Towards A Classification Model for Component-Based Software Engineering Research

Gerald Kotonya, Ian Sommerville and Steve Hall
Computing Department
Lancaster University, UK
gerald, is, halls1{@comp.lancs.ac.uk}

Abstract

Accurate and timely information is a key motivator in the widespread adoption of CBSE technology in Europe. Although there are overlaps and informal communications between researchers and adopters of CBSE technology in Europe, there is no systematic mechanism for information interchange between the two. CBSEnet is a European Union initiative to create an Internet-based forum for the exchange of information between researchers and adopters of CBSE. This paper describes a proposed classification model for CBSE research that will form the basis for structuring the CBSEnet knowledge base.

1 Introduction

Component-based software engineering is being proposed as a means of improving software quality and reducing development costs. The drive to use components to construct software systems stems from a 'parts' philosophy derived from traditional engineering disciplines that promises instant productivity gains, accelerated time to market and lower development costs. However lack of accurate and timely information is hampering the widespread adoption of CBSE technology in Europe. In the past 3 years the European Union sponsored more than 20 research projects in CBSE under its Information Society Technologies (IST) Programme [25]. Although there are overlaps and informal communications between research initiatives throughout Europe, there is no systematic mechanism for information interchange between researchers and adopters of CBSE technology. We believe that researchers and potential adopters of CBSE could benefit from a more systematic approach to information interchange. Consequently, CBSEnet is a European Union sponsored initiative, which aims to:

- Create a European-wide forum for the exchange of information between researchers, developers working in the area of CBSE.
- Suggest how CBSE technologies could improve software engineering processes in different domains.
- Propose future research requirements for the development and deployment of CBSE technologies.

Central to these aims is the creation of a knowledge base to publish results and research issues on CBSE electronically. The knowledge base is accessible via the Internet and will be structured according to a CBSE classification model, which will in turn, be organised according to issues specific to CBSE. The short-term objective of the classification model is to provide a mechanism for establishing the current state of CBSE. Its long-term objective is to provide a basis for identifying gaps between CBSE problems and existing solutions. This paper focuses mainly on the short-term aims of the classification model; to provide a scheme for classifying CBSE research results.

The model is expressed in terms of CBSE research results, application domains and CBSE aspects to provide a schema that relates research results to relevant CBSE aspects and software domains (Figure 1). The prototype is implemented as part of the CBSEnet Web portal and is supported by query and visualisation mechanisms [26].

Six types of research results are identified in the model, comprising conference and journal articles, technical reports, white papers, books and tools. Each research result is identified by bibliographical information, location and optional project. The model relates research results to 8 CBSE aspects comprising concepts, processes, roles, product concerns and business concerns, technology, off-the-shelf components and related development paradigms. These have further been decomposed into more than 50 sub-aspects. We have also identified 16 application domains ranging from avionics to embedded systems through to utilities. The model maps application domains to the relevant CBSE aspects through the problem being researched.

The rest of the paper is organised as follows: Section 2 defines the CBSE aspects used in the model. Section 3 describes the application domains. Section 4 describes a simple method and an example for using the classification model. Section 5 provides a summary and some concluding thoughts.

