Initial Investigations into UML Based Architectural Reference Patterns for Set-Top Boxes

Robert Deaves, Andrew Jones, Peter Steiglitz, Martin Habets and Stuart Ryan from STMicroelectronics

Abstract:

This paper analyses a leading-edge Set-top Box (STB) design for architecture reference patterns. Specifically, the following contributions are made: (i) identifying and documenting (in UML) STB architectural reference patterns, and (ii) providing empirical (quantitative) analysis of pattern use.

Four pattern groups are identified: Dataflow, Stack, Control and State Transition along with 18 individual patterns. These are based on a combination of hardware IP and software modules.

The paper concludes that architecture reference patterns give a useful perspective on the architecture. These contribute to improved architectural understanding, potential to highlight system areas for improvement and provide useful project management information. Collectively, these can contribute to reducing the time-to-market of STB products. Future developments to these initial investigations are documented.

1. Introduction

A critical goal for Set-Top Box (STB) development is to reduce time-to-market in order to ensure commercially competitive and successful products. Historically, this was achieved by focusing on the hardware aspects of the box. However, the demand for higher levels of functionality in STBs have lead to increases in the quantity, complexity and production effort associated with the STB software.

From an architecture perspective a number of developments are being pursued that contribute to dealing with these increased software demands. These include formalizing the software architecture specification through the use of defined processes, standardized modeling and use of CASE tools.

The initial investigations presented builds on previous developments by analyzing the architecture specification with respect to design patterns. Specifically, reference architecture patterns pertinent to STBs are documented, their frequency and use analyzed, and how this information can be applied to improve the understanding and quality of our STB products. The remainder of the document is organized as follows: Section 2 provides a brief review of design patterns. Section 3 summarizes the UML representation of the STB under investigation. Section 4 documents the architectural reference patterns. The STB pattern use is analyzed in Section 5. A discussion on the analysis and how they could be applied is provided in Section 6. Concluding remarks and future

work are given in Section 7.

2. Review of Design Patterns

Design patterns describe solutions to problems that occur frequently in a particular domain. This leads to a number of design advantages, including:

- Encouraging and promoting re-use.
- A vocabulary to describe complex designs.
- Abstraction from the lowest levels of design.
- Clarity and understanding of large systems.

The most prominent use of software design patterns has been in the domain of object-orientated (OO) code. The book [1] by Gamma, Helm, Johnson and Vlissides (referred to as the 'gang of four' or GoF) is a standard reference for most OO programmers. Here a pattern is described by its name, problem being solved, solution and consequences. These patterns do not describe primitive building blocks such as linked lists or hash tables; neither do they describe entire applications or sub-systems. These design patterns 'describe communicating objects that are customized to solve general design problems in a particular context'.

Seminal work on modern software architecture was provided by Garlan [2]. This work focused on the idea of architectural design patterns to provide familiarity and clarity to software architecture. Here several architectures were identified: Pipe and Filter (or Dataflow), Object Orientation Organization, Event based Implicit Invocation, Layered Systems (or Stacks), Repositories, Interpreters, Distributed Processes, Main Program/sub-routine Organization, State-transition Systems and Process Control Systems. Further, these architectures can be used in combination to provide Heterogeneous Architectures. Another contribution of Garlan [2] is to introduce the concept of domain specific software architectures. These allow 'reference' architectures to be defined for certain areas, such as avionics, vehicle management systems etc.

Our recent work [3] on the specification of software architecture for STBs applied 'views' to partition the problem [4] and UML [5] to represent the architecture, see Fig. 1 Vertical Architectural Development. Each view provides different characteristics of the specification. One of these characteristics is the abstraction of the architecture. At the higher abstraction views the architecture is represented (not exclusively) as UML Use Case and Component Models. The lower levels of abstraction realize each individual component diagram through a number of UML representations including Deployment Models. The design pattern analysis in this paper is primarily focused on the UML Deployment Models, see Fig. 1 Horizontal Architectural Development.

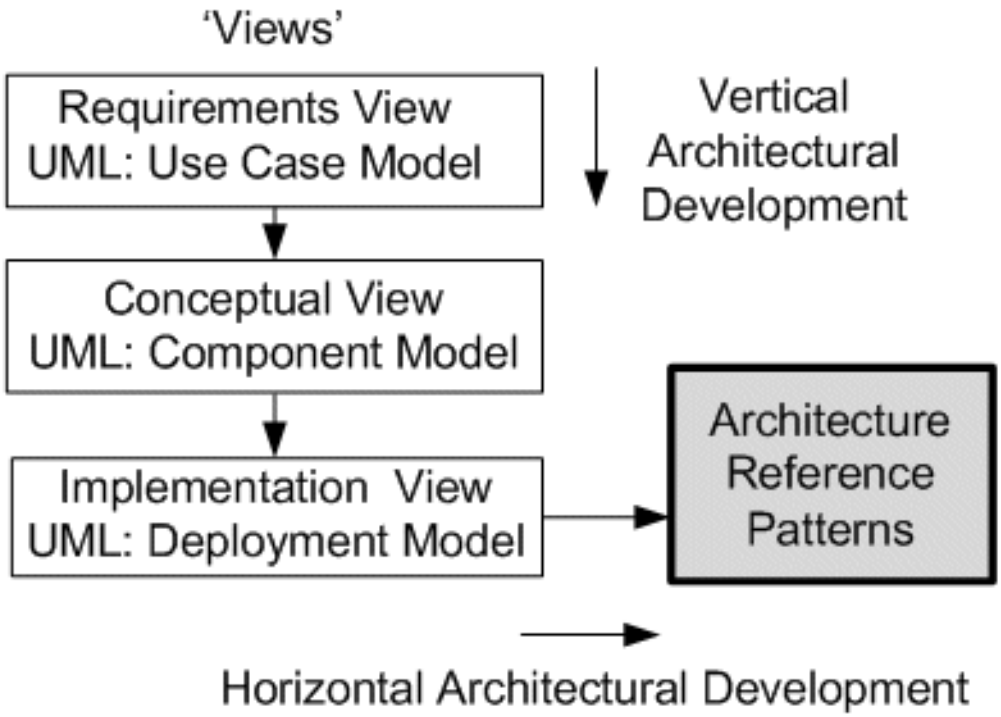

Fig.1: STB Specification and Reference Patterns

In the next section a quantitative view of the use of UML to represent the STB specification is summarized.

