Integrating Systems and Software Engineering Concepts in AP-233

Asmus Pandikow, Erik Herzog, Anders Törne
Real-Time Systems Laboratory
Linköpings Universitet
581 83 Linköping, Sweden
E-mail: {asmpa, erica, andto}@ida.liu.se

ABSTRACT

This paper presents ongoing work to extend the ISO 10303-233 (AP-233) systems engineering information model in order to integrate software-engineering concepts. The work is motivated by the increasing importance of software in contemporary systems and the gap between systems and software engineering's specification methods. Encoding entities supporting modern software engineering attains two objectives. Firstly, the model could be used for exchanging software specifications in addition to system specifications. Secondly, relationships between entities representing software and system concepts can be encoded which allows for traceability between requirements expressed in a systems specification and the software design specification.

INTRODUCTION

Requirements on and specifications of systems and software have traditionally been expressed in text, often augmented with illustrations in different formats. As long as this was the predominant method there were no explicit information exchange problems between system and software engineers. Expressing requirements and specification in natural language has distinct drawbacks, as most, if not all natural languages are ambiguous by nature. It is very difficult to express requirements in a format that cannot be misunderstood or misinterpreted. Furthermore, textual specifications cannot be simulated or verified formally. Consequently, it is not possible to make any statements about the correctness of such a specification.

Both the systems and software engineering communities have acknowledged the drawbacks of using natural language in specifications, but different methods have evolved in the respective communities. Systems engineering has adopted a function-centered approach as described in, e.g., (Hatley and Pirbhai, 1987) and (Oliver et al., 1997), while software engineering preferred the object-oriented approach. The result is a situation where moving information between the domains has become difficult. There exists a gap between the models at system and software levels, as described in (Cocks, 1999). As a consequence, decisions in the software development phases cannot be directly traced from the systems engineering point of view and vice versa. This might lead to mismatching specifications and hence result in software that might not be compliant with the original system requirements.

AP-233 is the reference within ISO 10303 (STEP) to a standardization project for systems engineering data exchange. The main aspect of this project is to create an information model that encompasses the main systems engineering information elements.

In this paper we describe ongoing work to interlink the systems engineering process and the software engineering process and to integrate object-oriented concepts into the existing elements of the current working draft of the AP-233 information model. An up-to-date overview of the AP-233 project is available in (Johnson et al., 2000) and the architecture of the AP-233 information model is presented (Herzog and Törne, 2000).

The rest of this paper is outlined as follows. The next section provides further motivation for the need to integrate object-oriented concepts with the systems engineering process and sketches the scope of the integration work. The subsequent sections describe the process of object-oriented software engineering followed by a brief description of the elements of the AP-233 information model. More information on AP-233 is available in (Herzog and Törne, 1999a), (Herzog and Törne, 1999b) and (Johnson, 1998). The object-oriented concepts are then compared to the matching parts of the AP-233 model and the degree of equivalence is identified. This is followed by a discussion of the integration work performed. A conclusion and an outline of future work finish the paper.

THE NEED TO INTEGRATE SE AND OO

In many domains, systems engineers are concerned with systems having substantial parts of their functionality implemented as software.

The increasing capacity of computer systems and the flexibility of software facilitates the realization of more system functions as software and hence increases a system's software/hardware ratio. The fact that different analysis and design methods for software and systems engineering are employed, makes it difficult to convey information from the system engineering process to the software engineering process. This also applies to the feedback from the software engineering process, which is delivered to the encompassing system engineering process.

Inherent software intensive domains, such as telecommunication, have started to look at their products as part of the encompassing system. This implies the same problem as mentioned above: conveying information between different processes, using different concepts and different notations.

Object-oriented software engineering has emerged as the dominant software engineering method. Combining the methods of object-oriented software engineering and system engineering would improve the information exchange between the engineering processes and hence improve the overall process.

We are involved in the development of AP-233 and believe that this provides an opportunity for integrating systems engineering and object-oriented concepts in order to achieve the objectives stated above. Including concepts of software engineering in the AP-233 information model will then provide the means for maintaining traceability of design decisions and their consequences throughout the complete engineering process, including software engineering. It may also improve the overall process by increasing the common understanding of the system to be developed.

One has to be aware of the fact that object-orientation originates from software engineering and is not originally intended to directly represent objects of the physical world as concepts like inheritance do not exist there. Nevertheless, the object-oriented approach might be used for representing physical structure and logical relationships between physical objects. In this paper we focus on the integration of object-orientation to support the flow of information from systems engineering to software engineering.

OBJECT-ORIENTED CONCEPTS

As object-oriented concepts originate from software engineering and as software is always only a part of the system to be developed, the early object-oriented methods such as (Booch, 1991) and (Rumbaugh et al., 1991) do not cover all aspects of the software's entire lifecycle. For example, requirements management and post-productive activities such as maintenance and disposal were usually expected to be covered by the methodology of the encompassing project.

The Unified Modeling Language UML (Booch et al., 1996) emerged from the major object-oriented methods by (Booch, 1994), (Rumbaugh et al., 1991) and (Jacobson et al., 1992). It is the first object-oriented notation that allows engineers to informally capture requirements (beginning with the help of Jacobson's "use cases") and also addresses aspects of physical architecture (in its implementation diagrams).

As the UML incorporates all important object-oriented concepts and as the Object Management Group (OMG Web Site, 2000) accepted it in 1997 as a standard, our further discussions and integration efforts are based on the current version 1.3 of the UML (OMG UML Specification, 1999).

The ideal object-oriented software engineering process can be described through three sequential phases: system analysis, design and implementation. Object-orientation provides basic concepts, which can be homogeneously used throughout all phases of the process. The basic concepts are class, object, attribute, operation, message, inheritance and polymorphism, see (Balzert, 1996) or (Muller, 1997) for an introduction. The main phases of the object-oriented software engineering process are further outlined below:

Analysis phase. Identifying the elemental use cases and the global logical packages of the system is the first step within the analysis phase of object-oriented software engineering. This is supported by the UML concepts for use cases and a system's static structure.

Completing the static model of a system (identifying classes, their associations and subsystems) and subsequently creating the dynamic model (detailed use cases and behavior descriptions) are concluding the analysis work. The UML provides additional concepts for this: use cases can be elaborated as message sequences and collaborations between objects. Moreover, state machines, activity diagrams and class operations can be used to describe behavior.

Design phase. In the design phase the system model from the analysis phase is further elaborated to gain a closer mapping to the final software. The analysis concepts are generally kept and only extended by details that allow for tailoring the model for implementation purposes. For this, the UML provides for example class templates, detailed interface and association descriptions as well as overloading and several kinds of polyphormism for class operations.

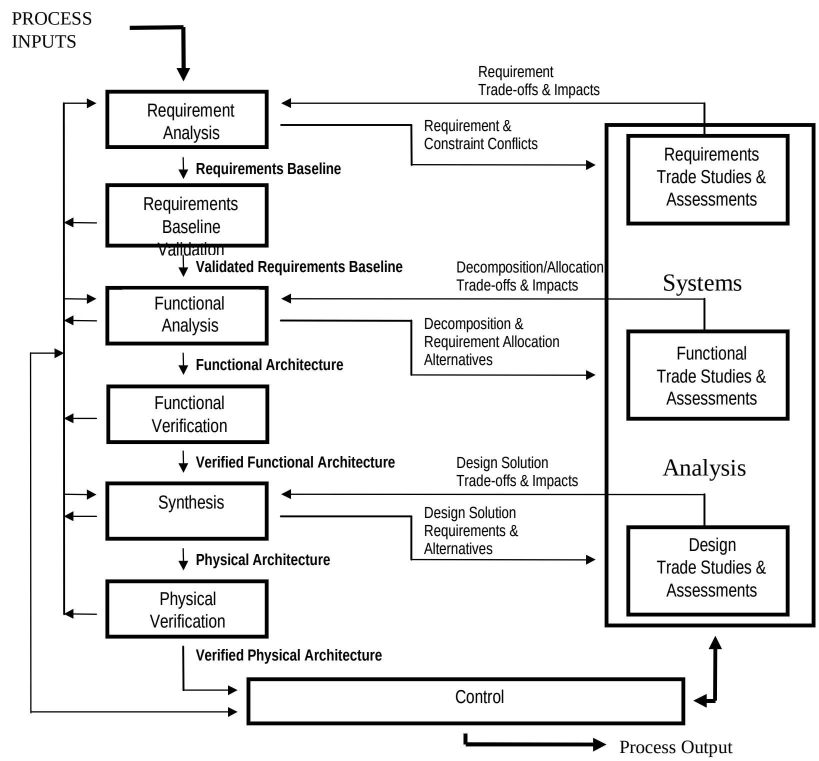

Figure 1: The IEEE 1220 Systems Engineering process

Implementation phase. Implementing the model, which emerged from the design phase in an object-oriented programming language, is the final phase of the object-oriented software engineering process. The UML supports implementation by a view of the components of the software, their interfaces and relations. It also provides a notation for the run-time deployment of software components among the system's hardware components.

UML in software engineering. The UML supports the software engineering phases with several notations, each giving a different view of the modeled system. One notation alone is not sufficient to describe the complete system and the notations usually do not require formal rigor. Instead, they permit the inclusion of semi-formal and informal elements, which allows modeling at different levels of granularity. This might lead to difficulties in interpreting and exchanging the model as it may contain ambiguities.

AP-233 CONCEPTS

The current AP-233 working draft information model, supports the core elements produced in a system engineering process based on structured analysis (Hatley and Pirbhai, 1987). The model itself is process independent, but there is a close relationship to the design elements mentioned in the process description in IEEE 1220 (Ptack, 1998). The engineering process is illustrated in Figure 1.

In the information model there is extensive support for the representation of requirements, functional and physical architectures. There are entities defined for maintaining traceability of how a system evolves.

The assumption in AP-233 is that the main elements of a complete specification for each life cycle of a system may be one of more of the following:

1. The set of requirements defining what a system shall perform, how well it shall be performed etc.
2. A functional architecture defining the functions, formally or informally, a system shall perform, including how it communicates with elements outside the system. The functional architecture may also contain elements for defining the dynamic behavior of a system, e.g., finite state machines, causal chains or formal statements.
3. A physical architecture defining the physical and/or logical architecture of the system under specification.
4. Traceability relationships defining traceability from requirements to elements in the functional or physical architecture.

5. Allocation relationship defining how elements in the functional architecture are mapped onto physical elements.

The main elements of the model are illustrated below in Figure 2 and different aspects of the model are further outlined in, e.g., (Johnson, 1998), (Herzog and Törne, 1999a) and (Herzog and Törne, 1999b).

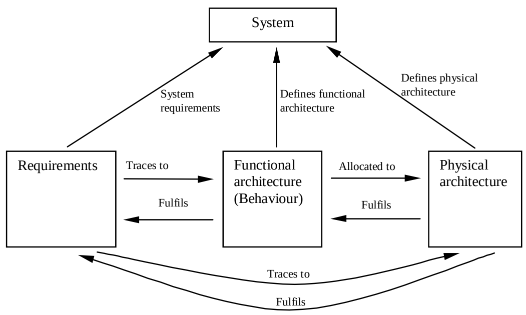

Figure 2: Information model, conceptual view

In addition to the elements presented above the information model supports representation of presentation information, version and configuration management and a wide variety of administrative information as described in (Herzog and Törne, 2000).

COMPARISON

Approach. To prepare the integration of object-oriented concepts into the AP-233 information model, we have started with comparing object-oriented concepts and their corresponding equivalents in the AP-233 information model using the following approach:

- Create a meta-model of the UML
- Identify semantically equivalent parts of the UML and the AP-233 model
- Compare the matching parts of both models and determine their degree of equivalence

The purpose of the comparisons is to support the identification of opportunities for integrating the models.

UML meta-model. The UML meta-model has been partly based on the existing meta-model in (OMG UML Specification, 1999). It has been captured in EXPRESS, just as the AP-233 information model was, in order to facilitate the integration efforts by an equivalent representation with respect to the modeling language.

UML constructs have been modeled in accordance to the modeling style selected for AP-233 in order to gain maximum congruence.

Equivalents. Table 1 gives an overview of the UML notations and how they roughly map to equivalent parts of the AP-233 information model.

UML Notation	AP-233 equivalent
Static structures	Packages, partially types, data items
Use cases	Partially in system views
Sequences and Collaborations	Partially causal chains
Activities	Causal chains
Statecharts	Finite state machines
Components and deployment	Physical architecture

Table 1: UML notation mapping

The mapping of single concepts of the UML notations to semantically equivalent concepts of the AP-233 information model is shown in Table 2.

AP-233	OO (UML)
Project management	(no equivalent)
Configuration management	(no equivalent)
Version management	(no equivalent)
Requirements	(no direct equivalent) Partially in use cases
System / sub-system	(no direct equivalent) Partially packages, classes and components
Partial system view	Partially in use cases
Functional analysis / breakdown	in classes
Causal chains	(no direct equivalent) Partially as message sequences object collaboration
Physical system structure	Partially in components and deployment
Data types	Data types, except class
State machines	Statecharts
Petri nets, causal chains	Activities and their transitions
(no equivalent)	Classes, objects and associations between them
(no equivalent)	Interactions between classes and objects

Table 2: Concept mapping

Global interconnections within the AP-233 model, such as version management and traceability had to be excluded from the mapping, as these are not supported by the UML.

Comparison. For each equivalence in comparison we evaluated whether they represent the same idea or if they represent genuinely different concepts.

Comparing the corresponding parts of the two models resulted in the following equivalence classes:

- Full equivalence
- Partial equivalence
- No equivalence

Full equivalence means that the respective UML concept can be represented by the corresponding AP-233 concept. Small extensions to the AP-233 model may have to be inserted in order to fully capture the UML characteristics. These extensions may in turn also be beneficial for systems engineering methodologies and for the systems engineering process.

Partial equivalence indicates that only some of the major characteristics of an UML concept can be found in the AP-233 equivalent. The AP-233 model would have to be extended substantially to be able to capture the UML counterpart.

No equivalence denotes that an UML concept cannot be identified in the AP-233 information model or that the corresponding concepts are semantically completely different and hence new model entities have to be created.

Table 3 summarizes the UML concepts grouped into the presented equivalence classes.

UML Concept	Equivalence class
Statecharts	Full equivalence
Activities & transitions	Full equivalence
Classes	Partial equivalence
Objects	Partial equivalence
Component view	Partial equivalence
Deployment view	Partial equivalence
Use cases	Partial equivalence
Class associations	No equivalence
Message sequences	No equivalence
Object collaborations	No equivalence

Table 3: Equivalence classes

For the cases of full or no equivalence the modeling solutions are obvious: AP-233 entities, which support fully equivalent UML concepts have possibly only to be adapted to the flavor of UML. Entities for capturing the cases of no equivalence have to be created and linked to existing context entities and global mechanisms of the AP-233 model.

Partial equivalencies require a more thorough investigation of possible integration alternatives in order to retain the semantics and modeling characteristics of both the UML and the existing AP-233 model. The following paragraphs describe the above mentioned partial equivalencies in more detail.

The UML class concept incorporates class operations and attributes, which partially can be compared to the existing AP-233 notions of functions, data items (variables) and their definition in data types. Also objects as instances of classes have similarities with the AP-233 data items and their definition in a data type. Thus, the AP-233 model provides some of the elements of a class but not a composite data type that combines them, i.e. the class as a data type is missing. See also the integration example further down.

The implementation notations in the UML, i.e. the component view and the deployment view, can for the most part be represented by AP-233 entities for physical architecture. However, the possibility of having incomplete component interface definitions is not included in the AP-233 model and the run-time character of the UML deployment view does not semantically match the AP-233 model.

The notion of use cases can for the greater part be represented by AP-233 entities for a partial view of a system, i.e. a perspective only including certain aspects of the system. Nevertheless, for the UML actors there is no semantically equivalent construct available in the AP-233 model.

INTEGRATION

We are currently integrating UML concepts into the AP-233 information model. We have decided to focus on some characteristic object-oriented concepts first and integrate the remaining parts after having gained experience from the first integration efforts. Use cases, classes and objects, associations between classes and statecharts have been selected for this, as they include at least one concept of each equivalence class in our integration taxonomy.

The integration of partial equivalencies such as classes and objects has to be carefully examined with respect to interference with AP-233 constructs. For example, to integrate the class concept we have extended the existing AP-233 data types with a new data type “class”, e.g. we consider a class to be a data type, which also can be found in other object-oriented notations such as programming languages.

To avoid side effects we have decided to introduce a new abstract super type for the class data type and the existing traditional AP-233 data types, as shown in figure 3.

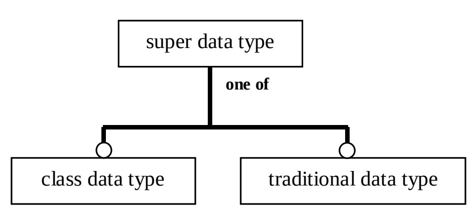

Figure 3: Integration example

The traditional data types again represent the traditional usage of data types within the AP-233 information model, whereas the object-oriented usage refers to the super data type. This means that an object-oriented data instance can either be of type class or of one of the traditional data types.

Once UML concepts are integrated in the information model, they are supported by general AP-233 concepts such as version and configuration management. Moreover, the information model can also be extended to link concepts which are already part of the information model to relevant UML concepts. For example, requirements can then also be associated with elements in the object-oriented model such as classes and objects.

This allows for the inclusion of object-oriented elements into the tracing capabilities of the AP-233 information model. Decisions taken in the software engineering phase can then be traced from a global systems engineering point of view.

CONCLUSIONS AND FUTURE WORK

It is likely that in the future software will take over even more functionality, which has so far only been realizable with hardware. Also additional tasks might be realized with the help of software, which increases the proportion of software in systems.

This development is an indication that it would be beneficial if systems engineering methods could be extended to incorporate advanced software-engineering methods. A desirable consequence is that paths for smooth information exchange between systems and software engineering could be established.

It has not been decided whether object oriented concepts as outlined in this paper shall be included into the AP-233 standard proposal. Regardless of the decision made it is important to close the gap between software and systems engineering methods – both from a method and a data exchange point of view.

We intend to continue integrating UML concepts into the AP-233 information model in order to identify examples of design data exchanges between traditional systems engineering tools and object-oriented tools supporting the UML as well as between object-oriented tools.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the hard work of participants in the supporting projects. The financial support from the European Commission for the SEDRES project (Esprit 20496) and Swedish National Board for Industrial and Technical Development (NUTEK) for the SEDEX project (Project no. IPII-98-06292) is also gratefully acknowledged.

REFERENCES

Balzert, H.: *"Methoden der objektorientierten Systemanalyse"*, Spektrum Akademischer Verlag, 1996.

Booch, G.: *"Object-Oriented Design with Applications"*, Benjamin / Cummings Publishing Company, 1991.

Booch, G.: *"Object-Oriented Analysis and Design with Applications"*, The Benjamin / Cummings Publishing Company, 1994.

Booch, G., Rumbaugh, J., Jacobson, I.: "The Unified Modeling Language for Object-Oriented Development", Rational Software Corporation http://www.rational.com/uml, 1996.

Cocks, D.: "The Suitability of Using Objects for Modeling at the Systems Level" in the *"Proceedings of the Ninth Annual International Symposium of the International Council on Systems Engineering"*, pages 1047-1054, INCOSE, 1999.

Hatley, D. and Pirbhai, I.: *“Strategies for Real-Time System Specification”*, Dorset House, 1987.

Herzog, E. and Törne, A.: “A Seed for a Step Application Protocol for Systems Engineering” in *“1999 IEEE Conference and Workshop on Engineering of Computer-Based Systems”*, pages 174–180, IEEE Computer Society Press, 1999.

Herzog, E. and Törne, A.: "Towards a Standardised Systems Engineering Information Model" in the *"Proceedings of the Ninth Annual International Symposium of the International Council on Systems Engineering"*, pages 909-916, INCOSE, 1999.

Herzog, E. and Törne, A.: "AP-233 Architecture" in the *"Proceedings of the Tenth Annual International Symposium of the International Council on Systems Engineering"*, INCOSE, 2000.

Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G.: *"Object-Oriented Software Engineering - A Use Case Driven Approach"*, Addison-Wesley, 1992.

Johnson, J.: “The Sedres Project: Producing a Data Exchange Standard Supporting Integrated Systems Engineering” in the *"Proceedings of the Eighth Annual International Symposium of the International Council on Systems Engineering"*, INCOSE, 1998.

Johnson, J., Herzog, E., Barbeau, S. and Giblin, M.: "The Maturing Systems Engineering Data Exchange Standard AP-233 & Your Role", in the *"Proceedings of the Tenth Annual International Symposium of the International Council on Systems Engineering"*, INCOSE, 2000.

Muller, P.-A.: *"Instant UML"*, Wrox Press Ltd., 1997

OMG UML Specification: "OMG Unified Modeling Language Specification", version 1.3, Rational’s UML Web Site at *http://www.rational.com/uml/*

resources/documentation/, Rational, 1999.

OMG Web Site: Internet homepage of the Object Management Group at http://www.omg.org, OMG, 2000.

Ptack, K.R. (editor): *"IEEE Standard for Application and Management of the Systems Engineering Process"*, IEEE Press, 1998.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: *"Object-Oriented Modeling and Design"*, Prentice Hall, 1991.

SEDRES Web Site: Internet homepage of the SEDRES-2 project at http://www.sedres.com, SEDRES-2 project, 2000.

BIOGRAPHIES

Asmus Pandikow is a Ph.D. student in the Real-Time Systems Laboratory in the Department of Computer and Information Science at Linköpings universitet, Sweden. His research addresses software engineering, systems engineering, information modeling and tool integration.

Erik Herzog is a Ph.D. student in the Real-Time Systems Laboratory in the Department of Computer and Information Science at Linköpings universitet, Sweden. His research include Systems engineering, specification methods, information modelling and tool integration techniques. He is also a nominated technical expert for ISO TC184/SC4/WG3.

Anders Törne, Ph.D., is an associate professor in the Real-Time Systems Laboratory in the department of Computer and Information Science, Linköpings universitet, Sweden, and manager of the Linköping office of Carlstedt R&D. His research interests are the design of real-time and embedded systems, systems engineering, and high-level specification languages.