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE

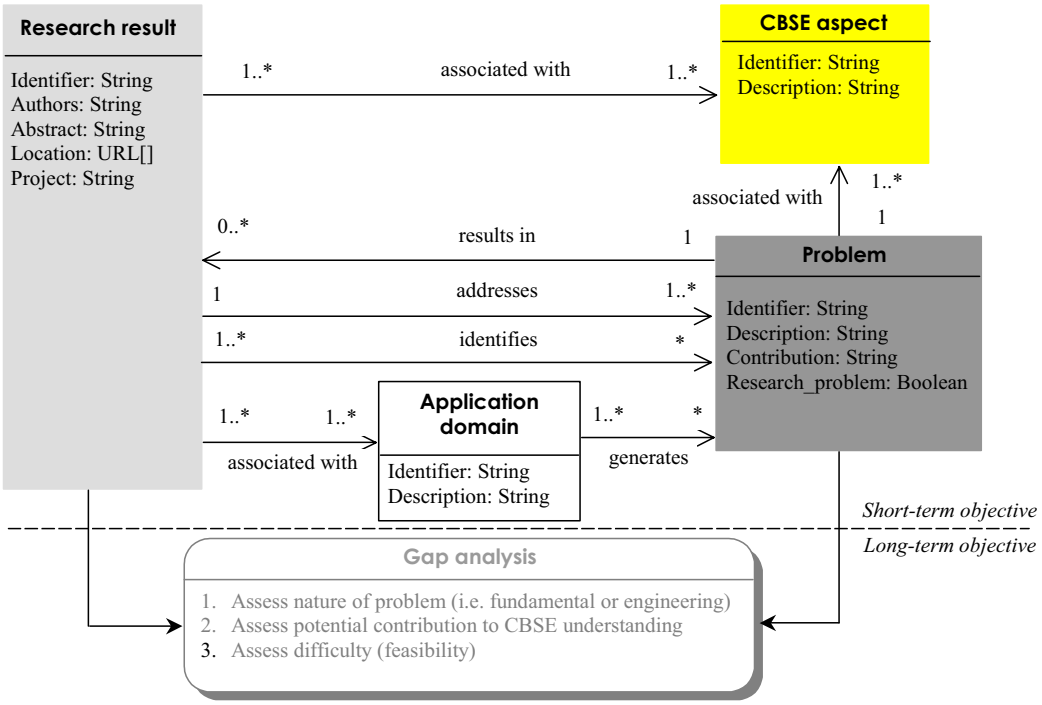

Figure 1 Classification model for CBSE research

2 CBSE Aspects

2.1 Concepts

We have identified eight principal concepts in component-based software engineering:

(i) **Component**. A software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard [2]. General examples of concrete components include interface, computational, memory, manager, controller components and Web services. Interface components transform the representation used by one system component into the representation used by another component. An example of an interface component is a human interface component that takes some system model and displays it for the human operator. Computational components perform a computation of some sort. Usually, the input and output to the component are fairly simple, e.g. procedure parameters. Examples of computational components are (mathematical) functions and filters. A memory component maintains a collection of persistent, structured data, to be shared by a number of other components. Examples are databases, a file system or symbol table. A manager component contains a state and a number of associated operations. When invoked, these operations use or update the state, and this state is retained between successive invocations of the manager's operations. Abstract data types and servers are examples of manager components. A controller component governs the time sequence of other events. A top-level control module and a scheduler are examples of controller components [22]. A Web service is a reusable software component that semantically encapsulates discrete functionality and is distributed and programmatically accessible over standard Internet protocols. Specifically, a web service is a stack of emerging standards that describe a *service-oriented*, component based application architecture.

(ii) **Component model**. Defines specific interaction and composition standards for components [2]. A component may have an explicit context dependency on the operating system or some other software component. An interaction standard specifies the type of explicit context dependencies a component may have. Interface standards are the mandatory requirements employed and enforced to enable software components directly interact with other software components. An interface standard declares what can comprise an interface. A component has two basic interfaces: a *required* and a *provided* interface. Composition standards define how the components can be composed to create a larger structure and how a producer can substitute one component to replace another that already exists within the structure. Examples of component models include EJB, COM+ (.NET) CORBA

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE

Component Model (CCM) and the Web service model [2, 24].

(iii) **Component model implementation.** Denotes the dedicated set of executable software elements required to support the execution of components that conform to the model. Model implementations enforce the interaction and composition standards defined by the model and provide run-time management services.

(iv) **Component interface**. Specifies a flow of dependencies from components that implement services to the components that use these services. A component supports a provided interface if the component contains implementations of all operations defined by that interface. A component needs a required interface if the component requests an interaction defined in that interface and expects some other component to support that interface. A component may be unable to provide an interface if one of its required interfaces is unfulfilled [2].

(v) **Contracts**. A contract specifies behavioural composition in terms of participating components, contractual obligations for each participant, invariants, and operations to instantiate the contract. Contracts provide a powerful mechanism for isolating, modifying and combining cohesive units of behaviour to support the reuse and refinement of software components [3].

(vi) **Service.** A *service* is an abstraction of a set of functions that is designed to achieve some logical purpose [23]. For example a printing service embraces functionalities related to printing.

A *service* is represented as a set of access interfaces. A printing service, for example, may be offered through a number of interfaces:

- Printing of text files
- Management of the printing spool area
- Installation of a new printing device.

(vii) **Architecture**. Components are designed on a pre-defined architecture so that they can interoperate with other components and/or frameworks. There are three different views of the architecture:

- *Runtime.* This includes frameworks and models that provide runtime services for component-based systems.
- *Design-time.* This includes the application-specific view of components, such as functional interfaces and component dependencies.
- *Compose-time.* This includes all the elements needed to assemble a system from components, including generators and other build-time services; a component framework may provide some of these services [4].