3. STB Software Architecture

The pattern analysis is performed on a STB targeted at the digital cable market supporting the DOCSIS standard [6].

This product combines the STB and Cable Modem functionality not only in a single box but within the same SoC, the STi7141 [7]. The functionality of the box includes live video, video-on-demand, local record/playback, IP applications (tele-shopping) and internet support for PC based home networks. In addition, the box provides support functionality such as software download (for upgrades) and power management. The software architecture for the box is represented as UML models and summarized in Fig. 2.

Moving from the abstract models (Use Cases) to the less abstract models (Deployment Nodes) increases the number of UML models used.

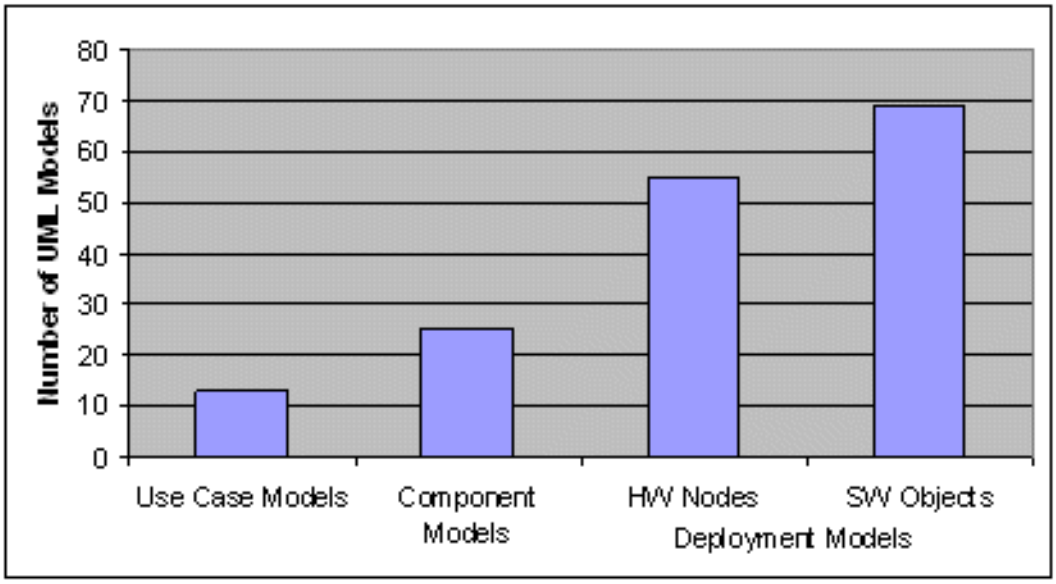

Fig. 2: STB UML Summary

In the remainder of this paper UML based reference architecture patterns pertinent to STB architecture are identified, analyzed and applied .

4. Architectural Reference Patterns

Each Deployment Model representation of the Component Model was analyzed to highlight architecture patterns for the STB.

The majority of the design could be captured through four architecture groupings. They are: Dataflow, Stack, Control and State-Transition, see Fig 3. These are discussed next. It should be noted that a full description of the groups/patterns and their characteristics is outside the scope of this paper.

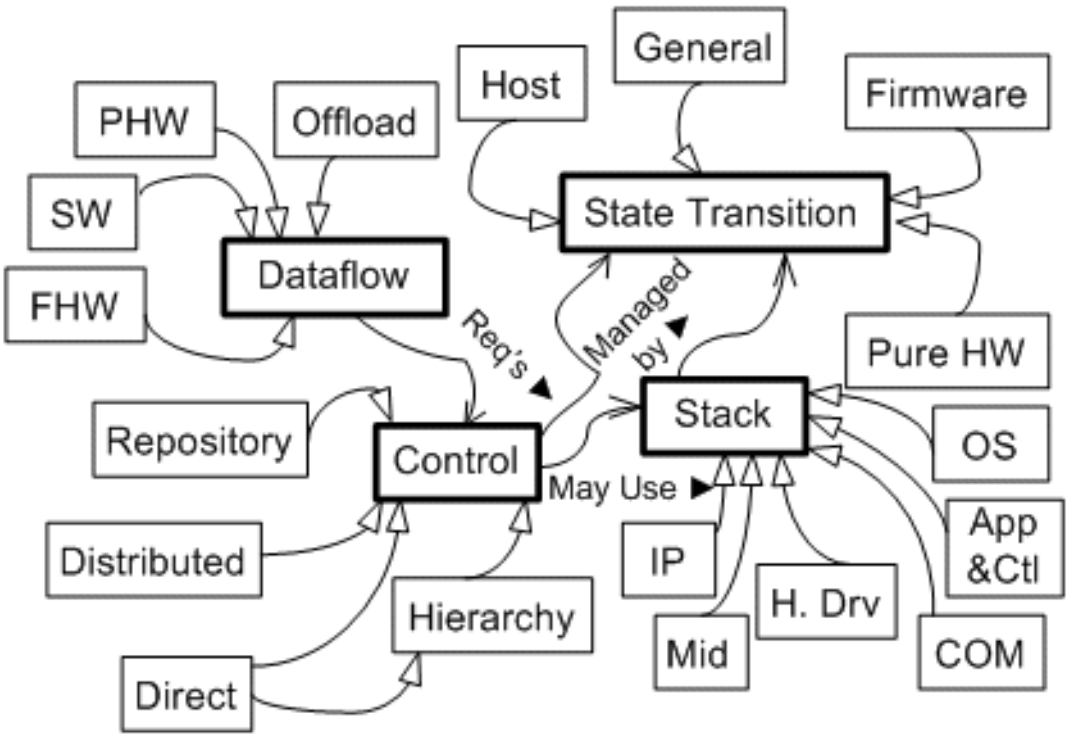

Fig.3: Architecture Pattern Relationships

4.1 Dataflow

A number of different implementations of the standard dataflow (or pipeline and filter) pattern have been identified in the STB architecture.

