Evolving System Architecture to Meet Changing Business Goals: an Agent and Goal-Oriented Approach

Daniel Gross & Eric Yu
Faculty of Information Studies
University of Toronto
{gross, yu}@fis.utoronto.ca

Abstract

Today's requirements engineering approaches focus on notation and techniques for modeling the intended functionality and qualities of a software system. Little attention has been given to systematically understanding and modeling the relationships between business goals and system qualities, and how these goals are met during architectural design. In particular, modeling must encompass changes to business goals over time and their effects upon a system's architecture. This paper reports on a case study, performed at a telecommunication company, that illustrates the decision-making process regarding architectural changes introduced into an existing switching system product. A notation including goals, strategic agents and intentional dependency relationships is used to support the architectural modeling and reasoning.

Keywords:
Goal, architecture, non-functional requirement, architectural evolution, knowledge-based design

1. Introduction

During architectural design, many of the quality aspects of a system are determined. System qualities are often expressed as non-functional requirements, also called quality attributes [1,2]. These are requirements such as reliability, usability, maintainability, cost, competitiveness, time to market and the like. Many of these originate at the business level, and are better viewed as business goals. Achieving business goals is crucial for system success. As business goals change, the system architecture needs to evolve to ensure continued satisfaction of business goals. Therefore, a systematic modeling framework needs to support linking business goals to architectural design.

Goal-oriented approaches, such as the NFR framework [3,4,5] that treats non-functional requirements as goals to be achieved during the design process, took a significant step in making explicit the relationships between quality requirements and design decisions. The NFR framework uses such goals to drive design [6], to support architectural design [7,8], and to deal with change [9]. While providing a systematic way to deal with the relationships between quality requirements and design, this approach has only limited support for dealing with the functional and structural aspects of the system under development. More recent approaches [8, 10] make a step to further incorporate functional and structural aspects into the design process

This paper proposes a strategic agent-oriented and goal-oriented approach that systematically relates business goals to architectural design decisions and architectural structures during software development and evolution.

This approach emphasizes goal modeling based on the observations that business goals that represent or give rise to non-functional requirements predominate during the architectural design deliberation process, and that changes in business goals may create a need to reevaluate and evolve the architectures of software systems. Goals serve as a guide in the search for design alternatives, and serve as criteria for choosing among them.

This approach uses the agent concept to model human organizations as well as technical components. The rationale for using agents for modeling social concepts is based on the observation that different stakeholders within the development and deployment organizations may have different business goals that they may wish to pursue. These differences may give rise to conflicting interests and rationales. By linking stakeholder goals to the design decision-making process, it becomes possible to express the positive and negative impacts of design decisions upon those goals during software development and evolution [8]. Agents enable the various interests within an organization to be expressed.

The rationale for using agents for modeling technical concepts is based on the observation that the computational elements within coarse-grained software structures, not unlike those within organizational structures, represent focal points for intentional properties, such as design goals and capabilities. Agent concepts lend themselves well to modeling and reasoning about the distribution of capabilities and allocation of responsibilities within a software system, and to show how computational elements are intended to contribute to the overall goals and objectives of the system and the business organization.

The approach uses the notion of strategic agents [15,16] based on the observation that designers of subsystems, concerned with achieving intended design goals, are at the

same time concerned with avoiding or at least mitigating vulnerabilities that might be imposed on them by design decisions taken within other subsystems. This approach models such vulnerabilities, which designers negotiate among themselves during the design process, and highlights how others are expected to contribute in achieving their respective subsystem design goals.

The approach is process-oriented, as it focuses on supporting an iterative decision-making process during design. Design goals are iteratively "reduced" to runtime structures. This is based on the observation that designers establish and refine architectural structures in an iterative manner, where structures first introduced establish coarse-grained partitioning of responsibilities, and iteratively refine to structures that are sufficiently fine-grained to guide implementation of the system.

Finally, based on the observation that designers often reapply previously known design solutions to achieve business- and system-related goals, this approach emphasizes the need to support capturing, generalizing and reapplying design knowledge. Previous design solutions can be sought, based on goals they met, tradeoffs they made, or system structures they created. This supports a knowledge-based approach to design.

The next section describes the modeling approach. Section three introduces the case study. Section four illustrates the modeling approach using the case study. Section five discusses the case study results, while section six concludes and points to future work.

2. An agent & goal-oriented approach

In order to relate business goals to the architectural decision-making process, and to the architectural structures during design, the modeling approach proposes the following main categories of features. Each category is represented as a separate view. All views are iteratively constructed during analysis and design.

- The *design process view* expresses how business goals relate to architectural choices and how changes in business goals invalidate architectural choices, and provides the basis for removing them to choose among alternative design options. This includes support for expressing alternative design paths, and relates alternative choices to the business and system goals that are traded-off against each other.
- The *structural view* provides an architectural description during design that expresses the principal roles played by architectural design elements within a system, and how roles are composed during the design process to arrive at the system design. Architectural elements are characterized by their capabilities, their expectations of other elements, and how they contribute in achieving system- and business-related goals. The notation of this view is taken from the strategic dependency model of the i* framework [15].