(viii) **Patterns**. A design pattern is a way of reusing abstract knowledge about a problem and its solution. It is a description of the problem and the essence of its solution. Patterns can be considered at different levels of abstraction. An architectural pattern expresses a fundamental structural organisation or schema for software components. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organising the relationships among them. A design pattern provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes commonly recurring structure of interacting components that solves a general design problem within a particular context [2, 9].

2.2 CBSE Processes

The CBSE process consists of two separate but related processes. The first is concerned with the analysis of application domains and the development of domain-related components (i.e. development for reuse). The second process is concerned with assembling software systems from prefabricated (off-the-shelf) components (development with reuse). The two processes are linked via a component market [7, 8,10,11].

2.3 Development for reuse

The component development process is concerned with developing generic and domain-specific components. To achieve successful software reuse, commonalities of related systems must be discovered and represented in a form that can be exploited in developing similar systems. Domain commonalities are used to develop models or software components that can be used to develop systems in the domain. Once reusable components are created, they can be made available within organisations or on the open market as commercial components.

For organisations to achieve the benefits of sharing and reusing components a successful reuse strategy must include the following:

1. *A reuse method that is consistently applied by component and application developers. The method that supports reuse contains the following steps:*
   - Classify market/business requirements by services (e.g. application, interface, support, or core).
   - Search repository for reusable components that support the requirements.
   - Analyse candidate components to ensure there is an acceptable match between them and the requirements.
   - Develop components based on standard component model(s). Placing the new component information into the repository.
   - Incorporate the reuse methodology into system development life cycle.
2. *The establishment of component review mechanisms whose function is to identify common components by identifying commonalities of related systems.*

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE

Within organisations, a framework needs to be put in place that allows for:

- Centralised management of reusable, shareable components
- Design reviews of new and existing projects for reusable components
- Enterprise access to information about reusable components

For market-driven software, a framework needs to be put in place that allows for:

- Reviews of market trends
- Reviews of commonalities of related systems
- Access to market information about reusable components

3. *Establishment of component design reviews of all ongoing projects.*
   - Determine whether the market/business requirements are met by any existing components.
   - If the market/business requirements are not met by existing components, determine if there are any components that can be modified/aggregated to satisfy the requirements.
   - Analyse existing applications for potential additions to the component repository.
4. *Establishment of a repository for maintaining the information about available reusable components.*
5. *Establishment of effective component "harvesting" mechanisms*. Harvesting involves the examination of legacy application for the purpose of identifying functions that can be isolated into standalone program modules. Harvesting should be supported with "wrapping" or encapsulation techniques for migrating the modules to reusable components (i.e. defining standard APIs for the services implemented by the modules).
6. *Adopting effective component management methodologies*. These include the tools to support component reuse and a method for managing the available components.

2.4 Development with reuse

Figure 2 shows our suggested component-based application development process. It develops some of the early ideas on component-based development [4] to provide a scalable process with a clear separation of concerns. The negotiation phase attempts to find an acceptable trade-off amongst multiple (often) competing development attributes. The planning phase sets out a justification, objectives, strategies (methods and resources to achieve development objectives) and tactics (start and end dates, and tasks with duration) for the project.

The development phase implements the agenda set out in the planning phase. Essential issues to be addressed in phase include:

(i) **Requirements**. The process of eliciting and defining requirements for a component-based system. The process includes requirements elicitation, requirements scoping, requirements modelling and specification.

(ii) **Designing system architecture**. The process of partitioning system requirements (services and constraints) into logical "components" or "sub-systems" [22]. Formal Architecture Description Languages (ADLs) have emerged as an effective mechanism for describing architectural elements and their relationships, and for supporting their refinement through levels of abstraction [21]

(iii) **Composition**. The process of replacing abstract design components with "concrete" off-the-shelf components. Off-the-shelf software components are packaged in many different forms (e.g. function libraries, frameworks and legacy applications). The composition process must devise mechanisms for integrating the different components without compromising the system quality (i.e. ensuring that the components conform to the adopted component model). The integration process may make use of some "gluing technology", which may be unrelated to the components, to provide an interface between components.