Group Name: Dataflow

Patterns: Pure Hardware (PHW), Firmware driven Hardware (FHW), Software (SW) and Offload.

Problem: Facilitates pipeline processing of data for reasons that include: (i) suites the algorithm structure, or (ii) alleviate CPU processing constraints by using dedicated or programmable hardware, or (iii) for achieving other system constraints such as low power.

Solution: Fig 4. represents a Hardware Dataflow. For the work presented in this paper this consists of Pure and Firmware driven Hardware. Here a number of hardware processing nodes are cascaded together to provide the pipeline. An alternative implementation has the dataflow implemented in software, see Fig. 5. The Offload Dataflow is represented in Fig. 6. Here the CPU offloads data processing to a companion CPU. Often a dataflow comprises a cascade of a number of PHW, FHW, SW and Offload pipelines.

Consequences: The Dataflow architecture needs to operate in real-time for most cases. A Control architecture, which is usually Hierarchical is required as part of the Dataflow.

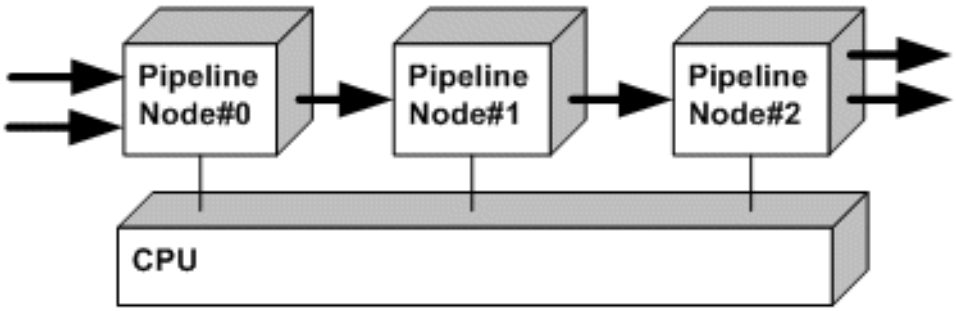

Fig. 4: Hardware (PHW and FHW) Dataflow Deployment Nodes

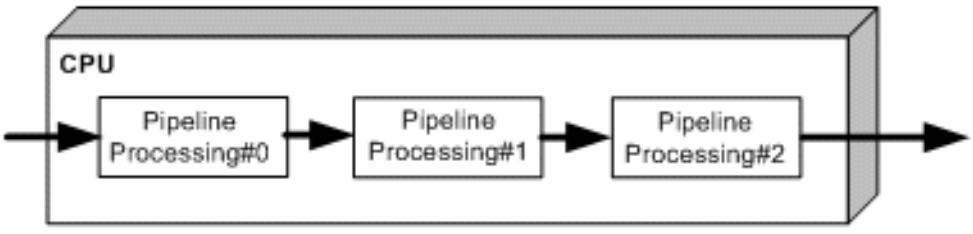

Fig. 5: Software Dataflow Deployment Model

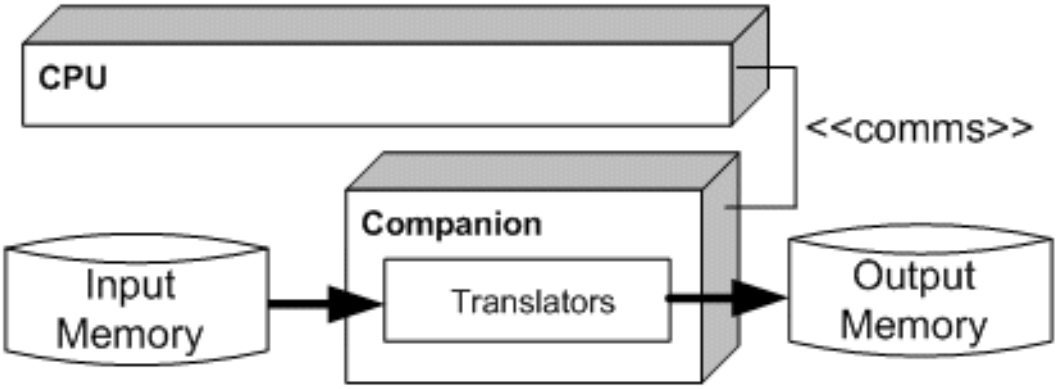

Fig. 6: Offload Dataflow Deployment Model

4.2 Stack

A variety of stack patterns exist in the STB architecture. These are represented as a specialized model of the UML Package Model.

Group Name: Stack

Patterns: Application and Control (App & Ctl), Middleware (Mid), Operating System (OS), Internet Protocol (IP) and Communications (Com) and Hardware Drivers (H. Drv).

Problem: Stacks are used in the STB to comply with software architectures defined by external standards (e.g. use of TCP/IP), to accommodate operating systems (e.g. Linux) and to provide abstraction.

Solution: Fig 7. represents the combination stack model used in the STB analyzed. These have differing characteristics

Consequences: The main constraint on the STB stacks is that they may have to comply with real-time constraints. For the communications stack this also includes bandwidth and latency.

