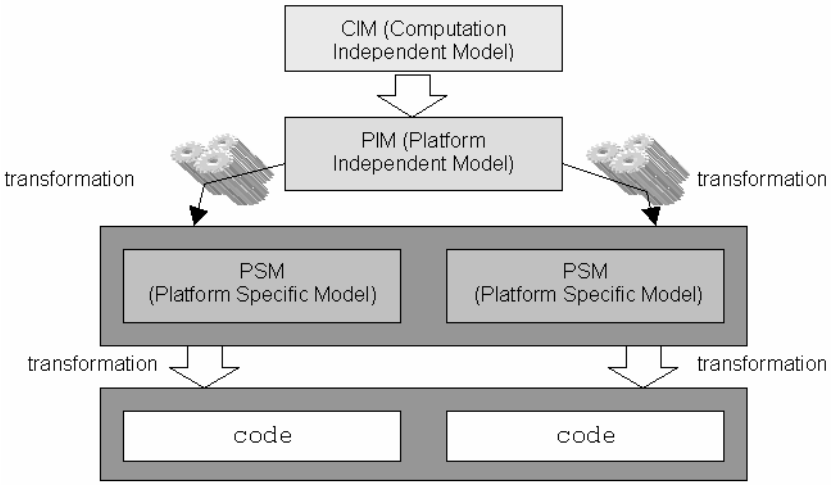

Figure 2
The Principle of Model-driven Architecture

2.3 Component-Based Architecture

Architectures based on development of components (*Component Based Development Architectures - CBDA*) are increasingly adopted by software engineers [17]. CBA complements architectures SOA, CBA, EDA – all approaches conforming to and supporting agile 'user centric' solutions (*User Centric Agile Solutions*) [6]. *Component Based Architecture* (CBA) is the base for distributed component architectures [6] and includes mechanisms and techniques for development of coarse (but still) reusable bussiness/technical implementation units, considering environment/container and decomposing service to more SBBs (Service Building Block), which are connectable distributed parts associated with: presentation logic, bussiness logic, resource management logic, integration logic, logic of net events, security logic and other logics [6].

Component is executable part of source code providing sum of services in the form of *black-box* [21]. Its services are accessible only through consistent

published interface which embodies communication standards. Component must be connectable to other components (through communication interface) to form more sizeable entities.

Components consist of following standalone parts (by reason of incremental development):

- specification – describes, what component provides,
- implementation (source code) – ensures operation of specified properties,
- executable code (binary files, libraries).

Granularity of component (Figure 3) should be appropriate. In case of high granularity, the system is too complex. On the contrary – in case of low granularity (high generality) the system gets to the boundaries of reusability [22].

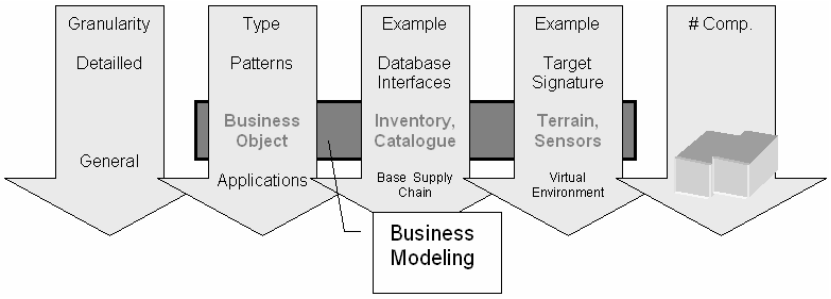

Figure 3
The Component Granularity

2.4 Current Problems and Our Motivation

Most of the current IS are deployed more-or-less only in binary form (with the exception of the several types of different auxiliary files and parts such as XML files, database files, etc.). However, the architectures of information systems stay transparent for their users and they're visible only for IS designers (by SOA, MDA and CBA) [2].

The design of IS (by means of different types of diagrams, formal or semi-formal notations) is separated from implementation parts of IS (for example source codes, system documentation, etc.). Maintenance of these two factors can be difficult by large and complicated IS.

Last but not least, in the company producing the IS, the documentation of the design process is usually not synchronized with the implementation documentation. This fact has consequences in case of later modifications of the IS - additional effort is needed by employees of the company to study the IS analysis, design and implementation. The similar situation occurs, when some analyst,

designer or developer, which collaborated on the IS development, left the company. To sum up, the customer simply depends on supply company when some changes in IS are needed.

3 Agents

Utilization of knowledge from agent technologies in ODA allows enhancement of system's quality (e.g. ensuring of security, response improvement). According to [3] if the entity is to be called agent, it should be:

• Autonomous - capable of acting without direct external intervention. Agent should have a degree of control over its internal state.

• Interactive - communicating with the environment using receptors and actuators.

• Learning - actions based on its experiences.

Different, but semantically similar descriptions of agent systems can be found in [3].

3.1 Software Agents

A software agent is a software entity that possesses an independent thread of execution and has a set of goals or tasks to complete, combined with an internal set of plans that are aimed at achieving these goals. The independent thread of execution means that the agent determines when to perform tasks based upon observing its 'environment' [1].

To the important properties of software agent belong:

- it gets nearer to the abstraction of actor than objects,
- it is higher-level component abstraction of modeling system,
- for its design and implementation it is possible to use object-oriented approach, or design patterns respectively.

The additional tiers of behavior such as learning, code mobility, etc. could be added in a straightforward way to the systems implemented with agent technology.

3.2 Artificial Intelligence Agents

Artificial Intelligence agent (AI agent) has compared to SW agent several extra properties, especially AI agent should be:

• Adaptive - capable of responding to other agents and/or its environment to some degree. More advanced forms of adaptation permit an agent to modify its behavior based on its experience.

• Intelligent - state is formalized by knowledge (i.e., beliefs, goals, plans, assumptions) and interacts with other agents using symbolic language.

• Rational - able to choose an action based on internal goals and the knowledge that a particular action will bring it closer to its goals.

• Transparent and accountable - must be transparent when required, yet must provide a log of its activities upon demand.

From extensive list of AI agents properties mentioned in [3], some meaningful for ODA are: Temporally continuous, Proactive, Character, Unpredictable (to some degree), Rugged, Trustworthy, Coordinative.

4 Proposed Architecture

As was outlined in introduction, every information system designed in context of ODA consists of two main parts:

- Application Logic (AL)
- System Design (SD)

Under the term application logic (AL) we intend specific group of services offered by particular IS, where the service is executable part of information system, covering exactly one functionality of system. The service takes input data from user of IS (or application representing the user) and produces output data (principle similar to web services [9], but unlike web services, AL is located in local IS services repository[1]). Every such service is responsible for designated area (e.g. Various types of database operations, calculations, security controls, data processing, etc.) and consists of interconnected and mutually interacting components called binary micro-components (BMC). BMC's have the form of individual executable unit (dynamically linked library satisfying defined conditions, ...) representing functionality fragment. This standalone unit (BMC) can be one class[2], implementing specified interface for example. Common interface allows communication with environment (ODA and other BMC's and services in ODA) and also allows the class to be the building block of services and finally running the service.

1 Of course, services can be placed in distributed environment – arbitrarily in Internet.
2 If considered object oriented paradigm.

Design of IS comprises of three separated sub-designs, which have the form of several diagram types. The first one, Application Logic Design – describes AL. The second one – Security design – designs security aspects – associating user roles with individual services in AL. The third sub-design describes organization structure (hierarchy) of respective services (based on types of service functionality) – Service Hierarchy Diagram. User-designer thus has the possibility of extracting systems design in case of required change. After design extraction, specific integrated development environment (IDE for modifications of design in ODA) can be used to apply required changes – even without breaking of IS's execution and disconnection of users (no consultations with previous designers of system are necessary, because AI agent manages relevant changes). Security considerations of runtime environment and all other security concerns are the responsibility of security manager represented by software agent (SW agent). On the other hand, AI agent's task is to tune-up performance and efficiency for users of IS. It's work includes profiling and setting priorities on behalf of users.

'**Design Provider**' component provides system's design to designer-user through integrated development environment (which also allows modifications of IS for authorized users). '**Agent layer**' component takes care of management of AI agents, namely it embraces creation, collaboration environment and destroying of AI agents. Component named '**Remoting Provider**' mediates communication between remote information systems (based on ODA) through heterogenous environment as networking environment or inter-process channels. So this component is important for communication between run-time environment (RE) and service users, and in the similar way for information interchange between potential user-designer (represented for runtime environment by IDE) and RE. Services of the IS can also be used by remote IS – if it has adequate access rights of course. '**Security manager**' is the component constituted by software agent administering security concerns, it manages public keys for encrypted communication, or protects from unauthorized accesses for example.

Since ODA strictly separates functionality of system from presentation logic, front-end client can be represented by Windows (or other operating system) application, web application, web service, etc. Only important requirement for client is to have proper interface for connecting and using of application functionality. Through this standardized interface, client is able to utilize all services of ODA and respective IS (services of binary micro-components, services of authentication, secure connection, ...). '**Information System Manager**' component serves as a manager of '**IS repository**', activities it incorporates include adding/removing of IS to/from repository and other actions for administration of all stored IS. IS repository is the storage place, where all information systems (their BMC's, application logic, documentation, ...) for local machine are warehoused (in ODA). '**Service Provider**' executes requested services, according to calls from clients or calls from other services in the same service hierarchy as executed service is. Component '**Run-time Controller**' is the

boss of runtime-environment. It loads RE and its components when started, shuts down all environment when stopped. This component also activates and deactivates other components of RE and thereby rules operation of all IS kept in IS repository. Figure 4 shows main parts of ODA.

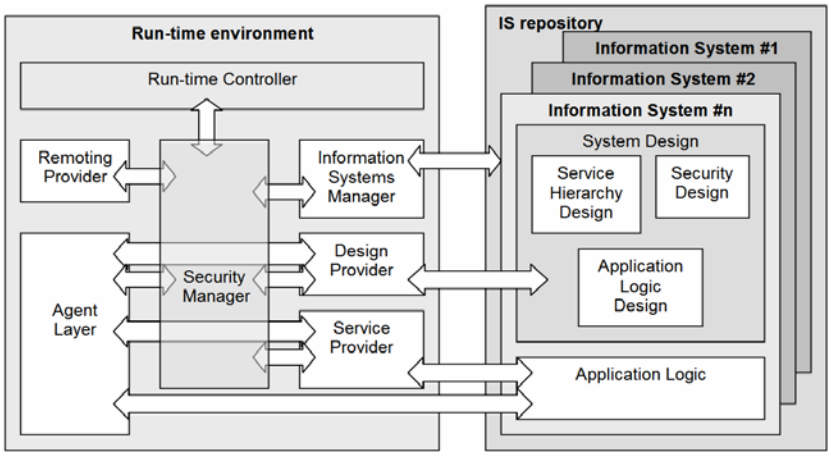

Figure 4
Open Design Architecture

Meaning of arrows in RE is as follows: All data incoming to RE (let's call it messages) from outside environment pass through component 'Remoting Provider'. Messages are filtered, verified and validated using component afterwards 'Security Manager'. This leads messages to 'Service Provider' component, executing particular service of application logic. Results of service execution are sent through 'Security Manager' and 'Remoting Provider' back to client. Analogous process is observed by adding/removing of information system, or accessing of some service's design, except that components 'Information Systems Manager', or 'Design Provider' are used instead of 'Service Provider'.

If change in IS design (and thereby in whole structure of IS) is requested, current design of IS is sent to designer-user through 'Design Provider'. After potential modifications by designer, the design is sent back to 'Design Provider' (through 'Remoting Provider' and 'Security Manager'), where modified design is transcribed to original design of IS. This change can be accomplished in running IS without disconnection of IS users. Changes in running system can bring many problems, like security risks, loss of data or inefficiency of system. This is the point, where properties of agents show handy. As was mentioned, software agent is responsible for security of whole RE. Software agent was chosen because determinism of software agents for such critical issues like security is better than non-determinism of AI agent. On the other hand, AI agent can show even better properties, at the cost of occasional inaccuracies (non-determinism). AI agent was therefore chosen to optimize speed, performance and availability of running

services in multi-user environment. Information, which is used for this purpose is called ‘user profile’ of service consumers (physical clients or external IS). AI agent thus stores evidence of each user including:

- number and times of log-on actions of client for particular IS
- sequence of services used by client after log-on
- periodicity of particular service usage by client
- time of execution for called services
- priorities of service consumers

These factors together (maybe more will be added later) comprise ‘user profile’, upon which AI agent decides on reallocating of resources, or even suspending some users for minor time intervals. Each role of service consumer is assigned its priority, expressed as number. As this priority is part of user profile in RE, AI agent considers this priority and precedence of high priority roles is ensured. This way, AI agent can improve overall performance of IS. Seeing that AI agent is not fully deterministic, it doesn't control execution directly, but instead it creates recommendations and sends them to ‘Service Provider’ through ‘Security Manager’. ‘Service Provider’ decides whether AI agent's recommendations are not suspicious, and decides whether to accept them. If ‘Service Provider’ receives suspicious recommendations (e.g. it receives none or distorted ones), it means that AI agent failed and safe mode of operation is activated to ensure services availability. One of such ‘safe’ algorithms used temporarily for IS is FIFO (in other words: first come, first served).

According to presented ODA principles, ODA combines properties of SOA [9] – regarding services, MDA – regarding design of IS independent from target platform of IS and CBA – services consist of binary micro-components.

5 Future Work

Next phase of work on successful realization of ODA will be directed at specification of essential diagrams of ‘System Design’ part and their precise semantics. The work will also be aimed at AI agent and software agent specification.

One of last parts, that we are able to identify now, will be implementation of mentioned components of runtime execution environment together with security mechanisms ensuring secure IS operation in context of ODA.

Last phase will be design and implementation of prototype IS, where expected properties of architecture will be checked and supposedly demonstrated. The goal

of the prototype will be to convey integrated view of possibilities offered by ODA.