(iv) **Verification.** The verification process is intended to ensure that there is an acceptable match between the software components used to build the system and the system being built. The verification process varies in focus and detail across the development cycle. At the *requirements* stage, verification is used to establish the availability of software components and viability of a component-based solution. At the *design* stage verification is concerned with ensuring that the design matches the system context (i.e. in terms of non-functional requirements, architectural concerns and business concerns). This may require detailed blackbox testing of the software components and architectural analysis of the design.

At the *composition* stage verification translates to design validation, through component assembly testing and system testing. In summary, component-verification regimes should serve 8 aims [11]:

- *Availability*. Establishing the availability of potential off-the-shelf software solutions.
- *Viability*. Establishing the viability of a component-based solution for the application.
- *Discovery.* Exposing undocumented features and faults in the software components.
- *Specification verification.* Verifying vendor data/specification.
- *Fitness for purpose.* Establishing how well the component capabilities fit in with the system needs.
- *Masking.* Establishing the extent to which it is possible to mask out unwanted component features.
- *Adequacy.* Setting out realistic test adequacy criteria that take into account the resources

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE

available and criticality of the component being tested.

- *Early validation.* Early component verification and validation to minimise repair delays caused by slow repair-response times.

(v) **Management**. The process of managing assets, system configurations, change, and assuring quality.

- *Asset management.* Providing a framework for managing the acquisition, usage and evolution software components. The inventory of software components, versions, where they reside, and the financial obligations licenses associated with them, are a critical aspect of system management [8].
- *Evolution.* Providing schemes for assessing the cost and difficulty of evolving component-based systems. Software component producers and customer organisations are motivated by different objectives to evolve their software, which makes it difficult for user organisations to anticipate and to plan for change. Successful commercial software components have a wide user base. One of the advantages of this is to spread the cost of the component across many users. However, this also means that for the most part, the evolution of third-party components is influenced by diverse user organisations whose interests may not overlap [2].
- *Quality control.* This involves providing quality control procedure and standards to address the problem of the fault identification, repair, and the tracking of system of fixes.
- *Configuration management.* Providing a framework for tracking and controlling the versions of blackbox software components and custom software installed at all locations for the system.
- *Market research.* The different vendor-customer evolution cycles present a particularly difficult system management problem. Vendor evolution cycles are largely driven by market changes. This process is intended to provide a framework for factoring market research into the process. Market research is essential in anticipating the availability of new commercial hardware and software products, and providing a framework for determining the timescale and kind of changes that will be made to the commercial components that are already in a component-based system [8].

.

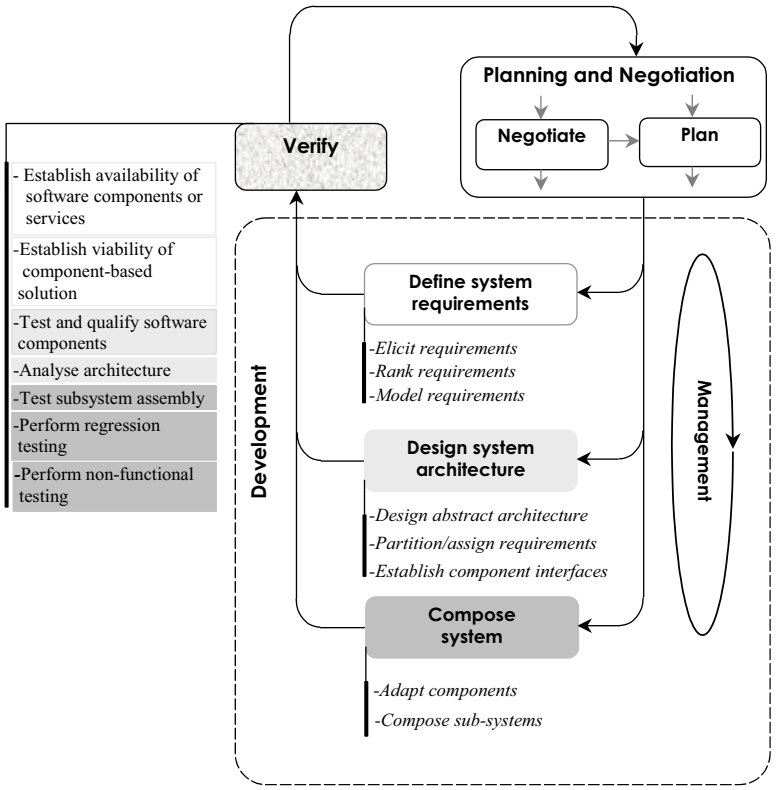

Figure 2 Component-based application development process

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE

2.5 Roles

The radical departure from the conventional software development process has created new roles and stakeholders. The most significant are the roles of the component developer and application developer. Unlike conventional software development where the two roles occupy the same development process, in CBSE, the component developer and the application developer may operate in separate processes [4,14]. This has created a situation in which the system developer has limited knowledge of the software components used to construct a system, and therefore little control over their functionality and the evolution of the system. Other roles created by CBSE include component brokers, component vendors, system integrators and component certification authorities.