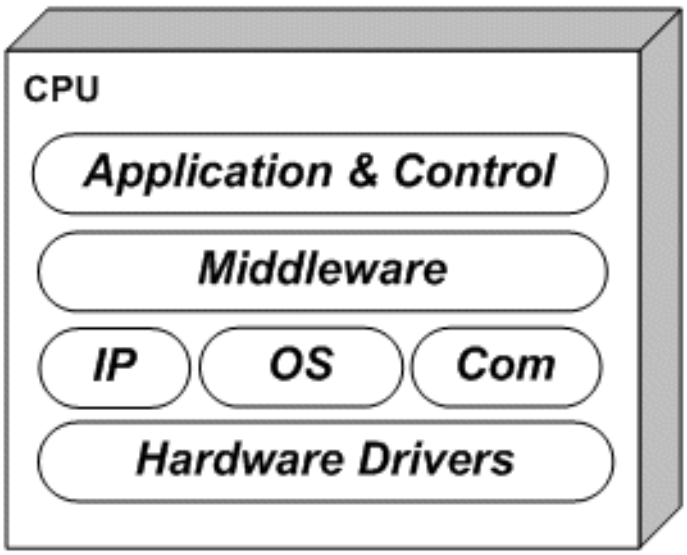

Fig. 7: Example Stack Model

4.3 Control

The STB requires a number of different control architectures. These are discussed next:

Group Name: Control

Patterns: Direct, Hierarchy, Repository and Distributed

Problem: In order to utilize the hardware IP and software modules of the STB some form of control architecture is required.

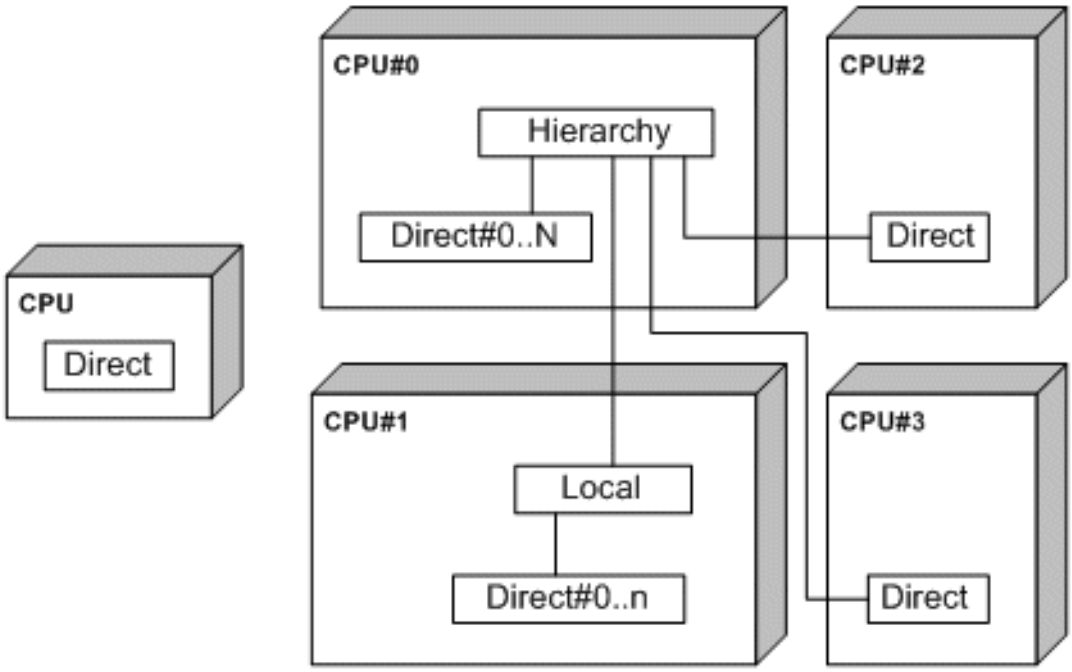

Fig. 8: Direct and Hierarchy Control Architectures

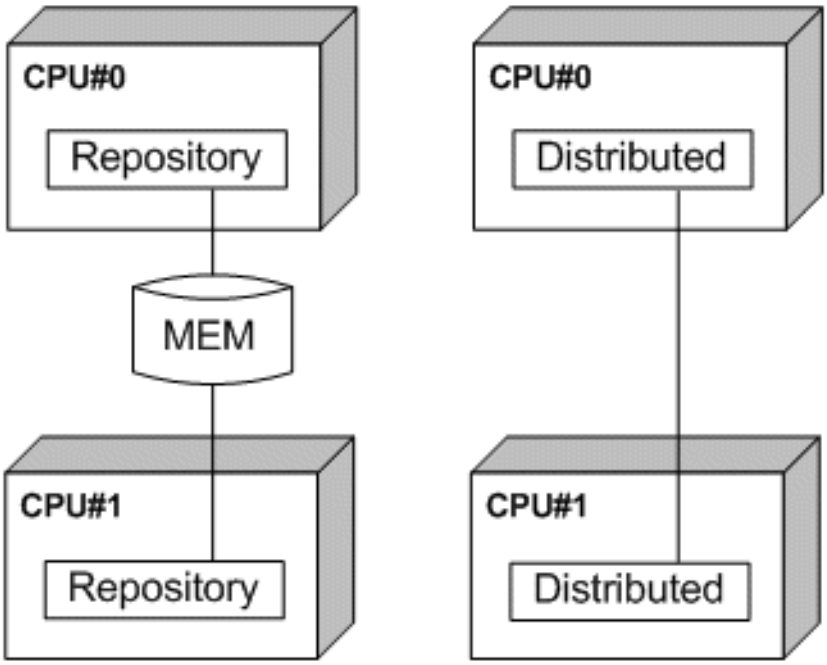

Fig. 9: Repository and Distributed Control Arch

Solution: Fig 8. provides a UML deployment model representing the Direct and Hierarchy control architectures. The Direct architecture is extensively used in the STB to control IP. In addition, the Direct control architecture is a key element of the Hierarchy control architecture. This provides centralized control of a number of Direct control architectures distributed among several CPUs. More specialized control architectures include the classical Repository and Distributed patterns. These are represented in Fig. 9.