  This view provides architectural descriptions of the system at several levels of abstraction, and how these are related to each other during the design process. This includes expressing architectural structures at different stages of completion, together with a description of where architectural structures need further refinement through design decision-making.
- The *organizational view* identifies stakeholders and their goals, and expresses how they depend on each other and on the emerging system design to achieve their goals. This includes support for deducing during the design process how, and upon whom, design choices have an effect. Due to space limitations, this view is not diagrammed in this paper.

This approach also provides knowledge-based support by enabling capturing, storage, retrieval and guidance in reapplying relationships between goals and design elements, when similar goals need to be met during future design efforts.

The organizational view is used to capture the pertinent stakeholders and their business and system related goals. Goals related to functional abilities provide the basis for system requirements, while goals related to business and system qualities provide the basis for non-functional requirements. Goals from the organizational view can be used as a starting point when constructing the design process view.

The design process view is used to construct a goal graph during the development process. The goal graph is used to search for and generate alternative design solutions. Goals denoting functional abilities are refined to alternative design options. Goals denoting non-functional requirements (called softgoals) are used to systematically drive the search for alternative solutions and to determine how each alternative solution relates to pertinent business- and system-related qualities, and to their respective stakeholders described in the organizational view.

The structural view is constructed in accordance with refinements of the goal graph. Existing or new design elements introduced within the structural view are related to architectural decisions described in the goal graph. Alternative refinements provide the basis for searching and identifying refinements within the goal graph.

3. The case study introduced

The case study was performed during the fall of 1999 at a multi-national telecommunication company. We studied a project that intended to utilize WAP/WML[1] technology to

[1] WAP - Wireless Application Protocol, WML - Wireless Markup Language

provide Internet browsing and service provision capabilities to telephone sets [2], which would require architectural changes within their "flagship" switching system.

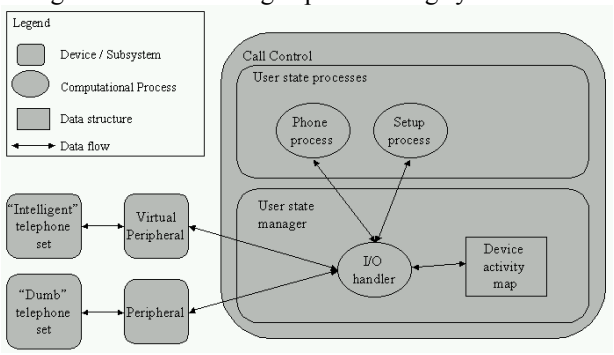

Figure 1: Telephone system architecture

Figure 1 shows the principal architectural elements of the telephone system analyzed during this study. The call control subsystem is responsible for all aspects of a telephone session: establishing calls; enabling features such as call forwarding, call waiting and the like; and terminating calls. All these are implemented by the "phone" process within the call control subsystem. Call control is also responsible for providing to users the set-up functionality for all desired services and features of the telephone set. The "setup" process within the call control subsystem implements this function. Call control is considered the main user application running within the switching system. Figure 1 also shows the peripheral component, which is a proprietary hardware device that connects proprietary telephone sets to the switching system; and the virtual peripheral components, which is software on standard PC-based hardware that emulates a peripheral device for "intelligent" telephone sets. These intelligent telephone sets are connected through a standard IP-based environment (such as an in-house LAN) to the virtual peripheral. The principal architectural question was to find where to place the WML browser component within the components or subsystems of the current telephone system architecture.

1. Within call control
2. Within the virtual peripheral[3]
3. Within the "intelligent" telephone set

It was assumed that the WML browser would be one of many future applications that would be made available on the telephone sets. The question discussed, therefore, was to find where future applications would reside within the telephone system, and what component or subsystem would control what application would interact at what time with the telephone set.

Figure 2 shows how moving from old to new business goals relates to the systems' architecture evolution path. In particular it shows:

- How business goals impact the architecture of a software system. This is shown by the impact links (straight arrows).
- How the current architecture may evolve to the different alternative architectures, each providing different support for adding and controlling new applications. This is shown through architectural evolution links (curved arrows).
- How alternative architectures resemble specializations of a common architectural pattern. This is shown through inheritance links (dotted arrows).

The "curved" links between the architectural alternatives in figure 2 show how "far" the proposed alternative architectures evolve away from the current set of business goals, toward the ideal appliance-based architecture that best achieves the new set of business goals.