2.6 Business and Product concerns

Concerns in component-based development range from risks in adopting the technology, to difficulties in establishing the quality of third party software components, through to problems of managing component-based systems [7]. These concerns may be broadly divided into product and business concerns. Business concerns are associated with the risk of adopting software component technology and events that may result in loss of business through failure to realise business objectives or, in the case of developer organisations, failure to deliver a system on time or within budget. Product concerns are related to the quality of component technologies used to build the system, for example, unreliable components [19, 20]. Technology and business risks are not mutually exclusive. Some risks may be related to both the business and technology. It is also important to note that business and product concerns sometimes oppose each other, for example, a business risk might be failure to meet a market window and a possible risk reduction strategy may be faster development using component-based development [13].

2.7 TECHNOLOGY

Technology reflects issues related to component technology standards, tool support, development environments and methods.

2.8 OFF-THE-SHELF SOFTWARE COMPONENTS

This aspect is concerned with issues that affect the procurement and utility of Off-The-Shelf software components including COTS (Commercial-Off-The-Shelf) components. These include documentation, classification, legal, market, repository, component quality (dependability, performance, timeliness, usability, maintainability, etc) and trust issues.

2.9 RELATED DEVELOPMENT PARADIGMS

Describes complimentary and related development paradigms. Of particular interest are relationships between CBSE and service-centric software engineering, model-driven, aspect-oriented and agent-based software development.

3 Application Domains

Several taxonomies for application domains have been proposed [1,19]. Perhaps the best known of these are the taxonomies proposed by Digital, IBM and Reifer [15,16,17]. Digital and IBM have proposed taxonomies that are focused on business information systems. Reifer has proposed a taxonomy that is driven by real-time and scientific applications. However, there is little consistent basis for classification, across the taxonomies and within them. The main reason for this is that many of them contain elements related to both the application (problem area) and the implementation of the solution.

In Digital's application taxonomy, "capital assets", "physical resources" and "cost accounting" are listed under "accounting". It is not clear whether these represent applications or application areas. Like other taxonomies, Reifer's application taxonomy suffers from the use of inconsistent classification criteria. For example, under data processing, we see "accounting" which is usually regarded as an application domain, followed by "banking and finance", which represents one or more industries. The lack of taxonomic structure suggests that the originators of the taxonomies did recognise this distinction. To address this problem, a number of task groups have been setup under the umbrella of the Object Management Group (OMG) to identify relevant standards, business architectures, research and technologies in various application domains [19].

The application taxonomy used in the CBSE classification model (Table 1) is a combination of the IBM, Reifer and those defined under the OMG task groups. Our taxonomy is intended to act as a starting point for identifying application domains and is by no means definitive. A section of this taxonomy is shown in Table 1.

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE

Domain	Application
Avionics	Air traffic control
	Electronic warfare
Command and Control	Space
	Satellite
	Other
Embedded systems	Operating systems
	I/O controllers
	ASIC
	Other
Electronic commerce	Agents
	Brokerage
	Electronic Data Interchange
Finance	Accounting
	Banking
	Insurance
Healthcare	Emergency care
	Home care
	Primary care
Real-time	Controllers
	Sensors
	Signal processors
Simulation	Environmental simulators
	War-gaming
Telecommunications	Network management
	Network engineering
Utilities	Transmission, distribution, marketing and retailing functions of electric, water and gas utilities.

Table 1 Application domains

4 Using the classification model

This section illustrates how the CBSE classification model works in practice. The example uses three tables to relate research result, application domain and CBSE aspects. The tables have been simplified to model only limited attributes in each case. A simplified example using research results from 4 EU sponsored IST projects on CBSE is used to illustrate the model. Because of space limitations we can only show a few details, however, more examples can be found in CBSEnet portal [25].