Consequences: The architecture may have to work in real-time.

4.5 State Transition

Within the STB the State Transition architectures are primarily used to 'glue' the other patterns together.

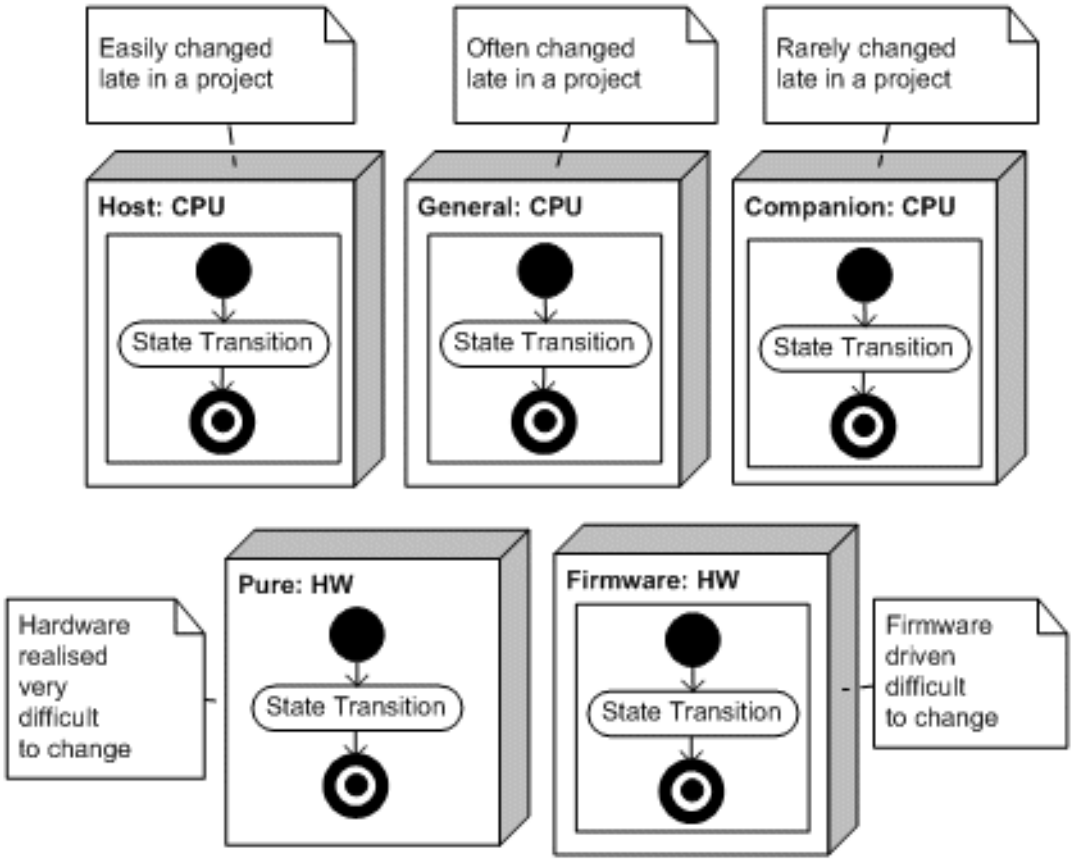

Fig. 10: State Transitions

Group: State Transition

Patterns: Host, General, Companion, Firmware, Pure Hardware

Problem: In order to provide a usable STB co-ordination of the other architecture reference patterns are required. In addition, a method of moving from one use case (e.g. Live Video to Video-On-Demand) is required.

Solution: Fig 10. provides a UML deployment model representing the state transition architectures. Five individual patterns have been identified. These include Pure and Firmware hardware; and Host, General and Companion CPUs. These have a number of different characteristics. One of these, their ease of change is represented in Fig. 10.

Consequences: Each State Transition pattern has its own characteristics. For example, changing the State Transitions of the pure hardware implementation will be difficult (if not impossible) towards the end of a project, whereas changes to the Host implementation would be achievable/expected throughout the project.

5. STB Pattern Analysis

Having identified the architectural reference patterns, this section provides an analysis of the usage of these patterns in a current STB design.

A summary of the results are provided in Fig 11. Note that for the control the Direct architectures that are part of the Hierarchy pattern are not double counted.

