INTEGRATING THE CONCEPT OF SYNTHESIS IN THE SOFTWARE ARCHITECTURE DESIGN PROCESS

Bedir Tekinerdogan
Mehmet Aksit
Department of Computer Science, University of Twente, The Netherlands

Synthesis is a widely applied problem-solving approach of mature engineering disciplines including the sub-processes of technical problem analysis, identification and composition of solution domain concepts, and alternative-space analysis. Current software development processes do not adopt an explicit synthesis process and as such may fall short in identifying, composing and evaluating the relevant concerns. In order to advance ad hoc *software development process to a mature engineering* discipline *it is necessary to integrate the concept of synthesis in current software engineering processes. In software engineering, software architecture design forms a key artifact including the early design decisions, which embodies the overall structure that impacts the quality of the overall system. For ensuring the quality of software architecture, it is necessary to identify and compose the relevant concerns. For this, we integrate the concept of synthesis in the software architecture design process and present the synthesis-based software architecture design process. This approach differs from existing software architecture design approaches since it explicitly includes the synthesis sub-processes of technical problem analysis, solution domain analysis and alternative space analysis, integrating these in a common process.*

Keywords: *synthesis, software architecture design.*

1. Introduction

In order to grasp the essence of software engineering and understand its inherent problems, a critical analysis from a broad perspective is required. To this aim, in this paper we consider software engineering basically as a problem solving activity, whereby software solutions are produced for technical problems. To explicitly reason about the concepts of problem solving, a model for problem solving that may be used for analyzing various problem-solving activities is presented. This model is used for analyzing problem solving in software engineering and comparing it with the more mature engineering disciplines.

A basic concept that is derived from our analysis process that may be essential for software engineering is the concept of *synthesis*. Synthesis is a well-known problem solving process that is broadly and successfully applied in the traditional engineering disciplines. It includes explicit processes for *technical problem analysis*, *solution domain analysis* and *alternative space analysis*. In the problem analysis process, technical problems are identified and structured into loosely coupled sub-problems that are first independently solved and later integrated in the overall solution. In the solution domain analysis process, solution abstractions are extracted from the corresponding solution domains. In the alternative space analysis process, different alternative solutions are searched and evaluated against explicit quality criteria. The synthesis process is basically defined as interplay among the three sub-

processes of technical problem analysis, solution domain analysis and alternative space analysis. In mature engineering, it has been shown that it is effective in designing robust systems that adhere to the corresponding quality attributes and constraints.

Unfortunately, in current software engineering practices the synthesis concept is not known and the three processes are not fully integrated. In general, an explicit problem analysis process does not exist, solution domain analysis is still not fully integrated in software design processes, and alternative management is usually done in an implicit manner and is practically missing. Obviously, synthesis is a useful concept in mature problem solving and it is worthwhile to integrate this in software engineering.

In software engineering, software architecture design (Aksit, 2001)(Tekinerdogan, 2000) forms a key artifact including the early design decisions that embody the overall structure that impacts the quality of the overall system. As such, it is expected that enhancing software architecture design processes using synthesis will improve the quality of the software. Our previous studies have shown that that the state-of-the-art architecture design approaches are not aligned with an explicit synthesis process. Usually in software architecture design processes during the problem analysis, solution domain analysis and alternative space analysis is either implicit or not well-defined. This causes a number of problems such as the difficulty in finding stable abstractions, difficulty in leveraging the architecture boundaries and poor semantics of the architectural components. We integrate the concept of synthesis in the software architecture design process and present the *synthesis-based software architecture design process* (*Synbad*). This approach differs from existing software architecture design approaches since it explicitly includes the synthesis sub-processes of technical problem analysis, solution domain analysis and alternative space analysis, integrating these in a common process.

The remainder of the paper is organized as follows. In section 2 we provide the *problem solving for engineering model* (*PSEM*) (Tekinerdogan, 2000) which is the underlying model for the concept of synthesis. In section 3 we show how the PSEM and the concept of synthesis are applied in mature engineering disciplines. In section 4 we analyze software engineering from the problem solving perspective and discuss the main differences with the synthesis process. In section 5 we apply the concept of synthesis to software architecture design and present the synthesis-based architecture design process. Finally section 6 concludes the paper.