The approach works as follows:

1. Identify the research item and its type (e.g. book, conference/workshop article, journal article, technical report, white paper, or tool) – Table 2.
2. Identify and rank the application domains associated with the research result in (1). For *core* application domains enter ●, for *peripheral* application domains enter ○. A research result associated with the *general* category should not appear in another category (Table 3).
3. Identify and rank the CBSE aspects addressed by or closely associated with the research result in (1). For *core* CBSE aspects enter ●, for *peripheral* aspects enter ○ (Table 4).

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE

(1) Identifying the research result item and its type

		Type					
		Book	Conference/ workshop article	Journal article	Technical report	White paper	Tool
Research Result	Item: CADL: Component Architecture Description Language Project: ECOADM URL: www.ecoadm.ccs.es				√		
	Item: COREx: Component Oriented Requirements Expression Project: ECOADM URL: www.ecoadm.ccs.es		√				
	Item: PECOS: Components for embedded software Project: PECOS URL: http://www.pecos-project.org/publications.html		√				√
	Item: OOSPICE-Software Process Improvement for Component-Based Software Engineering Project: OOSPICE URL: http://www.oospice.com/publications/publications.html		√				
	Item: BIT - Built-In-Test Vade Mecum : A common BIT architecture Project: COMPONENT+ URL: www.component-plus.org/sidor/reports.htm				√		

Table 2 Nature of research item

(2) Identifying and ranking application domains associated with research results

		Application Domain														
		Avionics	Command and control	Embedded systems	Electronic commerce	Enterprise computing	Finance	Healthcare	Real-time	Manufacturing	Software engineering	Scientific	Simulation	Telecommunications	Transportation	Utilities	General
Research result	CADL					●	●								○		
	Designer					●	●								○		
	COREx					●	●								○		
	PECOS			●													
	OOSPICE																●
	BIT			○		●			○								

Table 3 Relating and ranking research result to application domain

5 Summary and conclusion

Lack of timely and accurate information is hampering the widespread adoption of CBSE technology in Europe. In the last 3 years the EU under its IST programme has sponsored more than 20 research projects covering many aspects of CBSE. However, despite overlaps of interest between researchers and

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE

adopters of CBSE technology, there is no systematic mechanism for information interchange between researchers and adopters of CBSE technology.

This paper has provided an overview of CBSEnet, a EU sponsored initiative to create a European wide forum for the exchange of information between researchers and adopters of CBSE. We have also described a CBSE classification model, a key component of CBSEnet. The CBSE classification model has a short-term objective of providing a mechanism for establishing the current state of CBSE and a long-term objective of providing a basis for identifying gaps between problems in CBSE and proposed solutions as means for new research. In this paper we have focused largely on the short-term objective. We have proposed a simple but effective solution for classifying CBSE research results based on an integrated framework that relates research results to relevant application domains and specific aspects in CBSE. We have demonstrated using examples from 4 IST projects how the model can be used to classify research results. The solution has been implemented as part of the CBSEnet Web portal and includes extensive visualisation and query facilities. The implementation is being extended to include statistical analysis and a scheme for gap analysis.

The classification model represents our first pass at defining a scheme for structuring CBSE research results; we believe the model will be refined over the coming months and evolve to address other aspects of CBSE as the technology evolves. We are currently in the using the model to classify research results from 20 IST research projects on CBSE spanning the last 3 years.

(3) Identifying and ranking CBSE aspects associated with the research results
(We are unable to show all the aspects here due to lack of space)

			Research result					
			CADL	Designer	CORex	PECOS	OOSPICE	BIT
CBSE Aspect	**Concepts**	Architecture	●			●		●
		Component	●					●
		Component model	●					
		Component model implementation						
		Contracts						
		Interface	●					●
		Service	●		○			○
		Patterns	○					
	Process	Requirements			●	●		
		Design	●	●		●		
		Composition	●	●		●		
		Management	○		○			
		Verification	●					●
		Procurement			○			
		Deployment						
	Roles	Broker						
		Designer						
		Developer						●
		Integrator						●
		Maintainer						
	Business	Adoption						
		Cost						
		Risk						○
		Scalability						
	Product	Adaptability	●	●				
		Availability						
		Compositionality	●	●				
		Efficiency						
		Interoperability		●				
		Reliability						●
		Security						

Table 4 Mapping ranked research results to relevant CBSE aspects

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE

ACKNOWLEDGMENT

The work described here has been carried out under the EU IST Programme 2001-35485, Component-Based Software Engineering Network. We would like to acknowledge the contributions of our partners: Ingegneria Informatica S.p.A, Athens Technology Center S.A, ESI, SINTEF, and MTA SZTAKI.