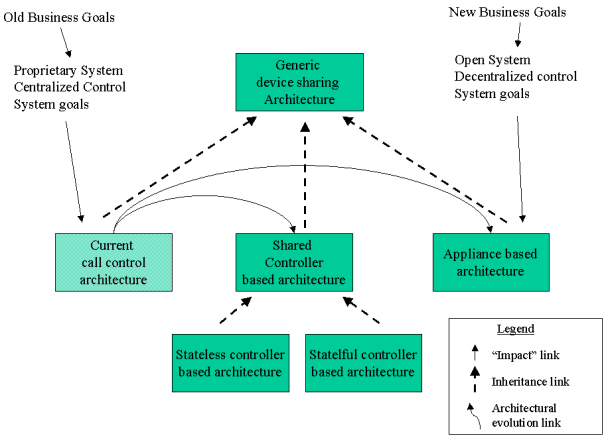

Figure 2: Architectural evolution paths

4. Illustrating the modeling approach

Figure 3 shows part of a goal graph produced during the case study. In the top half of the diagram are pertinent business goals that were voiced by stakeholders. The bottom half of the diagram shows design goals, and the architectural solution elements proposed.

The design goal service_creation_infrastructure_be-_WML_based, shown by the oval modeling element, denotes the overall functional goal to provide the current telephone system with a service creation infrastructure based on WAP/WML technology. This design goal is decomposed, through means-ends links, into the three alternative architectural design solutions. Means-ends links relate alternative design solutions (means) to design goals (ends). The design solutions proposed were master-

[2] Although WAP is used for mobile devices, the project considered its use for their non-mobile telephone sets.

[3] The "regular" peripheral, and the "dumb" phone devices did not support the addition of browser software.

_controlled_WML_based_infrastructure, shared_controller_based_WML_infrastructure, and appliance-_based_WML_infrastructure, each denoted by the hexagonal "design task" symbol.

The first architectural solution, master_controlled-_WML_based_infrastructure, is further decomposed, through task decomposition links, into design solution elements that describe how the WAP/WML architectural elements are added to the current switching system architecture. Since the switching system itself runs on Windows NT, this solution suggests adding the WML browser within the Windows NT environment outside of the switching system. It adds a Browser proxy component within call control as another user state process, and pertinent Browser state information within the user state manager subsystem of call control.

Figure 3 shows how all of these design elements relate through contribution or correlation links to business- or system-related quality goals. A contribution link shows that the design solution was chosen to achieve a business or system goal, while a correlation link denotes a side effect a design solution has on a goal. Both links can be either sufficiently or insufficiently positive, or to some extent, or sufficiently negative, to reject a design option. These degrees of contribution are denoted by the plus and minus signs, and dots within figure 3. They are used to evaluate design solutions through qualitative reasoning, and to direct the exploration of further design alternatives. Placing the browser within Windows NT, for example, has a sufficiently positive effect on reusing commercial software code, which reduces time to market. Placing browser proxy code within the user state process subsystem of call control allows maintaining architectural integrity, which in turn reduces time to market. Maintaining architectural integrity also aids in reducing the complexity of software code, which in turn reduces the cost of software development. However, placing the browser proxy within call control has a sufficiently negative impact on the architectural evolution goals for the switching system, by further entrenching the current architectural principles — rather than moving away from them or at least creating "evolvable" components that are reusable within next generation telephone systems.

In the middle of figure 3 we can see that for the shared_controller_based_WML_infrastructure design task two alternative design options were identified. This is shown by refining the design task into a corresponding design goal, WML_infrastructure_be_shared_controller_based, to denote that this design task, when further explored, raises further design alternatives. This design goal is then refined into the two alternatives: stateless_shared_controller_WML_infrastructure and stateful_shared_controller_WML_infrastructure.
Figure 3 shows how stateful_shared_controller_WML-_infrastructure is further refined, through task decomposition links, into design elements that are proposed as additions to the current switching system architecture. Each one of these design elements contributes to business and system goals. Figure 3 does not show all contribution or correlation links identified during the case study, but only the most pertinent ones for our discussion. For example, it shows that placing the Browser within the virtual peripheral contributes positively to the architectural evolution goal (namely the ability to provide "evolvable" state manager components to future switching systems). Adding the stream interpreter component, which is another design element, both affects adversely the performance of telephone sets attached to the system, and increases the likelihood of processing errors due to the difficulty of interpreting data streams without all the knowledge of its meaning, which resides within call control.