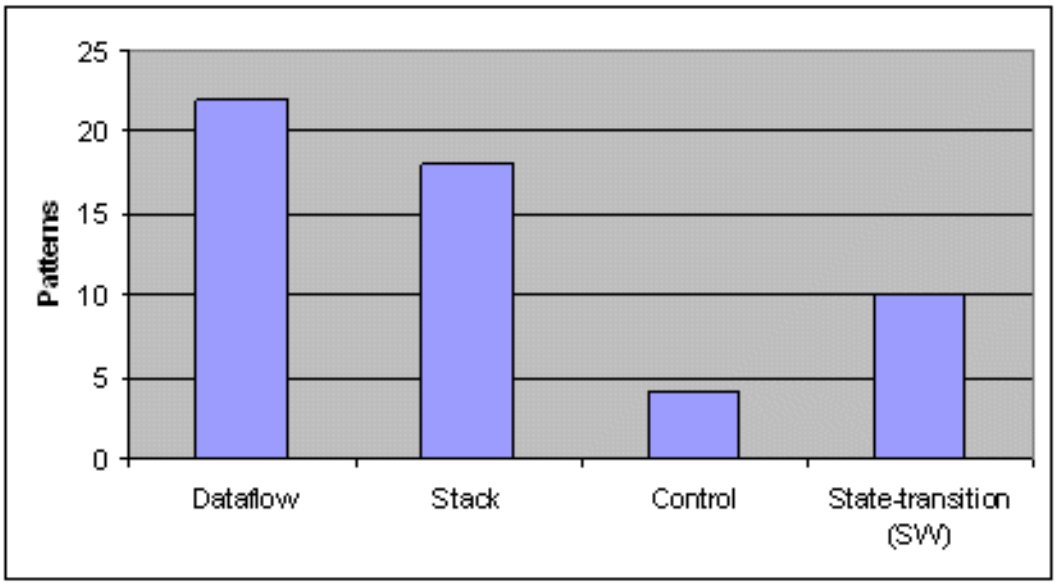

Fig. 11: Architecture Pattern Usage Summary

The Dataflow usage is broken down further in Fig. 12. The majority of the patterns are Hardware Dataflow. These partition further with the majority being single input/output types. One of the Hardware patterns was complex with 6 inputs, 2 outputs and an inter-dataflow feedback path. Another interesting observation was that most of the Dataflow architectures required real-time behavior.

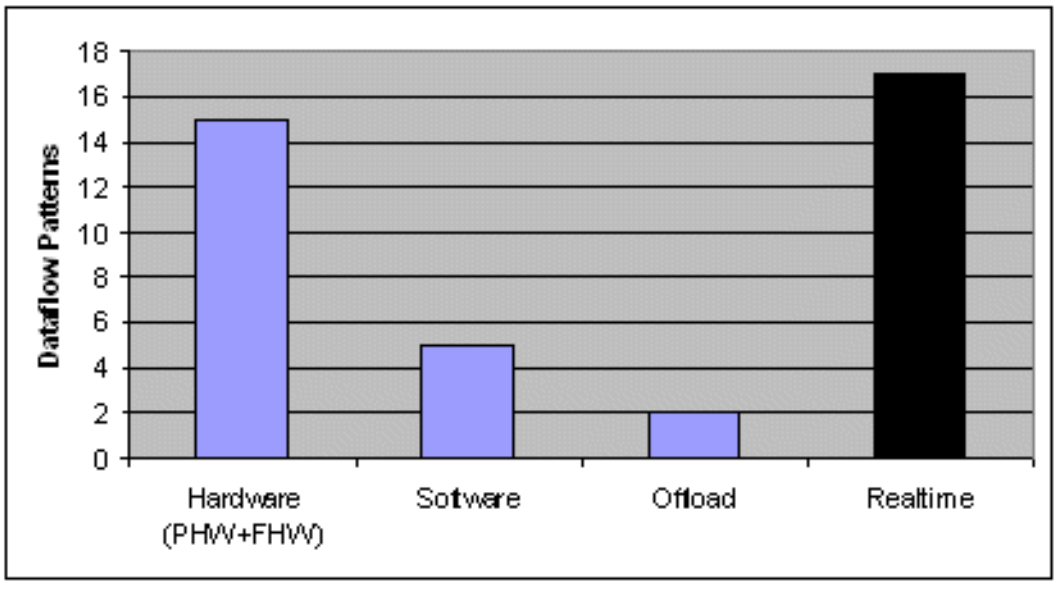

Fig. 12: Dataflow Pattern Usage Summary

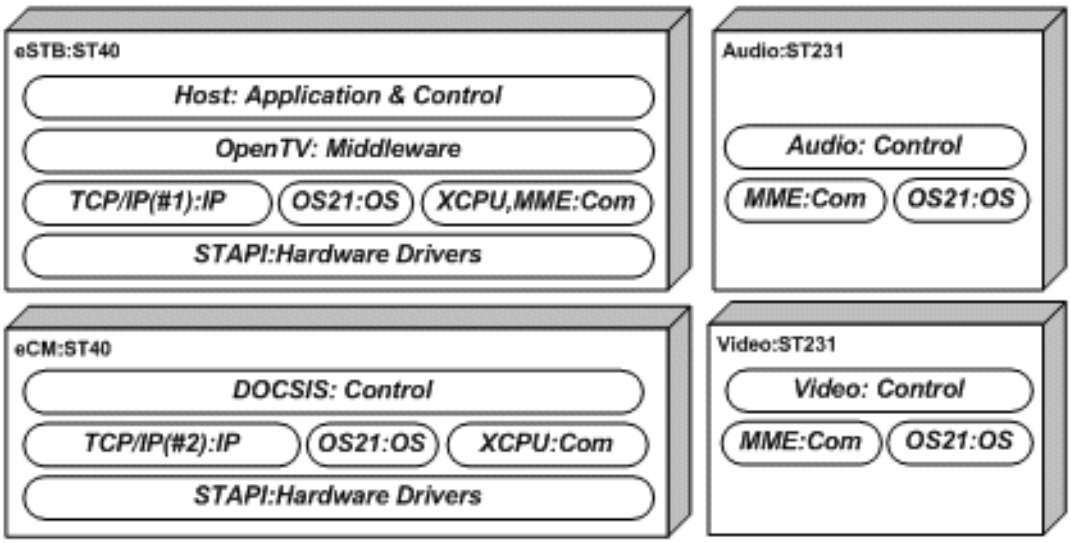

Fig. 13: Stack Pattern Usage Summary

Fig. 13 represents the Stack patterns used in the STB. This highlighted that two internally developed communications architectures were being supported. Further, the two IP stacks are from two different third party vendors.

The Control Pattern usage in the STB are summarized in Fig. 14. The primary control architecture when the STB is in run-time mode is a Hierarchy pattern. The Repository Control pattern is used during the boot process to determine the mode of the Box, e.g. deployment or service mode. The Distributed Control pattern is used to facilitate robustness scenarios. Here the eCM can detect eSTB faults and reset the eSTB. Conversely, the eSTB can determine eCM errors and perform the reset.

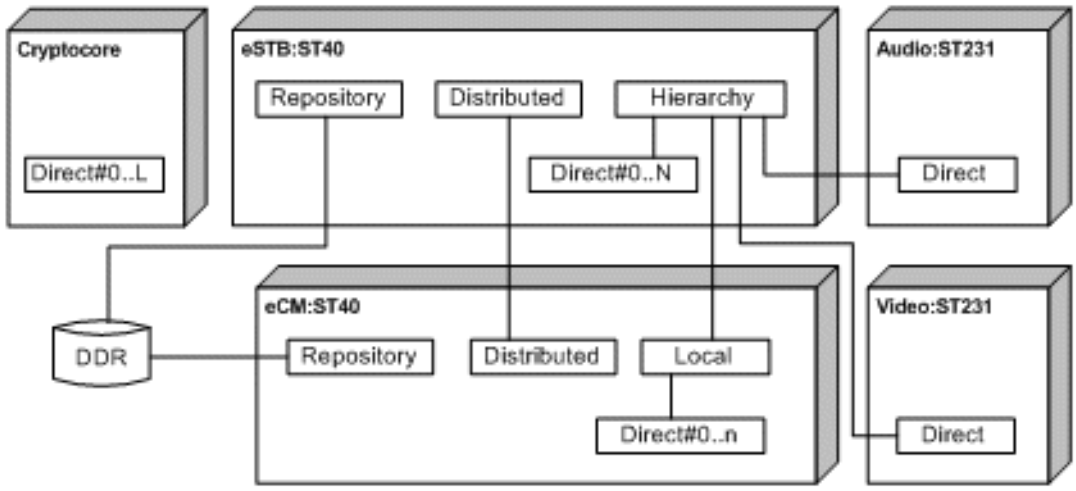

Fig. 14: Control Pattern Usage Summary

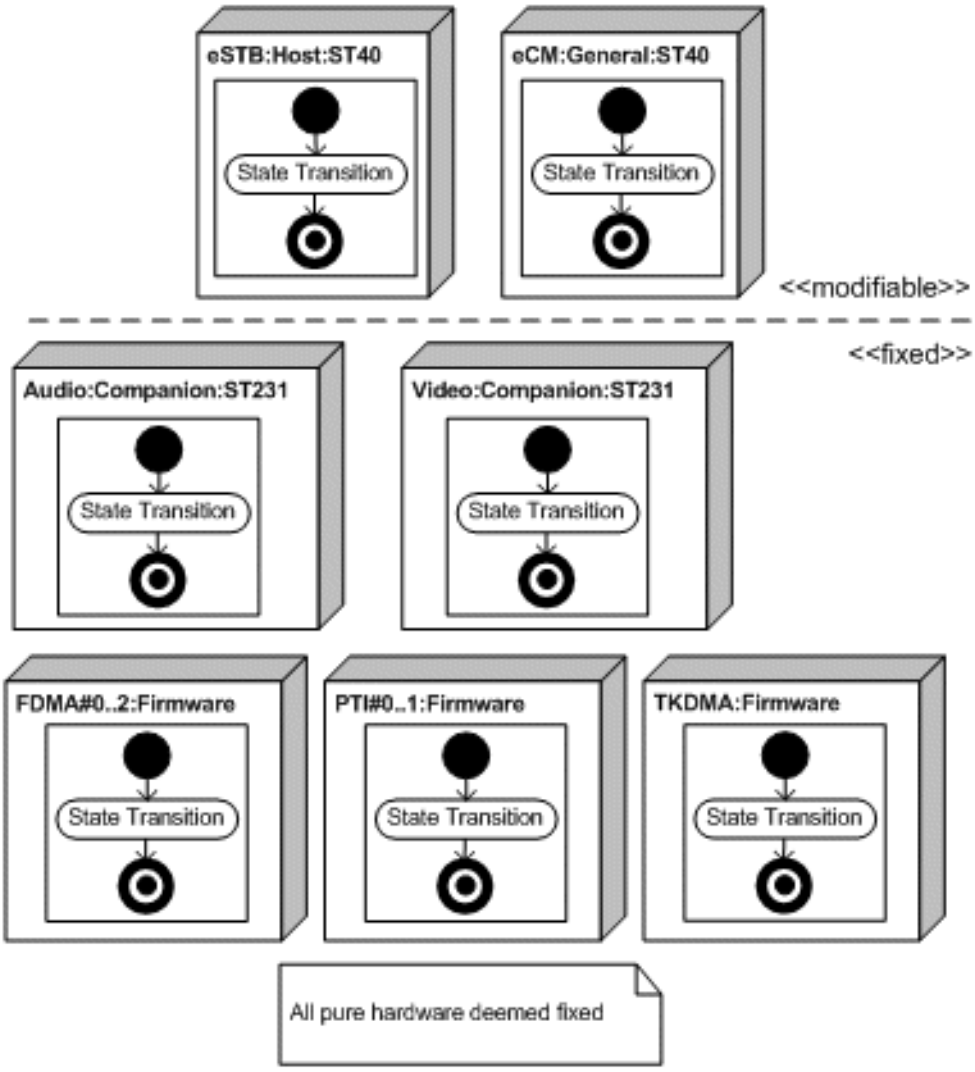

Fig. 15: State Transition Pattern Usage Summary

The State Transition Control architecture is represented in Fig. 15. Specifically, this provides the view during the integration stage of a project. Here only the Host and General state models can be changed. All the other State-Transition patterns are fixed. Of course, in 'show stopping' scenarios changes can be made to the Companion and Firmware software state models.