References

1. Glass, R. and Vessey, I.: "Contemporary application domain taxonomies", *IEEE Software*, 12(4), 1995, pp. 63-76.
2. Heinman, G.T. and Council, W.T: "Component-based Software Engineering: Putting the pieces together", Addison-Wesley, 2001.
3. D'Souza, D. F. and Wills, A. C.: "Objects, Components, and Frameworks With UML: The Catalysis Approach", Addison-Wesley, 1998.
4. Brown, A.W and Wallnau, K.C.: "The current state of CBSE", *IEEE Software*, 15(5), 1998.
5. Coker, l, Hayes-Roth, R.: "Services first, components second!", OMG-DARPA Workshop on Compositional Software Architectures, 1997.
6. Kotonya, G., Hutchinson, J., Onyino, W. and Sawyer, P.: "COTS-Based System Development: Processes and Problems", Proc. of 4th International Information Society Conference: Development and Reengineering of Information Systems, Oct. 2001, Ljubljana, Slovenia.
7. Voas, J.M.: "The challenges of using COTS software in component-based development", Computer, 1998, 31(6), pp. 44.
8. Vigder, M.R., Gentleman, W.M., and Dean, J.: "COTS Software Integration: State of the Art", Institute for Information Technology, National Research Council, Canada, 1996.
9. Johnson, R.E.: "Frameworks = (Components + Patterns)", *Communications of the ACM*, 40(10), 1997, pp. 39-42.
10. Voas, J.M.: 'Disposable information systems: The future of software maintenance?' *Journal Of Software Maintenance-Research And Practice*, 1999, 11(2), pp.143-150.
11. Harrold, M.J., Liang, D., and Sinha, S.: 'An Approach to Analysing and Testing Component-based Systems', International Workshop on Testing Distributed Component-based Systems, ICSE'99, Los Angeles, California, May 1999.
12. Ncube, C., and Maiden,N.A.M.: "PORE: Procurement-oriented requirements engineering method for the component-based systems engineering development paradigm", Proc. IEEE International Workshop on Component-Based Software Engineering, Los Angeles, California, USA, May 1999, pp.1-12.
13. Kotonya, G. and Rashid, A.: "A strategy for managing risk in component-based systems", Proceedings of IEEE 26th Euromicro Conference, September 2001, pp. 12-21, Warsaw, Poland.
14. Kozaczynski, W., and Booch, G.: "Component-based software engineering", *IEEE Software*, 1998, 15(5), pp.34
15. IBM Corp.: "Industry applications and Abstracts", IBM Corp, White Plains, N.Y., 1988.
16. Reifer Consultants: "Productivity and Quality Survey", El Segundo, California, 1990.
17. AFIPS: "Taxonomy of Computer Science Engineering", American Federation of Information Processing Societies, Arlington, Va. 1980.
18. OMG: "OMG in Motion", July 2000
19. ISO/IEC Standards 9126 (Information Technology) - Software Product Evaluation- Quality Characteristics and Guidelines for their use, 1991) and 14598 (Information Technology - Software Product Evaluation: Part I, General Overview: Part 4, Process for Acquirers; 1999)
20. IEEE Computer Society: "IEEE Standard for Software Quality Metrics Methodology. IEEE Std. 1061-1992", IEEE Computer Press, New York, 1992.
21. Medvidovic, N. and Taylor, R.N.: "A Classication and comparison Framework for Software Architecture Description Languages", Trans. IEEE Software Eng., 26 (1), January 2000, pp.70-93.
22. Sommerville, I: "Software engineering", Addison-Wesley, 6th Ed., 2001.
23. Saridakis, T: "Component-based system development lecture", http://www.cs.hut.fi/~saridaki/pdf/CBD-LectureNo5.pdf, 2002.
24. Gottschalk, K., Graham, S., Kreger, H. and Snell, J.: 'Introduction to Web Services Architecture', IBM Systems Journal Vol 41(2), 2002.
25. http://dbs.cordis.lu/fep-cgi/srchidadb?CALLER=PROJ_IST&QZ_WEBSRCH=component-based
26. www.cbsenet.org

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE