1. ABSTRACT
This paper presents experiences made while recovering the reference architecture of a system family. The case study is a subsystem of a train control system. The train control system is an embedded, real-time system that forms a family of systems. The architecture recovery approach and architectural representation is briefly described. The main topic of this position paper is the commonality analysis process that identifies and recovers the reference architecture of the system family. The two steps of the process are described together with implications. Since the recovery of the reference architecture facilitates the forward engineering process the mismatch between as-is-architecture and as-should-be-architecture is of special interest and described at the end of the paper.

1.1 Keywords
Architecture Recovery, Product Line Architectures, Case Studies and Experience

2. INTRODUCTION
The degree of complexity of software systems is constantly growing. This is caused by additional requirements for reliability, safety, and maintainability. Large and complex systems require a higher level of abstraction to be manageable. Software architectures are a tool for managing high-level abstractions of systems. Software architectures of existing systems however, are often not available and must therefore be recovered. The area of software architecture recovery focuses on the recovery of architectural information from existing systems. In case a family of related systems exists, not only the recovery of the systems architecture is of interest, but also the recovery of the general architecture of the family members, the reference architecture.

The recovery of the reference architecture discussed in this paper is research work within the ESPRIT project ARES (Architectural Reasoning for Embedded Systems). ARES is supported by the European Commission under ESPRIT framework IV contract no. 20477.

2.1 Case Study
The case study is the BTM (Balise Transmission Module) subsystem in a train control system (TCS). The BTM subsystem is used for controlling high speed train movement and for precision stop in metros. TCS is a fail-safe and fault-tolerant system that supports different development and target environments. It is a medium-size software application with around 150 kLOC (Lines Of Code). TCS is executed on two different hardware platforms and is controlled by an in-house developed monitor.

The train control system is an embedded real-time system developed for a number of railway companies. In a continuous process further members of the program family are under development. The case study forms a family of train control systems with a rather small amount of members. Three members of the system family were analyzed in the project. We are confident, that the three members will provide enough input for the reference architecture recovery process. Future family members may be developed for different hardware platforms and/or different operating systems.

2.2 Architectural problems related to the case study
Since the case study is a safety critical system, the ability to recover and model safety concepts on the architectural level is of high importance. The current practice is to specify the system and block levels with focus on the system
Architectural Description Languages (ADLs) are formal languages used to represent the architectures of a software system. ADLs are an important tool for communicating and system description and fulfill different purposes:

- **Basis for communication** - System shareholders (project team members, manager, customers) use architectural representation for understanding and development.
- **Description of earliest design decisions** - Earliest design decisions, as the mapping of requirements to components, are the hardest to change and must be chosen carefully.
- **Project blueprints** - Architectural decisions are the base for structuring development teams, work assignments, schedule and fine-grain design.
- **Product line development blueprint** - An architecture may be re-used in other systems using the reference architecture representation.

For modeling our case study, we used the ADL Darwin [6]. Darwin is a declarative language, which supports the modeling of a system as a set of loosely coupled, context-independent software components. Darwin has a graphical and textual representation. It represents a system in terms of component types and the services they provide. Hierarchical composition and decomposition is supported to enable complex components and their interfaces to be constructed out of simpler sub-components. Components interact by
accessing services. Interaction between components is represented by a connection between a required service and a provided service.

To support the specific requirements for embedded real-time software description and reference architecture recovery, we developed a specific ADL called Architecture Structure Description Language (ASDL). ASDL not only supports the description of basic attributes of the architectural elements but also additional attributes, so-called properties. ASDL is a flexible language that allows to adjust the language to different domains. More information about ASDL can be found in [2].

ADLs representations are stand-alone products making it difficult to transform architectural description to other representations. ACME (Architecture Description Interchange Language) [5] provides an interchange format that allows the transformation of separately-developed ADLs. ACME could be used in the future for transforming ASDL descriptions to other representations.

3.3 Commonality analysis

For the recovery of the reference architecture a method was developed which identifies the reference architecture based on the architectural description of the family members. The main input to the commonality analysis is the architectural description (in Darwin and ASDL) of the family members.

In a first step the commonality analysis identifies components from different family members that appear to be similar. The input to the process is the set of the components of the different family members. Components of one family member are sorted into sub-sets of components that conform to the same component properties (e.g. components that are safety critical component and primitive components). The number of components in the different component sets is rather small. In the case study, the maximum number was 8 components. This process is done for each of the family members. Components of different family members, that belong to the same component set, can now be compared. This is the second step of the commonality analysis.

In the second step, the re-architect compares a component with the set of similar components of the other family members. In the first step, the selection criteria’s were the component properties. In this second step, the selection criteria’s are the interface properties. The re-architect compares the interfaces of two components to identify similarities between components. Interface properties are hereby the services provided and/or required, the interface role (e.g. producer) and other properties, as information about the data flow and control flow.

Three possible outcomes of the comparison are identified:

(a) Full-match: In all product members, a component could be identified with the similar component and interface properties. The similarities are so strong that such components can directly be replaced by one reference component in the reference architecture (direct transformation of full-match components to a reference component).

(b) No-match: Such components have no similar properties to any other component of the other product families. No-match components provide specific services and are candidates for direct transformation into a reference component. The re-architect may however exclude no-match components because they do not conform to the reference architecture requirements. Reference architecture requirements are requirements developed in close cooperation with forward engineers. The forward engineers try to anticipate future changes and challenges and document this information in the reference architecture requirements. Transformed no-match components are connected with a constraint indicating under what conditions the component becomes a member of a new family member.

(c) Partial-match: A number of components of the different family members with similar properties are identified. An example is a component in family member A which provides similar services as two components in family member B. A direct transformation to one reference architecture component is therefore not possible. The human experts can join partial-match components to one component or divide them to a number of components. This means that the human experts (in this case reverse architect and forward architect) transform partial-match components so that the result conforms to the reference architecture requirements. About 75 % of all components of the case study are partial-match components. This special transformation problem needs therefore special attention.

Both steps of the commonality analysis are currently done manually but could be formalized and supported by a tool in the future. The ASDL language supports the commonality analysis by providing input in textual form that can directly be used for both steps of the analysis process.

3.4 as-is versus as-should-be architecture

The architectures of the family members represent the implementation architectures of a system (as-is-architecture). The reference architecture represents a conceptual architecture of a system (as-should-be-architecture) that probably never will be instantiated. The reference architecture is a template from which software architecture of new family members can be derived.

In the second step of the commonality analysis the process is not longer a pure re-architecturing process but is influenced by a forward-architecting process. The resulting reference architecture has lost the direct relation to the as-is-architectures of the existing family members. The re-architect must be aware of this fact and carefully validate the as-is-architectures before moving on the transformation towards the reference architecture.
4. CONCLUSION

This paper describes a software architecture recovery process including a commonality analysis for identifying the reference architecture. The process was developed while working on a case study, an embedded real-time system. Architectural problems related to the case study were presented. The problems related to the transformation of as-is-architecture information to as-should-be architecture are described.

ADLs are identified as important tools for representing architectures of software systems. The ADLs Darwin and ASDL were used for describing the software architecture of the case study.

Based on the approach presented in this paper, the reference architecture of the case study was recovered. It is difficult to identify the quality of the recovered reference architecture. Valuable information will be available after a new family members is derived from the reference architecture.

5. ACKNOWLEDGMENTS

The author is grateful to Harald Gall and Berndt Bellay for their input to the recovery process and to Håkon Beckman and Sturle Mastberg for their support in analyzing the case study.

6. REFERENCES