2. Engineering as Problem-Solving

In essence, engineering is a problem solving activity and these two are directly interdependent. A common model that represents engineering from a problem-solving standpoint will specifically show the important features of engineering. In this context, we could come up with a very abstract model for problem solving consisting essentially of two concepts: *Need* and *Artifact*. Given a particular need (*Problem*), an artifact (*Solution*) must be provided that satisfies the need. Because of its very abstract nature, all engineering disciplines, including software engineering, apply to this overly simple model. Of course, the counterpart of the abstract nature of the model is that it is less useful in identifying the differences between the existing engineering disciplines and for comparing these. Hence, we are interested in a concrete problem-solving model that describes the separate important concepts needed for understanding and expressing the concepts of engineering. To this aim, we propose the domain-specific *Problem Solving for Engineering Model* (*PSEM*), which is illustrated in Figure 1. In the subsequent sections, PSEM will serve as an objective basis for comparing engineering disciplines. The service industry, whether it is in the area of travel, leisure, entertainment or finance, involves personalized activities requiring interaction and intervention between humans and machines. The U.S. Bureau of Labor Statistics estimates that more than two thirds of all workers in the U.S. are involved in service functions at present and their numbers are increasing rapidly.

This domain-specific model has been developed after a thorough literature study on both problem solving and mature disciplines. In addition to the afore-mentioned problem-solving literature, we have

studied selected handbooks including chemical engineering handbook (Perry, 1984), mechanical engineering handbook (Marks, 1987), electrical engineering handbook (Dorf, 1997) and civil engineering handbook (Chen, 1998). Furthermore, we have studied several textbooks on the corresponding engineering methodologies of mechanical engineering and civil engineering (Cross, 1989, Ertas and Jones, 1996, Smith, *et al.,* 1983), electrical engineering (Dorf, 1997) and chemical engineering (Biegler, *et al.,* 1997).

The model consists of a set of concepts and functions, which are represented by means of rounded rectangles and directed arrows, respectively. Concepts are the necessary fundamental abstractions and the functions are the conceptual processes that describe the interactions between these concepts. The model consists of three fundamental parts: *Problem Solving*, *Control* and *Context*. In the following, we will explain these parts in more detail.

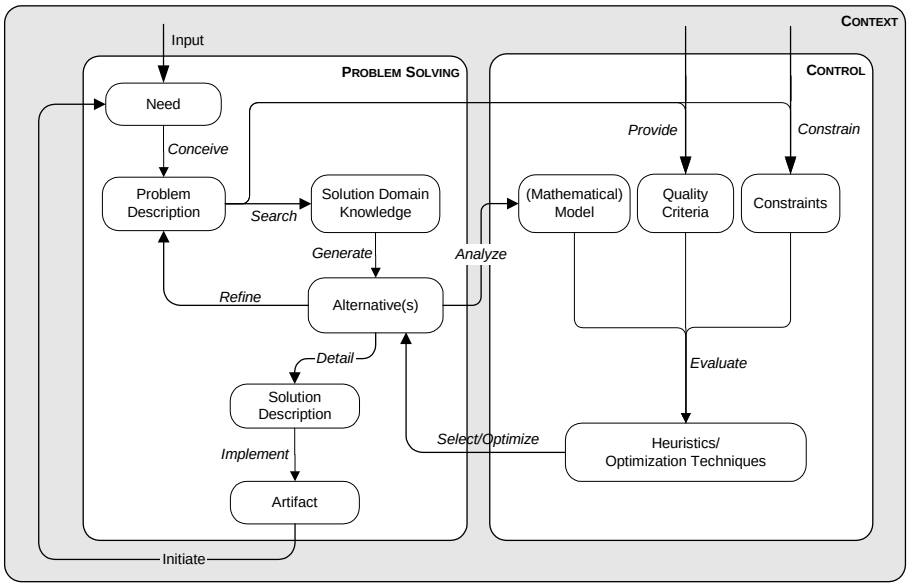

Fig. 1 Problem solving for engineering model (PSEM).

2.1. Problem Solving

The problem-solving part consists of five concepts: *Need*, *Problem Description*, *Solution Domain Knowledge*, *Alternative*, *Solution Description* and *Artefact*.

- *Need* represents an unsatisfied situation existing in the context. The function *Input* represents the cause of a need.
- *Problem Description* represents the description of the problem. The function *Conceive* is the process of understanding what the need is and expressing it in terms of the concept Problem Description.
- *Solution Domain Knowledge* represents the background information that is used to solve the problem. The function *Search* represents the process of finding the relevant background information that corresponds to the problem.
- *Alternative* represents the possible alternative solutions. The function Generate serves for the generation of different alternatives from the solution domain knowledge. After alternatives have

been generated, the problem description can be refined using the function *Refine*. The function *Detail* is used to detail the description of a selected alternative.

- *Solution Description* represents a feasible solution for the given problem. The function *Apply* requires two inputs, *Problem Description* and *Solution Domain Knowledge*. It uses the relevant background information to provide a solution description that conforms to the problem description.
- *Artefact* represents the solution for the given need. The function *Implement* maps the solution description to an artefact. The function *Output* represents the delivery and impact of the concept Artefact to the context. The function Initiate represents the cause of a new need because of the produced artefact.

2.2. Control

Problem solving in engineering starts with the need, while the goal is to arrive at an artifact by applying a sequence of actions. Since this may be a complex process, the concepts and functions that are applied are usually controlled. This is represented by the *Control* part in the model. A control system consists of a controlled system and a controller (Foerster, 1979). The controller observes variables from the controlled system, evaluates this against the criteria and constraints, produces the difference, and performs some control actions to meet the criteria. In PSEM, the control part consists of three concepts: *Representation of Concern*, *Criteria*, and *Adapter*.

- (*Mathematical*) *Model* represents a description of the concept Alternative. The function Analyse represents the process of analyzing the alternative.
- (*Quality*) *Criteria* represent the relevant criteria that need to be met for the final artifact. The function Evaluate assesses the alternative with respect to (Quality) Criteria and Constraints.
- *Constraints* represent the possible constraints either from the context or as described in Problem Statement.
- *Heuristics*/*Optimization Techniques* represents the information for finding the necessary actions to meet the criteria and constraints. The function Select/Optimize selects the right alternative or optimizes a given alternative to meet the criteria and the constraints.

2.3. Context

Both the control and the problem-solving activities take place in a particular context, which is represented by the outer rounded rectangle in Fig. 1 Context can be expressed as the environment in which engineering takes place including a broad set of external *constraints* that influence the final solution and the approach to the solution. Constraints are the rules, requirements, relations, conventions, and principles that define the context of engineering Fig. 1, that is, anything, which limits the final solution. Since constraints rule out alternative design solutions directing engineers into taking action on what is doable and feasible.

The context also defines the need, which is illustrated in Fig. 1 by a directed arrow from the context to the need concept. Apparently, the context may be very wide and include different aspects like the engineer's experience and profession, culture, history, and environment Fig. 1.

2.4. Scaleability of PSEM

Engineering problems are complex and include many and different kinds of concerns. A problem may include various needs, require different kinds of solution domain knowledge, various goals, different abstractions, etc. For large and complex problems, it is just practically impossible to cope with all these concerns at a time and by the same engineers. This means that the problem cannot be solved in one step. A traditional technique for coping with complexity is decomposition of the problem into sub-problems. The engineering disciplines apply this technique and decompose the overall

engineering process into so-called *phases*. A phase represents a set of related activities to solve a particular problem. As such each phase can itself be modeled using the PSEM. The decomposition into different phases may be modeled through the function *Initiate*. Each phase results in an intermediate artifact description that is used to produce subsequent artifact descriptions. This process will be continued until the final artifact, that is, the artifact directly used by the end-user, is produced. These observations are shown in Fig. 2. In the figure instances of the PSEM are represented through rounded rectangles with underlined names. Since each phase is a problem-solving process, it adopts the concepts and functions as described by the engineering model in Fig. 2. The concepts and functions, however, will have different and particular content.

Obviously, the decomposition of the overall process into several phases with a particular concern facilitates the problem solving effort. However, executing the whole process sequentially, that is, phase after phase, is generally complicated and therefore *iteration* between different phases is proposed in engineering disciplines. In Fig. 2, iteration is represented by feedback arrows between the different phases.

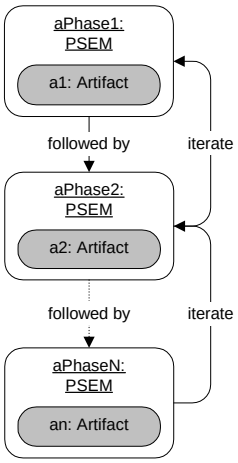

Fig. 2 The decomposition of the overall problem solving process into phases.

2.5. Notion of Synthesis

The PSEM model is a general model for representing problem-solving processes in engineering. The model can be applied to implement various problem solving processes. It appears that the model has the following three processes:

- **Technical Problem Analysis**, includes the definition of the problems and the sub-problems that need to be solved.
- **Solution Domain Analysis**, includes the search for the solution domain and its modeling in order to solve the problems.
- **Alternative Space Analysis**, includes the alternative space generation and alternatives evaluation of the composed solutions.

In traditional engineering, the interplay between these three processes is often termed *synthesis* (Maher, 1989). Synthesis in engineering often means a process in which a problem specification is transformed to a solution by first decomposing the problem into loosely coupled sub-problems that are

independently solved and integrated into an overall solution. Synthesis consists generally of multiple steps or cycles. A synthesis cycle corresponds to a transition (transformation) from one *synthesis state* to another and can be formally defined as a *tuple,* consisting of a problem specification state and a design state (Maimon, *et al.,* 1996). The problem specification state defines the set of problems that still needs to be solved. The design state represents the tentative design solution that has been lastly synthesized. Initially, the design state is empty and the problem specification state includes the initial requirements. After each synthesis state transformation, a sub-problem is solved. In addition a new sub-problem may be added to the problem specification state. Each transformation process involves an evaluation step whereby it is evaluated whether the design solutions so far (design state) are consistent with the initial requirements and if there are any additional requirements identified during the synthesis. In particular, the synthesis process includes an explicit phase for searching design alternatives in the corresponding solution domain and selecting these alternatives based on explicit quality criteria.

3. Synthesis Process in Mature Engineering

In the following we will describe synthesis and its three processes of technical problem analysis, solution domain analysis and alternative space analysis in mature engineering disciplines.

3.1. Technical Problem Analysis

Although initial client problems are ill-defined (Rittel, *et al*, 1984) and may include many vague requirements, the mature engineering disciplines focus on a precise formulation of the objectives and a quantification of the quality criteria and the constraints, resulting in a more well-defined problem statement. The objectives are often ordered into higher and lower-level objectives. The criteria and constraints are often expressed in mathematical formulas and equations. The quality concept is thus explicit in the problem description and refers to the variables and units defined by the International Systems of units (SI). Some of these variables are used by more than one engineering discipline; other variables are more specific to a particular engineering discipline. What matters, though, is that problem descriptions include quantified criteria and constraints and that quality is made explicit in this way. From the given specification, the engineers can easily calculate the feasibility of the end-product for which different alternatives are defined and, for example, their economical cost may be calculated.

3.2. Solution Domain Analysis

As a matter of fact mature engineering disciplines are based on a rich scientific knowledge that has developed over several centuries. The corresponding knowledge has been compiled in several handbooks and manuals that describe numerous formulas that can be applied to solve engineering problems. The handbooks we studied contain more than 2000 pages each and provide a comprehensive, in-depth coverage of the various aspects of the corresponding engineering field from contributions of dozens of top experts in the field. Using the handbook, the engineer is guided by hundreds of valuable tables, charts, illustrations, formulas, equations, definitions, and appendices, containing extensive conversion tables and usually sections covering mathematics. The handbooks not only describe properties of primitive elements such as material and energy but in addition describe well-known systems at a more gross level such as machines and mechanisms in mechanical engineering, control systems in electrical engineering, bridge design in civil engineering, and process design in chemical engineering. Together with engineering manuals they cover a wide range of scientific, mathematical and technological knowledge. Obviously, scientific knowledge plays an important role in the degree of maturity of the corresponding engineering.

3.3. Alternative Space Analysis

In mature engineering, alternatives are usually extracted from the related literature or composed from existing components for which extensive analyses are given in the related literature. In case no accurate formal expressions or off-the-shelf solutions can be found, heuristic rules are used (Coyne, 1990, Cross, 1989, Maher, 1989). Alternatives are evaluated by using *mathematical modeling*. A *mathematical model* is an abstract description of the artifact using mathematical expressions of relevant natural laws. One mathematical model may represent many alternatives. In addition different mathematical models may be needed to represent various aspects of the same alternative. It appears that mathematical models are widely used in mature engineering disciplines. The handbooks we studied each contain several chapters on mathematical theories predominantly on optimization. The selected alternatives are analyzed and evaluated using mathematical techniques such as differential calculus, linear programming, non-linear programming and dynamic programming.

4. Contemporary Perspective of Problem Solving in Software Engineering

We will now analyze software engineering using the PSEM model and the corresponding synthesis process.

4.1. Technical problem analysis

In software engineering, the phase for conceiving the needs is referred to as *requirements analysis* which usually is started through an initial requirement specification of the client. In mature engineering we have seen that the quality concept is already explicit in the problem description through the quantified objectives of the client. In software engineering this is quite different. Very often a distinction is made between *functional requirements* and *non-functional requirements*. As described in (Jacobson, 2000) functional requirements express the actions that a system must perform without considering the constraints. Non-functional requirements impose constraints on functional requirements and specify the required system properties, such as environmental, implementation and performance constraints and the expected quality criteria like maintainability and reliability. In contrast to mature engineering disciplines, however, constraints and the requirements are usually not expressed in quantified terms. Rather the quality concern is mostly implicit in the problem statement and includes terms such as ‘the system must be adaptable’ or 'system must perform well' without having any means to specify the required degree of adaptability and/or the performance.

It should be noted that the importance of requirements engineering has seriously changed over the last decade. There is an IEEE conference on RE, which has been running successfully since 1993, a Requirements Engineering journal, several serious textbooks on requirements engineering, and a lot of research, which deals with both formalizing and measuring functional and non-functional requirements. Although the community seems on the right track it is generally acknowledged that the aimed state of mature engineering is unfortunately not reached yet.

4.2. Solution Domain Analysis

It turns out that a common implicit assumption of the current approaches in software development is that the concept *Problem Description*, or requirement specifications, forms the basic input for the development of software solutions and scientific knowledge has only a minor role. The general idea is that requirements have to be specified using some representation and this should be refined along the software development process until the final software is delivered. Software development is thus seen as an evolutionary transformation process of the initial requirements until final software implementation. This approach resembles the early pre-mature phases of traditional engineering disciplines when scientific knowledge was not mature yet or not applied in practice. The lack of the explicit notion of solution domain could be related to the relatively young history of the field of

software engineering. In fact software engineering is only about 40 years old and obviously has not yet experienced the full maturation of the scientific and technological knowledge as in the traditional engineering disciplines. If we relate the quantity of knowledge to the supporting knowledge of mature engineering disciplines, the available knowledge in software engineering which is currently organized and actually used is quite meager. In that sense, the available handbooks of software engineering are not comparable to the standard handbooks of mature engineering disciplines. Moreover, on many fundamental concepts in software engineering, consensus among experts has still not been reached yet and research is ongoing. Similar to the developments in requirements engineering (and problem analysis), however, we can also observe progress in the solution domain analysis. Recently, *domain analysis* is introduced as the process of identifying, capturing and organizing domain knowledge about the problem domain with the purpose of making it reusable when creating new systems (Arrango, 1994). An increasing number of software design methods are now based on domain-driven design. Nevertheless, as in the case of requirements engineering it is still too early to state that software engineering applies solution domain analysis as in the synthesis process of mature engineering disciplines.

4.3. Alternative Space Analysis

The selection and evaluation of design alternatives in mature engineering disciplines is based on quantitative analysis through optimization theory of mathematics. Apparently, this is not common practice in software engineering. In general software methods do not easily apply mathematical optimization techniques to generate and evaluate alternative solutions. Moreover, the notion of quality in software engineering has still an informal basis. There is however a broad agreement that quality should be taken into account when deriving solutions. In software engineering quality factors are often divided into *external* and *internal* qualities that correspond to the distinction between internal and external attributes of entities. The external qualities are visible to the end-users of the system. The internal qualities concern the developers of the software system. Internal qualities deal largely with the structure of the system and help to achieve the external qualities. Quality factors may be attributed to the process, the product and the available resources. Some important software quality factors such as correctness, robustness, reliability, adaptability, reusability and extensibility are better defined (Ghezzi, *et al.,* 1991, Humphrey, 1989). However, in general, these quality factors are not quantified and as such cannot be explicitly used to generate, evaluate and optimize design alternatives.

5. Synthesis-Based Architecture Design

Obviously, despite its benefit for engineering, the concept of synthesis has not yet been fully implemented in software engineering processes (Tekinerdogan, *et al,* 2001). In this section we describe the *synthesis-based software architecture design process (Synbad)* that is an implementation of the PSEM model and as such an application of the concept of synthesis in software engineering. Synbad consists of five basic processes, which are respectively *Requirements Analysis*, *Technical Problem Analysis*, *Solution Domain Analysis*, *Alternative Design Space Analysis* and *Architecture Specification*. The process is illustrated in Fig. 3. The figure uses the graphical notation from Hierarchical Task Analysis (HTA) (Diaper, 1989) in which activities are represented in hierarchical order. Each numbered box represents an activity that can be refined using a plan. Each plan represents a flow diagram describing the causal sequencing of the activities. The double-headed arrows represent interaction between two activities. The diamond with a question mark represents the validation of a step. In the following Synbad is explained in more detail.

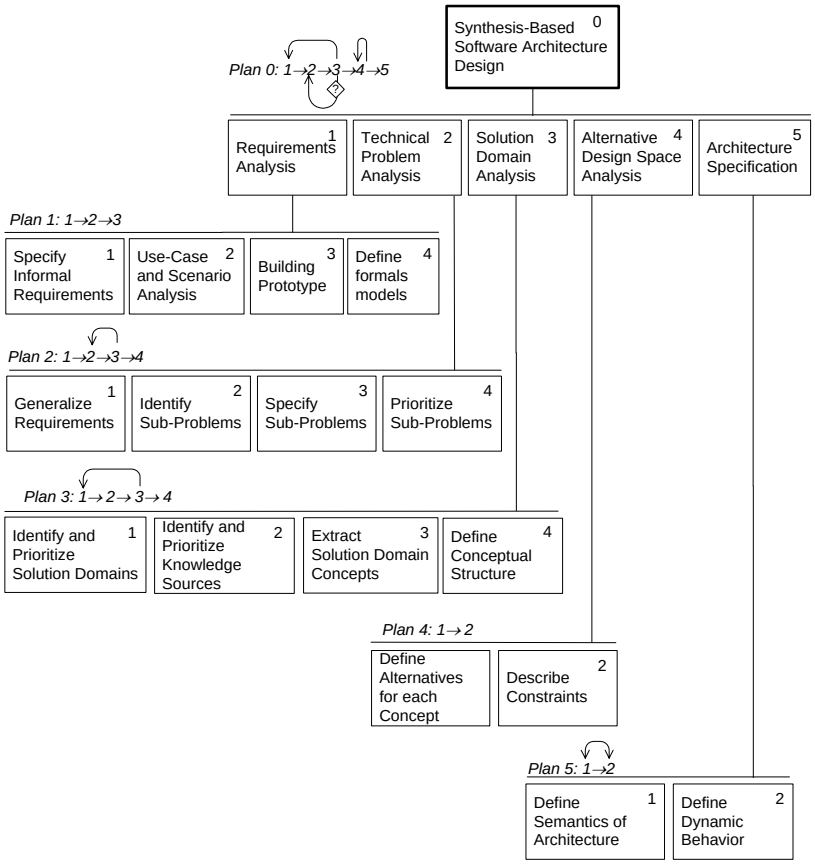

Fig. 3. Synthesis-based software architecture design.

5.1. Requirements Analysis

The first basic process of the synthesis process is the requirements analysis in which the basic goal is to understand the requirements from a client's perspective. For this purpose, Synbad adopts the well-known requirement analysis techniques such as informal requirement specifications, use-cases and scenarios, constructing prototypes and defining finite state machines. We will not elaborate on these techniques here and suffice to refer to existing software engineering textbooks (Sommerville, 2003).

5.2. Technical Problem Analysis

During the technical problem analysis process the client requirements are mapped to technical problems. Hereby, the requirements are abstracted and represented in a general form. From these abstracted requirements the technical problems are identified and specified. If necessary, the problem is decomposed into sub-problems, whereby these are prioritized to their relevance.

5.3. Solution Domain Analysis

The Solution Domain Analysis process aims to provide a solution domain model that will be utilized to extract the fundamental concepts of a problem. Hereby, for each problem the corresponding solution domains are identified. Subsequently, for each solution domain, knowledge sources are identified and prioritized with respect to their relevance and objectivity. From these knowledge

sources, the solution domain concerns are identified and structured by looking at commonalities and variations of the extracted abstractions. The relation between these concepts is shown in Fig. 4.

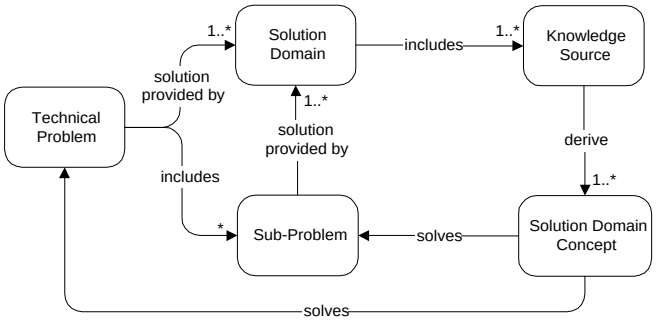

Fig. 4 The relation between technical problem, solution domain, knowledge source and solution domain concept.

5.4. Alternative Design Space Analysis

We define the *alternative space* as the set of possible design solutions that can be derived from a given conceptual software architecture. The *Alternative Design Space Analysis* aims to depict this space and consists of the sub-processes *Defining the Alternatives for each Concept* and *Describing the Constraints*. The architecture design alternatives are largely dealt with by deriving architectural abstractions from well-established concepts in the solution domain. Each architectural concept is an abstraction from a set of instantiations and during the analysis and design phases the architecture is realized by selecting particular instances of the architectural concepts. An instance of a concept is considered as an alternative of that concept. The total set of alternatives per concept may be too large and/or not relevant for solving the identified problems. Therefore, to define the boundaries of the architecture it is necessary to identify the relevant alternatives and omit the irrelevant ones. A reduction in the space is defined by the solution domain itself that defines the *constraints* and as such the possible combination of alternatives. The possible alternative space can be further reduced by considering only the combinations of the instantiations that are relevant from the client's perspective and the problem perspective. Constraints may be defined for the sub-concepts within a concept as well as among higher-level concepts. We first describe the constraints among the sub-concepts within a concept and later among the concepts.

5.5. Specification of Design

During the specification process the semantics of the concerns are extracted from the solution domain and the interactions and the additional dynamic behavior of the concerns are described. We consider each concept separately to derive its semantics from the solution domains to provide a more formal specification. The specifications of the architectural components are used to model the dynamic behavior of the architecture.

6. Conclusions

We have provided a problem solving model for engineering and discussed the concept of synthesis that is an implementation of this model. Our study has shown that synthesis is basically implemented in design processes of mature engineering but is still lacking in software engineering as an explicit concept. Obviously, synthesis plays a fundamental role in mature problem solving and for improving the maturity of software engineering it is necessary to integrate this concept synthesis within the

current software engineering practices. In this paper we have focused on software architecture design which is one of the key processes for defining quality software. The application of synthesis to architecture design process resulted in a novel approach that we termed *synthesis-based software architecture design approach (Synbad)*. This approach includes the explicit synthesis processes of *technical problem analysis*, *solution domain analysis* and *alternative space analysis* that are important for finding the stable architectural abstractions. During the technical problems analysis the initial requirement specifications are mapped to relevant technical problems. In the solution domain analysis, for each technical problem the necessary solution domains are identified and solution domain concepts are extracted by identifying commonalties and variabilities of the extracted knowledge from the solution domain. The solution domain concepts are mapped to the components of the conceptual architecture. In the alternative space analysis process, for each solution domain concept the set of possible alternative instances are depicted and the constraints among these are defined.

Unfortunately, due to space limitations we could not describe the Synbad process in detail. Detailed knowledge on Synbad and its application can be found in our earlier publications (Aksit, 2001, Aksit, *et al.*, 1999, Tekinerdogan, 2000). We have successfully adopted the approach in several projects (Ahsmann, 1995, Aksit, *et al.*, 1999) during the last decade, such as the design of an atomic transaction system architecture for a distributed car dealer information system, design of an insurance system, and the analysis and design of a digital TV architecture (Trader, 2005). Our future work will elaborate on the specialization of the specific sub-processes of the synthesis process.

7. Acknowledgements

This research has been financed by the Dutch National Organization for Science (NWO).

8. References

Ahsmann, F. and Bergmans, L., 1995, "I-NEDIS: New European Dealer System, Project plan I-NEDIS," Siemens-Nixdorf & University of Twente.

Aksit, M. (Ed.), 2001, "Software Architectures and Component Technology," Kluwer.

Akşit, M., Tekinerdoğan, B., Marcelloni, F. and Bergmans, L. 1999, "Deriving Object-Oriented Frameworks from Domain Knowledge," in: M. Fayad, D. Schmidt, R. Johnson (Eds.), John Wiley & Sons, Building Application Frameworks: Object-Oriented Foundations of Framework Design, pp. 169-198"

Arrango, G, 1994, "Domain Analysis Methods. In Software Reusability," in: R. Schäfer, R. Prieto-Díaz and M. Matsumoto (Eds.), Ellis Horwood, New York, New York, pp. 17-49.

Bass, L., Clements, P., and Kazman, R., 1998, "Software Architecture in Practice," Addison-Wesley.

Arrango, R., 1994, "Domain Analysis Methods". in: R. Schafer, R. Prieto-Diaz and M. Matsumoto (Eds.), Software Engineering Reusability, Ellis Horwood, New York.

Biegler, L.T., Grossmann, I.E., and Westerberg, A.W., 1997, "Systematic Methods of Chemical Process Design," Prentice Hall.

Braha, D., and Maimon, O., 1997. "The Design Process: Properties, Paradigms, and Structure". IEEE Transactions on Systems, Man, and Cybernetics, Vol. 27, No. 2.

Chen, W.F., 1998, "The Civil Engineering Handbook," CRC Press.

Coyne, R.D., Rosenman, M.A., Radford, A.D., Balachandran, M., and Gero, J.S., 1990, "Knowledge-based Design Systems," Addison-Wesley.

Cross, N., 1989, "Engineering Design Methods," Wiley & Sons.

Diaper, D., (ed.) 1989, "Knowledge Elicitation," Ellis Horwood, Chichester.

Dorf, R.C., 1997, "The Electrical Engineering Handbook," Springer Verlag, New York.

Dunsheath, P., 1962, "A History of Electrical Engineering," Faber & Faber, London.

Ertas, A., and Jones, J.C., 1996, "The Engineering Design Process," Wiley & sons.

Evans, E., 2004, "Domain-Driven Design: Tackling Complexity in the Heart of Software," Addison-Wesley.

Fenton, N.E., and Phleeger, S.L., 1997, "Software Metrics: A Rigorous & Practical Approach". PWS Publishing Company.

Foerster, H. von., 1979, "Cybernetics of Cybernetics," in: K. Krippendorff (ed.), Communication and Control in Society, New York: Gordon and Breach.

Ghezzi, C., Jazayeri, M., and Mandrioli, D., 1991, "Fundamentals of Software Engineering". Prentice-Hall.

Humphrey, W.S., 1989, "Managing the Software Process," Addison-Wesley, Oxford.

Jacobson, I, Booch, G., and Rumbaugh, J., 2000, "The Unified Software Development Process," Addison-Wesley.

Marks, L.S., 1987, "Mark's Standard Handbook for Mechanical Engineers," McGraw-Hill.

Maher, M.L, 1989, "Synthesis and Evaluation of Preliminary Designs," in: J.S. Gero (ed), Artificial Intelligence in Design, New York: Springer-Verlag.

Maimon, O and Braha, D., 1996, "On the Complexity of the Design Synthesis Problem," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 26, No. 1.

Newell, N., and Simon, H.A., 1976, "Human Problem Solving," Prentice-Hall, Englewood Clifss, NJ.

Perry, R., 1984, "Perry's Chemical Engineer's Handbook," McGraw-Hill, New York.

Rittel, H.W., and Webber, M.M., 1984, "Planning problems are wicked problems," Policy Sciences, 4, 155-169.

Smith, G.F. and Browne, G.J., 1993, "Conceptual Foundations of Design Problem Solving". IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 5.

Smith, A.A., Hinton, E., and Lewis, R.W., 1983, Civil Engineering Systems Analysis and Design, Wiley & Sons.

Sommerville, I., 2000, "Software Engineering". Addison-Wesley.

Tekinerdogan, B., 2000, "Synthesis-Based Software Architecture Design," PhD Thesis, Dept. of Computer Science, University of Twente.

Tekinerdoğan, B., and Akşit, M., 2001, "Classifying and Evaluating Architecture Design Methods, in Software Architectures and Component Technology: The State of the Art in Research and Practice, M. Akşit (Ed.), Boston:Kluwer Academic Publishers, pp. 3 - 27.

Trader, 2005, "Television Related Architecture Design for Enhancing Reliability (Trader) project," http://www.esi.nl/site/projects/trader

Upton, N., 1975, "An illustrated history of civil engineering," Heinemann, London.