Let us now describe the structural view, and how it relates to the modeling elements in the goal graph. Figure 4 shows the structural view of the master_controlled_WML-_based_infrastructure design alternative, and how it relates to the generic device sharing architecture. The top part of figure 4 shows the structures defined for the device sharing architecture. These are the shared_device, the device_controller and the application agent. An agent represents a computational component during design. It encapsulates the design goals it achieves, the capabilities it provides, the capabilities it offers to other parts of the system, and the quality constraints it depends on. Figure 4 shows how the design of each agent depends on other agents through goals, tasks and resource dependencies. For example, the resource dependency data_stream between the application and the device_controller agent denotes the expectation of each agent to receive such a data stream from the other during runtime. The goal dependency exclusive_ownership_granted between the application and the device_controller agent denotes the expectation of the application agent that the device_controller agent will provide it with exclusive access to the data stream received from, and sent to the shared_device. This expectation expressed by the goal dependency is a design goal that is directed from the application agent toward the device_controller agent. The dependency does not prescribe how the device_controller agent will achieve this design goal, but only expects that it will be achieved during further design. Furthermore, the goal dependency denotes that it is up to the designer of the device_controller to decide how to achieve that design goal, and thus how to implement such exclusive ownership over data streams within the device controller component. The two softgoal dependencies, performance and minimize_processing_errors are quality attributes that the application agent depends on and wishes to have satisfied. These quality attributes serve as design constraints imposed by the application agent on the

device_controller agent in its exploration of design alternatives. Only those design alternatives that provide good performance and minimize processing errors are deemed acceptable to the application agent.

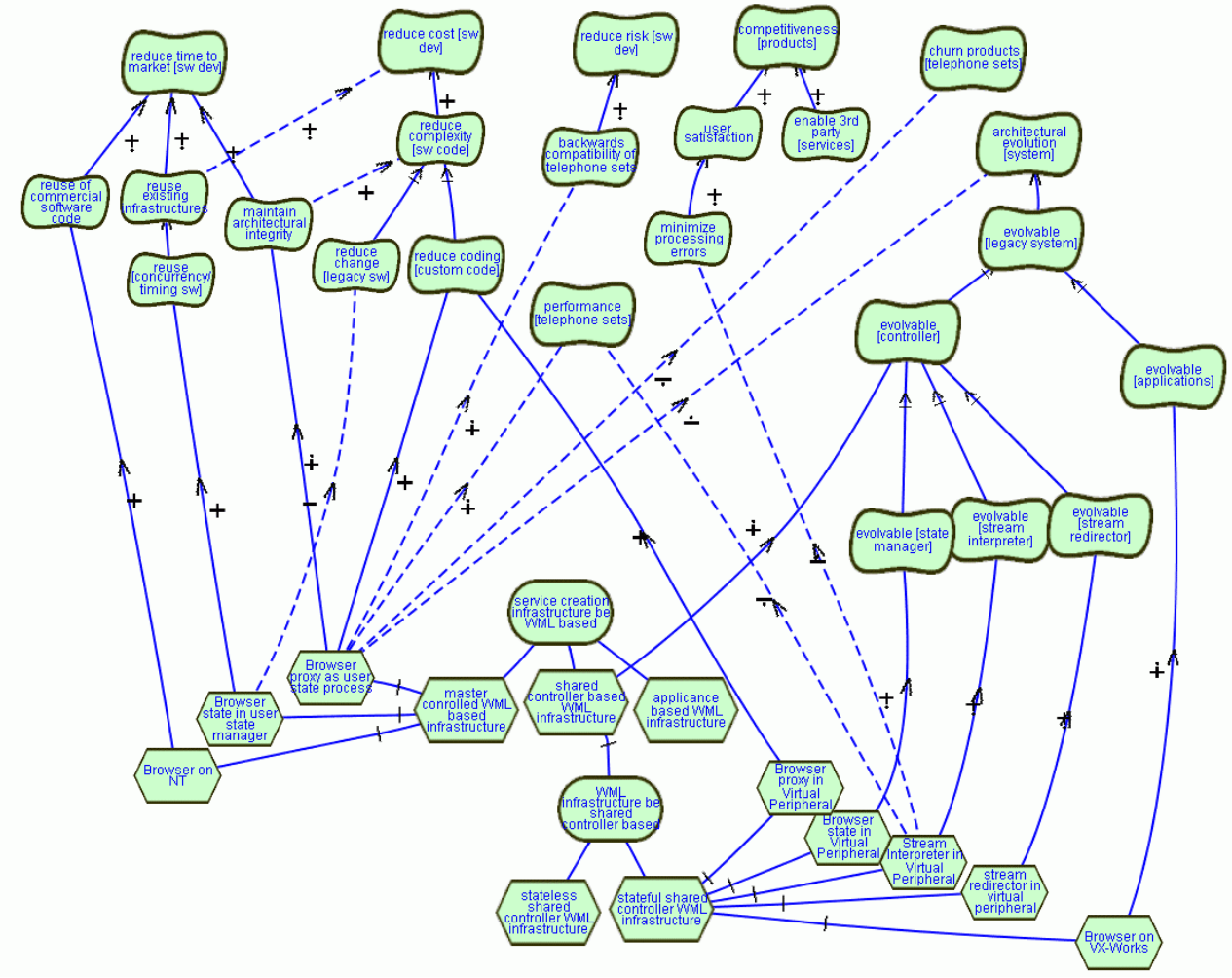

Figure 3: Goal graph denoting a design process with alternative architectural choices

For completeness, let us mention the send_state-_changed_commands task dependency between the device_controller and the shared_device. A task dependency denotes a design goal having constraints to a particular implementation. In our example, the device_controller agent expects the shared_device agent to send commands reflecting state change information, and expects that such commands will appear within the data stream.

This example highlights the difference between a structural view expressed in an agent-oriented manner and the common blocks-and-arrows diagrams. It shows how agents in conjunction with strategic dependencies are used to represent computational elements where design goals still exist and a design process still needs to unfold. Goal dependencies direct further design deliberations, while softgoals provide a means to constrain the selection of future proposed design alternatives in terms of quality requirements that need to be achieved within the system or the organization. Task dependencies provide a means to constrain design to exhibit particular functional features. Blocks-and-arrows diagrams represent final design choices and do not guide where and how further design choices need to be made.

The top part of figure 4 further shows that the device_controller agent is made out of three sub-agents, the command_interpreter, state_manager and data_stream_redirector agents, each performing part of the controller tasks. The command_interpreter scans the incoming data stream from the shared_device for commands to switch applications. The state_manager maintains a record of what application currently "owns" the shared device, and what application needs to be activated based on incoming commands. Finally, the data_stream_redirector agent directs the data stream between the shared device and the application that

currently has exclusive ownership. Any architecture that makes use of this generic device-sharing architecture needs to incorporate these three agents within its design. The bottom part of figure 4 shows how this device sharing architecture, and in particular how these three components within the device_controller agent, are allocated within the master_controlled_WML_based_infrastructure architectural alternative described in the goal graph in figure 3. It shows the call_control agent and its two sub-agents, the I/O_handler and the user_services agent. User_services is part of the user state processes subsystems and denotes all services available within call control. The user_services agent depends on I/O_handler to provide it with exclusive_telephone-_set_ownership and to receive a user_input_data. The I/O_handler in turn depends on the user_services to receive signal&response_data, which it directs to the telephone_set agent. The telephone_set depends on the I/O_handler to be shared, and to receive signal (i.e. commands in telephone set terminology) and response data streams.

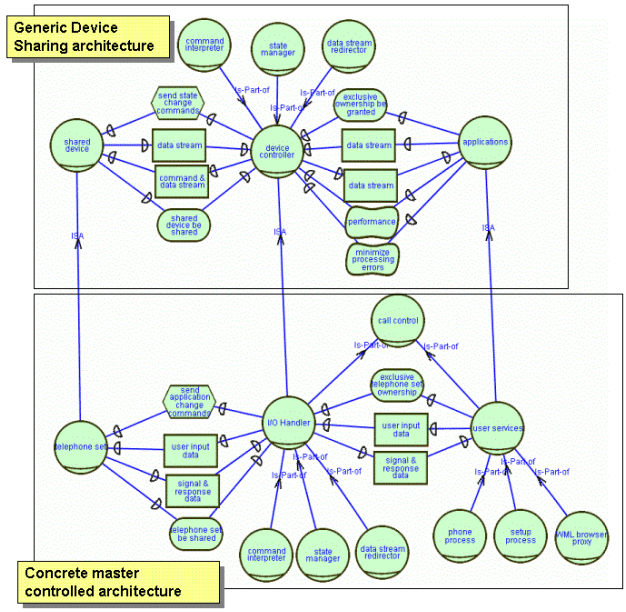

Figure 4: Abstract device sharing architecture and concrete master controlled WML infrastructure

Figure 4 further shows that the master_controlled-_WML_based_infrastructure architecture is a specialization of the generic device sharing architecture. The telephone_set agent is a shared_device, the I/O_handler is a device_controller, and user_services is an application. These relationships or "mappings" are denoted by "ISA" links. When mapping agents from the generic device sharing architecture to the more concrete master-controller architecture, the corresponding dependency links among agents may also be mapped. For example, the data_stream dependencies among the shared_device and the device_controller agents are created between their "counterparts", the telephone_set and the I/O_handler agents, albeit often renamed to fit the domain meaning of those dependencies. Mapping dependencies, through ISA links, from abstract to more concrete architectures is a design activity that needs judgment of designers. Unlike "conventional" inheritance, ISA links denote possible mappings available. Designers, in conjunction with the design process view, decide whether and what dependencies to map onto what agents, and what domain meaning and possible further constraining specializations to provide.

Sub-agents are also "inherited" from the abstract architectural view to the more concrete one. The state_manager, user_input_data_redirector and change_command_interpreter that are part of the I/O_handler are all inherited from the device-_controller agent. All these agents are allocated as described by design elements within the goal graph in figure 3, to achieve good performance and to minimize processing errors. Both good performance and minimizing processing errors are achieved by maintaining the centralized way that incoming signals from the telephone sets are interpreted, and by not having external computational elements performing similar tasks elsewhere. The other alternatives described in the goal graph allocate the state_manager, data_stream-_redirector and command_interpreter differently within the system to make different tradeoffs among these quality requirements, in particular to create an architecture that is more favorable to the architectural evolution goals. Finally, figure 4 shows that the WML_Browser_proxy agent is considered as a part of user services, since it is considered as an application, and in this architecture alternative, applications run within user services.

Let us now illustrate how the stateless and the stateful shared controller-based architectures are derived, through design steps described in the goal graph, from the generic device sharing architecture. We will see how goals and softgoal dependencies provide guidance in exploring alternatives during the design process. Each design task within the goal graph (denoted by the hexagonal symbol) refers to the structural view. Refining such tasks either through means-ends links or task-decomposition links into sub-tasks prompts the creation of additional components within the structural view. Goals and softgoals, both within the goal graph and within the structural views, guide the search for alternative design refinements.

The "legacy system with new extensions" structural view in figure 5 denotes an abstract architecture for extending legacy systems with new functionality. It defines two principal agents, the legacy_system and the new_system_extension agent. The goal and softgoal

dependencies between these two agents describe the design expectations each agent has of the other, which should be fulfilled during the subsequent design efforts. In particular, the view shows that the legacy_system agent is concerned with performance and maintain_architectural_integrity. On the other hand, the new_system_extension agent is concerned with creating "evolvable components" within the legacy system. These are components that are designed both to be implemented within the legacy system and to be reused within new systems ("next generation systems") that will comply with evolved system architectures.

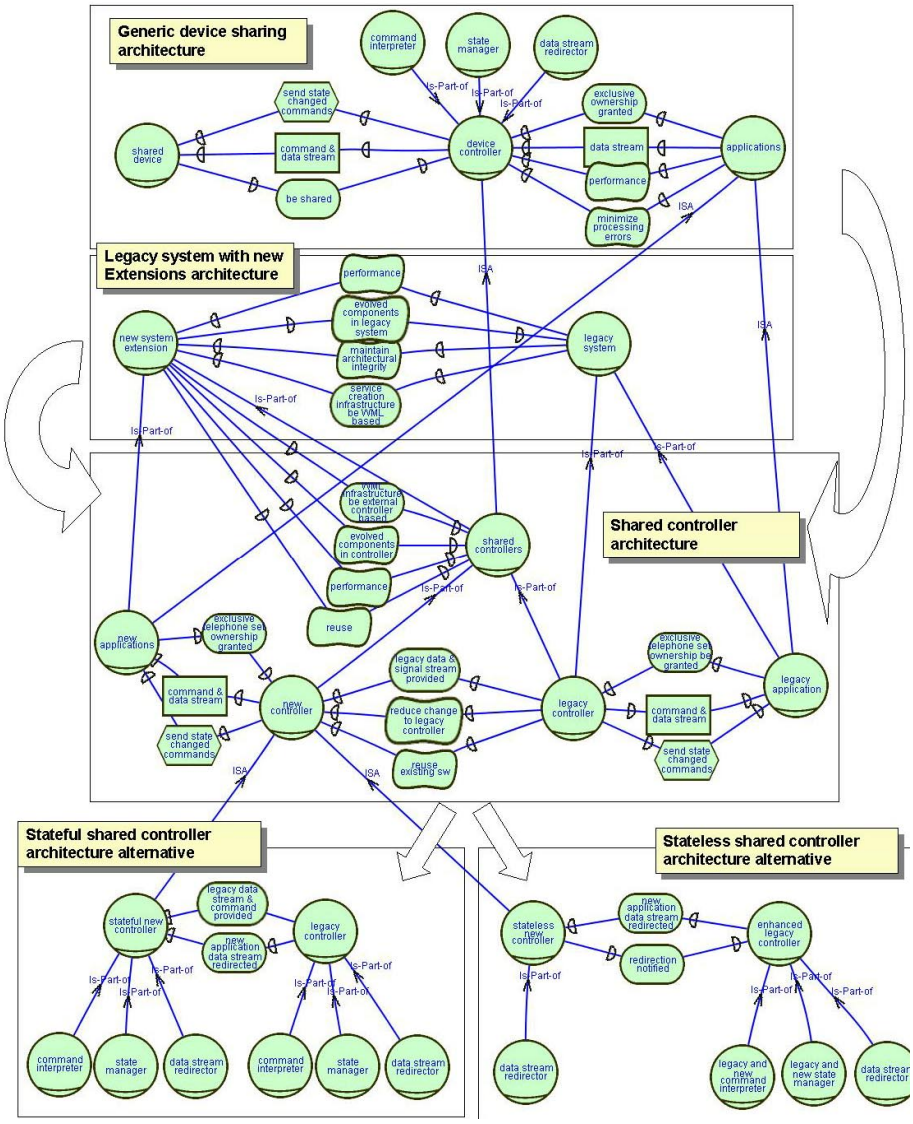

Figure 5: Shared-controller architecture alternatives

As discussed earlier, the goal graph in figure 3 shows that the WML_based_service_creation_infrastructure design solution can be achieved through three different architectures, each one based on a different specialization of the generic device-sharing architecture. Choosing this design task corresponds to consolidating the generic device-sharing architecture and the "legacy system with extension architecture" into the shared controller architecture structure described in figure 5. Note that choosing the shared controller architecture already achieves quality goals, such as creating evolvable controller components. This is shown in figure 3 through a contribution link from shared_controller_based-_WML_infrastructure to the evolvable [controller] softgoal. Having achieved this softgoal, further goals and softgoals are now identified that need to be achieved within the shared controller design, namely evolvable [state_manager], evolvable [stream_interpreter] and evolvable [stream_redirector] components. These are identified through the structure of the controller agent as shown in the structural view. The need to now achieve

these softgoals is shown by these softgoals and their contribution links in figure 3.

This shared-controller architecture introduces the shared_controller agent, which is composed of the new_controller agent and a legacy_controller agent. It further introduces two application agents, the new_application and legacy_application agents. The dependencies among new_application and new_controller, and legacy_application and legacy-_controller agents, correspond to the dependencies defined among the application and device_controller agents within the generic device-sharing architecture. These are inherited according to the inheritance links defined between the agents of both structural views. This shared-controller architecture provides architectural structure for any system that wishes to provide two focal points of control, for which legacy applications control is provided within the legacy system and for new applications control is provided within an additional component or subsystem.

A key question during the following design task is how exactly control is shared between the new_controller and legacy_controller agents such that the right tradeoffs are found among 1) maintaining the architectural integrity of the legacy system 2) optimizing performance of the system 3) providing further evolvable components 4) reducing change to the legacy controller and, finally, 5) reuse of existing software within the system. All these quality requirements are described in figure 5. The first ones (1-2) are inherited from the dependencies between the new_system_extension and legacy_system agents. The others (3-5) are represented by the dependencies between the new_controller and legacy_controller agents.

Figure 5 shows the structural view of the major components of the stateful shared controller and the stateless shared controller architectural alternatives. The goal graph in figure 3 shows how each alternative trades-off differently the above-mentioned quality requirements. Figure 5 shows in what way each alternative differs, in terms of allocating the device_controller sub-agents inherited from the generic device-sharing architecture between the new_controller and the legacy_controller. The stateful architectural alternative inherits all sub-agents to both the legacy and new controllers. The stateless architectural alternative inherits only the data_stream_redirector to the new controller (denoted by the stateless_new_controller), and makes it dependent on an enhanced version of the legacy_controller agent. This enhanced agent processes commands, manages the state of new applications and notifies the stateless controller of when to redirect and stop redirecting data streams. Figure 5, thus, demonstrates how dependencies among agents, in conjunction with the goal graph in figure 3, serve as criteria for searching and evaluating further alternative architectural designs.

5. Discussion

The requirements engineering research community has recognized the importance of goal modeling [11, 12, 13, 15,16,17]. However, goals are typically used to guide the establishing of requirements or designing of business processes, and serve as criteria for requirements completeness. The approach expounded in this paper recognizes the need to utilize goals during analysis and during the design process. This aids in representing the "unfolding" of the design decision process over time. Goals during design provide a focal point for unmet design requirements without (over) committing to particular design solutions.

This approach allows representing the many stages of completion through which design solutions move, and the stakeholder or system goals still to be addressed during further design. Goals denoting quality requirements provide an effective means for denoting constraints over further design efforts, and criteria for choosing among alternatives. Research in architectural design has given rise to notations that emphasize the compositional and behavioral aspect of coarse-grained system structures [14]. Quality attributes, or non-functional requirements, were identified as key driving forces, and rationales for different compositional system configurations. However, their treatment is often informal and not included in the architectural design notation. Both research communities recognize the importance of such links. However, little research has been done so far in bridging the requirements and architectural design gap.

The concepts of business goals and their relationships to functional and non-functional system requirements are not clear-cut. In this paper we took the stance that business goals are purposes that the business organization desires to achieve, both in the short and in the long term. Such goals are not necessarily tied to one product, but may relate to all product portfolios developed, maintained and evolved in the organization. Such goals originate from a variety of organizational and marketplace stakeholders. They are used to negotiate and determine functional and non-functional requirements, and, as we have seen, also architectural design decisions. For the purpose of modeling the architectural evolution process we did not feel the need to make a clear distinction between goals that originated from the business level and goals that represented system requirements. Both are seamlessly linked together through contribution (and correlation) links, and reside within the context of business and system development stakeholders. Precise boundaries might be needed for areas such as contracting and other legal purposes.

During the case study it was observed that the generic device-sharing architecture pattern, although being technical in nature, lent itself well to describing alternative business models pursued by the organization. System architectures that assigned the application and control components to one computational element in the target architecture pursued a centralized business model. Architectures that distribute these components, in particular among computational elements belonging to applications or devices under the jurisdiction of other organizations, pursue a decentralized and distributed business model. During the case study, the design decision to allow the organization's telephone sets to be operated by providers of competing switching systems would pursue both an open and decentralized business model.

An important feature of the "mapping" mechanism proposed is its ability to determine conformance among architectures. When changing the design of the concrete architecture, it can be determined whether it still conforms or violates one or more of the abstract architectures from it took over components and dependencies from. For example, figure 5 does not show how the WML browser proxy appeared within the switching system architecture. Two architectural patterns were, in fact, applied. One is the abstract architecture describing the WAP/WML reference architecture, which defines the WML browser agent, and the other describes how proxy components are utilized when wishing to split components among two spatial locations, while maintaining both parts as a logical computational unit. The structural view, in conjunction with the goal graph, allows representing such relationships among various "reference architectures" and how and why each contributes to the establishing of solution architectures.

6. Conclusion and future work

The case study highlighted the need for a modeling approach that supports modeling and analyzing how business goals relate to the architectural decision-making process, and how changing business goals give rise to alternative architectural choices and solution structures. It illustrated the need to describe the organizational stakeholders, their goals, and how these are affected by alternative choices during the design process. The case study highlighted the utility of goal modeling for expressing alternative design choices, and to serve as criteria during design deliberation. It showed the utility of using agents and goal concepts for modeling architectural solution structures. Agents were used to describe architectural distribution of capabilities, while goals were used as a focal point for expressing where within architectural structures further design choices needed to be made. Future work needs to focus on refining the integrated modeling framework, further formalizing the relationships among its diagrams, and investigating how its abstraction and mapping facilities can support knowledge-based tools that provide systematic design guidance and analysis support.

6. Acknowledgements

We like to thank the anonymous reviewers for their valuable comments; Tauba Staroswiecki, Douglas Anderson for their proof reading; and CITO, NSERC and our industrial research partner for their financial support.

7. References

[1] Boehm BW. Characteristics of software quality. North-Holland Pub. Co., Amsterdam New York 1978.

[2] Bowen TP. Wigle GB. Tsai JT. Specification of software quality attributes (Report RADC-TR-85-37).

[3] Chung L. Representing and using non-functional requirements: a process-oriented approach. Department of Computer Science University of Toronto. Toronto 1993.

[4] Chung L. Nixon B. Yu E. et al. Non-functional requirements in software engineering. Kluwer Academic, Boston 2000.

[5] Mylopoulos J. Chung L. Nixon B. Representing and using nonfunctional requirements: a process-oriented approach. IEEE Transactions on Software Engineering 1992; 18(6).

[6] Chung L. Nixon B. Yu E. Using quality requirements to systematically develop quality software. Proceedings of the 4th Int. Conf. on Software Quality. McLean, VA, USA. 1994.

[7] Chung L. Nixon B. Yu E. Using non-functional requirements to systematically select among alternatives in architectural design. Proceedings of the First International Workshop on Architecture for Software Systems. Seattle, Washington. 1995.

[8] Chung L. Gross D. Yu E. Architectural design to meet stakeholder requirements. In: Donohue, P(ed.). Software architecture. Kluwer Academic Publishers. San Antonio, Texas, USA 1999. pp 545-564.

[9] Chung L. Nixon B. Yu E. Dealing with change: An approach using non-functional requirements.

[10] D. Gross, E. Yu, From Non-Functional Requirements to Design through Patterns, Requirements Engineering. (to appear).

[11] A. I. Anton, "Goal-based Requirements Analysis." Proc.2nd IEEE Int. Conf. Requirements Eng. April 1996.

[12] A. Dardenne, A. van Lamsweerde and S. Fickas, Goal-Directed Requirements Acquisition, Science of Computer Programming, 20, pp. 3-50, 1993.

[13] S. Jacobs and R. Holten, "Goal-Driven Business Modelling – Supporting Decision Making within Information Systems Development," Proc. Conf. On Organizational Computing Systems, Milpitas, Calif. 1995, pp. 96-105.

[14] Shaw, M. and Garlan, D. (1996) “Software Architecture: Perspectives on an Emerging Discipline”, Prentice Hall.

[15] Yu E. Modelling strategic relationships for process reengineering. Ph.D. thesis, Dept. of Computer Science, University of Toronto. 1995.

[16] E. Yu Agent Orientation as a Modelling Paradigm, Wirtschaftsinformatik. 43(2) April 2001. pp. 123-132.

[17] E. Yu and J. Mylopoulos `Why Goal-Oriented Requirements Engineering', Proceedings of the 4th Int. Workshop on Requ. Engineering: Foundations of Software Quality (8-9 June 1998, Pisa, Italy). E. Dubois, A.L. Opdahl, K. Pohl, eds. Presses Universitaires de Namur, 1998. pp. 15-22