Conclusions

Designing and implementing of flexible, secure and easy scalable information systems isn't simple process. We must solve a lot of problems such as security, scalability, performance, etc. These challenges can require invariably knowledge from artificial intelligence area (with knowledge of system architectures in IT area).

We presented a new architecture for information system and its runtime environment. Information systems will provide services that are characterized with consistent modularity, which beside others separates functionality from presentation logic.

To the potential advantages of such architecture we anticipate quick development, simplified modification of individual IS services.

To achieve introduced features the information system will consist (except its own functionality, which is generally in binary form) also from meta-information called 'System Design', which can be edited in order to effectively modify its structure and behavior.

Acknowledgement

This work is supported by this project:

VEGA 1/2176/05 – Technologies for Agent-based and Component-based Distributed Systems Lifecycle Support

References

[1] What is software agent? Thales, Research & Technology (UK) Ltd

[2] Belicak M., Paralič M., Havlice Z.: Distributed Service-Oriented Information Systems with Open Architecture, in Proceedings of Electronic Computers and Informatics ECI 2006, The Seventh International Scientific Conference, Košice – Herľany, Slovakia, September 20-22 2006, pp. 8-12

[3] Object Management Group, Agent Platform Special Interest Group: Agent Technology (Green Paper), OMG Document agent/00-09-01, Version 1.0, September 1, 2000

[4] Lüders F: Use of Component-Based Software Architectures in Industrial Control Systems, Mälardalen University Licentiate Thesis No. 18, Department of Computer Science and Engineering, Mälardalen Univ., 2003

[5] Object Management Group, OMG Model Driven Architecture, May 24, 2006, http://www.omg.org/mda

[6] Sriraman B., Radhakrishnan R.: Component-based Architecture Supplementing SOA, Sun Microsystems, March 2005

[7] ZapThink, LLC, Solving the IT Impasse with Service Orientation, Whitepaper, ID: WP-0110, March 2003, http://www.zapthink.com

[8] Tsai W. T., Fan Ch., Chen Y., Paul R., Chung J.-Y.: Architecture Classification for SOA-based Applications, in Proc. of ISORC'06, April 2006, pp. 295-302

[9] Dragoni N., Gaspari M.: Integrating Agent Communication Languages in Open Services Architectures, Technical Report UBLCS-2003-12, Department of Computer Science, University of Bologna, Italy, 2003

[10] Erl T.: Service-Oriented Architecture - Concepts, Technology, and Design, Prentice Hall Professional Technical Reference, July 2005

[11] Agrawal R., Bayardo R. J. Jr., Gruhl D., Papadimitriou S: Vinci: A Service-Oriented Architecture for Rapid Development of Web Applications, in Proc. of the 10th International Conference on WWW, 2001

[12] Taylor K., Austin T., Cameron M.: Charging for Information Services in Service-Oriented Architectures, in Proc. of the IEEE EEE05 international workshop on Business services networks BSN '05, March 2005

[13] Cuomo G.: Databases and Service-oriented Architectures: IBM SOA "on the edge", in Proc. of the ACM SIGMOD Management of data, June 2005

[14] Alonso G., Casati F.: Web Services and Service-Oriented Architectures, ICDE (archive), in Proceedings of the 21st International Conference on Data Engineering (ICDE'05), Volume 00, 2005, p. 1147

[15] Papazoglou M. P., Georgakopoulos D., Guest Editors: Introduction to Service-oriented Computing. Communications of the ACM, October 2003, Vol. 46, pp. 24-28

[16] O'Toole A.: Web Service-Oriented Architecture – The Best Solution to Business Integration, IT Management News, 2003 (SecurityProNews, August 15, 2003)

[17] Nunes Rodrigues G.: A Model Driven Approach for Software Systems Reliability, in Proceedings of the 26th International Conference on Software Engineering (ICSE'04), May 2004, pp. 30-32

[18] Sheng Q. Z., Benatallah B.: ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services, In Proc. of the (ICMB'05), July 2005, pp. 206-212

[19] Hemme-Unger K., Flor T., Vögler G.: MDA Development Approach for Pervasive Computing, OOPSLA'03, Anaheim, CA, USA, 18th ACM SIGPLAN conf. on OOP, systems, languages, and app., Oct. 26-30, 2003

[20] Abd-Ali J., El Guemhioui K.: An MDA-Oriented .NET Metamodel, in Proc. of the 9th IEEE Int. EDOC E. Computing Conf., 9/2005 pp. 142-156

[21] Kanisová H., Müller M.: UML srozumitelně, Computer Press a.s., 635 00 Brno, CZ, 2004, pp.145-152

[22] Component-based Architecture And Modeling and Simulation, Keane, NDIA, In Proc. of 5th Simulation-Based Acquisition/Advanced Systems Engineering Environment Conf., June 24-27, 2002