6. Discussion

The architecture reference patterns documented in this paper contributes to STB developments in three main areas: (i) improves understanding of the architecture, (ii) highlights technical improvements and anomalies, and (iii) provides useful project management information. These areas are now discussed in more depth:

Four primary architectural reference patterns have been identified for a current STB: Dataflow, Stack, Control and State Transition. In addition, each group has a number of individual reference patterns that total 18. A UML description of these reference architectures has been provided, see Section 4.

Subtle interaction exists between the patterns, e.g. Dataflow architecture requires a Control architecture which in turn may interact with State-Transition architecture. These pattern relationships have been documented, see Fig. 3.

The description and documentation (in UML) of the STB architectural patterns promotes their re-use, introduces a descriptive vocabulary, allows design abstraction and improves clarity and understanding of the STB.

A complete set of STB deployment model architectures have been analyzed and bisected to determine its contents in terms of the architecture reference patterns. This has highlighted opportunities for several technical improvements:

1. A complex dataflow based on 6 inputs and 2 outputs was identified. A more detailed analysis of this area highlights that it may be better served with a dedicated General CPU. This would reduce the complexity and processing burden of the Host software.
2. A number of real-time dataflow implementations are serviced by the Host CPU. These might be better served by another General CPU. This would reduce the valuable processing requirements on the Host CPU.
3. Two internal communications stacks are currently maintained. Combining these would reduce the maintenance/development burden.
4. Two third party IP stacks are used in the design (#1 and #2). Using the same IP stack could reduce licensing costs for the design.
5. Additional flexibility could be introduced to the control architecture by including the Distributed modules in the Hierarchy implementation.

These examples highlight the technical opportunities for improvements.

A number of architectural details have been determined through the investigation. These include quantitative values that document the use of: (i) over 50 hardware IPs, (ii) over 70 software modules, (iii) 15 hardware dataflow, (iv) 18 stack layers, (v) 6 main distributed software state-transition requirements, and (vi) real-time requirements.

This information assists the project manager in determining a number of characteristics. These include: (i) the scope of the STB project, (ii) engineering skills required, and (iii) partitioning of the work.

7. Concluding Remarks

This paper has identified architectural reference patterns for a leading edge STB design. These are represented as UML along with pattern details of name, problem area, solution and consequences. This work builds on, and extends the work of others in the general area of software engineering and architecture by defining specific STB architectural reference patterns.

This work also develops and extends our own work on STB architectural specification. Our previous work included the use of ‘views’ to partition the architectural specification, and UML to represent the architecture. The work presented in this paper analyses the UML representations providing additional and complementary information on the architecture.

The usual benefits associated with generating reference patterns: promoting re-use, common vocabulary, complex design description, system abstraction, and system clarity/understanding are achieved. In addition, the quantitative analysis of the use of these patterns provides useful insight to STB architecture. Very experienced STB architects could idiomatically identify the patterns that have been identified. However, no formal documentation exists for their definition.

Further, even experienced STB architects would have difficulty in providing accurate figures of the use of all the patterns in an STB, for example numbers of types of Dataflow used. This work begins to overcomes these shortfalls.

Having concise pattern information provides an opportunity to identify architectural improvements that can be made during the specification and design stages of development . These can (or have the potential to) lead to reduced design complexity, better resource use, reduced cost, and increased flexibility. In addition, architectural pattern analysis can be useful post-implementation. Here project decisions may be taken for milestone, timescale and manning resource issues that result in architectural compromises being made. Post-implementation, architectural pattern analysis can aid re-factoring in re-gaining architecture compromises and design deviations.

Project management activities may also gain from the use of architectural reference patterns in improving the accuracy of plans and reducing the risk of projects.

Both identifying architectural improvements and having more accurate plans with less risk contribute to reducing time-to-market for our STB products. Our future work in this area will include analysis of other STBs to extend and refine the architecture reference patterns.

In addition, the STB architectural reference patterns provide a concise representation that could be used for developing executable system level models (e.g. executable UML [8]) that can be used to ensure model consistency and move towards providing automated performance prediction.

Our future work in this area will include (i) more detailed analysis of the patterns (those identified already and 'new' patterns) and their characteristics, (ii) analysis of the influence of customer application and middleware on overall system analysis, and (iii) pattern trends (e.g. reduction in PHW/FHW and increase in software dataflow). Other related pattern areas for investigation include (iv) identification and documentation of anti-patterns, i.e. patterns that eventually lead to undesired or problematic architectures, and (v) analysis on non-patterns, i.e. architecture areas that do not align with the identified patterns (sometimes referred to as 'one-offs'), which can have a substantial impact on the project.

Acknowledgements

The authors would like to acknowledge valuable comments and support from Roger Shepherd, Bill Fletcher and Yann Garnier concerning aspects of this work.

References

[1] Gamma, E., Helm, R. Johnson, R., and Vlissides, J., 'Design Patterns: Elements of Re-usable Object-Orinetated Software', ISBN 0-201-63361-2, Addison-Wesley, 1994.

[2] Garlan, D. and Shaw, M., 'An Introduction to Software Architecture', Vol 1 Advances in Software Engineering and Knowledge Engineering, Singapore: World Scientific Publishing Company, 1993.

[3] Deaves, R.H. et al., 'Embedded Software Architecture Specification Developments in Support of SoC Design and Re-use', IPSOC 08, Dec 2008.

[4] Hofmeister, C., Nord, R., and Soni, D., 'Applied Software Architecture', ISBN 9-780201-325713, Addison-Wesley, 2000.

[5] Pilone, D., 'UML 2.0 Pocket Reference', O'Reilly, ISBN 978-0-596-10208-1, 2006.

[6] DOCSIS: Data Over Cable services Interface Specifications, http://docsis.org.

[7] STi7141:HD decoder for cable STB

[8] Mellor, S.J. and Balcer, M.J., 'Executable UML: A Foundation for Model-Driven Architecture', Addison-Wesley, ISBN 0-201-74804-5, 2002